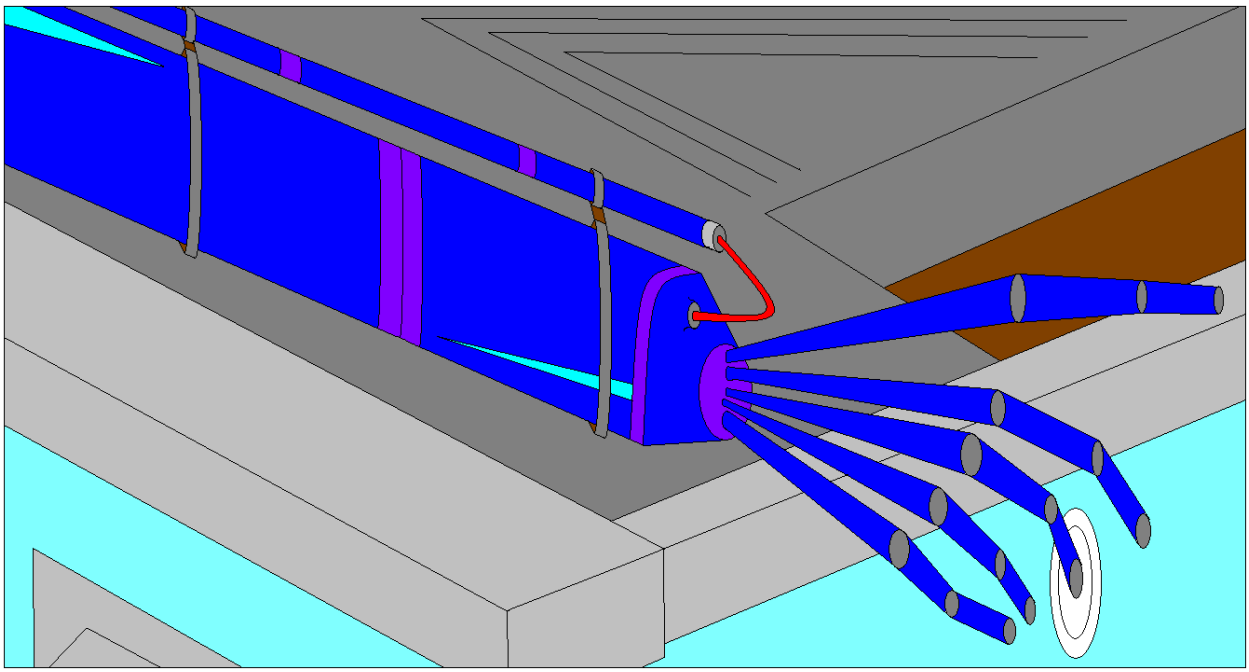
May 2023

Terminator 2 ROM L8.4



ROM L8.4 Manual Includes

Change Log

Technical Details

Appendix

ROM Image Changes

Terminator 2 L8.4

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.
L8.4	May 17, 2023	9A08	Fixed several long standing bugs. Improved Tournament Mode behavior. Adjustments for more challenging behaviors at the hunter-ship and super jackpot. Adjustment for skill-shot autofire time. Attract mode gun/start sound effects.

L8.4 Beta and Release Candidate ROM Image History

Revision	Date	Checksum	Info
L8.4	December 31, 2022	B408	Initial Beta containing: Pre-menu checksum report. Removed sound text 05 explicit stop. Fixed the L8.3 function ID overlap. Fixed 5-bank “Targets remaining” discrepancy issue. Added “Cannon 1 hit” adjustment. Fixed PAPA Lost Super Jackpot bug. Fixed Security Level inserts problem.
L8.4	January 6, 2023	9608	Fix Security Level inserts issue in beta B408 when both arrows should remain lit after Security Pass award. Fixed Bonus-X and Hold Bonus lamp issue at tilt lamps now get cleared.
L8.4	March 11, 2023	ED08	Added “SS Autofire Time” adjustment. Added “Super Jackpot” adjustment. Minor update to German text during game adjustments “sek.” changed to “Sek.”
L8.4	March 13, 2023	DF08	Prototype “Scattered Pixels” fix. Power-up init code performs a clear of memory and updates DMD board in a way more closer to what Indiana Jones L-7 does.
L8.4	March 19, 2023	D808	Investigative ROM for “Scattered Pixels” fix. Delay based on the configured value of A2.09 “Drop Target Timer”. During the delay the game is spending time clearing DMD memory.
L8.4	May 6, 2023	A608	Initial release candidate. Found to need update to L8.4 attract mode start-button so can play sounds during “Insert Coin”.

L8.4 Change Log

- Added pre-menu report of CPU U6 Checksum value to report the ROM stored checksum value.
- Removed the L8.3 Sound Test “Sound 05 Explicit Stop” code which was inadvertently in L8.3.
- Fixed the L8.3 Function ID Overlap issue, now using non overlapping ID for the L8.3 function.
- Fixed 5-bank “Targets Remaining” discrepancy w/lit targets when targets hit simultaneously.
- Added feature adjustment “Cannon 1 hit” to restrict hunter ship hits per cannon shot to 1.
- Fixed the PAPA Lost Super Jackpot bug when ball-popper hit during lengthy animation sequence.
- Fixed the Payback Time Security Levels inserts misalignment issue after Security Pass award.
- Fixed Bonus lamps at Tilt. Bonus-X lamps and Hold Bonus lamps cleared when player tilts.
- Added feature adjustment “SS Autofire Time” to allow adjustment of the skillshot autofire timer.
- Text correction, German “sek.” Changed to “Sek.” In menu system for abbreviated “seconds”.
- Added feature adjustment “Super Jackpot” to adjust how 5-bank lamps behave for super jackpot.
- Fixed auto-fire ball saver during first multiball so it always returns ball without loading cannon.
- Tournament Mode Enhancement: Database awards same awards for all players.
- Tournament Mode Enhancement: 5-bank lamp patterns for multiball start same for all players.
- Tournament Mode Enhancement: 5-bank single lamp for jackpot same for all players.
- Tournament Mode Enhancement: Video Mode same for all players.
- Tournament Mode Enhancement: Gun-trigger during attract mode shows previous game scores.
- Fix when Adjustment A.1 03 is set to “NO EX. BALL”, no EB animations and no lit-EBs.
- Fixed issue where start-button would play sound when zero credits and attract sounds are off.
- Added “L8.4” as new adjustment value for feature adjustment “Attract Mode”
- For L8.4 attract mode, start-button will play sounds when there are zero credits.
- For L8.4 attract mode, gun-trigger will play sounds.

Document Revision History

Revision 0.1, November 2022, Initial framework.
Revision <untracked>, Active document updates as L8.4 progressed.
Revision 1.0, May 2023, Completed document encompassing all of L8.4 content.

L8.4 Overview

The L8.4 update initially was intended to be a small update mainly focused on the last few bugs that have been reported in various online forums. As work proceeded and additional bugs were added to the L8.4 schedule, the L8.4 became more complicated as the L-8 code was more deeply investigated especially during the analysis into the 255-Hits issue, and the PAPA lost super jackpot issue. Eventually we determined the nature of the bugs, were able to reproduce the problems in a controlled environment, and developed fixes where needed for the various coding issues.

After bugs were fixed, some time was spent adding new player-requested enhancements to the L8.4. One complaint was that the super jackpot is too predictable. Players can easily get super jackpot solely based on the position of the cannon during its swing. For L8.4 a new "Super Jackpot" adjustment was added to provide different behaviors in the 5-bank target lamp movements during super jackpot, thus adding to the challenge and excitement of game play. In order to make game play fair, the new super jackpot lamp behaviors also ensure that all players in a multi-player game get to have the identical experience, so that one player doesn't get a more easier super jackpot attempt than another.

Since the PAPA super jackpot bug is considered a major reason that T2 was not a preferred tournament game, and since the bug is now deemed as fixed, the L8.4 efforts also focused on how the game behaves when Tournament Mode is enabled. With the L8.4 super jackpot code to give all players the same experience for new super jackpot attempts, it made sense to extend the idea into Tournament Mode for all game features so that all players in a multi-player game get the same experience, leveling the playing field, or so to speak. For L8.4 when game is in tournament mode, all players get the same database awards, video modes, 5-bank patterns for multiball and 5-bank lamp for jackpot.

Since a lot of tournaments forbid play of extra balls, initial consideration was given to also block extra balls when Tournament Mode is enabled however it was later noticed that an existing adjustment is already present. The "Max E.B. Count" adjustment can be set for no extra balls. It was noticed, however, when the game is set this way, the game still teases the player with extra ball lit and animations (while never actually giving it to the player) so L8.4 updates game behavior when the setting is for no extra balls. No longer will the extra ball be lit or its animation be shown. The idea is that when setting a game up for tournament mode, the operator should also set "Max E.B. Count" to no extra balls.

Lastly, for L8.4 some fun was added to attract mode, allowing prospective players to interact with the game during attract mode similar to Indiana Jones. The gun trigger can be pulled to trigger some sound effects and the start-button can also play some sounds (when zero or fractional credits). Like IJ, there is a quiet period involved to prevent overuse of this feature.

At 571 hours logged into L8.4 engineering efforts and countless hours by our community of beta testers, we are pleased to provide the pinball community with L8.4, setting a new standard for code quality, player experience and tournament mode action!

Contents

Terminator 2 L8.4 2

L8.4 Beta and Release Candidate ROM Image History 2

L8.4 Change Log 3

Document Revision History 3

L8.4 Overview 4

Pre-Menu report of the CPU U6 Checksum Value 9

Removal of the L8-3 Sound 05 Explicit Stop 15

Fix the L8.3 Function ID Overlap Issue 16

- Survey of CancelAllCallbacksIdMaskParameterBytes() 18
- Survey of SearchLinkedListAndMaskParameterBytes() 19
- ROM Survey of New Function ID 00B1 21
- Code Change For Fixing Function ID Overlap Issue 22

The Terminator 2 -- 255-Hits Issue 23

- Computer Representation of 8-Bit Numbers 23
- Analysis of L-8 Hits Remaining Code 24
  - Analysis of Older T2 ROMs 25
- Reproducing the 255-Hits Issue in L-4 27
  - Recipe for Reproducing the 255-Hits issue in L-4 29
- Reproduction Attempts of the 255-Hits Issue in L-8 33
- Summary of Observations from the 255-Hits Investigation 34
- Hunter Ship 5-Bank Target Logic 35
  - Case 1: Lit target hit followed by another lit target hit 39
  - Case 2: Unlit target hit followed by a lit target hit 43
  - Case 3: Lit target hit followed by an unlit target hit 48
  - Case 4: Unlit target hit followed by unlit target hit 51
- Hunter Ship 5-Bank Target Logic Summary 54
  - Hunter Ship 5-Bank Target Logic Summary Example 1 55
  - Hunter Ship 5-Bank Target Logic Summary Example 2 56
- Hunter Ship 5-Bank Target L8.4 Updates 56
  - Fix the lit-lamp/targets remaining discrepancy problem 56
  - Add feature adjustment to restrict target hits to 1 per cannon shot 58

Hunter Ship 5-Bank Target L8.4 Code Changes 60
The PAPA Lost Super Jackpot Bug 69
PAPA Lost Super Jackpot Investigation and Analysis 69
PAPA Lost Super Jackpot Investigation Summary 81
PAPA Lost Super Jackpot Code Fixes 82
PAPA Lost Super Jackpot Logic Update: State Check 0x0A Animation Check 82
PAPA Lost Super Jackpot Logic Update: Timeout 0x48 Flag Retention 85
PAPA Lost Super Jackpot Code Fixes 87
PAPA Lost Super Jackpot Code Fix: Add Animation Wait to 0092 Function 87
PAPA Lost Super Jackpot Code Fix: Add Multiball Conditional To Timeout Function 88
Payback Time Insert Misalignment Bug 90
Payback Time Summary 90
Payback Time Security Levels Bug 93
Payback Time Security Levels Bug Recipe 93
Security Levels Code Design 94
Security Levels Bug Fix 100
Security Levels Bug Fix – Code Changes 102
Post-Tilt Bonus Multipliers Lamp Bug 105
Terminator 2 Bonus and Bonus Multipliers Logic 105
Terminator 2 Bonus and Bonus Multipliers Logic Flowcharts 109
Terminator 2 Bonus and Bonus Multipliers Bug 110
Terminator 2 Bonus and Bonus Multipliers Bug Fix 111
HSTD Table Update Investigation 113
HSTD Full Problem Statement 114
HSTD Code Walk Through 114
Adjustable Autofire Timer 138
Autofire Mode 00, Skill Shot 139
Autofire Timer Code 140
Autofire Timer Adjustment 142
Autofire Timer Adjustment – L8.4 Code Changes 143
Autofire Timer Adjustment – L8.4 Code Changes for Adjustment Management 143
Autofire Timer Adjustment – L8.4 Code Changes for Autofire Timer Startup 146

Text String Corrections L8.4 148
Fix German Adjustment Menu Text For Abbreviated Seconds 148
Super Jackpot Lamp Movement Update for L8.4 149
Super Jackpot Lamp Design Considerations for L8.4 149
Super Jackpot Lamp Movement Possibilities 149
Super Jackpot Levels of Difficulty 149
Super Jackpot Multi-Player Considerations 149
Super Jackpot Variable Lamp Movements 150
Super Jackpot Lamp Enhancement for L8.4 151
Super Jackpot L8.4 Configuration 152
Super Jackpot Adjustment – L8.4 Code Changes for Adjustment Management 154
Super Jackpot Lamp Movement – Original L-8 Code 157
Super Jackpot Lamp Movement – Bookkeeping Statistics To Derive Variable Lamp Behaviors 158
Multiball Auto-Fire Problem Fix L8.4 168
Multiball Auto-Fire Code Analysis 169
Multiball Auto-Fire Code Correction 177
Tournament Mode Enhancements, L8.4 179
Tournament Mode Enhancement: Database Award, L8.4 179
Database Award Logic L-8 179
Per-Player In-Game Statistics Analysis L-8 187
Database Award Logic Enhancements for L8.4 189
Database Award Logic Code Update for L8.4 192
Tournament Mode Enhancement: Multiball Lamps, L8.4 197
Multiball Lamps Logic Enhancements for L8.4 197
Tournament Mode Enhancement: Jackpot Lamp, L8.4 207
Tournament Mode Enhancement: Video Mode, L8.4 212
Extra Ball Award Improvements for L8.4 230
Extra Ball Award Improvement: No Extra Ball Animation 233
Extra Ball Award Improvement: No EB Lit at Bonus-X 235
Extra Ball Award Improvement: No EB Info in Flipper-Button Status-Report 239
Extra Ball Award Improvement: No Lit Consolidation Ball 240
Skill-Shot 5-Bank Failure Investigation, L8.4 243

Attract Mode Improvements, L8.4 243
Attract Mode Improvement: Adding “L8.4” Attract Mode Setting 246
Attract Mode Improvement: Overall Changes for Start-Button and Gun-Trigger 247
Attract Mode Improvement: Overall Changes: Identifying available RAM Bytes 247
Attract Mode Improvement: Overall Changes: Identifying 2 Function ID values 248
Attract Mode Improvement: Overall Changes: Initialize the RAM bytes at attract mode start 251
Attract Mode Improvement: Start-Button Handling 254
Attract Mode Improvement: Gun-Trigger Handling 261
Appendix 267
Indexed Display Effect Functions 267
ROM Image Changes 270
Corrections to the L8.3 Document 286
L8.4 Test/Verification 287

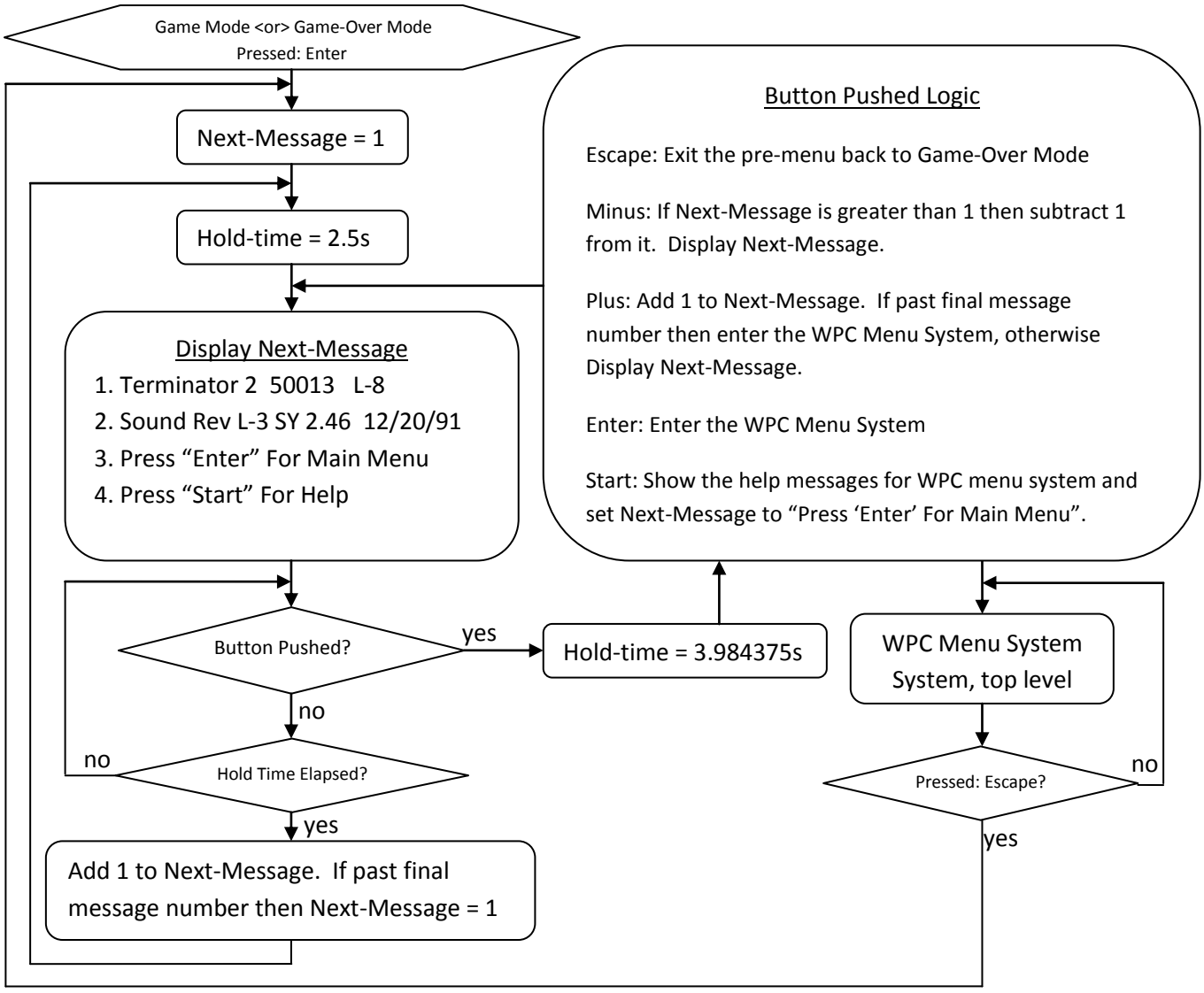
Pre-Menu report of the CPU U6 Checksum Value

For L8.4 the pre-menu report of game system information adds a new panel of information to report the CPU board U6 checksum value. This will report the actual value from ROM location 0x7FFEE and 0x7FFEF. The "U6 CHECKSUM" text is shown this way regardless of game language selection.

This panel of information was added to aid in troubleshooting and to help beta-testing for L8.4 go easier by allowing the checksum to be easily discovered to ensure correct/expected software is in use.

The report of checksum was added into the existing pre-menu report, as shown below. The new panel of information is shown between the regular report of game ROM revision and sound board information.

In order to update the pre-menu report with the checksum, the existing pre-menu code was analyzed. The existing logic for pre-menu is shown below.

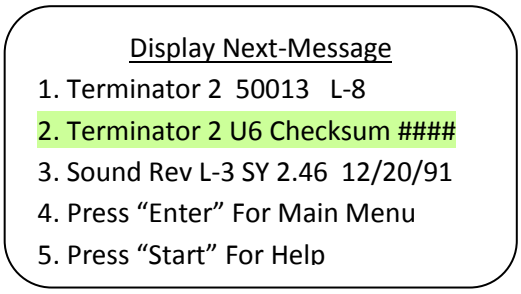


The L-8 code that manages the pre-menu loop is depicted below. This code is located at $41CE,3A, ROM offset 0x681CE. This is depicting the main loop of the flowchart with comments to describe what the non-depicted functions are also doing.

```
;---------------------------------------;-------------------------------------------
                                        ;
41CE: 34 02           PSHS  A           ;
                                        ;
41D0: 4F              CLRA              ; Reset button-press index to 0, no buttons pressed
                                        ;
41D1: 8D 74           BSR   $4247       ; ReportGameTitleIdCpuRevision()
41D3: 8D 4E           BSR   $4223       ; SystemInfoDisplayPause() Hold-Time wait/button pushed
41D5: 24 08           BCC   $41DF       ; C-clear? No menu buttons pushed, do sound board info
41D7: 8D 5B           BSR   $4234       ; CoinDoorButtonPressIndexValidate() Check button press
41D9: 25 3B           BCS   $4216       ; C-set? goto escape/enter/start button handler
41DB: 8D 63           BSR   $4240       ; CoinDoorMinusButtonPressCheck() Check minus press
41DD: 24 F2           BCC   $41D1       ; C-clear? Re-show game title and cpu revision report
                                        ;
41DF: BD 42 A3        JSR   $42A3       ; ReportSoundBoardRevision()
```

```
41E2: 8D 3F          BSR   $4223       ; SystemInfoDisplayPause() Hold-Time wait/button pushed
41E4: 24 08          BCC   $41EE       ; C-clear? No menu buttons pushed, do "PRESS ENTER"...
41E6: 8D 4C          BSR   $4234       ; CoinDoorButtonPressIndexValidate() Check button press
41E8: 25 2C          BCS   $4216       ; C-set? goto escape/enter/start button handler
41EA: 8D 54          BSR   $4240       ; CoinDoorMinusButtonPressCheck() Check minus press
41EC: 24 E3          BCC   $41D1       ; C-clear? Go back to game title and cpu revision report
                                       ;
41EE: BD 89 F8       JSR   $89F8       ; ReportTwoLinesSystemStatusInfo()
41F1: 80 FD 93                         ; String index 0xFD "PRESS "ENTER"" / "FOR MAIN MENU"
                                       ; 0x93 = line 1/centered
41F4: 8D 2D          BSR   $4223       ; SystemInfoDisplayPause() Hold-Time wait/button pushed
41F6: 24 08          BCC   $4200       ; C-clear? No menu buttons pushed, do "PRESS START"...
41F8: 8D 3A          BSR   $4234       ; CoinDoorButtonPressIndexValidate() Check button press
41FA: 25 1A          BCS   $4216       ; C-set? goto escape/enter/start button handler
41FC: 8D 42          BSR   $4240       ; CoinDoorMinusButtonPressCheck() Check minus press
41FE: 24 DF          BCC   $41DF       ; C-clear? Go back to display of sound board info
                                       ;
4200: BD 89 F8       JSR   $89F8       ; ReportTwoLinesSystemStatusInfo()
4203: 81 02 9F                         ; String index 0x102 "PRESS "START"" / "FOR HELP"
                                       ; 0x9F = line 1/centered
4206: 8D 1B          BSR   $4223       ; SystemInfoDisplayPause() Hold-Time wait/button pushed
4208: 24 C7          BCC   $41D1       ; C-clear? No menu buttons pushed, do title/CPU revision
420A: 8D 28          BSR   $4234       ; CoinDoorButtonPressIndexValidate() Check button press
420C: 25 08          BCS   $4216       ; C-set? goto escape/enter/start button handler
420E: 8D 30          BSR   $4240       ; CoinDoorMinusButtonPressCheck() Check minus press
4210: 24 DC          BCC   $41EE       ; C-clear? Go back to display "PRESS ENTER"...
4212: 1C FE          ANDCC #$FE        ; C-set = plus pushed, clear C and exit for main menu
4214: 20 0B          BRA   $4221       ; goto done
                                       ;
                                       ; Escape or Enter or Start was pressed, handle it here
4216: 81 04          CMPA  #$04        ; A is 0x04 when Start was pressed
4218: 26 05          BNE   $421F       ; If it wasn't Start then go to end
421A: BD 42 F6       JSR   $42F6       ; Display menu "help" information panels
421D: 20 CF          BRA   $41EE       ; Help is done, now goto "PRESS ENTER FOR MAIN MENU"
421F: 81 01          CMPA  #$01        ; Escape: returns c-set to return to Game-Over mode
4221: 35 82          PULS  A,PC        ; Enter: return c-clear to enter menu system
                                       ;
;--------------------------------------;-----------------------------------------
```

To add a new panel to the pre-menu information, the code was updated in a way as to retain the same logic except with an additional panel of information. The flowchart is effectively updated with a single new entry in the following section. All existing flowchart logic remains the same:



The updated code replaces the first portion of the loop with a jump to previously unused code at the end of bank $3A. The new/replaced code is highlighted in green. As shown below, the code now jumps

to a new function located at $7A0B,3A and the string “U6 CHECKSUM” is also added. The relocated code left enough room for this text string that will be used when displaying the new checksum panel of information. The new code will return to the $41DF,3A, bypassing this text string in the ROM image.

```
;--------------------------------------;-----------------------------------------
                                       ;
41CE: 34 02           PSHS  A          ;
                                       ;
41D0: 4F              CLRA             ; Reset button-press index to 0, no buttons pressed
                                       ;
41D1: 7E 7A 0B        JMP   $7A0B      ; Goto new logic to report game title then checksum
41D4: 55 36 20 43                      ; "U6 CHECKSUM"
41D8: 48 45 43 4B                      ;
41DC: 53 55 4D                         ;
                                       ;
41DF: BD 42 A3        JSR   $42A3      ; ReportSoundBoardRevision()
41E2: 8D 3F           BSR   $4223      ; SystemInfoDisplayPause() Hold-Time wait/button pushed
41E4: 24 08           BCC   $41EE      ; C-clear? No menu buttons pushed, do "PRESS ENTER”...
41E6: 8D 4C           BSR   $4234      ; CoinDoorButtonPressIndexValidate() Check button press
41E8: 25 2C           BCS   $4216      ; C-set? goto escape/enter/start button handler
41EA: 8D 54           BSR   $4240      ; CoinDoorMinusButtonPressCheck() Check minus press
41EC: 24 E3           BCC   $41D1      ; C-clear? Go back to game title and cpu revision report
                                       ;
41EE: BD 89 F8        JSR   $89F8      ; ReportTwoLinesSystemStatusInfo()
41F1: 80 FD 93                         ; String index 0xFD "PRESS "ENTER"" / "FOR MAIN MENU"
                                       ; 0x93 = line 1/centered
41F4: 8D 2D           BSR   $4223      ; SystemInfoDisplayPause() Hold-Time wait/button pushed
41F6: 24 08           BCC   $4200      ; C-clear? No menu buttons pushed, do "PRESS START”...
41F8: 8D 3A           BSR   $4234      ; CoinDoorButtonPressIndexValidate() Check button press
41FA: 25 1A           BCS   $4216      ; C-set? goto escape/enter/start button handler
41FC: 8D 42           BSR   $4240      ; CoinDoorMinusButtonPressCheck() Check minus press
41FE: 24 DF           BCC   $41DF      ; C-clear? Go back to display of sound board info
                                       ;
4200: BD 89 F8        JSR   $89F8      ; ReportTwoLinesSystemStatusInfo()
4203: 81 02 9F                         ; String index 0x102 "PRESS "START"" / "FOR HELP"
                                       ; 0x9F = line 1/centered
4206: 8D 1B           BSR   $4223      ; SystemInfoDisplayPause() Hold-Time wait/button pushed
4208: 24 C7           BCC   $41D1      ; C-clear? No menu buttons pushed, do title/CPU revision
420A: 8D 28           BSR   $4234      ; CoinDoorButtonPressIndexValidate() Check button press
420C: 25 08           BCS   $4216      ; C-set? goto escape/enter/start button handler
420E: 8D 30           BSR   $4240      ; CoinDoorMinusButtonPressCheck() Check minus press
4210: 24 DC           BCC   $41EE      ; C-clear? Go back to display "PRESS ENTER"...
4212: 1C FE           ANDCC #$FE       ; C-set = plus pushed, clear C and exit for main menu
4214: 20 0B           BRA   $4221      ; goto done
                                       ;
                                       ; Escape or Enter or Start was pressed, handle it here
4216: 81 04           CMPA  #$04       ; A is 0x04 when Start was pressed
4218: 26 05           BNE   $421F      ; If it wasn't Start then go to end
421A: BD 42 F6        JSR   $42F6      ; Display menu "help" information panels
421D: 20 CF           BRA   $41EE      ; Help is done, now goto "PRESS ENTER FOR MAIN MENU"
421F: 81 01           CMPA  #$01       ; Escape: returns c-set to return to Game-Over mode
4221: 35 82           PULS  A,PC       ; Enter: return c-clear to enter menu system
                                       ;
;--------------------------------------;-----------------------------------------
```

To complement the new code, a new function starting at previously unused ROM space $7A0B,3A, ROM offset 0x6BA0B is added to handle the display of checksum information. *Note, in L8.3 this portion of the ROM image was used by the T05 Explicit Stop code. In L8.4 this T05 Explicit Stop code was removed. To*

prevent gap of unused ROM bytes, this checksum code is placed here where the T05 Explicit Stop code had previously resided.

```
;--------------------------------------;-----------------------------------------
                                       ;
7A0B: 4D             TSTA              ; A = 0x00 = show initial "REV. L-8" message
7A0C: 26 15          BNE   $7A23       ; A != 0x00 = minus button pressed, show csum message
7A0E: BD 42 47       JSR   $4247       ; ReportGameTitleIdCpuRevision()
7A11: BD 42 23       JSR   $4223       ; SystemInfoDisplayPause() Hold-Time wait/button pushed
7A14: 24 0D          BCC   $7A23       ; C-clear? No menu buttons pushed, report checksum
7A16: BD 42 34       JSR   $4234       ; CoinDoorButtonPressIndexValidate() Check button press
7A19: 24 03          BCC   $7A1E       ;
7A1B: 7E 42 16       JMP   $4216       ; C-set? goto escape/enter/start button handler
7A1E: BD 42 40       JSR   $4240       ; CoinDoorMinusButtonPressCheck() Check minus press
7A21: 24 EB          BCC   $7A0E       ; C-clear? Re-show game title and cpu revision report
                                       ;
7A23: BD 7A 3B       JSR   $7A3B       ; ReportCpuChecksumMessage()
7A26: BD 42 23       JSR   $4223       ; SystemInfoDisplayPause() Hold-Time wait/button pushed
7A29: 24 0D          BCC   $7A38       ; C-clear? No menu buttons pushed, do sound board info
7A2B: BD 42 34       JSR   $4234       ; CoinDoorButtonPressIndexValidate() Check button press
7A2E: 24 03          BCC   $7A33       ;
7A30: 7E 42 16       JMP   $4216       ; C-set? goto escape/enter/start button handler
7A33: BD 42 40       JSR   $4240       ; CoinDoorMinusButtonPressCheck() Check minus press
7A36: 24 D6          BCC   $7A0E       ; C-clear? Go back to game title and cpu revision report
                                       ;
7A38: 7E 41 DF       JMP   $41DF       ; Jump back to regular message code, do sound board info
                                       ;
;--------------------------------------;-----------------------------------------
                                       ;
                                       ; ReportCpuChecksumMessage()
                                       ;
7A3B: BD 42 4C       JSR   $424C       ; ReportGameTitleString()
7A3E: 8D 03          BSR   $7A43       ; ReportGameChecksumValue()
7A40: 8D 41          BSR   $7A83       ; ReportGameChecksumText()
7A42: 39             RTS               ;
                                       ;
;--------------------------------------;-----------------------------------------
                                       ;
                                       ; ReportGameChecksumValue()
                                       ;
7A43: 34 36          PSHS  Y,X,B,A     ;
7A45: 32 7B          LEAS  $FFFB,S     ; Make room for 5 bytes on stack, csum and null
7A47: B6 FF EE       LDA   $FFEE       ; A get 1st checksum byte from ROM
7A4A: 8D 1C          BSR   $7A68       ; ConvertAHexToDAsciiNibbles()
7A4C: A7 E4          STA   ,S          ; Store first nibble of checksum
7A4E: E7 61          STB   $0001,S     ; Store second nibble of checksum
7A50: B6 FF EF       LDA   $FFEF       ; A get 2nd checksum byte from ROM
7A53: 8D 13          BSR   $7A68       ; ConvertAHexToDAsciiNibbles()
7A55: A7 62          STA   $0002,S     ; Store third nibble of checksum
7A57: E7 63          STB   $0003,S     ; Store fourth nibble of checksum
7A59: 6F 64          CLR   $0004,S     ; Null terminate the checksum string
                                       ;
7A5B: 31 E4          LEAY  ,S          ; Y gets pointer for checksum string
7A5D: BD 8A DA       JSR   $8ADA       ;
7A60: 81 01                            ; String index 0x101, "%SY" String pointer Y
7A62: 94 02                            ; 94=line2, 02=right justified and clear line first
7A64: 32 65          LEAS  $0005,S     ; Restore stack pointer
7A66: 35 B6          PULS  A,B,X,Y,PC  ;
                                       ;
                                       ;
;--------------------------------------;-----------------------------------------
```

```
                                        ;
                                        ; ConvertAHexToDAsciiNibbles()
                                        ;
7A68: 34 02           PSHS  A           ; Save a copy of the byte on the stack
7A6A: 8D 0B           BSR   $7A77       ; ConvertAHexLowNibbleToAscii()
7A6C: 1F 89           TFR   A,B         ; Put the low ASCII nibble into B
7A6E: 35 02           PULS  A           ; Restore the original byte from the stack
7A70: 8D 01           BSR   $7A73       ; ConvertAHexHighNibbleToAscii()
7A72: 39              RTS               ;
                                        ;
7A73: 44              LSRA              ; ConvertAHexHighNibbleToAscii()
7A74: 44              LSRA              ;
7A75: 44              LSRA              ;
7A76: 44              LSRA              ; Now A has the high nibble stored as a byte value 0..F
                                        ;
7A77: 84 0F           ANDA  #$0F        ; ConvertAHexLowNibbleToAscii()
7A79: 81 09           CMPA  #$09        ;
7A7B: 23 03           BLS   $7A80       ;
7A7D: 8B 37           ADDA  #$37        ; Make a..f values ASCII 'A'..'F'
7A7F: 39              RTS               ;
7A80: 8B 30           ADDA  #$30        ; Make 0..9 values ASCII '0'..'9'
7A82: 39              RTS               ;
                                        ;
;---------------------------------------;-----------------------------------------
                                        ;
                                        ; ReportGameChecksumText()
                                        ;
7A83: 34 36           PSHS  Y,X,B,A     ;
7A85: 32 74           LEAS  $FFF4,S     ; Make room for 12 bytes on the stack
7A87: BD 7A A0        JSR   #$7AA0      ; GetGameChecksumTextStringIntoX()
7A8A: D6 11           LDB   $11         ; Current bank into B
7A8C: 31 E4           LEAY  ,S          ; Y gets starting of the destination buffer on the stack
7A8E: 86 0B           LDA   #$0B        ; A gets value 11, # of bytes to copy from ROM to stack
7A90: BD 91 39        JSR   $9139       ; Copy 11 bytes from X to 12-byte buffer on the stack
7A93: 6F 6B           CLR   $000B,S     ; Null terminate the string on the stack
7A95: BD 8A DA        JSR   $8ADA       ;
7A98: 81 01                             ; String index 0x101, "%SY" String pointer Y
7A9A: 94 09                             ; 94=line2, 09=left justified and don't clear line first
7A9C: 32 6C           LEAS  $000C,S     ; Fix stack back to normal, done with 12-byte buffer
7A9E: 35 B6           PULS  A,B,X,Y,PC  ;
                                        ;
;---------------------------------------;-----------------------------------------
                                        ;
                                        ; GetGameChecksumTextStringIntoX()
                                        ;
7AA0: 8E 41 D4        LDX   #$41D4      ; Address of pointer to "U6 CHECKSUM" in this rom bank
7AA3: B6 17 4D        LDA   $174D       ; A gets Power-up status-byte. 00=good 01=checksum error
7AA6: 85 01           BITA  #$01        ; Test if we had a checksum error on power-up
7AA8: 27 03           BEQ   $7AAD       ; If A 01 bit is not set, skip to the end, all is good
7AAA: 8E 7A AE        LDX   #$7AAE      ; Address of pointer to "U6 BAD CSUM" in this rom bank
7AAD: 39              RTS               ; Done
                                        ;
7AAE: 55 36 20 42                       ; "U6 BAD CSUM"
7AB2: 41 44 20 43                       ;
7AB6: 53 55 4D                          ;
                                        ;
;---------------------------------------;-----------------------------------------
```

As evident by the code shown, the checksum report will use “U6 BAD CSUM” instead of “U6 CHECKSUM” when the software has detected a checksum error such as depicted below.

When the software, at power up, detects a checksum failure it sets the 0x01 bit in RAM address $174D. When this bit is set, the game will show the credit-dot and include the “U6 CKSUM ERROR” message in the test report.

This new L8.4 pre-menu report will also check for the 0x01 bit in RAM $174D and use this to report the fact that the game checksum is bad. This was done to reduce confusion when the pre-menu code is reporting checksum value while the calculated checksum does not match the checksum contained in the ROM image itself. In such case, instead of declaring the “U6 CHECKSUM” it will explicitly report “U6 BAD CSUM” along with the checksum value contained in the ROM image. *It will not report what the checksum ‘should’ be. It only reports the 2 checksum bytes currently stored in the ROM image.*

These strings “U6 CHECKSUM” and “U6 BAD CSUM” are used regardless of the language adjustment value (adjustment A.1 21).

Removal of the L8-3 Sound 05 Explicit Stop

For L8.4 the code depicted in L8.3 as “Sound 05 Explicit Stop” has been removed. This returns the sound test code back to L-8 where the sound test advancement from T05 (Database background) to T06 (100K award) no longer has special code to ensure the T05 sound is stopped. *Refer to “Sound 05 Explicit Stop” section of the L8.3 documentation for historical details on this code change*.

For L8.4 this change will un-do the L8.3 code changes, effectively restoring code back to original L-8 at the following two locations:

- $6D95,3A JMP instruction returned back to original jump point, and
- $7A0B,3A Function/code is returned back to unused/available ROM region.

The $6D95,3A, ROM offset 0x6AD95, ROM change in L8.4 is depicted below, restoring the original L-8 JMP instruction:

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; 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()← L8.3 code
6D95: BD 6D F3        JSR   $6DF3        ; StopCurrentSound()       ← L-8, L8.1, L8.2, L8.4 code
6D98: 81 02           CMPA  #$02         ;
6D9A: 35 82           PULS  A,PC         ;
                                         ;
-----------------------------------------;------------------------------------------------------
```

The new function at $7A0B,3A that was added in L8.3 is no longer included in L8.4. The function that is removed from L8.4 is depicted 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 06           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 16 bytes starting at $7A0B,3A, ROM offset 0x6BA0B are no longer needed since the code at $6D95,3A no longer jumps here. These bytes are now considered unused ROM region in bank $3A. As depicted in the L8.4 "Pre-Menu report of the CPU U6 Checksum Value" feature, this location is repurposed for code used to support the display of the pre-menu game checksum report.

Fix the L8.3 Function ID Overlap Issue

The L8.3 software documentation, "Multiball-Start Drop-Target Action Adjustment Code Analysis", describes how the L8.3 code ended up using a function ID value for new drop-target 'up' logic that was already being used during ball-search routine. This can potentially lead to unexpected behavior when subsequent code goes to check for the running function or cancel the running function based on its function ID value. For L8.4 this overlap issue is corrected. Below is the analysis and discovery of the new function ID value that will be used to remove the overlap issue identified in L8.3.

Below is a list of some of the applicable function IDs and location of each function with the conflicting ID, 00B9, highlighted:

Function ID	Function WPC Address	Function Purpose	ROM
00 B2	$45D7,31	Kick a ball out of shooter lane and checks shooter-lane switch	L-8, L8.1, L8.2, L8.3
00 B3	$6816,31	Reset drop-target up and checks drop-target switch	L-8, L8.1, L8.2, L8.3
00 B5	$4A77,31	TBD, Called when left-loop lock shot is hit	L-8, L8.1, L8.2, L8.3

00 B6	$4A28,31	TBD, Called when right-loop is hit	L-8, L8.1, L8.2, L8.3
00 B7	$7894,3B	Drop-Target Down, New Scheduled Function in L8.3	L8.3
00 B8	$6C80,31	TBD, Called as part of multiball startup	L-8, L8.1, L8.2, L8.3
00 B9	$78B7,3B	Drop-Target Up, New Scheduled Function in L8.3	L8.3
00 B9	$60B9,3B	Ball-Search 3.5 second wait and drop-target scheduler	L-8, L8.1, L8.2, L8.3

As shown, the issue is that in L8.3 there are two different functions that are both using the ID 00B9. The ball-search code uses a function with ID 00B9, and the new L8.3 code elected to use function ID 00B9 in the drop-target up function.

For L8.4, the code was surveyed for an available function ID that is not currently in use. The table below shows the results of such survey, initially looking for an available function ID that is not already referenced in bank $31. Once an unused function ID is identified in bank $31 then the rest of the ROM will be checked to make sure the function ID is not in use anywhere in the ROM image.

Surveyed Function ID	Function WPC Address	Function Purpose
00 B0	$48FA,31	Load the Gun countdown timer handler function.
00 B1	-	Function ID 00 B1 not referenced anywhere in bank $31
00 B4	$61EF,31	Scheduled after gun solenoid is fired, possibly checks for hunter ship hit.
00 BA	$45F0,31	Scheduled while handling gun-shooter switch, related to hunter ship/MB.
00 BB	$4609,31	Scheduled while handling gun-shooter switch, related to hunter ship/MB.
00 BC	$60CA,3B	Referenced in bank $31 as a function ID. Details TBD.
00 BD	$6675,31	Referenced in bank $31 as a function ID. Details TBD.
00 BE	$634C,31	Referenced in bank $31 as a function ID. Details TBD.
00 BF	$6553,31	Referenced in bank $31 as a function ID. Details TBD.

As shown above, the ID 00B1 is not immediately cited in bank $31 as a callback function. With the notion that this could be an unused function identifier, a full search of the entire ROM was done, using the various function signatures, to see if ID 00B1 is cited as a function ID. Below are the known functions that reference function ID numbers and below is the results of the search for function ID 00B1 being used anywhere in the L-8 ROM image.

Function Name	L-8 Function Usage Signature	Description	Search Result
ScheduleFunctionStart()	BD 8B 77 xx xx yy yy yy	Schedules function ID xx xx to start at WPC Addr yy yy yy	No occurrence of: BD 8B 77 00 B1
SearchLinkedListForId()	BD 86 90 xx xx	Searches for scheduled function ID xx xx	No occurrence of: BD 86 90 00 B1
CancelScheduled CallbackFunction()	BD 86 9E xx xx	Cancels scheduled function ID xx xx	No occurrence of: BD 86 9E 00 B1
UpdateCurrentRunningSchedule	BD 86 AC xx xx	Sets currently running	No occurrence of:

FunctionIDParameterBytes()		function ID to xx xx	BD 86 AC 00 B1
TBD()	BD 86 BA xx xx	TBD, where xx xx is function ID	No occurrence of: BD 86 BA 00 B1
CancelScheduledCallbackID ParameterBytes()	BD 86 D0 xx xx	Cancels scheduled ID xx xx	No occurrence of: BD 86 D0 00 B1
CancelAllCallbacksIdMask ParameterBytes()	BD 8A 9A xx xx yy yy	Cancels scheduled functions matching ID pattern of xx xx bitwise-and yy yy	No occurrence of: BD 8A 9A 00 Bx, See further analysis, below
SearchLinkedListAndMask ParameterBytes()	BD 8A AA xx xx yy yy	Searches for schedule functions matching ID pattern of xx xx bitwise-and yy yy	No occurrence of: BD 8A AA 00 Bx See further analysis, below
AddLinkedListEntry()	BD 8B 3D xx xx yy yy yy	Adds function yy yy yy to linked list as ID xx xx	No occurrence of: BD 8B 3D 00 B1
ScheduleFunctionStart()	BD 8B 77 xx xx yy yy yy	Schedules function yy yy yy ID xx xx	No occurrence of: BD 8B 77 00 B1
TBD()	BD 8B 9D xx xx yy yy yy	TBD, where xx xx is ID and yy yy yy is addr.	No occurrence of: BD 8B 9D 00 B1
ScheduleFunctionCallback()	BD 8B C3 xx xx yy yy yy	Schedules function yy yy yy ID xx xx	No occurrence of: BD 8B C3 00 B1
TBD()	BD 8B F7 xx xx yy yy yy	Schedules function yy yy yy ID xx xx	No occurrence of: BD 8B F7 00 B1

The inspection, above, shows there are no specific references to ID 00B1 as a function ID, thus adding to the confidence that the ID 00B1 can be used to resolve the ID conflict from L8.3. To further this analysis, the functions that accept ID and Mask values to cover multiple function IDs are surveyed and reported below.

Survey of CancelAllCallbacksIdMaskParameterBytes()

This function takes as parameters function ID xxxx and mask yyyy. This function allows a range of function IDs to be cancelled as a group. This might be useful when code has a group of functions that might be running and they all need to be ended at the same time. For example, this could be done at end of a timed feature where certain playfield or display sequences are running and need to be cancelled at the same time.

To ensure the proposed function ID 00B1 safe to use, analysis is done here to see if any calls to this cancel function might somehow apply to function ID 00B1 (but not apply to the other set of drop-target related callback functions). The table below is the result of a survey of all occurrences of this function call with signature BD 8A 9A, along with the xxxx and yyyy values in each function call.

It appears that all running function IDs are sampled, and those that have an ID that matches the set of bits identified by xxxx AND yyyy are cancelled. *This analysis is subject to further investigation/correction.*

For each running function ID:

IF <running ID> AND <yyyy> EQUALS <xxxx> AND <yyyy>, then function is cancelled.

ROM Offset	ID xxxx	Mask yyyy	Applies to ID 00B1?	Description, x-bit = don't care
0x44593	00 85	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1000 0101 (085)
0x44D78	00 85	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1000 0101 (085)
0x45064	00 A8	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1010 1000 (0A8)
0x4535A	00 89	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1000 1001 (089)
0x46653	00 A5	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1010 0101 (0A5)
0x46A6D	00 AB	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1010 1011 (0AB)
0x46A74	00 8D	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1000 1101 (08D)
0x46AB7	00 A6	01 FF	No	Cancels all function IDs that end with xxxx xxx0 1010 0110 (0A6)
0x606C9	00 00	08 00	Yes	Cancels all function IDs that have xxxx 1xxx xxxx xxxx bit cleared
0x60751	00 00	10 00	Yes	Cancels all function IDs that have xxx1 xxxx xxxx xxxx bit cleared
0x6078E	00 00	10 00	Yes	Cancels all function IDs that have xxx1 xxxx xxxx xxxx bit cleared
0x6593E	00 00	20 00	Yes	Cancels all function IDs that have xx1x xxxx xxxx xxxx bit cleared
0x7C808	00 60	01 FC	No	Cancels all function IDs that end with xxxx xxx0 0110 00xx (060)
0x7C967	00 60	01 FC	No	Cancels all function IDs that end with xxxx xxx0 0110 00xx (060)
0x7CAEE	00 60	01 FC	No	Cancels all function IDs that end with xxxx xxx0 0110 00xx (060)

As shown in the table above, the masked cancel function calls which affect the proposed ID 00B1 are encompassing many functions including all of the existing function IDs that start with 00Bx. These are likely functions that cancels all active functions at certain times such as tilt, slam-tilt and at menu-button push, etc. These large-encompassing cancellations do not uniquely affect function ID 00B1 therefore adding to the idea that ID 00B1 is a safe candidate for resolving the L8.3 function ID overlap.

Survey of SearchLinkedListAndMaskParameterBytes()

A secondary function also performs lookups of functions using the ID xxxx and Mask yyyy method described in previous function analysis. This masked function lookup performs a group search of such functions to determine if one or more matching functions are currently running. This function returns C-bit clear when one or more such functions have been found in the scheduler queue.

Below is the result of a L-8 ROM survey of all such occurrences of this function, with the BD 8A AA signature.

ROM Offset	ID xxxx	Mask yyyy	Applies to ID 00B1?	Description, x-bit = don't care
0x4457C	00 86	01 FF	No	Finds function IDs that have xxxx xxx0 1000 0110 (086)
0x45187	00 93	01 FF	No	Finds function IDs that have xxxx xxx0 1001 0011 (093)
0x456B7	00 89	01 FF	No	Finds function IDs that have xxxx xxx0 1000 1001 (089)

0x458FF	00 91	01 FF	No	Finds function IDs that have xxxx xxx0 1001 0001 (091)
0x45909	00 A4	01 FF	No	Finds function IDs that have xxxx xxx0 1010 0100 (0A4)
0x45912	00 89	01 FF	No	Finds function IDs that have xxxx xxx0 1000 1001 (089)
0x4591B	00 A9	01 FF	No	Finds function IDs that have xxxx xxx0 1010 1001 (0A9)
0x46060	00 A2	01 FF	No	Finds function IDs that have xxxx xxx0 1010 0010 (0A2)
0x46125	00 86	01 FF	No	Finds function IDs that have xxxx xxx0 1000 0110 (086)
0x46692	00 AB	01 FF	No	Finds function IDs that have xxxx xxx0 1010 1011 (0AB)
0x4675E	00 A4	01 FF	No	Finds function IDs that have xxxx xxx0 1010 0100 (0A4)
0x4676A	00 86	01 FF	No	Finds function IDs that have xxxx xxx0 1000 0110 (086)
0x46780	00 86	01 FF	No	Finds function IDs that have xxxx xxx0 1000 0110 (086)
0x46866	00 90	01 90	Yes	Finds function IDs that have xxxx xxx0 1xx1 xxxx
0x46A93	00 96	01 FF	No	Finds function IDs that have xxxx xxx0 1001 0110 (096)
0x46F9B	00 95	01 FF	No	Finds function IDs that have xxxx xxx0 1001 0101 (095)
0x47001	00 A4	01 FF	No	Finds function IDs that have xxxx xxx0 1010 0100 (0A4)
0x4708B	00 92	01 FF	No	Finds function IDs that have xxxx xxx0 1001 0010 (092)
0x51B98	00 89	01 FF	No	Finds function IDs that have xxxx xxx0 1000 1001 (089)
0x60740	80 00	80 00	No	Finds function IDs that have 1xxx xxxx xxxx xxxx
0x6E67B	00 AA	01 FF	No	Finds function IDs that have xxxx xxx0 1010 1010 (0AA)
0x6E684	00 AB	01 FF	No	Finds function IDs that have xxxx xxx0 1010 1011 (0AB)
0x6E68D	00 86	01 FF	No	Finds function IDs that have xxxx xxx0 1000 0110 (086)
0x6E6C5	00 86	01 FF	No	Finds function IDs that have xxxx xxx0 1000 0110 (086)
0x6EFAE	00 40	01 F0	No	Finds function IDs that have xxxx xxx0 0100 xxxx (04x)
0x6EFC4	00 40	01 F0	No	Finds function IDs that have xxxx xxx0 0100 xxxx (04x)
0x6F031	00 40	01 F0	No	Finds function IDs that have xxxx xxx0 0100 xxxx (04x)
0x6F62E	00 40	01 F0	No	Finds function IDs that have xxxx xxx0 0100 xxxx (04x)
0x7C8DE	10 61	01 FF	No	Finds function IDs that have xxxx xxx0 0110 0001 (061)
0x7C97D	00 60	01 FC	No	Finds function IDs that have xxxx xxx0 0110 00xx (06x)

As highlighted in the table above, only one such lookup function will find the prospective function ID 00B1. This entry will also find the other 00Bx ID functions no differently than the existing L-8 and L8.3 so it seems relatively benign as it applies to the use of ID 00B1 as the new function identifier.

The highlighted lookup at ROM offset 0x46866, $6866,31, is oddly looking for any function ID that have only 3 bits matching the pattern of xxxx xxx0 1xx1 xxxx. Based on the other lookups it raises suspicion as possible incorrect coding in the original L-8. More research into code in this ROM bank $31 is necessary to say with any amount of certainty.

This unusual function ID lookup takes place in the drop-target down function responsible for tracking the timed drop-target down period before automatically kicking it back up (and stopping its blinking lamp). This is during a moment outside of multiball when the drop-target is hit and then becomes subject to such timer that automatically kicks the target back up after a short period.

```
---------------------------------------;------------------------------------------------
                                       ;
                                       ; DropTargetDownSwitchTimerLoop() function ID 00A6
```

```
                                        ;
684A: BD 85 B2    JSR   $85B2           ;
684D: 10                                ;
684E: 8E 06 18    LDX   #$0618          ;
6851: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6854: 7E 68 57    JMP   $6857           ;
6857: E6 84       LDB   ,X              ;
6859: D7 C7       STB   $C7             ;
                                        ;
685B: BD 83 46    JSR   $8346           ; -\ -\ Sleep()
685E: 40                                ;  |  |  0x40 == 1 second
685F: BD 86 90    JSR   $8690           ;  |  | SearchLinkedListForId() // c-bit clear = found
6862: 00 86                             ;  |  |  Search for 0x0086, c-clear = multiball running
6864: 24 19       BCC   $687F           ;  |  |  C-clear, cancel the drop-target timer now
6866: BD 8A AA    JSR   $8AAA           ;  |  | SearchLinkedListAndMaskParameterBytes()
6869: 00 90                             ;  |  |  ID   0090
686B: 01 90                             ;  |  |  Mask 0190
686D: 24 EC       BCC   $685B           ;  | -/ C-clear means search found a match go to $685B
                                        ;  |
686F: 0A C7       DEC   $C7             ;  | Decrement number of seconds for drop-target
6871: 26 E8       BNE   $685B           ; -/
                                        ;
6873: BD 83 46    JSR   $8346           ; Sleep()
6876: 20                                ;  0x20 == 1/2 second
6877: BD 68 08    JMP   $6808           ; ScheduleDropTargetUp()
687A: BD 87 22    JSR   $8722           ; ClearLamp()
687D: 0E 40                             ; Lamp 0x0E 0x40
687F: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;------------------------------------------------
```

The unusual search, as shown above, keeps the drop-target down as long as any scheduled function is running which matches the pattern xxxx xxx0 1xx1 xxxx (with x=don't care bits). Interestingly, in bank $31 there is a function with id 0090. Could it be that this highlighted code was intending to check if the function ID 0090 was running but used incorrect function call parameters which capture a variety of other function IDs as well? Certainly more study and analysis is needed to understand whether this function ID search code was intentional or incorrect.

ROM Survey of New Function ID 00B1

Lastly, a survey of the L-8 ROM image for all occurrences of the 2 bytes 00B1 was done to see if, by chance, the function ID 00B1 is used in any function not previously identified as a Function ID handler.

ROM Offset	WPC Addr.	Surrounding Bytes	Analysis
0x43C4E	$7C4E,30	C0 3D 10 25 00 B1 8D A3	This 00B1 is part of a LBCS instruction in attract-mode loop
0x488EB	$48EB,32	00 60 74 04 00 B1 33 00	Not executable code. Data code (dmd/font data, other?)
0x49DE5	$5D35,32	BF F7 04 0A 00 B1 44 A9	Not executable code. Data code (dmd/font data, other?)
0x684A6	$44A6,3A	C5 08 33 01 00 B1 34 46	Data table cited in prior function. Not likely a function id.
0x6E61E	$661E,3B	00 B0 80 4C 00 B1 80 4D	Data table cited in prior function. Not likely a function id.

As shown, there are only 5 occurrences in the entire L-8 ROM image of the 2-byte sequence "00 B1" and only one such occurrence appears to be used directly in executable code but it is in the opcode of a LBCS

instruction. The other 4 occurrences of 00B1 are in data tables that are not likely to be related to the use of a function ID since all other uses of function IDs are not in data tables, but in running code as function parameter bytes.

This and the previous findings make it most likely that 00 B1 may safely be used as the new Function ID for resolving the function ID overlap issue identified in the L8.3 document.

Code Change For Fixing Function ID Overlap Issue

The L8.3 code introduced two places in the code where the 00 B9 function ID was used for its new drop-target up function. Both of these places are updated in L8.4 to use the new function ID 00 B1.

The new L8.3 function at $7FBA,31, ROM offset 0x47FBA, is shown below with the old L8.3 code highlighted in red and the updated function ID for L8.4 highlighted in green. This is the L8.3 code that schedules the drop-target up on behalf of the new L8.3 feature to have the drop-target reset to the up position at the start of multiball.

```
----------------------------------------;--------------------------------------------------
7FBA: 34 04       PSHS  B               ;
7FBC: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
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~~
7FE1: 00 B1                             ; 0x00B1 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 secondary location of L8.3 code where the function ID 00B9 is cited is shown below. This function is at $7FA2,31, ROM offset 0x47FA2. This is code that is used during the handling of the drop-target switch whereby code will check if the drop-target 'up' function is scheduled and, if so, ignores the drop-target switch with assumption that the switch may be ignored during the period of time that the drop-

target is being reset. This is to prevent the software from treating a drop-target switch closure as a player-initiated drop-target hit during this period.

```
----------------------------------------;--------------------------------------------------
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
7FAC: 00 B1                            ; 0x00B1 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
----------------------------------------;--------------------------------------------------
```

With these 2 changes of 00B9 to 00B1 in place, the function ID overlap issue is effectively resolved.

The Terminator 2 -- 255-Hits Issue

For L8.4, one goal was to fully understand the long-standing report of T2 reporting 255 Hits remaining on the hunter ship to start multiball. This issue was mostly elusive and not something that has been observed during the testing of L8.3. For L8.4 we want to identify (and correct) the coding issues or determine whether such issues have already been fixed in prior T2 software updates. What we discovered was a little bit of both (it was fixed in L-6 but there is a coding issue to fix in L8.4).

Computer Representation of 8-Bit Numbers

To understand the reason the game would report 255 hits remaining will require a little bit of understanding in how computers store and interpret numbers in an 8-bit (byte) memory storage. The table below shows how the numbers 5 through 0 are stored and how they are interpreted.

Bit Position								Total Numeric Value (signed)	Total Numeric Value (unsigned)
7	6	5	4	3	2	1	0		
0	0	0	0	0	1	0	1	5	5
0	0	0	0	0	1	0	0	4	4
0	0	0	0	0	0	1	1	3	3
0	0	0	0	0	0	1	0	2	2
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0

As hinted in the last two columns in the table above, the software can choose to read and represent a number from memory as if it were a *signed value* or an *unsigned value*. A *signed value* means the code is intentionally treating the piece of memory as a value that can contain negative value. An *unsigned value* means the code is intentionally treating the piece of memory as a value that only represents

values 0 and above. When an 8-bit value is represented as a signed number, it can contain values in the range of -128 to 127. When an 8-bit value is represented as an unsigned number, it can contain values in the range of 0 through 255.

The number 255 enters the picture when the value zero is decremented by 1. The actual result of 0 minus 1 is always all 8 bits are set to 1. It is a matter of how the subsequent code accesses such memory value is what depends on whether the number is treated as -1 or 255. Below is a continuation of the table showing the interpreted values when the memory is decremented below 0.

Bit Position								Total Numeric Value (signed)	Total Numeric Value (unsigned)
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	-1	255
1	1	1	1	1	1	1	0	-2	254
1	1	1	1	1	1	0	1	-3	253
1	1	1	1	1	1	0	0	-4	252
1	1	1	1	1	0	1	1	-5	251

With the above information, it is fairly evident that the game could report “255 Hits Remaining” on the hunter ship due to the subtraction of 1 from an internal “hits remaining” counter while the value is zero. This would result in a value being reported as 255 by code that treats the 8-bit “hits remaining” value as an unsigned number. It is evident that the code that reports the “hits remaining” value would not have considered the value being a negative value and, therefore, treats the memory location as an unsigned value which is why it would report 255 hits remaining instead of -1 hits remaining.

Analysis of L-8 Hits Remaining Code

In order to understand whether the 255-hits issue could occur, an analysis of the L-8/L8.3 software was done to see if it is possible for the L-8 to actually decrement the ‘Hits remaining’ value below zero. What was discovered is that the “Hits remaining” value is specifically checked if it is non-zero prior to decrementing. Below is an example of such code from L-8 that handles hunter ship hits at $4DCE,31, ROM offset 0x44DCE.

```
4DCE: 8E 06 03    LDX   #$0603          ; #$0603 is Hunter ship hits remaining for multiball
4DD1: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
4DD4: 7E 4D D7    JMP   $4DD7           ; <nop>
4DD7: 6D 84       TST   ,X              ; Check value pointed to by X
4DD9: 27 02       BEQ   $4DDD           ; If value is zero, skip down to $4DDD
4DDB: 6A 84       DEC   ,X              ; Decrement per-player Hunter Ship Hits remaining
4DDD: 8E 05 FF    LDX   #$05FF          ;
```

As shown in the code, above, the software gets the hunter ship hits remaining value from memory location $0603. This is the starting address for player 1’s hits remaining. A function, $FB29, is called to increment X by the current player number to get the unique value for the current player. **The value in X is then checked to be non-zero prior to decrementing it by 1.** This means this code appears to have been specially coded to ensure the “255 Hits” problem won’t occur.

Additionally, another piece of code was observed to be present in L-8 which may also be part of extra code to help prevent the possibility of "255 Hits" problem. During the multiball startup code the following code from L-8 is at $6C88,31.

```
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              ; Ensure # of hunter ship hits is zero
```

The code, above, is specifically ensuring the per-player "hits remaining" value is specifically forced to value 0 at multiball start. This code is possibly put in place to help reduce possibility of incorrect hits being stored in memory and, possibly, part of a code enhancement in original T2 code to prevent unexpected problems related to the "hits remaining" value.

Analysis of Older T2 ROMs

The two code sections, mentioned above, served as the basis for subsequent code searches in older T2 ROM images to see at what point the fixes may have been added, and as such, when the "255 Hits" problem was most likely corrected. With assumption that each ROM revision likely used different memory addresses, searches were done with wildcard bytes, represented below as 'xx', to match on any byte values.

The discovered "xx xx" memory address identified with the 8E xx xx instruction in the "Decrement When Non-Zero" is then used in the search pattern for "Reset To Zero at MB Start" search.

The pattern was found in L-6 with slightly different format, instead of a JSR to non-banked ROM followed by JMP to next instruction, the L-6 has a JSR to the WPC banked function caller which means the 7E xx xx is replaced with WPC address to the actual function where the memory value in X gets incremented by the current player index.

ROM	Search	Search Pattern	Search Result
L-8	Decrement When Non-Zero	8E xx xx BD xx xx 7E xx xx 6D 84 27 02 6A 84	Found at ROM offset 0x44DCE w/8E 06 03
L-8	Reset To Zero at MB Start	8E 06 03 BD xx xx 7E xx xx 6F 84	Found at ROM offset 0x46C88
L-6	Decrement When Non-Zero	8E xx xx BD xx xx xx xx 6D 84 27 02 6A 84	Found at ROM offset 0x4D30 w/8E 05 D1
L-6	Reset To Zero at MB Start	8E 05 D1 BD xx xx xx xx xx 6F 84	Found at ROM offset 0x6C27
L-4	Decrement When Non-Zero	8E xx xx BD xx xx xx xx xx 6D 84 27 02 6A 84	Not found
L-4	Reset To Zero at MB Start	8E xx xx BD xx xx xx xx xx 6F 84	Not found
L-3	Decrement When Non-Zero	8E xx xx BD xx xx xx xx xx 6D 84 27 02 6A 84	Not found
L-3	Reset To Zero at MB Start	8E xx xx BD xx xx xx xx xx 6F 84	Not found
L-2	Decrement When Non-Zero	8E xx xx BD xx xx xx xx xx 6D 84 27 02 6A 84	Not found
L-2	Reset To Zero at MB Start	8E xx xx BD xx xx xx xx xx 6F 84	Not found

The search results, above, would lead one to believe that the “255 Hits” bug was most likely fixed starting in L-6. As we further explore, below, we discover this to be *mostly* a correct observation. *Note L-5 is not immediately available and, as such, was not checked as part of this exercise.*

Comparison of code from L-4 to L-6

To demonstrate the code was fixed, starting in L-6, below are the comparisons of the two sections of code mentioned above, between L-4 and L-6.

Shown below is the faulty code from L-4 where the number of hunter ship hits is allowed to decrement by 1 regardless of its current value (compare with that shown, above, for L-8). This L-4 code is from $4D30,31 (ROM offset 0x$4D30):

```
4D30: 8E 05 D3    LDX   #$05D3          ; #$0D53 is Hunter ship hits remaining for multiball
4D33: BD 88 8B    JSR   $888B           ; <calls the function specified in next 3 bytes>
4D36: FA BF FF                          ; IncrementXByPlayerIndexNumber()
4D39: 6A 84       DEC   ,X              ; Decrement per-player Hunter Ship Hits remaining
4D3B: 8E 05 CF    LDX   #$05CF          ; Per-player target-map for hits remaining
```

As seen in the L-4 code, above, the number of hunter ship hits remaining is allowed to decrement no matter what its current value might be. Below is the comparable section of code from L-6 at $4D30,31 (ROM offset 0x4D30) *interestingly, location starts at the same WPC address and same ROM offset between L-4 and L-6*:

```
4D30: 8E 05 D1    LDX   #$05D1          ; #$05D1 is Hunter ship hits remaining for multiball
4D33: BD 88 94    JSR   $8894           ; <calls the function specified in next 3 bytes>
4D36: FB AB FF                          ; IncrementXByPlayerIndexNumber()
4D39: 6D 84       TST   ,X              ; Check value pointed to by X
4D3B: 27 02       BEQ   $4D3F           ; If value is zero, skip down to $4DDD
4D3D: 6A 84       DEC   ,X              ; Decrement per-player Hunter Ship Hits remaining
4D3F: 8E 05 CD    LDX   #$05CD          ;
```

Next the second part of code that cites the ‘hunter ship hits remaining’ value, as described above for L-8, is during multiball start where it was observed that this hits-remaining value is reset to 0 at start of multiball. Below will compare this portion of code between L-8 and L-4.

In the L-8 code at multiball start, the clearing of the ‘hunter ship hits remaining’ value is preceded by a function call to survey game state and branch to end if not ready for multiball. After the applicable code is a call to system function with parameter bytes 00 1B, as shown below from L-8 starting at $6C83,31 (ROM offset 0x46C83):

```
6C80: BD F7 59    JSR   $F759           ; Checks state variables. Z-set means okay to proceed.
6C83: 7E 6C 86    JMP   $6C86           ; <nop>
6C86: 26 52       BNE   $6CDA           ; If Z-clear, branch to end of function
                                        ;
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              ; Ensure # of hunter ship hits is zero
                                        ;
6C93: BD 88 D5    JSR   $88D5           ; Call5253,39WithXParameterBytes()
6C96: 00 1B                             ;
```

Observing the corresponding code from L-4 reveals that, indeed, there is no such clearing of the 'hunter ship hits remaining' value at this same location (nor can such clearing be found anywhere in the L-4 ROM image). Shown below is the same region of code from L-4 starting at $6C1B,31 (ROM offset 0x6C1B):

```
6C1B: BD 88 8B    JSR   $888B          ; <calls the function specified in next 3 bytes>
6C1E: F6 E8 FF                        ; Checks state variables. Z-set means okay to proceed.
6C21: 26 4B       BNE   $6C6E         ; If Z-clear, branch to end of function
                                      ;
6C23: 7F 05 D3    CLR   $05D3         ; clear $05D3
                                      ;
6C26: BD 88 6B    JSR   $886B         ; Calls function with 00 1B parameter bytes
6C29: 00 1B                           ;
```

As shown, above, the L-4 code had a single clearing of a single byte in ram at $05D3 where the L-8 code (above) has the clearing of the per-player 'hunter-ship-remaining' value. It's not immediately clear what the $05D3 byte tracks but it certainly is not a per-player value since no function is called to select 1 of 4 bytes from the address. The L-4 just cleared $05D3 and this was replaced. There doesn't appear to be any corresponding single-byte clear in the L-8 around this same part of the code, so it seems evident that this was clearly a coding bug that somebody replaced with the correct code we see in L-8. Below is the corresponding code from L-6 at $6C1F,31 (ROM offset 0x6C1F):

```
6C1F: BD 88 94    JSR   $8894         ; <calls the function specified in next 3 bytes>
6C22: F7 D4 FF                        ; Checks state variables. Z-set means okay to proceed.
6C25: 26 53       BNE   $6C7A         ; If Z-clear, branch to end of function
6C27: 8E 05 D1    LDX   #$05D1        ; #$05D1 is Hunter ship hits remaining for multiball
6C2A: BD 88 94    JSR   $8894         ; <calls the function specified in next 3 bytes>
6C2D: FB AB FF                        ; IncrementXByPlayerIndexNumber()
6C30: 6F 84       CLR   ,X            ; Ensure # of hunter ship hits is zero
6C32: BD 88 74    JSR   $8874         ; Calls function with 00 1B parameter bytes
6C35: 00 1B                           ;
```

As shown above, the new code started in L-6 where the multiball start includes the reset of per-player 'hunter ship hits remaining' value to 0.

These examples also help show the reason why a lot of the L-8 code has "JMP" instructions that simply jump to the very next instruction. The older code utilized the function that calls the function specified in the following 3 bytes, however in cases where the following 3 bytes are in non-banked ROM, the code can simply call such function directly instead of using the intermediate function. It seems the newer code elected to leave the JMP instructions as placeholder instead of omitting such instructions entirely.

In conclusion, the text above demonstrates that the 255-Hits bug was corrected in L-6 and continues to be fixed in L-8, and the subsequent s/w releases as well.

Reproducing the 255-Hits Issue in L-4

With the assumption being that the problem was most likely fixed starting in L-6, the goal shifted to reproduction of the "255 Hits" issue in an L-4 ROM image. Once the problem can be reliably reproduced in L-4 then the same procedure can be done on subsequent ROM images to support the conclusion that the "255 Hits" issue no longer occurs on ROM images starting at L-6.

A description of the "255 Hits" problem came in with the best clues that eventually lead to discovery for a way to reproduce the original problem. **A big Thank You is deserved to the source of this information.** The description mentioned the idea that the hunter ship targets sometimes are not lit consistently as compared with the reported number of targets remaining. It is when there are 2 targets lit while the game reports 1 target remaining is when the trouble is most likely to occur. When there are 2 targets lit but 1 target remaining, the targets remaining will report at 255 if at the next "fire at will" attempt both of the 2 lit targets are hit at, seemingly, the same time.

After a lengthy period of reproduction failures and code analysis, it became apparent that the pinball simulator, which was the primary sandbox where bug reproduction efforts were being attempted, was most likely in need of modification in order to hit the issue. With analysis leading us to believe that the problem was most likely due to hunter targets being hit soon after the initial, successful, hunter ship hit, the simulator was modified in the following ways:

- The on-screen display reports *which* simulated hunter-target will be triggered when gun-trigger is pulled. This makes it easier to "aim" the cannon on the simulator.
- Immediately after the gun-trigger pull, the two targets adjacent to the 'hit' target are then repeatedly toggled, back and forth, for a few seconds (i.e. while the hunter ship explosion animation is playing if ship was hit).

These modifications help expose bugs in the game code that occur when the ball, on a real machine, hits the hunter ship targets immediately after hitting a lit target. For clarity, the modified simulator has the following characteristics (where "T1" is the top-most target and "T5" is the bottom-most target):

Five-Bank Target Hit from Cannon	Modified Simulator Behavior After the intended Target-Hit from Cannon, the following behavior ensues
Top, T1	Targets T5 and T2 alternately are hit, 25 times each, over a period of a couple seconds
T2	Targets T1 and T3 alternately are hit, 25 times each, over a period of a couple seconds
T3	Targets T2 and T4 alternately are hit, 25 times each, over a period of a couple seconds
T4	Targets T3 and T5 alternately are hit, 25 times each, over a period of a couple seconds
Bottom, T5	Targets T4 and T1 alternately are hit, 25 times each, over a period of a couple seconds

With this modification in place, for example, after successfully hitting lit target T3, the two other targets T2 and T4 automatically get hit, back and forth, rapidly over a couple seconds. This happens regardless of the lit state of these two targets. The modified simulator is simulating a ball or human manually hitting the other two targets very quickly after the initial hunter ship target had been hit. Although this particular scenario is not realistic, it does appear to exercise the problematic code path which the real game sometimes experiences with the ball hitting the hunter ship targets in such a way as to experience the "255 Hits" issue on the real game (running L-4 or older) from time to time. *Later it is evident that it is the most immediate target hits that contribute to the problem so hitting the adjacent targets only 1 time instead of 25 times each should have been sufficient to reproduce the issue.*

Recipe for Reproducing the 255-Hits issue in L-4

Although there are many possible combinations of targets and possible ways for the 5-bank targets to be hit and likely encounter the problem, the following procedure was crafted using the modified pinball simulator to repeatedly and predictably encounter the 255-Hits bug. This recipe could be used as the basis for a real Terminator 2 machine to also hit the 5-bank targets so as to also cause the problem.

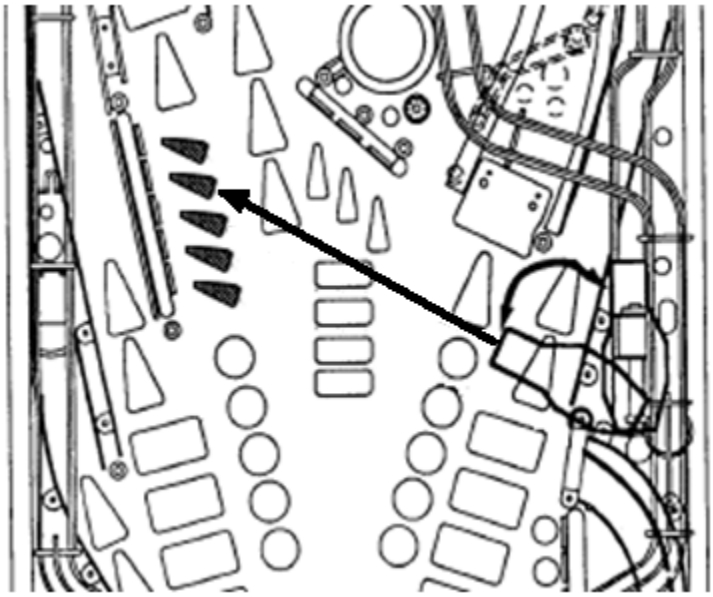
To hit the issue, you must be running L-4 or older software, such as shown below

Step 1. Reach the point of "5 Hits Remaining"

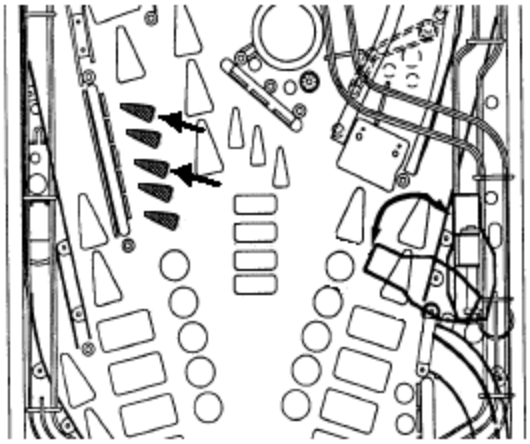
The problem could happen earlier than this, however it is easier to describe a procedure to reproduce the issue starting at this point at the 5th multiball where the game reports that you need 5 hits to start multiball.

Step 2. Cannon Shot 1: Shoot the Second, T2, Target

Refer to the modified image pulled from the T2 manual. Here all five target lamps are lit and the second target ,T2, is hit for a successful hunter ship strike.



As previously described, *immediately* after the successful hunter ship hit on the T2 target, the simulator will automatically toggle the target immediately above and below the hit target, rapidly, during the next few seconds while the hunter ship explosion animation is playing.

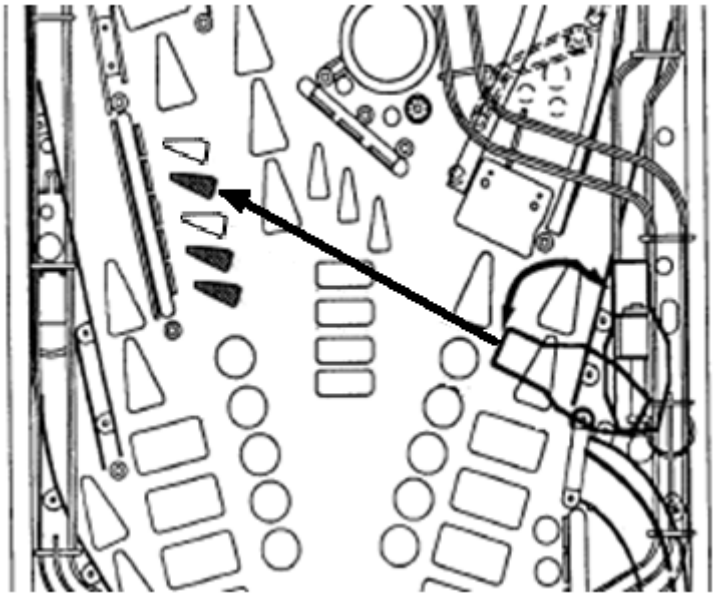


Immediately after the above takes place, the game performs its usual report of targets remaining. In this case, the result of the above activity is that the game reports 2 targets remaining. Since the previous cannon shot hit target T2 and (with simulator mod) also hit T1 and T3, it is reasonable to expect that the game registered 3 hit targets with 2 remaining.

Step 3. Cannon Shot 2: Shoot the Second, T2, Target (again)

When the cannon is reloaded for the 2nd shot attempt, the first sign of trouble appears. The game reports "2 Targets Remaining" on the game display, **however the actual lit targets show 3 lamps lit**. In this case, the game will light lamps for **T2,** T4**,** and T5. This is especially interesting since the first target that was hit at cannon shot #1, was the target T2 but T2 is still lit. *Further text in this document, below, will explain why this happened (essentially, the game forgets that T2 was hit when a secondary lit target is detected as being hit immediately afterwards).*

For this step of the procedure, hit the (lit) target T2 with the cannon shot, again.



In this case, the modified pinball simulator repeatedly hits the targets immediately above and immediately below the T2 target for a few seconds, however in this case the targets are not lit and no perceivable effect occurs from these unlit target hits.

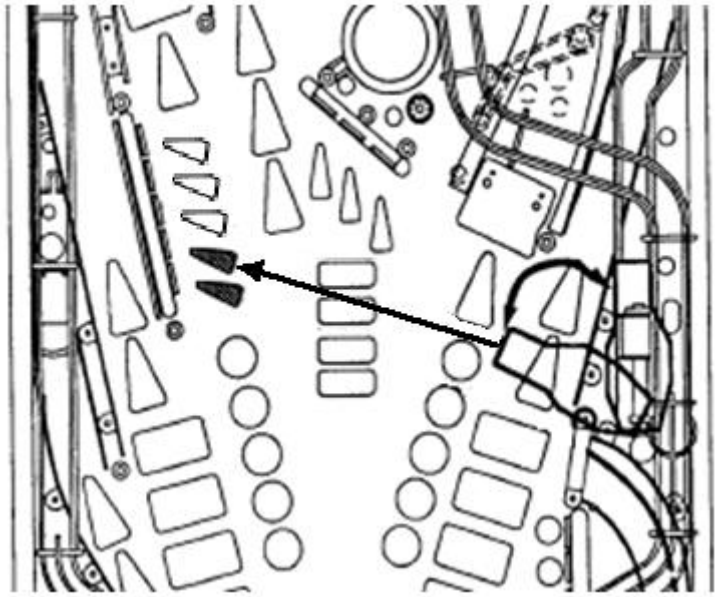
After this 2nd cannon shot strikes the T2 lit target (again) the display then reports 1 hit remaining.

Step 4. Cannon Shot 3: Shoot the Fourth, T4, Target

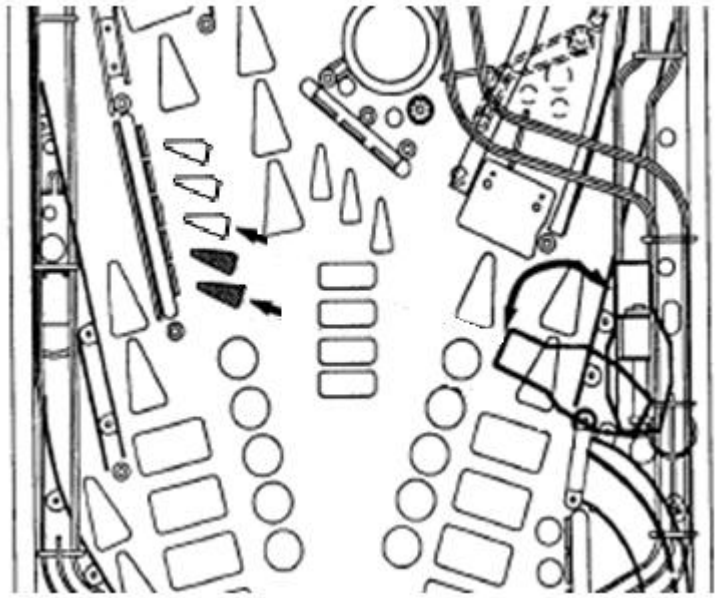
At the 3rd cannon load, the game reports, again, that 1 target hit remains before multiball will start. This time, the game continues to show mismatch with the actual lit targets. In this case, the lit targets are now the bottom two targets, T4 and T5.

It is worth noting that at this time the game is in the state that was previously described as the main trigger for hitting the 255-hits problem. The game is reporting 1 hit remaining but showing two lit lamps. The information that was provided is that hitting both of the lit targets at this point will cause the 255-hits issue. Using the modified pinball simulator we will see that this is, indeed, the case. *On a real game, this is when the ball would strike both targets at the same time.*

At this step, hit target T4 with the cannon shot.



After successfully hitting the lit T4 target, the game shows the hunter ship explosion while the modified pinball simulator rapidly hits the target immediately above and below the T4 target. In this case, the simulator hits T3 and T5 immediately after the T4 target was hit.



When the T4 target was hit, the game subtracted 1 from the hits remaining, taking the value to 0. When the T5 target is hit immediately afterwards, the game subtracts 1, again, from the hits remaining which takes it to value 255 since, as previously described, the game code treats the value as an *unsigned 8-bit number* so we get the report of 255 instead of negative 1.

For this procedure, this is the end of the demonstration of the "255 hits" issue. A remaining step can also be applicable for other scenarios in which the game might have reached this state, as described next.

Step 5. Cannon Shot 4: Last Lit Lamp

In some scenarios when this problem occurs, there could be 1 lamp remaining lit. Loading the cannon again may reveal one of the previously two lit targets is still lit. Shooting this remaining lit target will cause the game to subtract 1 from the target-remaining counter and report the following:

Subsequent cannon loads reveal the futility of subsequent cannon loads since there are no longer any more lit targets. The "hits remaining" value can no longer be reduced by using the cannon. For the player to get their "hits remaining" value to reset back to normal value, they need to start a multiball using the left loop or from the database award. After starting a multiball in these ways, the subsequent multiballs via the cannon will then resume at a normal value at "5 Targets Remaining" in this case.

Reproduction Attempts of the 255-Hits Issue in L-8

With the procedure defined for hitting the problem in L-4, it can now be attempted on L-8. As previously analysis has shown, it is not expected to ever see the "255 Targets Remaining" on any T2 running L-6 and above.

The attempts to reproduce the issue in L-8 using the same steps described for L-4 are shown in the table below. These steps include the modified simulator that hits adjacent targets immediately after the successful hunter ship hit.

Step	Description	Result
1	Reach the point of "5 Hits Remaining"	Regular game play to reach this point. No problems.
2	Cannon Shot 1: Shoot the Second, T2, Target	"2 Targets Remaining" Next lit targets T2, T4, T5, same as L-4
3	Cannon Shot 2: Shoot the Second, T2 Target (again)	"1 Target Remaining" Next lit targets T4, T5, same as L-4
4	Cannon Shot 3: Shoot the Fourth, T4, Target	Multiball starts, NO occurrence of the 255-hits problem.

As highlighted in the table above, it is immediately apparent that the game has fixed the problem where the "hits remaining" value will no longer decrement below zero and, as such, never report "255 targets remaining". It is also apparent that **there is still a separate bug** whereby the number of lit targets are not always consistent with the number reported in the "Targets Remaining" message. **This issue exists even in L-6, L-8, L8.1, L8.2 and L8.3.**

Note, it took several attempts to adjust the pinball simulator to trigger the adjacent targets at precisely the right moment in order to reproduce the issue. After several adjustments it was shown, as highlighted above, that a coding problem can be exposed in L-8 where lit 5-bank lamps are not in sync with the reported targets remaining.

It is evident in the code samples that the code was fixed to prevent the possibility that the counter could decrement to 255. Because of this, multiball will always be achieved; however we have this obvious remaining issue where the player can occasionally get incorrect and confusing set of lit targets.

Summary of Observations from the 255-Hits Investigation

The overall analysis into the 255-Hits problem investigation results in several distinct observations.

- Immediately after a successful hunter ship hit, there is a small window of time in which other lit 5-bank targets may also be hit which will reduce the "Targets Remaining" count. This certainly appears to be intentionally designed this way. In L8.4 an adjustment is being added to make this adjustable so that only a single hunter ship hit is allowed per cannon shot.
- Immediately after a hunter ship **miss**, the same small window of time exists where a successful target hit can be detected which will proceed to be processed as a hunter ship **hit**. This behavior is in all T2 software. This also appears to be intentionally coded this way to provide a little benefit to the player and to add some fairness when an unlit target and a lit target are hit seemingly simultaneously. Further analysis of this behavior will follow, below.
- Immediately after a successful hunter ship hit, the same small window of time exists where subsequent 5-bank target hits can result in a mismatch between the number of lit targets and the number reported for "Targets Remaining". This issue presumably exists in all T2 software through L8.3. It is subject to being corrected in L8.4 as a bug fix.
- The "Targets Remaining" counter was being allowed to decrement below zero. As shown above, this issue has been corrected starting in L-6 and requires no further analysis or correction.

The following text will further analyze these issues and explore the code changes needed to address them, as needed.

Hunter Ship 5-Bank Target Logic

In order to understand the nature of the observations that have been made as part of the 255-hits investigation, the logic that is used in L-8 (and L8.1, L8.2 and L8.3) has been analyzed and is summarized below.

For readers not interested in technical details, refer to the subsequent section "Hunter Ship 5-Bank Target Logic Summary" for an abbreviated description of the 5-bank target logic.

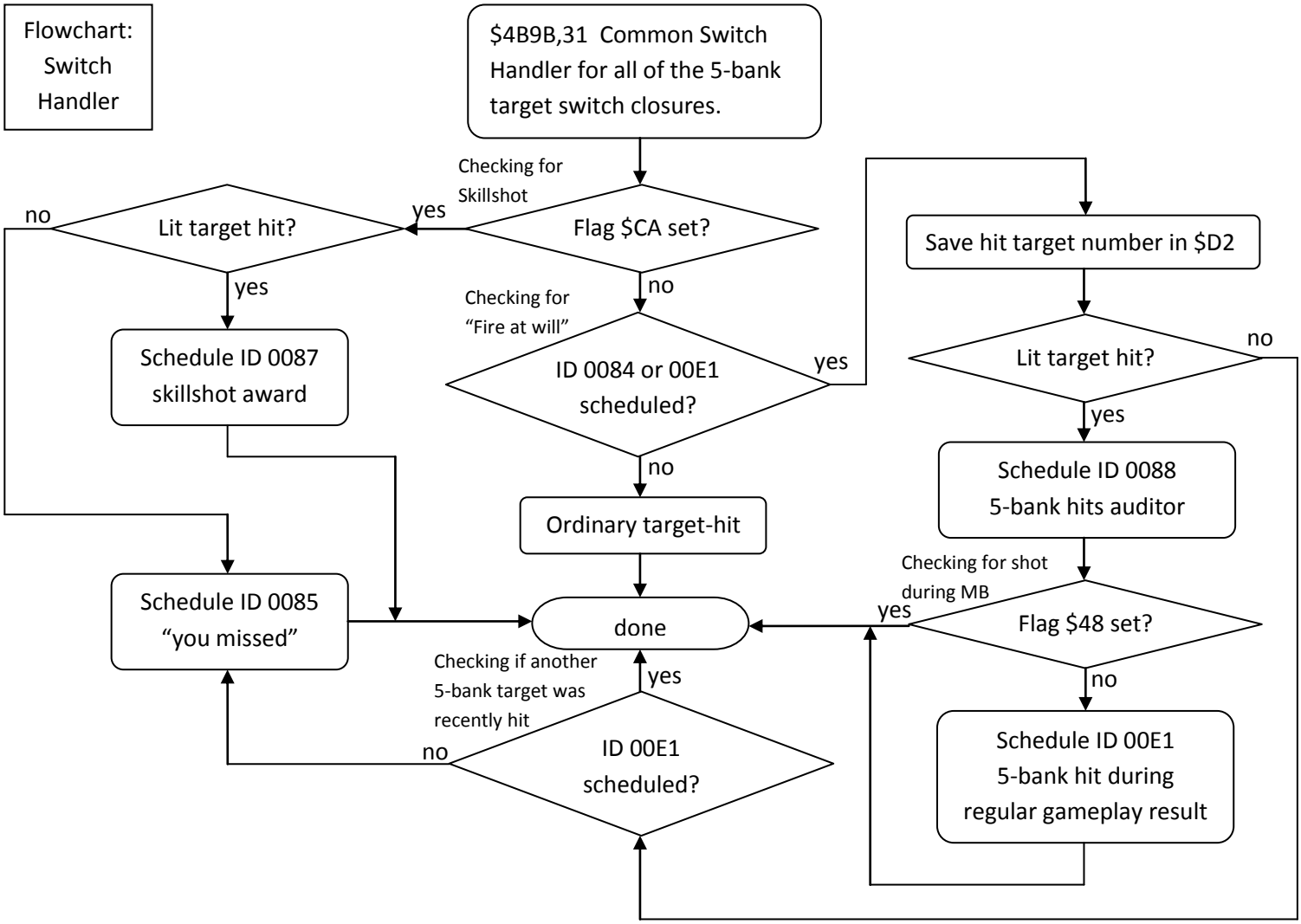
Refer to the L8.3 document where the switch table is described where it is shown how each game switch is associated with an 11-byte table entry where 3 of the bytes represent the WPC address where code will call to handle such switch closure *(or switch opening, or both in some cases)*. Shown below is the part of the table corresponding to the 5-bank targets. This portion of the table is at $4BA7,3D, ROM offset 0x74BA7:

```
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                 ;
```

As highlighted in the table portion, above, the WPC address for each of the 5 switches is a function located in bank $31 at closely addressed functions: $4B93,31, $4B95,31, $4B97,31, $4B99,31 and $4B9B,31. These correspond to ROM offsets 0x44B93, 0x44B95, 0x44B97, 0x44B99 and 0x44B9B, respectively.

Shown later in this section, these all end up calling a common switch handler function. *It's unclear why these 5 switches don't use the exact same address of the switch handler function.* All switch handler functions are called with indicator of which switch has been closed so that a common switch handler can be called. The switch handler itself can then discover which switch closure is responsible for the invocation of that function.

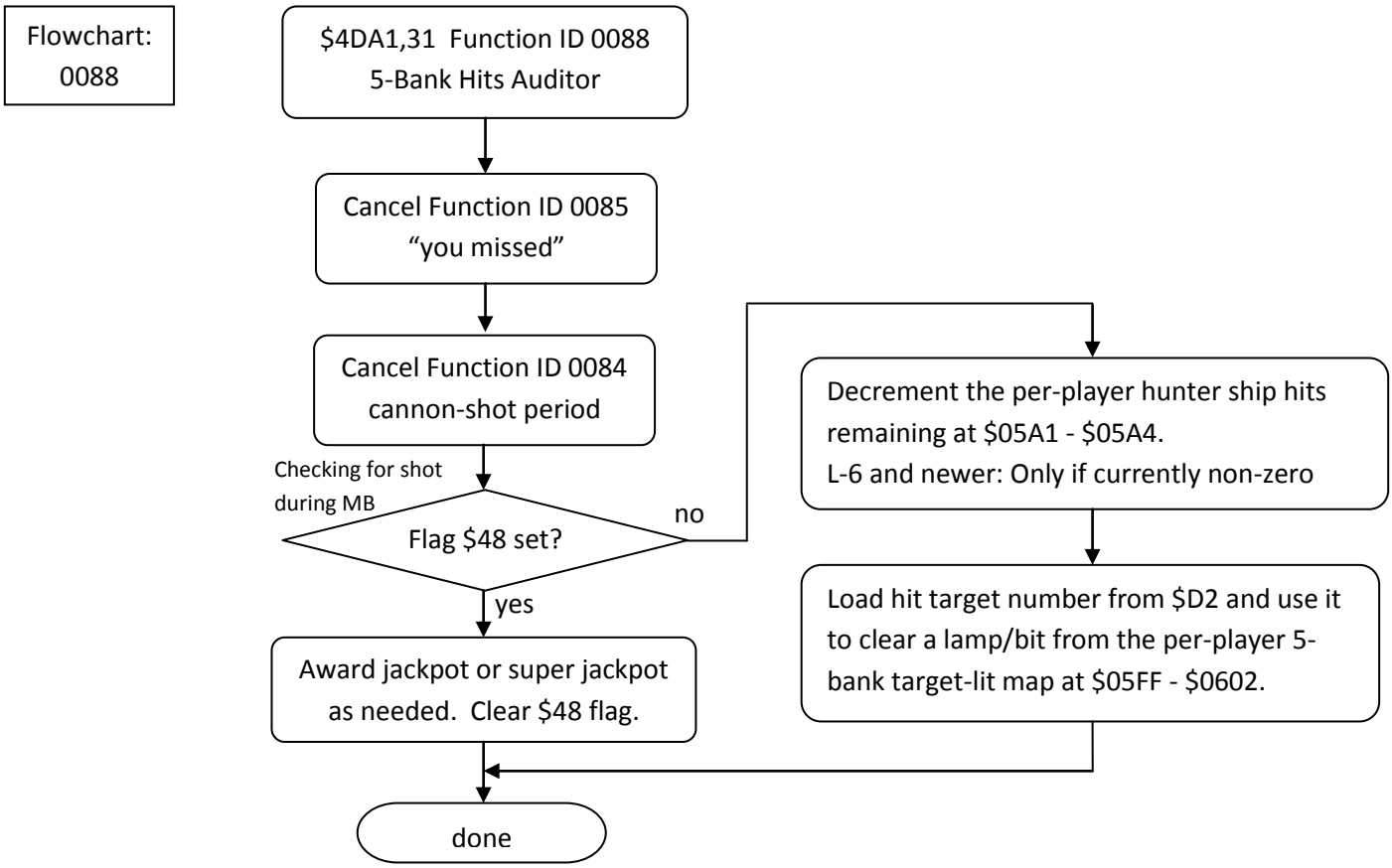
Shown here is a simplified flowchart depicting what happens in $4B9B,31 when a 5-bank target is hit.



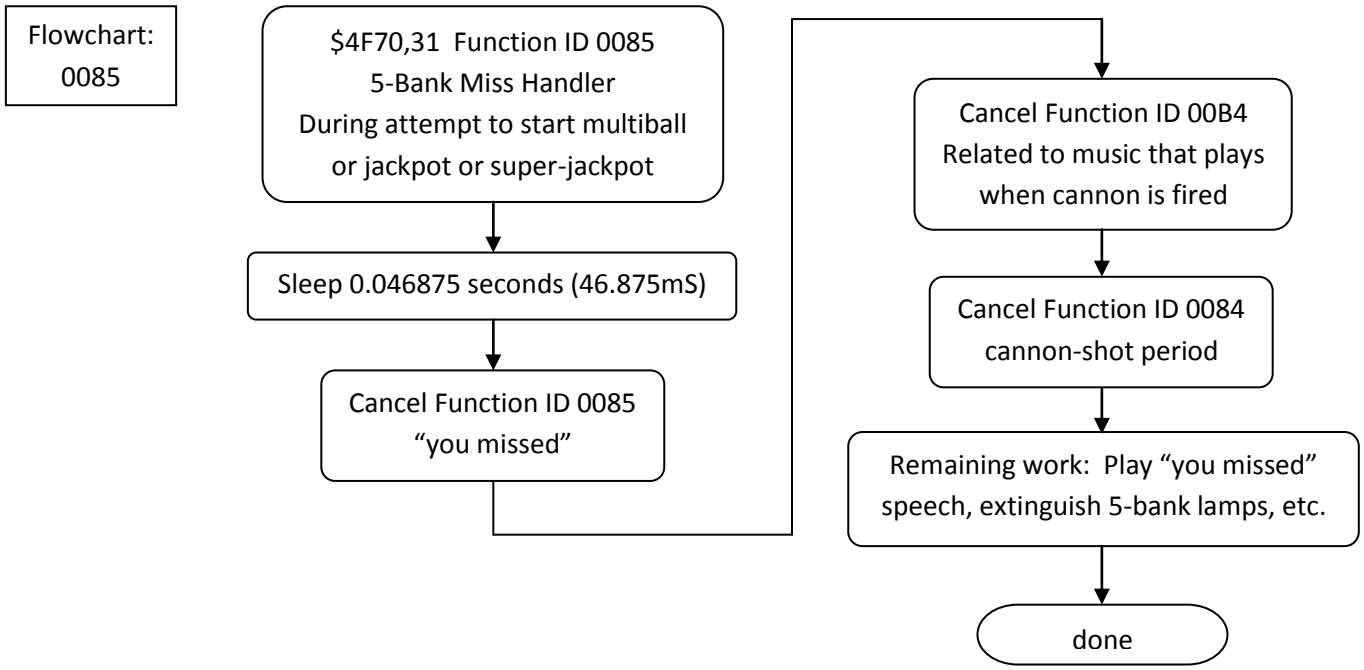
The logic depicted above shows how the handler for the 5-bank has a few memory flags and scheduled function ID lookups to determine how to proceed. Refer to the legend, below for details.

Resource	Description
Memory Flag $CA	Set when skill shot is being attempted
Memory Flag $48	Set when cannon shot is being attempted during multiball
Function ID 0087	Function that awards skill shot hit
Function ID 0084	Short lived function that runs when cannon is fired tracking time until target hit or timeout/miss
Function ID 0085	Function that runs when cannon shot 'miss' has been determined, plays "you missed", etc.
Function ID 00E1	Function that surveys hits results and reports "Targets Remaining" or starts Multiball
Function ID 0088	Function that handles hit target, decrements counter, awards Jackpot/Super Jackpot as needed

To understand how the various “255-hits” observations could happen, further understanding into some of the scheduled functions is needed. As depicted, during a “Fire at will” when a lit target is hit, the function ID 0088 is first scheduled. In the L-8 software this function is located at $4DA1,31, ROM Offset 0x44DA1. Depicted below is an abbreviated version of its logic with some logic steps bundled together as such details are not imperative for the purpose of illustrating the “255-hits” observations.

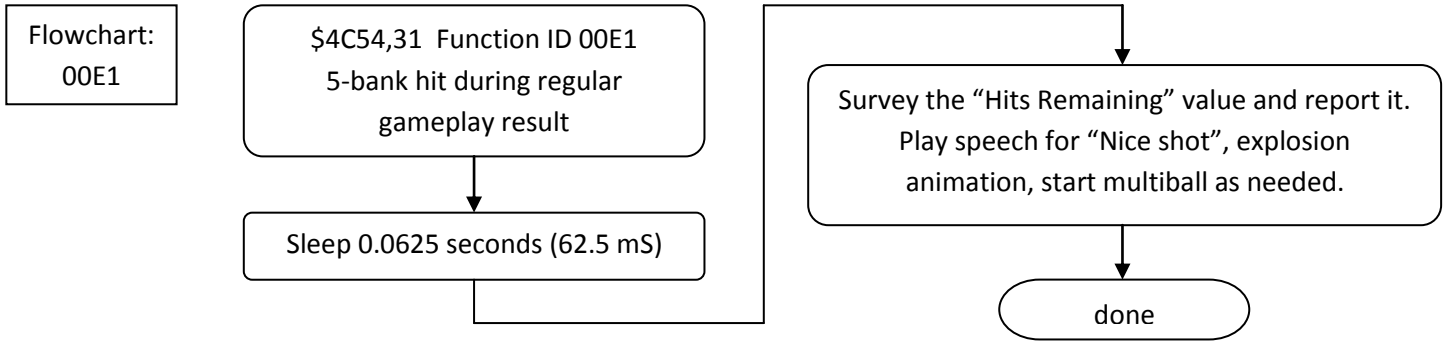


Next, the function ID 0085 is depicted below to show the scheduled function logic for when a 5-bank target is missed. As indicated in the 5-bank handler flowchart, there are multiple ways in which the 0085 function can be scheduled. There is an added detail in that the L-8 code has multiple functions that get called with the ID value 0085, depending on from where the function is scheduled (such as from skill shot miss or hunter ship miss). The ID 0085 function depicted in the following flowchart is from the logic path where the hunter ship miss takes place while trying to start multiball or while trying to get jackpot or super-jackpot. This is the most relevant version of the 0085 function depiction as it applies to the “255-Hits” observations.



As shown, for the 0085 function the "you missed" handler performs a small sleep followed by the cancellation of the 0085, 00B4 and 0084 functions along with other cleanup and "you missed" speech call. The important part of this function as it relates to the "255-Hits" observations is the small window of time in which the function sleeps. During this sleep is when other scheduled functions are allowed to run.

Lastly the ID 00E1 function logic is depicted below as it also plays a role in how the various "255-Hits" observation can take place:



The sleep that takes place at the start of the 00E1 function plays a role in how the various logic scenarios can take place. This sleep gives time for other scheduled functions to perform their work.

With the above logic now established, each of the "255-Hits" observations can now be described in terms of how and when the previously described functions are invoked. Below is a summary of thee following cases and scenarios. Scenarios involve specific delays, in milliseconds, between the 1st and 2nd target hits.

- Case 1: Lit target hit followed by lit target hit
  - Scenario 1: Second target detected 0mS after first target
  - Scenario 2: Second target detected 50mS after first target
  - Scenario 3: Second target detected 75mS after first target
- Case 2: Unlit target hit followed by lit target hit
  - Scenario 1: Second target detected 0mS after first target
  - Scenario 2: Second target detected 25mS after first target
  - Scenario 3: Second target detected 46.875mS after first target
  - Scenario 4: Second target detected 50mS after first target
- Case 3: Lit target hit followed by unlit target hit
  - Scenario 1: Second target detected 0mS after first target
  - Scenario 2: Second target detected 50mS after first target
  - Scenario 3: Second target detected 75mS after first target
- Case 4: Unlit target hit followed by unlit target hit
  - Scenario 1: Second target detected 0mS after first target
  - Scenario 2: Second target detected 25mS after first target
  - Scenario 3: Second target detected 50ms after first target

After the following detailed analysis, a summary will be provided to simply describe the timing windows and behaviors that the L-8 software provides with regard to simultaneous target hits.

In all such cases and scenarios, timing differences could be simply due to the switch gaps on the standup targets causing slightly longer period of time between each target hit, or perhaps due to a ball hitting one target more predominantly before hitting the second target, or even a rapid ricochet of the ball from the first target to a playfield post and back to the second target.

Some notes about function schedule behavior:

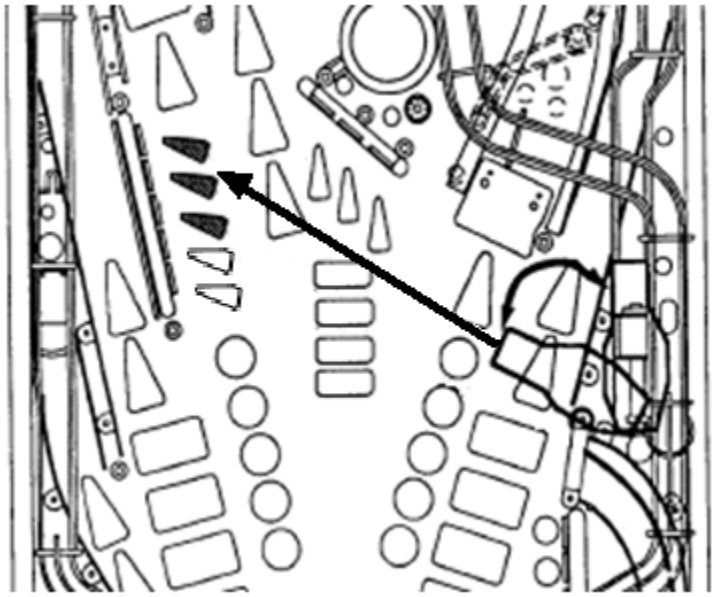
- Function ID 0088 is allowed to schedule multiple instances in the scheduler, simultaneously
- Function ID 0088 gets scheduled at the top of the schedule list, in front of the line.
- Function ID 00E1 gets scheduled in a way where existing 00E1 is cancelled first.
  - This means only a single 00E1 function is ever in the scheduler at a time.
- Function ID 0085 gets scheduled in a way where existing 0085 is cancelled first.
  - This means only a single 0085 function is ever in the scheduler at a time.

Case 1: Lit target hit followed by another lit target hit

In this case, the ball hits both lit targets at, seemingly, the same time causing 2 switch closures to be registered and processed back-to-back. Depending on the timing, the result can be:

- The player's "hits remaining" value is decremented for both lit target hits.
- The lit lamps for the remaining hits may become inconsistent with reported "hits remaining".

For this case, imagine a case where the top three targets are lit, T1, T2, T3 and 3 hits are remaining for multiball. In this scenario the ball hits between T1 and T2 causing the game software to detect T1 followed by switch T2.



There are multiple ways in which the sequence can go, depending on timing of how quickly the software detects the secondary hit of switch T2. The text below will go over multiple timing scenarios:

- Scenario 1: Second target detected 0mS after first target
- Scenario 2: Second target detected 50mS after first target
- Scenario 3: Second target detected 75mS after first target

Scenario 1: Second target detected 0mS after first target

In this scenario, the secondary hit on switch T2 is detected immediately after the switch T1 was processed. In this case, the game processes the 5-bank switch handler for switch T2 before any of the scheduled functions that were set up during the T1 switch handling had a chance to start:

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T1 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T1 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler

d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: Switch Handler

2. The 5-bank switch-handler called on behalf of T2 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T2 switch number into $D2 as the hit target, overwriting T1 that was in $D2.
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler
        i. Since step 1.d already added it, the same 00E1 function gets re-scheduled.

Flowchart: 0088

3. The ID 0088 Function is called due to the T1 switch handler logic at step 1.c.
    a. There is currently no function ID 0085 running (no ID 0085 to cancel)
    b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
    c. Flag $48 is not set:
        i. Decrement the per-player "hits remaining" value from 3 to 2
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to Step 2.b, this clears lamp for target T2 and not T1

Flowchart: 0088

4. The ID 0088 Function is called due to the T2 switch handler logic at step 2.c.
    a. There is currently no function ID 0085 running (no ID 0085 to cancel)
    b. The ID 0084 was cancelled at step 3.b (no ID 0084 to cancel)
    c. Flag $48 is not set:
        i. Decrement the per-player "hit remaining" from 2 to 1
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to step 3.c.ii, the lamp for the value at $D2 was already cleared

Flowchart: 00E1

5. The ID 00E1 Function is called due to step 2.d.i
    a. Sleep for 0.0625 seconds starts and completes
    b. Function reports "1 Hit Remaining"

For readers who are following along, it should be evident from the above scenario how it is that the game can get into a situation where the reported "Hits remaining" doesn't match the set of lit targets. The game saves the hit target number in ram address $D2 allowing multiple target hits to overwrite the value at $D2 before the value of $D2 has been processed by the 0088 function. The above sequence results in the player "hits remaining" to reach value 1 but the player's 5-bank target lit-map only extinguished the lamp for T2, which means the lamps for T1 and T3 are still illuminated at the next cannon shot attempt.

Scenario 2: Second target detected 50mS after first target

In this scenario, the secondary hit on switch T2 is detected 50mS after the switch T1 was processed. Prior to this 2nd target hit on T2, the game had already completed the 0088 function which was added to the scheduler by the T1 switch hit. This scenario demonstrates the logic that allows multiple target hits to count during the same cannon shot without incurring any mismatch on the 5-bank lit target map.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T1 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T1 switch number into $D2 as the hit target

c. Detected a lit target was hit, add function ID 0088 to the scheduler
d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0088

2. The ID 0088 Function is called due to the T1 switch handler logic at step 1.c.
   a. There is currently no function ID 0085 running (no ID 0085 to cancel)
   b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
   c. Flag $48 is not set:
      i. Decrement the per-player "hits remaining" value from 3 to 2
      ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
      In this case, due to Step 1.b, this clears lamp for target T1, as expected

Flowchart: 00E1

3. The ID 00E1 Function is called, it was originally scheduled at step 1.d
   a. Sleep for 0.0625 seconds (62.5 mS) starts

4. Approximately 50mS elapses

Flowchart: Switch Handler

5. The 5-bank switch-handler called on behalf of T2 switch closure
   a. ID 0084 is NOT found in the scheduler, was cancelled at step 2.b (cannon shot period)
   b. ID 00E1, however, is found in the scheduler, added at step 1.d
   c. Save T2 switch number into $D2 as the hit target
   d. Detected a lit target was hit, add function ID 0088 to the scheduler
   e. Flag $48 is not set, add function ID 00E1 to the scheduler
      i. Since step 1.d already added it, the same single function is re-scheduled.
      ii. **The existing 00E1 from step 3 is cancelled** and new one is scheduled to begin.

Flowchart: 0088

6. The ID 0088 Function is called due to the T2 switch handler logic at step 5.d.
   a. There is currently no function ID 0085 running (no ID 0085 to cancel)
   b. The ID 0084 was cancelled at step 2.b (no ID 0084 to cancel)
   c. Flag $48 is not set:
      i. Decrement the per-player "hit remaining" from 2 to 1
      ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
      Due to step 5.c, this clears lamp for target T2, as expected

Flowchart: 00E1

7. The ID 00E1 Function is called, it was scheduled at step 5.e.ii.
   a. Sleep for 0.0625 seconds starts and completes
   a. Function reports "1 Hit Remaining"

The scenario above shows the case when a player hits multiple targets with a single cannon shot. The result is the correct number of "hits remaining" and a correct lit target map on the subsequent cannon shot attempt. The most interesting part of this depiction is the highlighted step that reveals the secondary lit target hit effectively restarts the 62.5mS timer during which a subsequent lit target could be registered and counted against the hits remaining.

Scenario 3: Second target detected 75mS after first target

In this scenario, the secondary hit on switch T2 is detected 75mS after the switch T1 was processed. Prior to this 2nd target hit on T2, the game had already completed the 0088 function which was added to the scheduler by the T1 switch hit. Additionally, since 75mS elapsed, the 00E1 function was also allowed

to run to completion. This scenario is provided out of completeness to show how the sequence would occur in the case where the second target hit was delayed (such as a ball hitting first T1 target then bouncing off a post to come back and hit the secondary target T2).

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T1 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T1 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0088

2. The ID 0088 Function is called due to the T1 switch handler logic at step 1.c.
    a. There is currently no function ID 0085 running (no ID 0085 to cancel)
    b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
    c. Flag $48 is not set:
        i. Decrement the per-player "hits remaining" value from 3 to 2
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to Step 1.b, this clears lamp for target T1, as expected

Flowchart: 00E1

3. The ID 00E1 Function is called due to step 2.d
    a. Sleep for 0.0625 seconds starts and completes
    b. Function reports "2 Hits Remaining"
4. Approximately 75mS elapses

Flowchart: Switch Handler

5. The 5-bank switch-handler called on behalf of T2 switch closure
    a. ID 0084 is not found in the scheduler, was cancelled at step 2.b (cannon shot period)
    b. ID 00E1 is not found in the scheduler, it ran and completed after step 3
    c. Target T2 hit is treated as an ordinary game-play target hit

The example shown above shows how the delayed 5-bank hit is simply treated as an ordinary target hit after the 00E1 function, after having been scheduled due to T1 target hit, has been given its chance to run to completion. The follow-up hit on target T2 gets doesn't count toward "hits remaining" and is treated as a regular target hit that happens during game play.

Case 2: Unlit target hit followed by a lit target hit

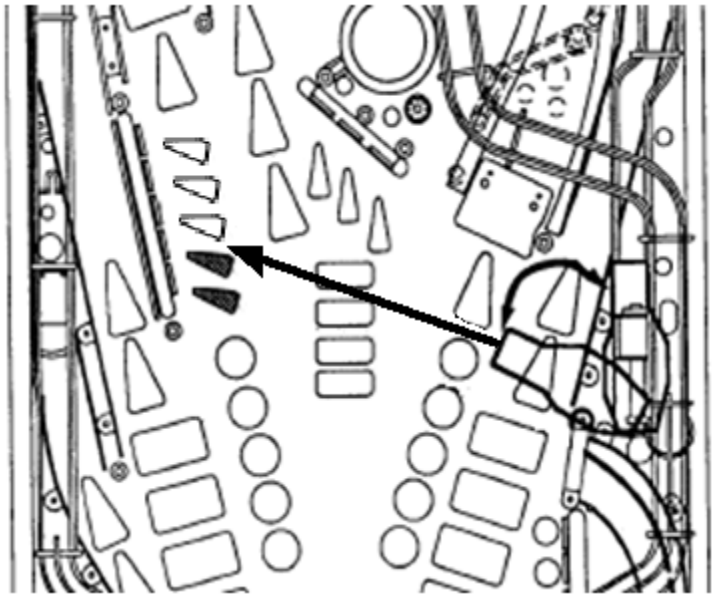
In this case, the ball hits two targets at, seemingly, the same time, one is unlit and the other is lit. The game software detects the unlit target first followed by the lit target hit.

In this case the result can be:

- The player's "hits remaining" value is decremented for the lit target hit.

This scenario is worth examining because it shows how the game technically detected a 'miss' at first however it allows for a tiny period of time for the second hit on a lit target to get counted as a successful target hit.

Imagine a case where the bottom two targets are lit, T4 and T5 and 2 hits are remaining for multiball. In this scenario the ball hits between T3 and T4 causing the game software to detect hit on unlit T3 followed by lit target T4.



There are multiple ways in which the sequence can go, depending on timing of how quickly the software detects the secondary hit of switch T4. The text below will go over multiple timing scenarios:

- Scenario 1: Second target detected 0mS after first target
- Scenario 2: Second target detected 25mS after first target
- Scenario 3: Second target detected 46.875mS after first target
- Scenario 4: Second target detected 50mS after first target

Scenario 1: Second target detected 0mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the unlit target T3 immediately followed by processing of the lit target T4.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T3 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T3 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled

i. Add function ID 0085 to the scheduler for “you missed”

Flowchart: Switch Handler

2. The 5-bank switch-handler called on behalf of T4 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T4 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0088

3. The ID 0088 Function is called due to the T4 switch handler logic at step 2.c.
    a. There IS currently function ID 0085 “you missed’ scheduled and it is cancelled.
    b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
    c. Flag $48 is not set:
        i. Decrement the per-player “hits remaining” value from 2 to 1
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to Step 2.b, this clears lamp for target T4, as expected

Flowchart: 00E1

4. The ID 00E1 Function is called due to step 2.d
    d. Sleep for 0.0625 seconds starts and completes
    e. Function reports “1 Hit Remaining”

The sequence above shows how the initial “miss” gets overruled by the immediate “hit” that takes place afterwards. This allows for a ball strike that hits two targets at seemingly the same time, gives preference to the lit target.

Scenario 2: Second target detected 25mS after first target

In this scenario, both targets are hit a seemingly the same time and the game processes the unlit target T3 shortly followed by processing of the lit target T4 25mS later. In this scenario the T4 target is detected after the 0085 “you missed” function is started but not completed. The secondary hit on T4 takes place during the sleep that occurs at the start of the 0085 “you missed” function.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T3 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T3 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Add function ID 0085 to the scheduler for “you missed”

Flowchart: 0085

2. The ID 0085 Function is called due to the T3 switch handler logic at step 1.c.i
    a. The 0.046875 sleep is started

3. Approximately 25mS elapses

Flowchart: Switch Handler

4. The 5-bank switch-handler called on behalf of T4 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T4 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0088

5. The ID 0088 Function is called due to the T4 switch handler logic at step 4.c.
    a. There IS currently function ID 0085 “you missed’ scheduled and it is cancelled.

b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
c. Flag $48 is not set:
    i. Decrement the per-player "hits remaining" value from 2 to 1
    ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
    In this case, due to Step 4.b, this clears lamp for target T4, as expected

Flowchart: 00E1

6. The ID 00E1 Function is called due to step 4.d
    a. Sleep for 0.0625 seconds starts and completes
    b. Function reports "1 Hit Remaining"

The scenario depicted above has same result as previous scenario. In this scenario the 0085 "you missed' function was allowed to start but during its sleep period, the subsequent T4 target hit was processed and handled which resulted in the cancellation of the 0085 function and subsequent treatment of T4 as the hit target.

Scenario 3: Second target detected 46.875mS after first target

This scenario explores what seemed, at first, to be potential trouble area that could happen if the timing was made slightly longer between the unlit target T3 and the subsequent lit target T4 hit.

In this scenario, both targets are hit at seemingly the same time and the game processes the unlit target T3 shortly followed by processing of the lit target T4, 46,875mS later. In this scenario the T4 target is detected after the 0085 "you missed" function is started but not completed. The secondary hit on T4 takes place during the sleep that occurs at the start of the 0085 "you missed" function. In this case the timing is such that the 0085 function awakens from its sleep after the secondary T4 target hit is processed and before the 0088 function is allowed to run. *Upon further analysis it turns out that this situation cannot happen, as described below.*

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T3 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T3 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Add function ID 0085 to the scheduler for "you missed"

Flowchart: 0085

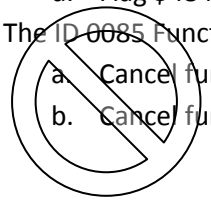
2. The ID 0085 Function is called due to the T3 switch handler logic at step 1.c.i
    a. The 0.046875 sleep is started
3. Approximately 46.875mS elapses

Flowchart: Switch Handler

4. The 5-bank switch-handler called on behalf of T4 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T4 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0085

5. The ID 0085 Function sleep is completed and its function resumes
    a. Cancel function ID 0085 (cancels any other scheduled 0085 functions)
    b. Cancel function ID 00B4



c. Cancel function ID 0084 (cannon shot period)
d. Plays the "you missed" speech and extinguishes 5-bank lamps.

Flowchart: 0088

6. The ID 0088 Function is called due to the T4 switch handler logic at step 4.c.
   d. There is no 0085 "you missed" scheduled, nothing to cancel
   e. The function ID 0084 was cancelled at step 5.c, nothing to cancel
   f. Flag $48 is not set:
      i. Decrement the per-player "hits remaining" value from 2 to 1
      ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
      In this case, due to Step 4.b, this clears lamp for target T4, as expected

Flowchart: 00E1

7. The ID 00E1 Function is called due to step 4.d
   a. Sleep for 0.0625 seconds completes
   b. Function reports "1 Hit Remaining"

In the depicted scenario, the game detected the target miss and plays the "you missed' speech however the 0088 and 00E1 are subsequently allowed to proceed which will treat it as a target hit.

Upon further analysis it was found that the way in which the 0088 is added to the scheduler, it is inserted into the top of the list and, because of this, if the 0085 function is about to wake up from its sleep, the 0088 function will still be allowed to run prior to 0085. When 0088 runs prior to 0085, the 0088 function will cancel the 0085 function, thus preventing the possibility of the game declaring "you missed". Due to this subsequent analysis, the circle-slash symbol is put over the steps above that will not actually take place. The actual flow will be the same a previously depicted scenario.

Scenario 4: Second target detected 50mS after first target

In this scenario, the unlit target T3 is hit and its code is allowed to run to completion and the lit target T4 is detected 50mS after the unlit target T3 was hit and processed. This scenario is shown for completeness and to depict how subsequent target hits are treated simply as ordinary 5-bank hits.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T3 switch closure
   a. ID 0084 is found in the scheduler (cannon shot period)
   b. Save T3 switch number into $D2 as the hit target
   c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
      i. Add function ID 0085 to the scheduler for "you missed"

Flowchart: 0085

2. The ID 0085 Function is called due to the T3 switch handler logic at step 1.c.i
   a. The 0.046875 sleep is started and completes.
   b. Cancel function ID 0085 (cancels any other scheduled 0085 functions)
   c. Cancel function ID 00B4
   d. Cancel function ID 0084 (cannon shot period)
   e. Plays the "you missed" speech and extinguishes 5-bank lamps.
3. Approximately 50mS after the first switch hit T3 has elapsed

Flowchart: Switch Handler

4. The 5-bank switch-handler called on behalf of T4 switch closure
   a. ID 0084 is not running due to step 2.d having cancelled it
   b. ID 00E1 is not running due to not having been added to scheduler

c. Treat T4 hit as an ordinary target hit.

This scenario is essentially a single ‘miss’ followed by a regular game-play target hit of T4. No unexpected behaviors take place in this scenario.

Case 3: Lit target hit followed by an unlit target hit

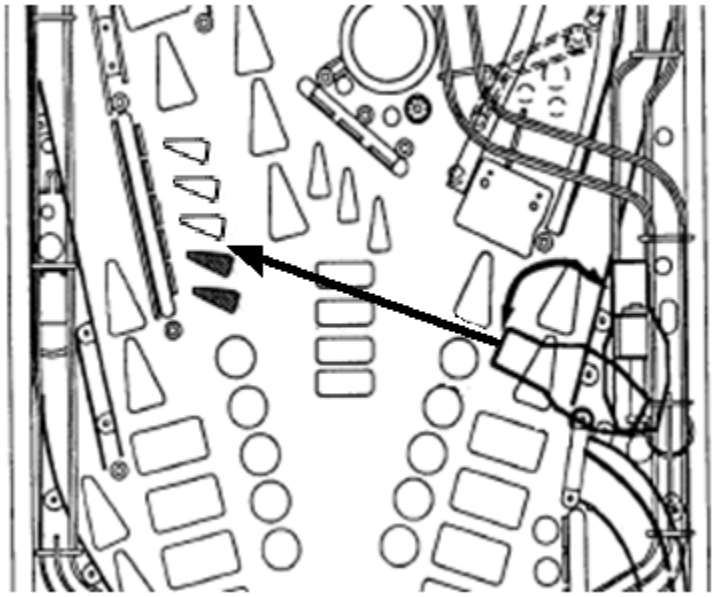
In this case, the ball effectively hits two targets at the same time, one is lit and the other is unlit. The game software detects the lit target first followed by the unlit target.

In this case the result can be:

- The player’s “hits remaining” value is decremented for the lit target hit.

This case is examined for completeness and in contrast to the previous case. In this case the first lit target hit wins as opposed to the previous case where the first unlit target hit was overruled by the follow-up lit target hit. Both cases are in favor of awarding the player a hit. In the previous case this is true even though the game detected a ‘miss’ at first.

Imagine a case where the bottom two targets are lit, T4 and T5 and 2 hits are remaining for multiball. In this scenario the ball hits between T3 and T4 causing the game software to detect lit T4 followed by unlit switch T3.



There are multiple ways in which the sequence can go, depending on timing of how quickly the software detects the secondary hit of switch T3. The text below will go over multiple timing scenarios:

- Scenario 1: Second target detected 0mS after first target
- Scenario 2: Second target detected 50mS after first target
- Scenario 3: Second target detected 75mS after first target

Scenario 1: Second target detected 0mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the lit target T4 immediately followed by processing of the unlit target T3.

Depending on the timing, the result can be:

- The player's "hits remaining" value is decremented for the lit target hit.
- The lit lamps for the remaining hits may become inconsistent with reported "hits remaining".

This scenario is interesting because it depicts another way in which the lit lamps can become inconsistent with the reported "hits remaining" value.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T4 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T4 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: Switch Handler

2. The 5-bank switch-handler called on behalf of T3 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T3 switch number into $D2 as the hit target, overwriting the T1 value in $D2
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Add function ID 0085 to the scheduler for "you missed"

Flowchart: 0088

3. The ID 0088 Function is called due to the T4 switch handler logic at step 1.c.
    a. There IS currently function ID 0085 "you missed' scheduled (step 2.c.i) and it is cancelled.
    b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
    c. Flag $48 is not set:
        i. Decrement the per-player "hits remaining" value from 2 to 1
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to Step 2.b, this re-clears lamp for target T3 instead of the lit T4.

Flowchart: 00E1

4. The ID 00E1 Function is called due to step 1.d
    a. Sleep for 0.0625 seconds starts and completes
    b. Function reports "1 Hit Remaining"

The sequence above shows how the initial "hit" is sustained and not overruled by the immediate "miss" that takes place afterwards. This allows for a ball strike that hits two targets at the same time, giving preference to the lit target.

The result of this, also, is that the two lamps T4 and T5 are still lit at the next cannon shot even though the reported hits remaining is now 1. As in the previous case/scenario where this happened, **this is because the game allows multiple switch handlers to overwrite the hit target value at $D2 before the stored value in $D2 is processed by the 0088 function**. This means the 0088 function only processes whatever was the most recent target hit as it updates the player's 5-bank lit-lamp map.

Scenario 2: Second target detected 50mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the lit target T4 followed by processing of the unlit target T3 approximately 50 mS later. The purpose of this scenario is to show how a delayed secondary hit to an unlit target can get discarded and not even processed as a regular target hit.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T4 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T4 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0088

2. The ID 0088 Function is called due to the T4 switch handler logic at step 1.c.
    a. There is no 0085 "you missed" scheduled, nothing to cancel
    b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
    c. Flag $48 is not set:
        i. Decrement the per-player "hits remaining" value from 2 to 1
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to Step 1.b, this clears lamp for target T4, as expected

Flowchart: 00E1

3. The ID 00E1 Function is called due to step 1.d
    a. Sleep for 0.0625 seconds starts

4. Approximately 50mS elapses

Flowchart: Switch Handler

5. The 5-bank switch-handler called on behalf of T3 switch closure
    a. ID 0084 is not found in the scheduler, was cancelled at step 2.b (cannon shot period)
    b. ID 00E1 IS found in the scheduler, it was scheduled at step 1.d and is currently in sleep
    c. Save T3 switch number into $D2 as the hit target
    d. Detected a lit target was NOT hit, ID 00E1 IS currently scheduled, so no further work

Flowchart: 00E1

6. The ID 00E1 Function, scheduled at step 1.d, awakes from its sleep
    a. The sleep started at step 3.a completes
    b. Function reports "1 Hit Remaining"

The sequence above shows how the initial "hit" is sustained and not overruled by the immediate "miss" that takes place afterwards. This allows for a ball strike that hits two targets at the same time, gives preference to the lit target.

This scenario is interesting because it shows that subsequent target misses are discarded and not treated as an ordinary target hit if they are received during the window of time in which the 00E1 is performing its sleep.

Scenario 3: Second target detected 75mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the lit target T4 followed by processing of the unlit target T3 approximately 75 mS later. The purpose of this scenario is to show how a delayed secondary hit to an unlit target can get treated as an ordinary target hit.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T4 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T4 switch number into $D2 as the hit target
    c. Detected a lit target was hit, add function ID 0088 to the scheduler
    d. Flag $48 is not set, add function ID 00E1 to the scheduler

Flowchart: 0088

2. The ID 0088 Function is called due to the T4 switch handler logic at step 1.c.
    a. There is no 0085 “you missed” scheduled, nothing to cancel
    b. The function ID 0084 is cancelled from the scheduler (cannon shot period)
    c. Flag $48 is not set:
        i. Decrement the per-player “hits remaining” value from 2 to 1
        ii. Load hit target number from $D2 and clear its lamp from per-player 5-bank map
        In this case, due to Step 1.b, this clears lamp for target T4, as expected

Flowchart: 00E1

3. The ID 00E1 Function is called due to step 1.d
    a. Sleep for 0.0625 seconds starts and completes
    b. Function reports “1 Hit Remaining”
4. Approximately 75mS after the first switch hit T4 has elapsed

Flowchart: Switch Handler

5. The 5-bank switch-handler called on behalf of T3 switch closure
    a. ID 0084 is not found in the scheduler, was cancelled at step 2.b (cannon shot period)
    b. ID 00E1 is not found in the scheduler, it ran to completion after step 3
    c. Process the T4 hit as an ordinary target hit.

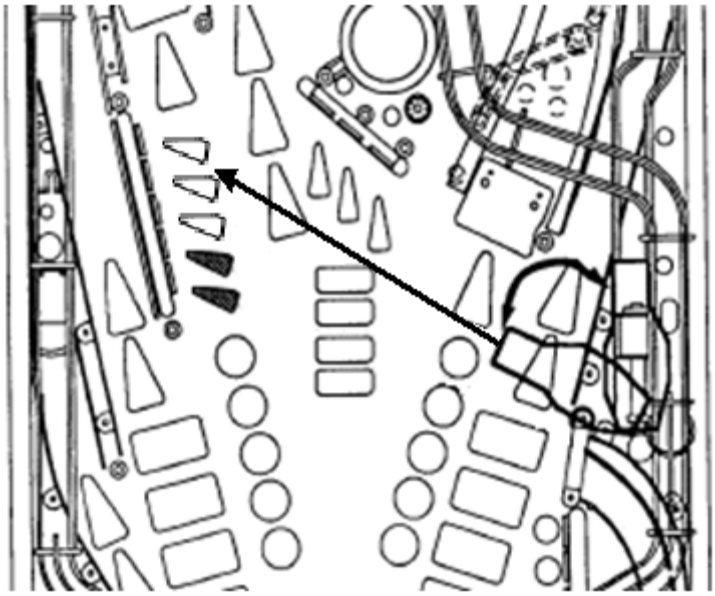
This scenario depicts what is effectively an ordinary subsequent hit to an unlit target T3 after having been awarded the successful hit on target T4. There is nothing out of the ordinary in this scenario.

Case 4: Unlit target hit followed by unlit target hit

In this case, the ball effectively hits two targets at the same time, both are unlit.

This case is being analyzed out of completeness and to, perhaps, determine if any potential troubles are present in this part of the game code.

Imagine a case where the bottom two targets are lit, T4 and T5 and 2 hits are remaining for multiball. In this scenario the ball hits between T1 and T2 causing the game software to detect unlit T1 followed by unlit switch T2.



There are multiple ways in which the sequence can go, depending on timing of how quickly the software detects the secondary hit of target T2. The text below will go over multiple timing scenarios:

- Scenario 1: Second target detected 0mS after first target
- Scenario 2: Second target detected 25mS after first target
- Scenario 3: Second target detected 50ms after first target

Scenario 1: Second target detected 0mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the unlit target T1 immediately followed by processing of the unlit target T2.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T1 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T1 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Add function ID 0085 to the scheduler for “you missed”

Flowchart: Switch Handler

2. The 5-bank switch-handler called on behalf of T2 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T2 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Re-add function ID 0085 to the scheduler for “you missed” (re-schedules 0085)

Flowchart: 0085

3. The ID 0085 Function is called due to the T2 switch handler logic at step 2.c.i.
    a. The 0.046875 sleep is started and completes.
    b. Cancel function ID 0085 (cancels any other scheduled 0085 functions)
    c. Cancel function ID 00B4
    d. Cancel function ID 0084 (cannon shot period)
    e. Plays the "you missed" speech and extinguishes 5-bank lamps.

The sequence above is fairly uneventful except to note that the 0085 "you missed" function gets re-scheduled when the secondary hit to unlit target T2 takes place. The way in which the 0085 gets scheduled, it will first cancel any existing 0085 and then add the new 0085 function to the scheduler.

Scenario 2: Second target detected 25mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the unlit target T1 followed by processing of the unlit target T2 25mS later. In this case, the 0085 "you missed" function is allowed to start and the hit to unlit target T2 happens while the 0085 function is performing its sleep.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T1 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T1 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Add function ID 0085 to the scheduler for "you missed"

Flowchart: 0085

2. The ID 0085 Function is called due to the T2 switch handler logic at step 1.c.i.
    a. The 0.046875 sleep is started

3. Approximately 25mS elapses

Flowchart: Switch Handler

4. The 5-bank switch-handler called on behalf of T2 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T2 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Re-add function ID 0085 to the scheduler for "you missed" (re-schedules 0085)
        ii. The re-schedule will cancel existing 0085 which is sleeping, and start a new 0085.

Flowchart: 0085

5. The ID 0085 Function is called due to the T2 switch handler logic at step 4.c.i.
    a. The 0.046875 sleep is started and completes.
    b. Cancel function ID 0085 (cancels any other scheduled 0085 functions)
    c. Cancel function ID 00B4
    d. Cancel function ID 0084 (cannon shot period)
    e. Plays the "you missed" speech and extinguishes 5-bank lamps.

The sequence above is interesting because it shows how the secondary hit to unlit target T2 effectively restarted the period in which subsequent target hits will be accepted as a possible hit or miss on the hunter ship. The hit on unlit target T2 caused the existing sleep from step 2 to get cancelled and such time period is restarted at step 5.a.

Scenario 3: Second target detected 50mS after first target

In this scenario, both targets are hit at seemingly the same time and the game processes the unlit target T1 followed by processing of the unlit target T2 50mS later. In this case, the 0085 "you missed" function is allowed to start and complete before the secondary hit to unlit target T2 happens.

Flowchart: Switch Handler

1. The 5-bank switch-handler called on behalf of T1 switch closure
    a. ID 0084 is found in the scheduler (cannon shot period)
    b. Save T1 switch number into $D2 as the hit target
    c. Detected a lit target was NOT hit, ID 00E1 is not scheduled
        i. Add function ID 0085 to the scheduler for "you missed"

Flowchart: 0085

2. The ID 0085 Function is called due to the T2 switch handler logic at step 1.c.i.
    a. The 0.046875 sleep is started and completes.
    b. Cancel function ID 0085 (cancels any other scheduled 0085 functions)
    c. Cancel function ID 00B4
    d. Cancel function ID 0084 (cannon shot period)
    e. Plays the "you missed" speech and extinguishes 5-bank lamps.
3. Approximately 50mS after the until T1 switch closure has elapsed.

Flowchart: Switch Handler

4. The 5-bank switch-handler called on behalf of T2 switch closure
    a. ID 0084 is not found in the scheduler, was cancelled at step 2.d (cannon shot period)
    b. ID 00E1 is not found in the scheduler
    c. Treat T2 hit as an ordinary target hit.

The sequence above effectively depicts a single target miss followed by an ordinary hit on another unlit target afterwards. The secondary hit is handled as an ordinary target hit during gameplay since the initial "miss" has already been handled in full.

Hunter Ship 5-Bank Target Logic Summary

The previous analysis boils the logic of 5-bank hunter ship hit/miss to the following:

- After an initial "hit" there is a 62.5mS window for subsequent target hits/misses.
    - Each "hit" during the 62.5mS window restarts the window for another 62.5mS period.
    - Any "miss" hits during the 62.5mS window is discarded.
- After an initial "miss" there is a 46.875mS window for subsequent target hits/misses.
    - Subsequent "miss" hits during the 46.875mS window cause the window to restart.
    - Any "hit" during the 46.875mS window overrides the "miss", starting a 65.5mS window
        - Subsequent "hit" or "miss" are handled per the case of initial "hit", above.

The set of rules the software appears to be enforcing can be described as follows:

- During a cannon shot attempt, the game allows for the possibility of the ball to hit multiple targets in the same shot.
- There is a very small window of time the game allows for such multiple target hits to be detected by the software.

- If any of the hit targets during this window are lit, then the game will award the hunter ship strike (even if the first hit target was not lit).
- Any lit targets that are hit during this window count against the “hits remaining” value to start multiball.

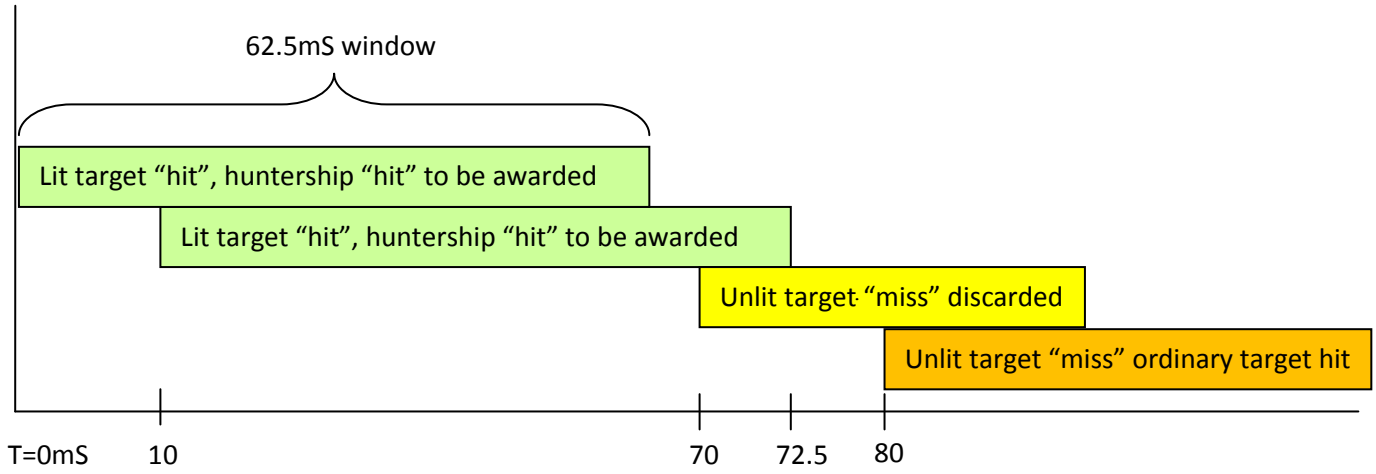
Furthermore:

- Additional target hits during the window can restart the window, increasing its overall time.
- A coding flaw can result in multiple target hits during this window to result in mismatch in the remaining lit targets as compared to the reported “hits remaining” value. *Will be fixed in L8.4.*

It is worth pointing out that when you consider how the game will read and process switches in a given column, there is always a chance that the order in which the switches are processed may not be the same as the order in which 2 switches have physically closed. Additionally, the switch contact gaps can affect which of two simultaneously hit targets gets detected as closed first. Because of this, the use of this window mechanism provides for some fairness in the benefit of the player by awarding the hit even if an unlit target may have, technically, been hit first.

Hunter Ship 5-Bank Target Logic Summary Example 1

The timeline example below shows the case of an initial hit with subsequent hits/misses.

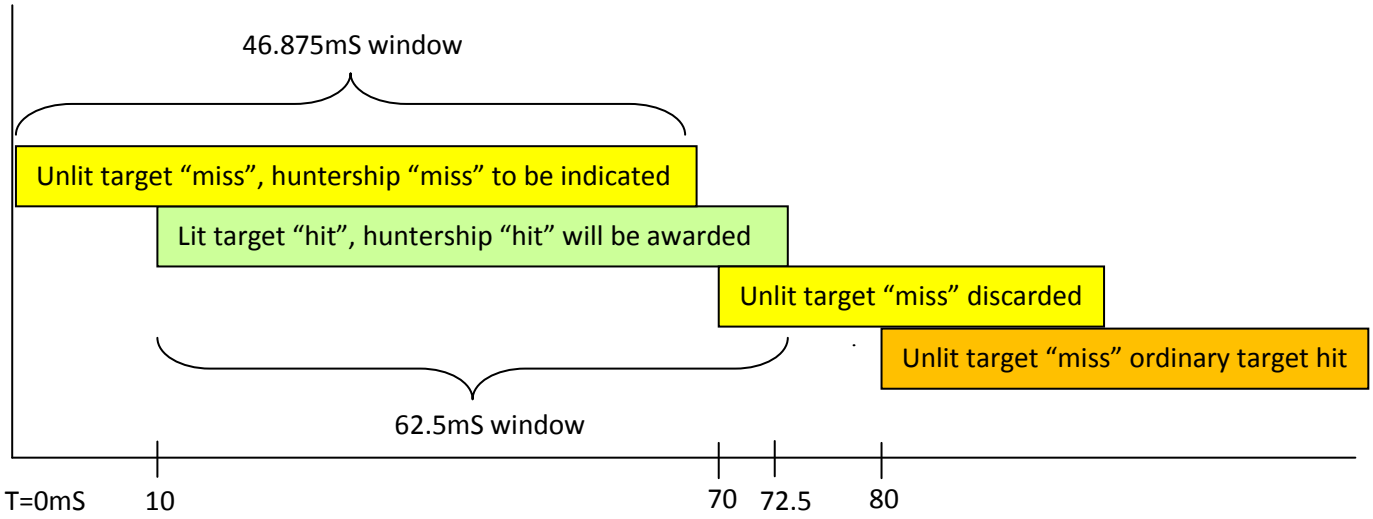


Summary:

- At T=0, the first lit 5-bank target is hit, game will award a hunter ship hit.
- 10mS, A secondary lit 5-bank target is hit, game restarts 62.5mS window.
- 70mS, A third 5-bank target is it, it is an unlit target. This “miss” is discarded.
- 72.5mS, The (restarted) 62.5mS timer expired and the game awards hunter ship hit (2 target hits)
- 80mS, Another 5-bank target is hit, it is an unlit target and is handled as ordinary gameplay hit.

Hunter Ship 5-Bank Target Logic Summary Example 2

The timeline below shows the case of an initial miss with subsequent hits/misses.



Summary:

- At T=0, the first unlit 5-bank target is hit, game set to declare a hunter ship miss.
- 10mS, A secondary lit 5-bank target is hit, game starts a 62.5mS window and will award a hit.
- 70mS, A third 5-bank target is it, it is an unlit target. This “miss” is discarded.
- 72.5mS, The 62.5mS timer expired and the game awards hunter ship hit (1 target hit)
- 80mS, Another 5-bank target is hit, it is an unlit target and is handled as ordinary gameplay hit.

Hunter Ship 5-Bank Target L8.4 Updates

Based on the information presented above, the changes for L8.4 regarding the hunter ship 5-bank targets are as follows:

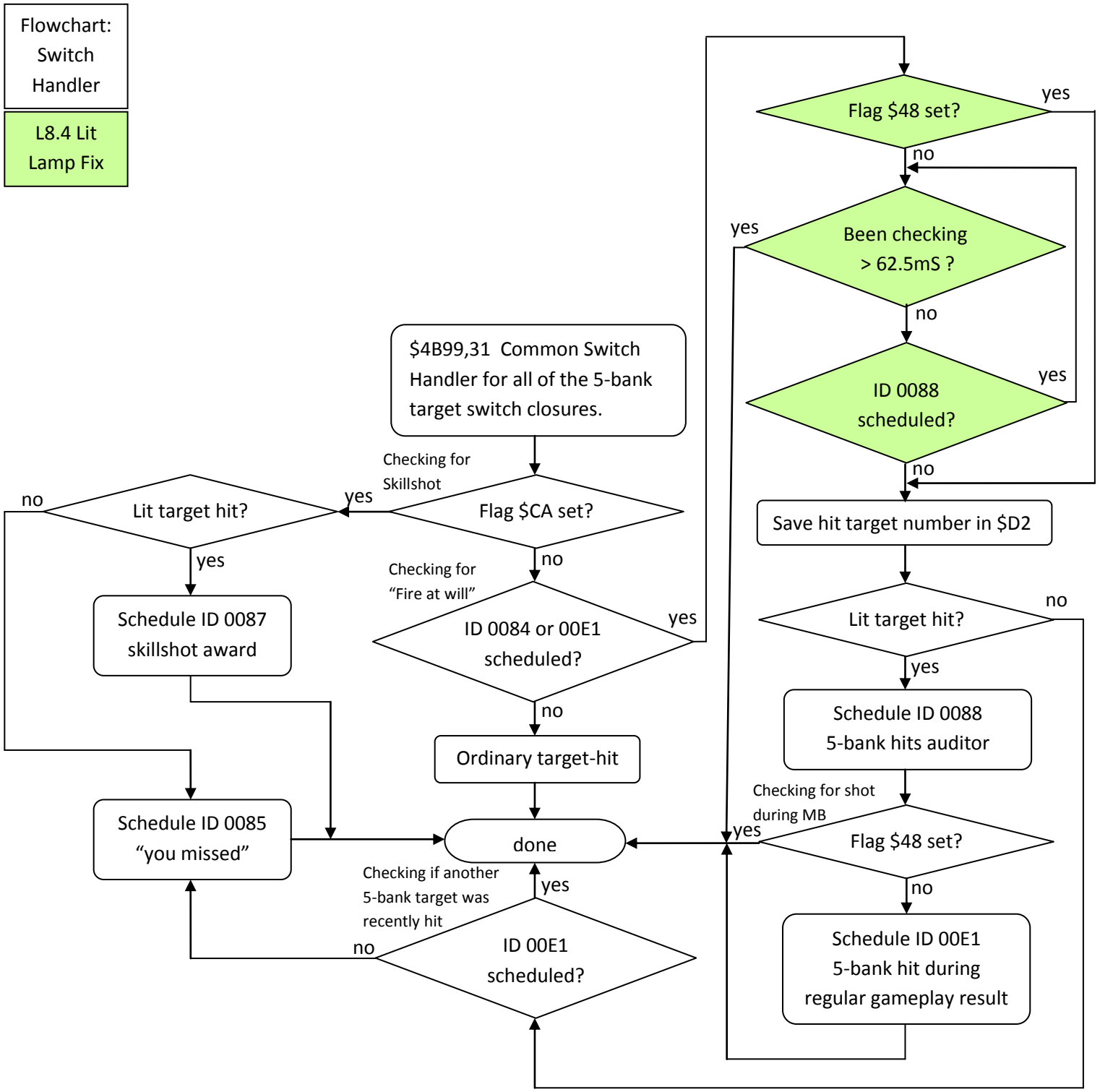
- Fix the lit-lamp/targets remaining discrepancy problem
- Add feature adjustment to restrict target hits to 1 per cannon shot

Each of these changes are described in more detail, below.

Fix the lit-lamp/targets remaining discrepancy problem

The problem with lit-lamps not matching the reported “Targets Remaining” is being fixed in L8.4 as a distinct bug fix. A fairly minimal code change is being added to update the previously depicted switch handler flowchart for the 5-bank targets.

The updated logic will always result in correctly lit target lamps since the 0088 function will always be ensured to process the correct target-hit value in the $D2 memory location. The updated flowchart depicting the bug fix is shown below.



The updated logic, as highlighted in green in the flowchart above, will check if the 0088 function is in the scheduler and, if so, repeatedly check until the 0088 function is no longer in the scheduler. This will only take place if the $48 flag is *not* set since the $48 flag is set when the cannon shot is during multiball (for jackpot or super jackpot attempt). The $48 flag is clear for cannon shots that are being done not during multiball. The $48 flag clear is when cannon shots are in progression towards multiball. The problem with lit-lamp mismatch is only for non-multiball cannon shot attempts, and as such, the fix logic is only done when $48 flag is clear.

Not depicted in the flowchart is that the code will perform small sleep of 15.625mS between each check of whether function 0088 is in the scheduler. This will attempt 4 times before giving up which is why the flowchart indicates it will wait up to 62.5mS before giving up. The smallest sleep period allowed is 15.625mS which is why the total wait time is 62.5mS.

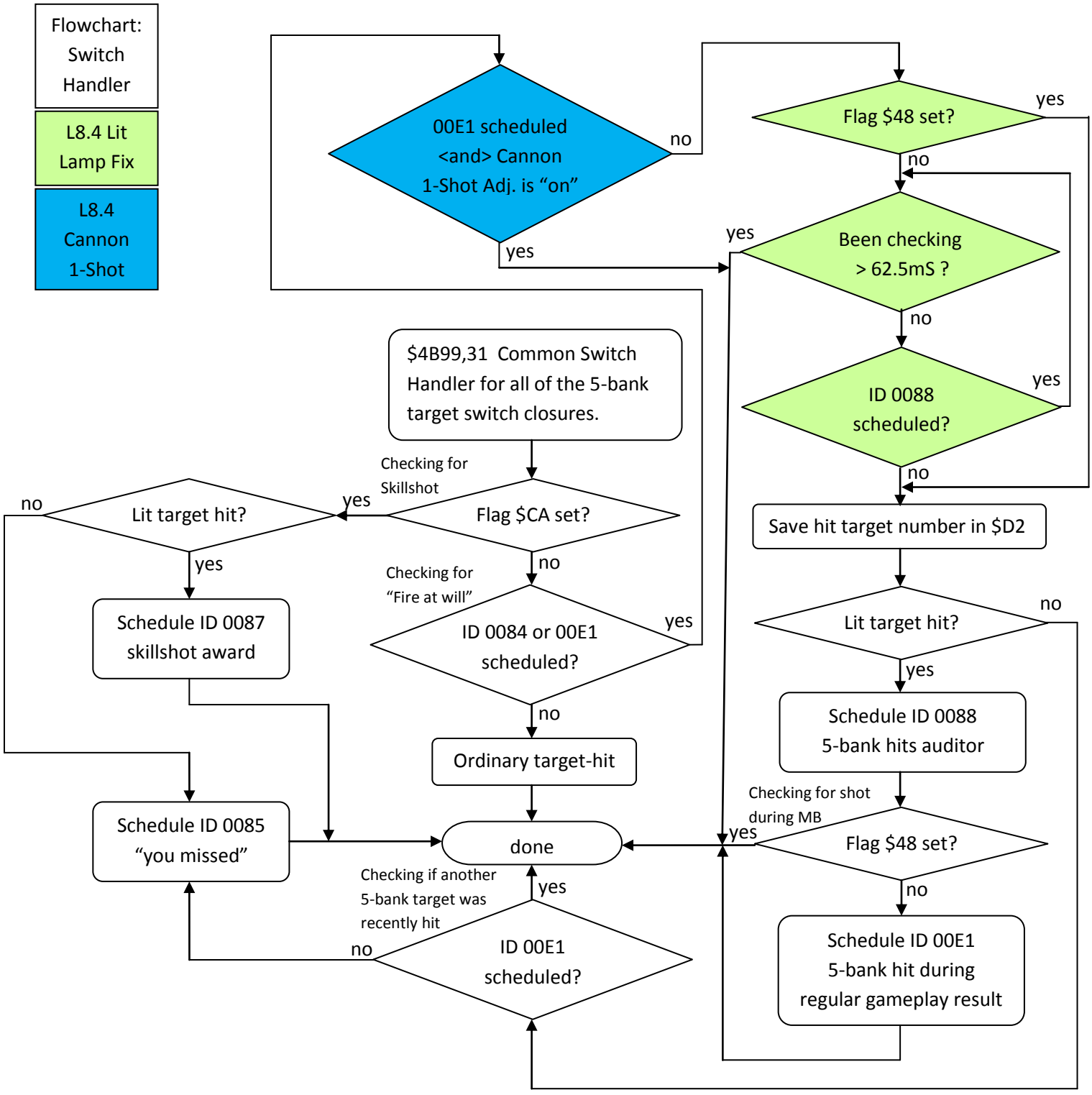
The new logic will ensure that the 0088 function, if currently scheduled, is allowed to run to completion before the current switch handler code will proceed. This means the 0088 function will be ensured to process the $D2 value and update the lamp map *prior* to the current switch handler update of $D2 with the new target number. When the switch handler code is allowed to proceed, it loads $D2 with the currently hit target number at a moment *after* the 0088 function had finished reading the previous $D2 value.

In testing on the simulator, the rare times that the 0088 function is in the scheduler during processing of a 5-bank target, after only a single 15.625mS sleep, is when the subsequent check determines that the 0088 function is no longer in the scheduler (meaning it had ran to completion so the current switch-handler can proceed). In the event that the 0088 function is in the scheduler for the entire 62.5mS wait, then the logic will simply discard the current target hit. It will not accumulate any points. Since the 0088 function is in the scheduler from *a previous* lit target hit, this discarded target-hit would not be detrimental to the player since a hit is being awarded (the discarded target hit, in this case, is a secondary target hit that came in after the initial target hit). The analysis of the 0088 function doesn't show any potentially lengthy function run times in this scenario. It is not expected that the 0088 function will be in the scheduler for the full 62.5mS in this scenario and, as such, it is not expected that a target hit will be discarded by this mechanism.

Add feature adjustment to restrict target hits to 1 per cannon shot

As an enhancement to L8.4, a new adjustment is added in L8.4 to allow the cannon shot to only accumulate, at most, a single hunter ship strike per cannon shot attempt. This is being added for game operators who want to increase the difficulty of acquiring multiballs and for players who want to add an extra level of challenge to the game play.

The inclusion of this new adjustment into the switch handler logic adds new logic to the flowchart, as depicted below.


Flowchart: Switch Handler
L8.4 Lit Lamp Fix
L8.4 Cannon 1-Shot
00E1 scheduled <and> Cannon 1-Shot Adj. is "on"
no
yes
Flag $48 set?
yes
no
Been checking > 62.5mS ?
yes
no
ID 0088 scheduled?
yes
no
Save hit target number in $D2
Lit target hit?
no
yes
Schedule ID 0088 5-bank hits auditor
Checking for shot during MB
Flag $48 set?
yes
no
Schedule ID 00E1 5-bank hit during regular gameplay result
$4B99,31 Common Switch Handler for all of the 5-bank target switch closures.
Checking for Skillshot
Flag $CA set?
yes
no
Lit target hit?
no
yes
Schedule ID 0087 skillshot award
Checking for "Fire at will"
ID 0084 or 00E1 scheduled?
yes
no
Ordinary target-hit
Schedule ID 0085 "you missed"
done
Checking if another 5-bank target was recently hit
ID 00E1 scheduled?
yes
no

The logic for limiting each cannon shot to a single target hit is highlighted in the blue addition to the switch handler flowchart, above.

As indicated, the logic checks whether the 00E1 function is in the scheduler and, if so, if the new adjustment for single cannon hit is 'on' then the target hit is discarded and no points awarded. If the 00E1 function is not in the scheduler or if the single cannon hit is 'off' then logic advances to the previously described logic to ensure the 0088 function runs to completion.

The 00E1 function is only added to the scheduler when a cannon shot has been attempted during regular game play outside of multiball and such cannon shot had hit a lit target, thus scheduling the 00E1 function to tally the target hits and report to the user the remaining number of targets for multiball or launching multiball when all of the lit targets have been hit.

Because of this logic in how 00E1 is used, this updated logic can simply check if a previously scheduled 00E1 function is in the scheduler and, if so, this means that the current switch handler routine is for purposes of potentially accumulating a subsequent target hit to bundle along with the existing target hit. If the desire is to not allow subsequent hunter ship hits then the new logic simply needs to discard the target hit when 00E1 is already running.

Hunter Ship 5-Bank Target L8.4 Code Changes

Shown below is the original L-8 switch handler function for the 5-bank targets. As previously mentioned, each of the 5-bank targets calls a slightly different starting point into this function but they all end up running the same code that starts at $4B9B,31, ROM offset 0x44B9B. The following function is mostly, but not fully annotated.

```
----------------------------------------;--------------------------------------------------
                                        ;
                                        ;
4B93: 20 06       BRA   $4B9B           ; SwitchMatrixHdlr_Target1High()  A=0x11 B=0x39 top
4B95: 20 04       BRA   $4B9B           ; SwitchMatrixHdlr_Target2()      A=0x12 B=0x3A
4B97: 20 02       BRA   $4B9B           ; SwitchMatrixHdlr_Target3()      A=0x13 B=0x3B
4B99: 20 00       BRA   $4B9B           ; SwitchMatrixHdlr_Target4()      A=0x14 B=0x3C
                                        ; SwitchMatrixHdlr_Target5Low()   A=0x15 B=0x3D bottom
                                        ;
4B9B: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
4B9E: CA                                ; 0xCA == Skill Shot
4B9F: 25 2C       BCS   $4BCD           ;
                                        ; Get here when 0xCA flag is set (C-bit was clear)
                                        ; This happens during skill shot
4BA1: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4BA4: 00 83                             ; ID 0083 (TBD)
4BA6: 24 03       BCC   $4BAB           ; The c-bit is clear (0083 is running) during skill shot
                                        ;
4BA8: BD 45 47    JSR   $4547           ;
                                        ;
4BAB: C0 39       SUBB  #$39            ; B=sw idx. Subtract 0x39 makes it 0-based target 0..4
4BAD: F1 05 FA    CMPB  $05FA           ; $05FA has skill shot winning index, compare hit target
4BB0: 27 11       BEQ   $4BC3           ; If skill shot is hit, branch to $4BC3
                                        ;
                                        ; Skill shot was missed
4BB2: F6 05 FA    LDB   $05FA           ; B gets actual winning target
4BB5: CB 11       ADDB  #$11            ; Add 0x11 to winning target
```

```
4BB7: 1F 01       TFR   D,X             ; Put result into X
4BB9: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
4BBC: 00 85                             ; ID 0085 == This is the "you missed" scheduled function
4BBE: 4D 74 31                          ;
4BC1: 20 08       BRA   $4BCB           ;
                                        ;
                                        ; Skill shot was hit, give award
4BC3: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
4BC6: 00 87                             ;
4BC8: 4C E5 31                          ;
4BCB: 20 79       BRA   $4C46           ; Branch to the end, all done handling switch
                                        ;-------------------------------------------------------
                                        ;-------------------------------------------------------
                                        ; Here handling hunter ship hit and check if we're in
                                        ; a "fire at will" period.
                                        ;-------------------------------------------------------
                                        ;-------------------------------------------------------
                                        ;
4BCD: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4BD0: 00 84                             ;  ID 0084 Ball-gun-to-Target-Period function is running
4BD2: 24 07       BCC   $4BDB           ; If 0084 is running = Fire-At-Will mode
                                        ; Otherwise, if 0084 isn't running then check if 00E1
                                        ;
4BD4: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4BD7: 00 E1                             ;  ID 00E1 Hunter ship hit scheduled handler
4BD9: 25 57       BCS   $4C32           ; If 00E1 is not running then ordinary target hit
                                        ; If 00E1 is running then Fire-At-Will mode
                                        ;
                                        ; Here when hunter ship crosshairs is running.
                                        ;
4BDB: C0 39       SUBB  #$39            ; Make B 0-based index of hit target 0..4
4BDD: D7 D2       STB   $D2             ; Store hit-target in $D2
4BDF: 34 02       PSHS  A               ;
4BE1: 96 D3       LDA   $D3             ;
4BE3: 81 FF       CMPA  #$FF            ; $D3 == 0xFF means we're going for super jackpot.
4BE5: 35 02       PULS  A               ; Moving target bit is in $05F9
4BE7: 26 07       BNE   $4BF0           ;
4BE9: F1 05 F9    CMPB  $05F9           ;
4BEC: 27 1C       BEQ   $4C0A           ; If B target index = super jackpot target, we have hit
4BEE: 20 07       BRA   $4BF7           ; Super Jackpot missed, go to $4BF7
                                        ;
                                        ; -----------------------------------------------------
                                        ; Here for "fire at will" check target hit
                                        ; -----------------------------------------------------
4BF0: C6 40       LDB   #$40            ;
4BF2: BD 9E DD    JSR   $9EDD           ; Call function that checks if target was hit
4BF5: 24 13       BCC   $4C0A           ; If c-clear, Go to hunter ship hit, had a genuine hit
                                        ; -----------------------------------------------------
                                        ; -----------------------------------------------------
                                        ;
                                        ; When Super-Jackpot is missed we land here
                                        ; When genuine hunter ship miss happens, we land here
4BF7: 1F 89       TFR   A,B             ;
4BF9: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4BFC: 00 E1                             ;
4BFE: 24 30       BCC   $4C30           ;
4C00: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
4C03: 00 85                             ; ID 0085 == This is the "you missed" scheduled function
4C05: 4F 70 31                          ; Sleep 03, Cancels 0085, 00B4, 0084, Clears 0x48 flag
4C08: 20 26       BRA   $4C30           ; Done handling hunter ship miss, branch down to return
                                        ;
                                        ;
```

```
                                        ; ---------------------------
                                        ; --== SUPER JACKPOT HIT ==--
                                        ; ---------------------------
                                        ; -----------------------------
                                        ; --== ORDINARY JACKPOT HIT ==--
                                        ; -----------------------------
4C0A: BD 8B 3D    JSR   $8B3D           ; AddLinkedListEntry()
4C0D: 00 88                             ; ID 0088 == 5-bank hits auditor
4C0F: 4D A1 31                          ;
4C12: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-clear when flag set
4C15: 48                                ; Cannon shot during multiball when flag set (c-clear)
                                        ; 0x48 bit is 0x08 bit of flags starting at $0328.
                                        ; So this is 0x80 bit of $0328.
4C16: 24 18       BCC   $4C30           ; Branch when c-clear, when flag is set, multiball
4C18: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()  $48 flag is clear (C is set)
4C1B: 00 E1                             ;
4C1D: 4C 54 31                          ; Hunter ship multiball determinator
4C20: BD 48 8D    JSR   $488D           ;
4C23: BD 48 9E    JSR   $489E           ; Lock2StateSet()
4C26: 86 04       LDA   #$04            ;
4C28: C6 01       LDB   #$01            ;
4C2A: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
4C2D: 6E 5A 3B                          ;
4C30: 20 14       BRA   $4C46           ; Branch to the end, all done handling switch
                                        ;
                                        ;-------------------------------------------------------
                                        ; Here for ordinary target-hit during game play
                                        ;-------------------------------------------------------
4C32: BD FA A8    JSR   $FAA8           ;
4C35: 7E 4C 38    JMP   $4C38           ; <nop>
4C38: BD 85 1F    JSR   $851F           ;
4C3B: 38                                ;
4C3C: BD 50 DA    JSR   $50DA           ;
4C3F: 80 11       SUBA  #$11            ;
4C41: 8B 2D       ADDA  #$2D            ;
4C43: BD CC 3A    JSR   $CC3A           ;
4C46: 7E 99 A2    JMP   $99A2           ;
                                        ;
                                        ;
----------------------------------------;-------------------------------------------------
```

In order to add new logic to the switch handler code, existing code is replaced with a single call to new function that will handle the new logic. Since there is no available space in the same bank $31 for new code, the new function is located in bank $3A and called using the WPC function to call function located in another bank. The changes to the above function are highlighted below:

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ;
4B93: 20 06       BRA   $4B9B           ; SwitchMatrixHdlr_Target1High()  A=0x11 B=0x39 top
4B95: 20 04       BRA   $4B9B           ; SwitchMatrixHdlr_Target2()      A=0x12 B=0x3A
4B97: 20 02       BRA   $4B9B           ; SwitchMatrixHdlr_Target3()      A=0x13 B=0x3B
4B99: 20 00       BRA   $4B9B           ; SwitchMatrixHdlr_Target4()      A=0x14 B=0x3C
                                        ; SwitchMatrixHdlr_Target5Low()   A=0x15 B=0x3D bottom
                                        ;
4B9B: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
4B9E: CA                                ; 0xCA == Skill Shot
4B9F: 25 2C       BCS   $4BCD           ;
                                        ; Get here when 0xCA flag is set (C-bit was clear)
                                        ; This happens during skill shot
```

```
4BA1: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4BA4: 00 83                             ; ID 0083 (TBD)
4BA6: 24 03       BCC   $4BAB           ; The c-bit is clear (0083 is running) during skill shot
                                        ;
4BA8: BD 45 47    JSR   $4547           ;
                                        ;
4BAB: C0 39       SUBB  #$39            ; B=sw idx. Subtract 0x39 makes it 0-based target 0..4
4BAD: F1 05 FA    CMPB  $05FA           ; $05FA has skill shot winning index, compare hit target
4BB0: 27 11       BEQ   $4BC3           ; If skill shot is hit, branch to $4BC3
                                        ;
                                        ; Skill shot was missed
4BB2: F6 05 FA    LDB   $05FA           ; B gets actual winning target
4BB5: CB 11       ADDB  #$11            ; Add 0x11 to winning target
4BB7: 1F 01       TFR   D,X             ; Put result into X
4BB9: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
4BBC: 00 85                             ; ID 0085 == This is the "you missed" scheduled function.
4BBE: 4D 74 31                          ;
4BC1: 20 08       BRA   $4BCB           ;
                                        ;
                                        ; Skill shot was hit, give award
4BC3: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
4BC6: 00 87                             ;
4BC8: 4C E5 31                          ;
4BCB: 20 79       BRA   $4C46           ; Branch to the end, all done handling switch
                                        ;-------------------------------------------------------
                                        ;-------------------------------------------------------
                                        ; Here handling hunter ship hit and check if we're in
                                        ; a "fire at will" period.
                                        ;-------------------------------------------------------
                                        ;-------------------------------------------------------
                                        ;
~~4BCD: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found~~
~~4BD0: 00 84                             ;  ID 0084 Ball-gun-to-Target-Period function is running~~
~~4BD2: 24 07       BCC   $4BDB           ; If 0084 is running = Fire-At-Will mode~~
~~                                        ; Otherwise, if 0084 isn't running then check if 00E1~~
~~                                        ;~~
~~4BD4: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found~~
~~4BD7: 00 E1                             ;  ID 00E1 Hunter ship hit scheduled handler~~
~~4BD9: 25 57       BCS   $4C32           ; If 00E1 is not running then ordinary target hit~~
~~                                        ; If 00E1 is running then Fire-At-Will mode~~
                                        ;-------------------------------------------------------
                                        ;-------------------------------------------------------
                                        ; Call new L8.4 function to fix lamp mismatch, etc
                                        ;-------------------------------------------------------
;---------------------------------------;-------------------------------------------------------
4BCD: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
4BD0: 7A B9 3A                          ; $7AB9,3A == ROM offset 0x6BAB9
4BD3: 25 5D       BCS   $4C32           ; C-set == treat it as ordinary target hit.
4BD5: 27 59       BEQ   $4C30           ; C-clr and Z-set == silently discard the hit.
4BD7: 7E 4B DB    JMP   $4BDB           ; C-clr and Z-clr == check for hunter ship hit.
4BDA: 12          NOP                   ; filler instruction
;---------------------------------------;-------------------------------------------------------
                                        ;-------------------------------------------------------
                                        ;
                                        ; Here when hunter ship crosshairs is running.
                                        ;
4BDB: C0 39       SUBB  #$39            ; Make B 0-based index of hit target 0..4
4BDD: D7 D2       STB   $D2             ; Store hit-target in $D2
4BDF: 34 02       PSHS  A               ;
4BE1: 96 D3       LDA   $D3             ;
4BE3: 81 FF       CMPA  #$FF            ; $D3 == 0xFF means we're going for super jackpot.
4BE5: 35 02       PULS  A               ; Moving target bit is in $05F9
```

```
4BE7: 26 07       BNE   $4BF0           ;
4BE9: F1 05 F9    CMPB  $05F9           ;
4BEC: 27 1C       BEQ   $4C0A           ; If B target index = super jackpot target, we have hit
4BEE: 20 07       BRA   $4BF7           ; Super Jackpot missed, go to $4BF7
                                        ;
                                        ; ------------------------------------------------------
                                        ; Here for "fire at will" check target hit
                                        ; ------------------------------------------------------
4BF0: C6 40       LDB   #$40            ;
4BF2: BD 9E DD    JSR   $9EDD           ; Call function that checks if target was hit
4BF5: 24 13       BCC   $4C0A           ; If c-clear, Go to hunter ship hit, had a genuine hit
                                        ; ------------------------------------------------------
                                        ; ------------------------------------------------------
                                        ;
                                        ; When Super-Jackpot is missed we land here
                                        ; When genuine hunter ship miss happens, we land here
4BF7: 1F 89       TFR   A,B             ;
4BF9: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4BFC: 00 E1                             ;
4BFE: 24 30       BCC   $4C30           ;
4C00: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
4C03: 00 85                             ; ID 0085 == This is the "you missed" scheduled function.
4C05: 4F 70 31                          ; Sleep 03, Cancels 0085, 00B4, 0084, Clears 0x48 flag
4C08: 20 26       BRA   $4C30           ; Done handling hunter ship miss, branch down to return
                                        ;
                                        ;
                                        ; ----------------------------
                                        ; --== SUPER JACKPOT HIT ==--
                                        ; ----------------------------
                                        ; -------------------------------
                                        ; --== ORDINARY JACKPOT HIT ==--
                                        ; -------------------------------
4C0A: BD 8B 3D    JSR   $8B3D           ; AddLinkedListEntry()
4C0D: 00 88                             ; ID 0088 == 5-bank hits auditor
4C0F: 4D A1 31                          ;
4C12: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-clear when flag set
4C15: 48                                ; Cannon shot during multiball when flag set (c-clear)
                                        ; 0x48 bit is 0x08 bit of flags starting at $0328.
                                        ; So this is 0x80 bit of $0328.
4C16: 24 18       BCC   $4C30           ; Branch when c-clear, when flag is set, multiball
4C18: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()  $48 flag is clear (C is set)
4C1B: 00 E1                             ;
4C1D: 4C 54 31                          ; Hunter ship multiball-determinator
4C20: BD 48 8D    JSR   $488D           ;
4C23: BD 48 9E    JSR   $489E           ; Lock2StateSet()
4C26: 86 04       LDA   #$04            ;
4C28: C6 01       LDB   #$01            ;
4C2A: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
4C2D: 6E 5A 3B                          ;
4C30: 20 14       BRA   $4C46           ; Branch to the end, all done handling switch
                                        ;
                                        ;------------------------------------------------------
                                        ; Here for ordinary target-hit during game play
                                        ;------------------------------------------------------
4C32: BD FA A8    JSR   $FAA8           ;
4C35: 7E 4C 38    JMP   $4C38           ; <nop>
4C38: BD 85 1F    JSR   $851F           ;
4C3B: 38                                ;
4C3C: BD 50 DA    JSR   $50DA           ;
4C3F: 80 11       SUBA  #$11            ;
4C41: 8B 2D       ADDA  #$2D            ;
4C43: BD CC 3A    JSR   $CC3A           ;
```

```
4C46: 7E 99 A2    JMP   $99A2           ;
                                        ;
                                        ;
----------------------------------------;------------------------------------------------
```

As shown in the highlighted code addition, above, the new code takes place at the start of non-skill shot handling where it is about to check for whether or not “Fire at will” mode is in effect. The code replaces the 0084 and 00E1 scheduler lookup with a call to function at WPC address $7AB9,3A at ROM offset 0x6BAB9.

This new function can return 3 different states to control subsequent code flow:

- Returning C-bit set causes code to treat target hit as ordinary non-multiball 5-bank hit.
- Returning C-bit clear and Z-bit set causes the target hit to be silently discarded.
- Returning C-bit clear and Z-bit clear mean “Fire at Will” mode and game proceeds accordingly

The new function at $7AB9,3A, ROM offset 0x6BAB9, is as follows:

```
;---------------------------------------;-------------------------------------------
                                        ;
                                        ; Hunter ship hit mode determinator
                                        ;
                                        ; Return c-set to proceed with ordinary 5-bank hit
                                        ; Return c-clr and z-set to have hit silently discarded
                                        ; Return c-clr and z-clr to proceed with hunter ship hit
                                        ;
7AB9: 34 12       PSHS  X,A             ; $7AB9,3A == ROM offset 0x6BAB9
                                        ;
7ABB: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
7ABE: 00 E1                             ;  ID 00E1 Hunter ship hit scheduled handler
7AC0: 25 09       BCS   $7ACB           ; If c-set then hit-period not running, check for 0084
                                        ;
                                        ;-------------------------------------------------------
                                        ; Getting here means the 00E1 function is running which
                                        ; means a hit has already been registered during this
                                        ; period.  We can now allow normal code to proceed
                                        ; and allow additional hits to accumulate, OR if the new
                                        ; adjustment is set to allow only a single hit per
                                        ; cannon shot, then discard this target hit by
                                        ; returning C-clr and Z-set.
                                        ;-------------------------------------------------------
7AC2: BD 86 5B    JSR   $865B           ; LookupGameAdjParameter1andCheckIfEqualsParameter2()
7AC5: 1C 00                             ; 0x1C, $1BF1:$1BF2 FeatureAdj028, Cannon 1 Hit Adj=0x1C
7AC7: 27 09       BEQ   $7AD2           ; If adj is OFF then allow multiple hits per cannon shot
7AC9: 20 23       BRA   $7AEE           ; Adjustment is ON, single-hit mode, skip down to return
                                        ; C-clr and Z-set to discard target hit
                                        ;-------------------------------------------------------
                                        ;
7ACB: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
7ACE: 00 84                             ;  ID 0084 Ball-gun-to-Target-Period function is running
7AD0: 25 20       BCS   $7AF2           ; If C-set we are NOT in Fire-At-Will mode,
                                        ; Return C-set to caller to proceed with ordinary 5-bank
                                        ; target hit during game play
                                        ;
                                        ;-------------------------------------------------------
                                        ; We are in a "fire at will" mode.  Normally code loads
                                        ; the hit target index into $D2 at this point but here
```

```
                                        ; is where the fix code will wait until 0088, if found
                                        ; in the scheduler, runs to completion before this code
                                        ; proceeds.  This will fix problem where the $D2 value
                                        l could be overwritten with different target number
                                        ; before the 0088 function had a chance to process it
                                        ; (to clear lamp from map).
                                        ;-------------------------------------------------------
7AD2: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
7AD5: 48                                ; 0x48 flag is set when gun is loaded during multiball
                                        ;  flag 0x48 is RAM $0328 bit 0x80
7AD6: 24 12       BCC   $7AEA           ; If c-clr then 0x48 flag is set, branch to code that
                                        ; returns c-clr and z-clr for normal hunter ship hit.
                                        ; This is done since 0x48 flag being set means the 0088
                                        ; function won't care about the value of $D2
                                        ;
                                        ; Getting here means the 0088 function WILL care about
                                        ; the value of $D2 so we need to make sure there are no
                                        ; 0088 functions in the scheduler before proceeding.
                                        ; This will perform a simple look to wait for the 0088
                                        ; function to finish its work before proceeding.
                                        ;
7AD8: 86 04       LDA   #$04            ; Will loop this many times waiting for function 0088 to
                                        ; complete, 62.5mS worth of sleeps
                                        ;
7ADA: BD 86 90    JSR   $8690           ; -\ SearchLinkedListForId()  // c-clear = ID is found
7ADD: 00 88                             ;  |  ID 0088 == Cleanup function at multiball start
7ADF: 25 09       BCS   $7AEA           ;  | C-set means 0088 not running, so proceed normally,
                                        ;  | C-clr means 0088 running, wait for it to complete
7AE1: 4A          DECA                  ;  |
7AE2: 27 0A       BEQ   $7AEE           ;  | For some reason 0088 took longer than expected.
7AE4: BD 83 46    JSR   $8346           ;  | Sleep()
7AE7: 01                                ;  |
7AE8: 20 F0       BRA   $7ADA           ; -/ Keep checking
                                        ;
                                        ; Only get here when 0088 is absolutely not currently
                                        ; scheduled/running and "fire at will" target hit
                                        ; handler can proceed
                                        ;
7AEA: 1C FA       ANDCC #$00FA          ; Return c-clr and z-clr to proceed with hunter ship hit
7AEC: 20 04       BRA   $7AF2           ;
                                        ;
7AEE: 1C FA       ANDCC #$00FA          ; Return c-clr and z-set to discard the target hit
7AF0: 1A 04       ORCC  #$0004          ; This is for paths where the target hit is ignored
                                        ;
                                        ;
7AF2: 35 92       PULS  A,X,PC          ;
                                        ;
                                        ;
;---------------------------------------;-----------------------------------------
```

As a reminder, the applicable function IDs for this code change are as follows:

- 0084 is the function that is scheduled when cannon is shot. Cannon-to-target period function.
- 00E1 is the function that is scheduled to tally hit targets and report remaining or start multiball
- 0088 is the function that is scheduled when a target is hit, decrements counter & updates lamps

The new L8.4 code, shown above, performs the following tasks:

- Checks if 00E1 function is in the scheduler:

    - If 00E1 is in the scheduler then the Cannon 1 Hit adjustment is checked, if 'on' then the code returns C-clear and Z-set which will cause the target hit to be discarded.
    - If 00E1 is NOT in the scheduler then code proceeds to next check.
- Checks if 0084 function is in the scheduler:
    - If 0084 is NOT in the scheduler then the code returns C-set to cause the target hit to be treated as an ordinary game play 5-bank target hit.
    - If 0084 is in the scheduler then code proceeds to the next check.
- Checks if 0088 function is in the scheduler:
    - If 0088 is in the scheduler then code will wait until it is not in the scheduler or give up waiting. The wait consists of 4 loops, each consisting of a brief 15.625mS sleep followed by another check if 0088 is in the scheduler. This loop happens for 4 passes for a wait up to 62.5mS. If the 0088 is in the scheduler after 62.5mS then code returns C-clear and Z-set to cause the target hit to be discarded.
    - If 0088 is not in the scheduler or the previous loop discovered the 0088 is no longer in the scheduler then code returns C-clear and Z-clear to allow the target hit to be processed as a hunter ship hit or miss by the original calling code depicted earlier.

This logic in the new function, above, effectively implements the updated switch handler flowchart previously depicted. The end result of this added logic is:

- The lit-lamp problem will no longer take place, and
- If the new adjustment is set to 'on' the game will only allow a single hit per cannon shot.
- If the new adjustment is set to 'off' the game will allow multiple hits per cannon shot which is the same as the original T-2 code, allowing the player to more rapidly reach multiball with multiple target hits by a single cannon shot.

Hunter Ship 5-Bank Target L8.4 New Adjustment

Refer to the original text from L8.3 document describing *The L-8 Adjustments Memory Map*. The table with all adjustments is updated to accommodate the new L8-4 adjustment as highlighted in the *table portion*, below:

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

N/A	$1BF9:$1BFA	<Adjustments Checksum>

** New adjustments in L8.3 shown for reference.*

*** New L8.4 adjustment (additional new L8.4 adjustments are depicted later in this document)*

The *Feature Adjustments Metadata* table is updated as indicated in the abbreviated table content, below. Below is depicted the increased table size and new content. Additional updates in L8.4 will add additional adjustments causing the resulting L8.4 table size to also increase so 'xx' is depicted here.

```
;----------------------------------------;-----------------------------------------------------
7000: 00 xx, Incremented by 1              ; Table entries is xx (xx entries), Incremented by 1
7002: 0C                                   ; Bytes per table entry is 0C (12 bytes)
...
70FF: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.21, Drptrgt 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 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.28, Cannon 1 Hit
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>
;----------------------------------------;-----------------------------------------------------
```

The metadata table, above, is located at $7000,3D, ROM offset 0x77000 and is updated with the new Boolean adjustment to utilize existing 'On/Off' text and appropriately translated German/French texts the same as other Boolean adjustments. The name of this new adjustment is added to the existing English, German, and French menu string tables as indicated in the table below.

	Table Start	New Pointer Bytes	New String Address	New String Value
English	$6700,3D	$673B,3D	$68EB,3D	"CANNON 1 HIT"
German	$6A00,3D	$6A3B,3D	$6B7F,3D	"KANONE 1 TREFFER"
French	$6D00,3D	$6D3B,3D	$6ED4,3D	"CANON 1 COUP"

With all of the above changes in place, the L8.4 has a fully implemented new feature adjustment and corrected bug-fix for the lit-lamp mapping at the 5-bank targets.

The PAPA Lost Super Jackpot Bug

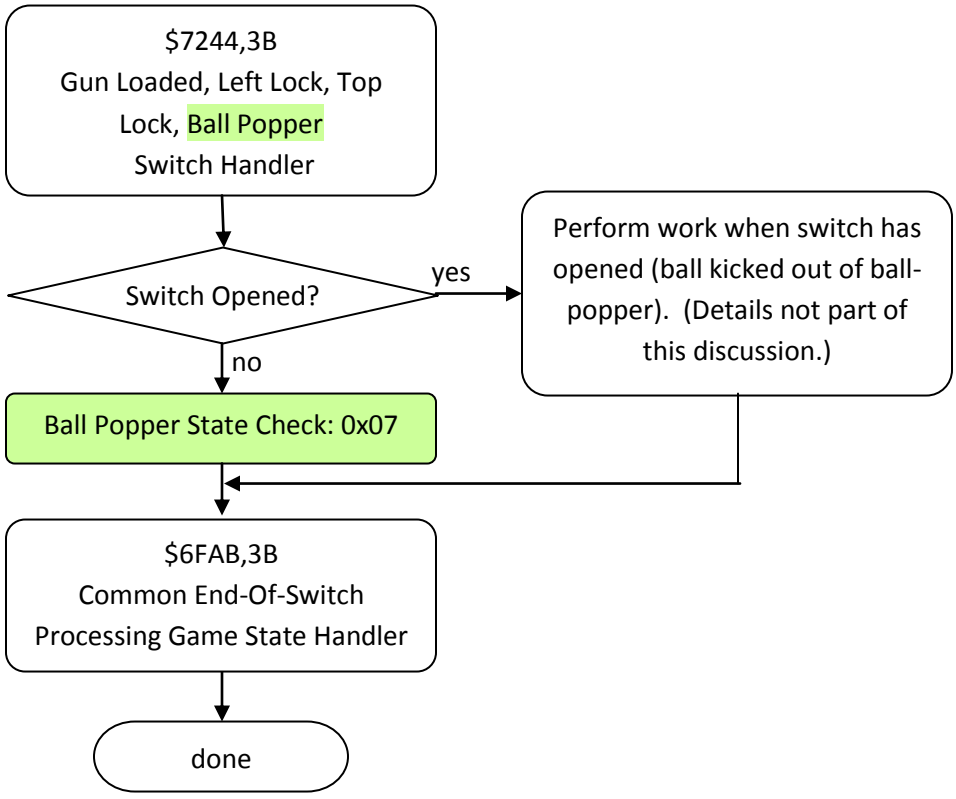
For L8.4 another goal is to understand and fix the bug referred to here as the PAPA Lost Super Jackpot bug. This bug was recorded on video and, at time of this writing, available through a simple online search. This occurred during the PAPA 14 event which took place August 11-14, 2011. In a online video titled "PAPA 14 World Pinball Championships Final Round – Terminator 2" at roughly 6:00 into the 30:34 video, what is shown is an obvious bug where the Super Jackpot was not awarded when it should have been.

For readers not interested in technical details, refer to the section "PAPA Lost Super Jackpot Investigation Summary" later in this document for a brief summary of the problem.

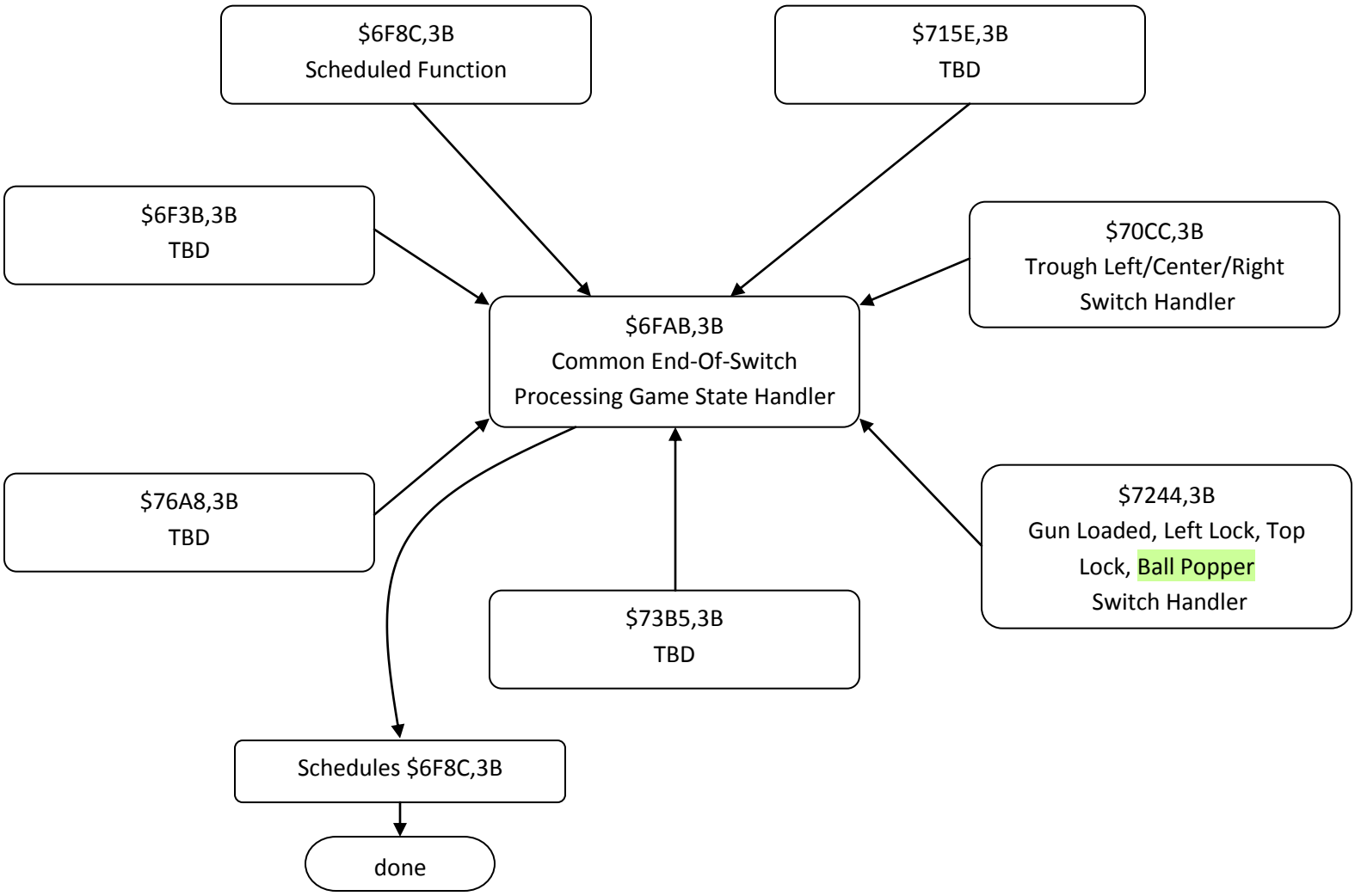
PAPA Lost Super Jackpot Investigation and Analysis

For L8.4 a close examination of the code was done along with various attempts to reproduce the issue on a pinball simulator. Eventually the nature of the problem became apparent. The problem and the solution are being documented here for posterity and for future refinement as needed.

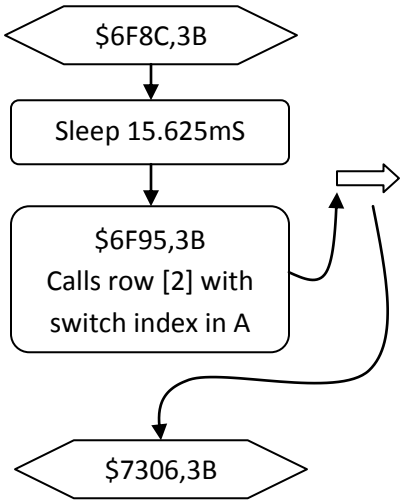
To describe what leads up to the problem, some high level flowcharts are shown depicting the various logic that takes place when the ball lands in the ball popper. The focus on these descriptions is the case where the ball is launched into the popper for purposes of scoring a super jackpot although this may also be applicable to cases where the ball is shot into the popper at other times during game play. Shown below is some logic when ball lands in the ball-popper:



The simplified flowchart, above, shows how the ball-popper handler, when switch is closed (ball landed on ball popper switch), performs a state check 0x07 (details further below) followed by a $6FAB,3B end of switch processing. This end-of-switch processing is actually called from a variety of sources as depicted below. This common function then spawns a separate function to perform further work as described below.



As shown above, the common function schedules the $6F8C,3B to finish the work. This is probably done this way to prevent the single switch handler from taking too much time doing its tasks.

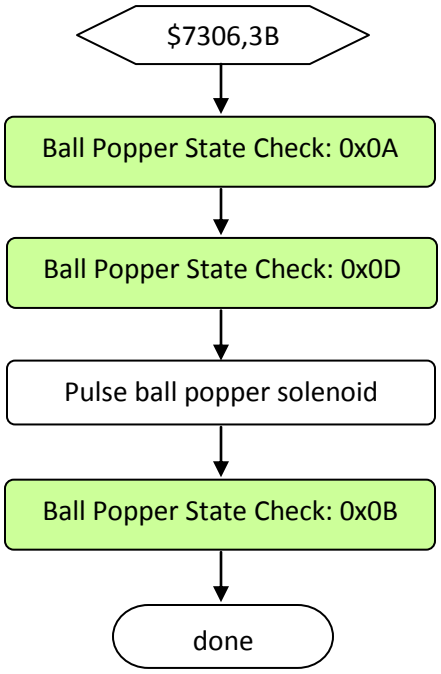


Function Callback Table $75AF,3B	
Index	**Use**
[0] $8EBC	
[1] $8EBC	
[2] $7306,3B	Always called
[3] $744C,3B	
[4] $778D,3B	
[5] $8EBC	
[6] $8EBC	
[7] $72D7,3B	
[8] $7429,3B	
[9] $776F,3B	

Survey of $6F8C,3B and its call of $6F95,3B appears that entry [2] of the table is always called. It's not clear under what circumstances the other rows are used.

The $6F8C,3B function appears to lookup a function pointer from table $75AF,3B and then call the function at index [2]. Initial analysis seems that this [2] is always the function that gets called. Somebody can correct this observation if wrong but it appears code looks at $75AF,3B table and uses its data-size value as the index to use. Since the table $75AF,3B always reports a table entry size of 2 (2 bytes per table entry for each 16-bit address of a function), this is the value used as index into the array.

The $7306,3B function then performs the remaining work as part of the end-of-switch processing. There is an array in RAM that tracks additional state information which is not fully analyzed and part of this analysis. For purposes of explaining the PAPA Lost Super Jackpot bug a simplified description of what the $7306,3B function does while processing the ball-popper prior to super jackpot is shown next.



As shown, above, when the ball popper switch goes through the end-of-switch processing and gets into the $7306,3B scheduled function, there are 3 additional state checks that are done: 0x0A, 0x0D, and 0x0B. These three state checks are performed after the initial state check 0x07, described earlier when the ball popper switch was initially closed. *Details on this ball popper state check are to follow*. *Also, note the term 'state check' is informally used here as the function call purpose rather informally. It is merely a call to a function with a given reference number.*

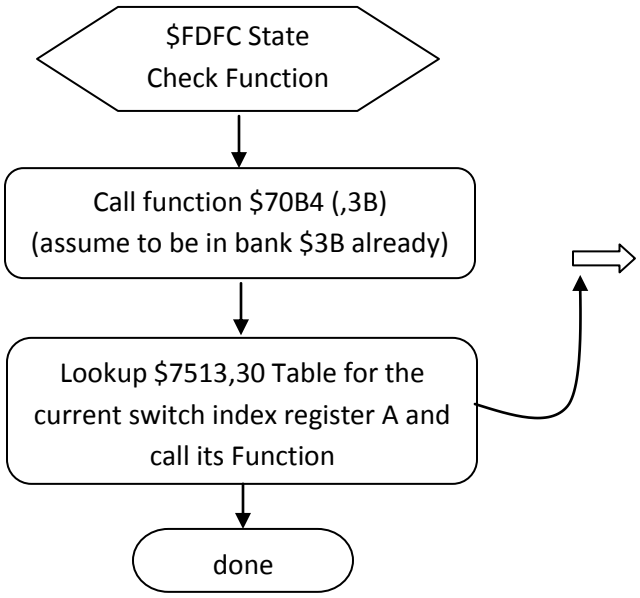
A survey of the total state checks that are done for various ball-popper activities were observed and recorded in the table below. These findings include the address from where the state check was initiated in the bank $3B.

Ball Popper Activity	**Ball Popper State Checks**
Ball popper hit during ordinary game play in advancement towards starting a multiball	State check: 0x07, from $7293,3B
	State check: 0x0A, from $734E,3B
	State check: 0x0D, from $735E,3B
	State check: 0x0B, from $73AD,3B
Ball popper hit during multiball where ball is retained in popper in anticipation for a jackpot shot when other balls are locked or drained	State check: 0x07, from $7293,3B
	State check: 0x08, from $72C8,3B
Ball popper switch opens when locked ball is kicked from ball-popper to cannon for a jackpot attempt	State check: 0x05, from $727A,3B
	State check: 0x0A, from $734E,3B
	State check: 0x0D, from $735E,3B
	State check: 0x0B, from $73AD,3B
Ball popper hit during multiball and a super jackpot is to be attempted.	State check: 0x07, from $7293,3B
	State check: 0x0A, from $734E,3B

	State check: 0x0D, from $735E,3B
	State check: 0x0B, from $73AD,3B

With regard to the PAPA Lost Super Jackpot bug, the above analysis shows how there are 4 state checks that are performed when the ball popper is hit when a super jackpot is about to be attempted. *As shown, these same 4 state checks are also performed when ball popper is hit during ordinary game play, while attempting to advance towards a multiball.*

To give a little more detail to what happens when this state check is called, below are some additional high level flowcharts and drawings to help give additional clarity.



Function Callback Table $7513,30		
Index	**Use**	**Function**
[0] $7F22,30	TBD/unused(?)	$8EBC
[1] $7F26,30	Shooter (?)	$461B,31
[2] $7F31,30	Left Lock	$5133,31
[3] $7F3B,30	Top Lock	$58BA,31
[4] $7F45,30	Ball Popper	$6F4D,31
[5] $7F4F,30	Gun Loaded	$6A5E,31

This table will be cited later as the "Gun Loaded" switch will also be discussed briefly. The "Gun Loaded" will have a similar way of handling the state check as described here for the "Ball Popper" switch.

The $FDFC function in unbanked ROM region is interesting because it makes a direct function call to $70B4 without any notion of bank switching. This means the caller to this function must have a function in its bank at $70B4 for this to work. In this case, the callers of $FDFC are from bank $3B so this will work as there exists the subsequent lookup function at $70B4,3B.

This lookup function takes in an index at register A and uses it as lookup into a table located at $7513,30 to perform the state check. In the case of ball-popper, the register A has value 0x04 (from the switch handling function) so the function at $6F4D,31 is called to handle the requested state check.

In the case of ball-popper switch-hit for super jackpot (and ordinary hit in advancement toward multiball) this $6F4D,31 function is called once for each state check:

- $6F4D,31 called for state check 0x07
- $6F4D,31 called for state check 0x0A

- $6F4D,31 called for state check 0x0D
- $6F4D,31 called for state check 0x0B

The $6F4D,31 function, ROM offset 0x46F4D, is the dispatch function for the various state checks that might be performed. Each handled state check and its associated function are shown in the table below:

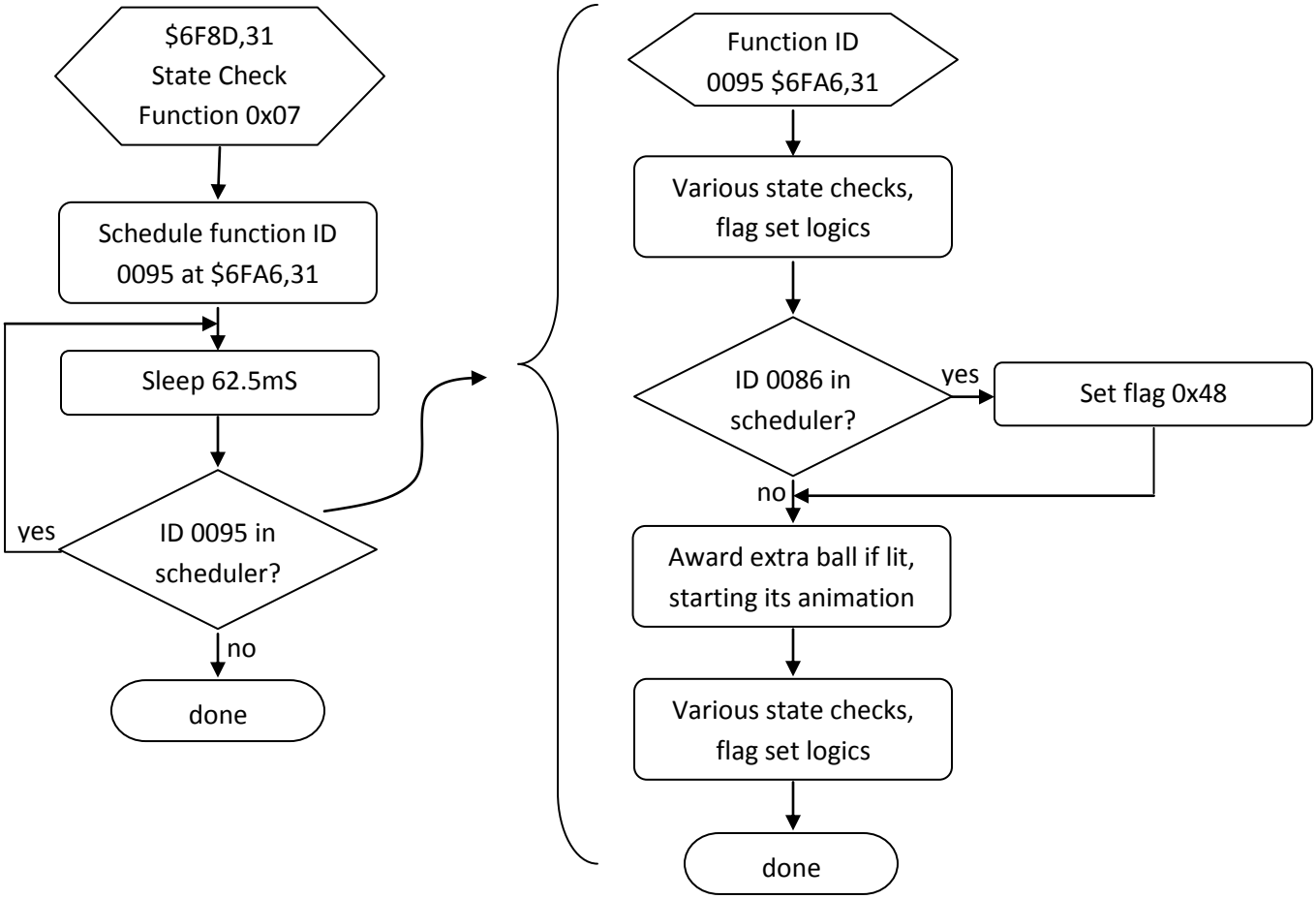
State Check Function for "Ball Popper" Switch, at $6F4D,31		
State	**Purpose**	**Function**
0x07	Ball popper switch closed	$6F8D,31
0x08	Ball popper switch is remaining closed (ball locked)	$6F75,31
0x05	Advancing to cannon for jackpot(?)	$6F86,31
0x0A	Advancing to cannon (1/3)	$707F,31
0x0D	Advancing to cannon (2/3)	$7050,31
0x0B	Advancing to cannon (3/3)	$70F7,31
0x04	Error state handler(?)	$6F70,31

The table, above, shows the different functions that get called on behalf of the ball-popper switch closure for each of the previously described state checks. Each of the called functions are designed to handle the different state check, setting up the game for the imminent "fire at will", ball lock, jackpot or super jackpot shot attempt.

Regarding the PAPA Lost Super Jackpot problem the following state check functions are briefly described at least in part where they contribute to the bug:

- State check 0x07 at $6F8D,31
- State check 0x0A at $707F,31
- State check 0x0B at $70F7,31

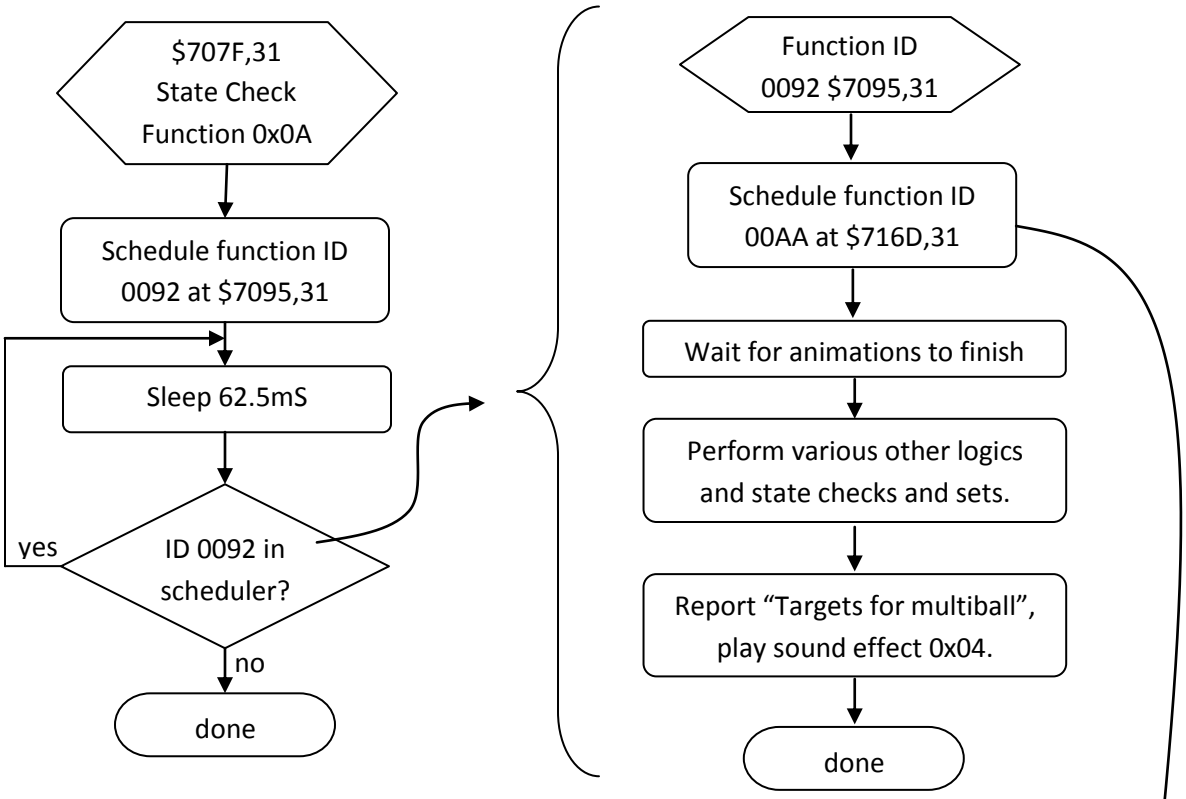
The simplified flowcharts below highlight the interesting part of these functions as they apply to the PAPA Lost Super Jackpot bug.



The logic for the ball popper state check 0x07 function is depicted above, greatly simplified. The important part in relation to the PAPA Lost Super Jackpot bug is the fact that the 0x07 state check function will schedule the helper function ID 0095 that sets the 0x48 flag when the 0086 function is in the scheduler. The 0086 function is the multiball loop function that maintains multiball state. This means that the 0x48 flag is set when the ball popper is hit during multiball and, as such, the 0x48 flag means that the imminent cannon shot is during multiball and is going to be for jackpot or super jackpot.

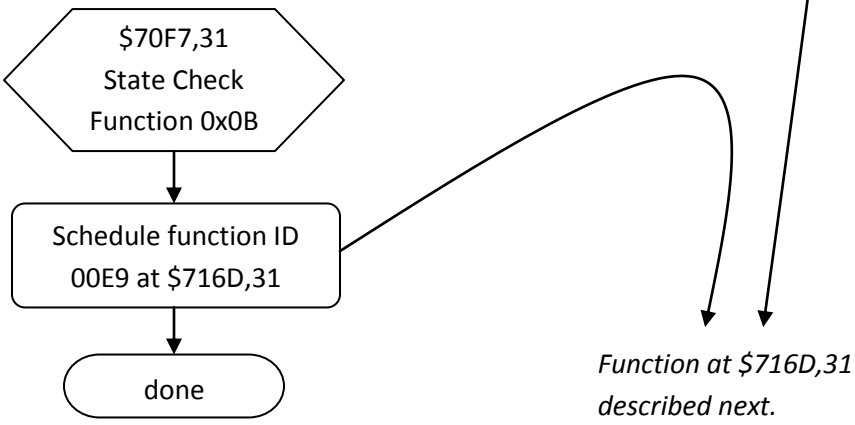
Observant readers may also notice references to this 0x48 flag in previous descriptions and code samples related to the 255-Hits problem description and analysis. While handling the 5-bank hunter ship targets the 0x48 flag is checked as part of determining what actions to take. When the flag is set, the game knows the ball-popper was hit during multiball. This and other game state information is then used to determine whether to award a jackpot or super jackpot or declare a target miss, as appropriate.

Shown next is flowchart logic depicting the ball popper state check 0x0A function.



The ball popper state check 0x0A function, above, schedules a helper function ID 0092 which performs various logic as part of the imminent transition of the ball from the ball popper to the cannon. For the PAPA Lost Super Jackpot bug, the important part is how this helper function then adds another function to the scheduler with ID 00AA starting at $716D,31. *After* doing this, the 0092 function then waits for any animations that are currently playing to complete before continuing *(this will be discussed in more detail as part of the solution analysis)*.

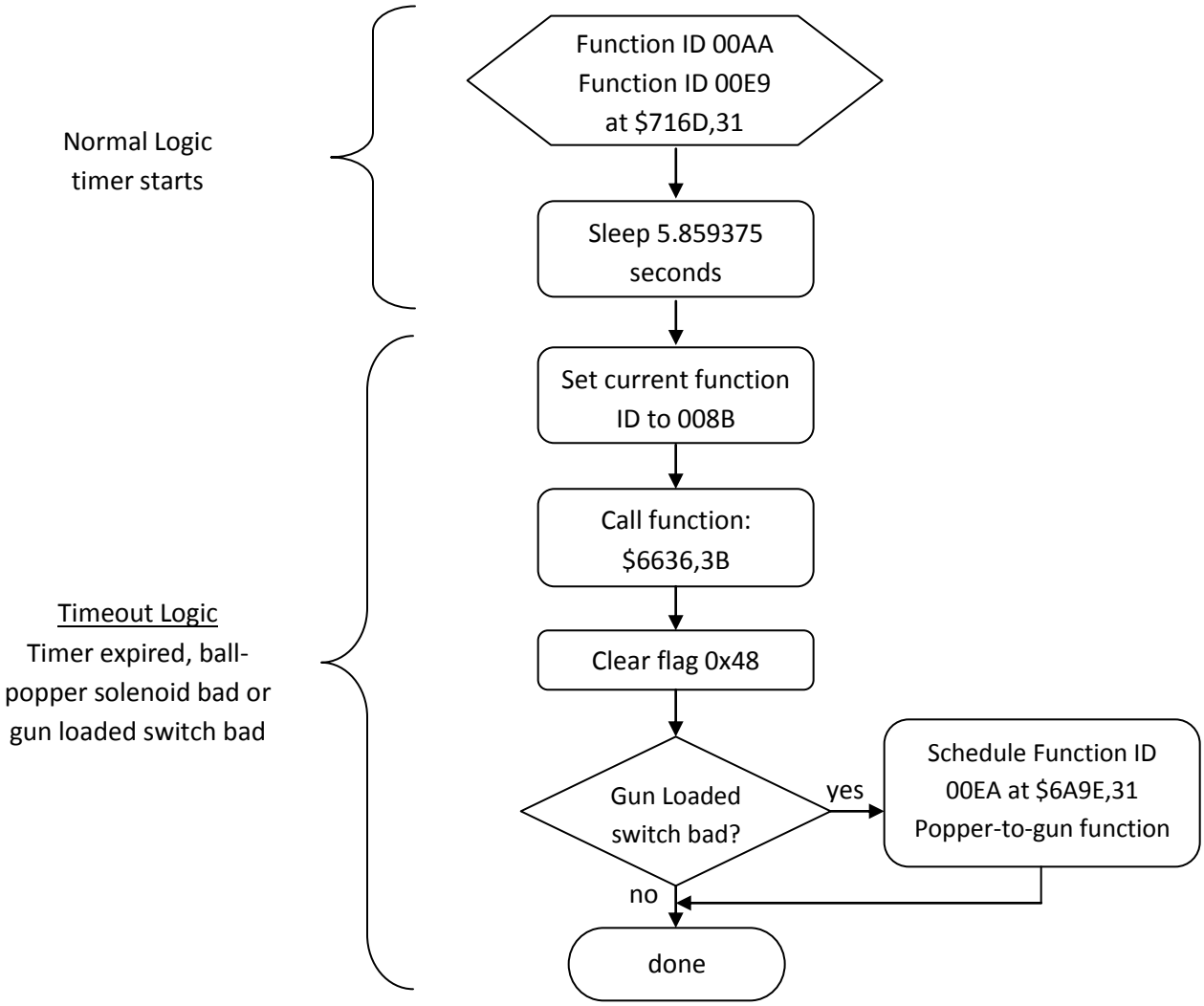
The ball-popper state check 0x0B function is depicted below.



As the previous ball popper state check flowcharts have depicted, there are two occasions where the $716D,31 function is added to the scheduler:

- During state-check 0x0A the $716D,31 is added to the scheduler with ID 00AA
- During state-check 0x0B the $716D,31 is added to the scheduler with ID 00E9

The logic of this function is depicted below:



The function depicted above appears to be a timeout handler. This function sets a timer of 5.859375 seconds before proceeding with what appears to be an attempt for code to handle the scenario where the timer expires. **In normal scenarios, the timeout is never reached**. As further shown below, these function IDs 00AA and 00E9 normally get cancelled during their 5.859375 sleep period and removed from the scheduler by subsequent code that takes place as part of the Gun Loaded switch handler.

In a scenario where the timeout DOES happen, it is shown here that the **0x48 flag gets unconditionally cleared**. The clearing of this 0x48 flag due to the 5.859375 second timeout of is why the subsequent

cannon shot is not treated as a super jackpot shot and, as such, **is the reason for the PAPA Lost Super Jackpot bug**.

This timeout could happen if, for example, the ball popper solenoid is not working or if the Gun Loaded switch is not working (game does not detect the ball in the cannon). The original L-8 software has this mechanism to make an attempt to recover in such a situation in some form of a predictable manner, allowing the game to proceed as best it can.

This timeout should only happen in a situation of mechanical failure. The PAPA Lost Super Jackpot bug shows that, instead, the timeout can also occur while the game is free of mechanical failures. When such timeout takes place when the game is otherwise functioning normally, the result is that the cannon shot is treated as an ordinary cannon shot instead of a jackpot or super jackpot shot attempt. *In theory this problem can happen for both jackpot or super jackpot attempts. Also, depending on conditions, it is also possible that instead of "Targets Remaining" the game could seemingly restart multiball (this seems to happen if the problem occurs more than once during the same ball in play).*

In the case of the PAPA video, the ball was shot into the popper when:

- The game starts the extra ball award animation
- The ball-trough was handling the previously drained ball, and,
- The game may have been processing the long flipper hold time for "Status Report" to display

In the pinball simulator the exact sequence to trigger the exact scenario of the PAPA Lost Super Jackpot hasn't been reproduced. It is seemingly an extremely rare sequence of events that happened to take place during the PAPA tournament. The bug, however, can be reproduced by a different method. In simulator, the 5.859375 second timeout can occur by scoring the super jackpot and then *immediately* locking the ball into the popper again. In this scenario, the game is playing the lengthy super jackpot animation sequence. If the ball was shot into the ball popper at the very start of this animation sequence, it is possible for the 5.859375 second timer to expire which causes the next cannon shot, if successful, to report the "Targets Remaining" message instead of awarding another super jackpot.

The following text continues the description of the behavior that normally takes place as part of the ball popper handling. Shown above was the point in which the ball popper switch handling caused the pulse of the ball popper solenoid during the $7306,3B function between the state check 0x0D and 0x0B. As mentioned, the cannon switch (aka the "gun loaded" switch) closure is what is responsible for ensuring the 5.859375 second timeout functions are cancelled before they expire.

The "gun loaded" switch actually shares the same switch handler callback. As depicted in the information above the same set of functions is called for "gun loaded" switch as for the "ball popper" switch. There are some checks that take place during the switch handling so that only certain logic is executed depending on the hit switch.

Also, shown previously was a function callback table located at $7513,30 which contains a function address that depends on which switch is being processed. The table is depicted below with the "gun loaded" row highlighted.

Function Callback Table $7513,30		
Index	**Use**	**Function**
[0] $7F22,30	TBD/unused(?)	$8EBC
[1] $7F26,30	Shooter (?)	$461B,31
[2] $7F31,30	Left Lock	$5133,31
[3] $7F3B,30	Top Lock	$58BA,31
[4] $7F45,30	Ball Popper	$6F4D,31
[5] $7F4F,30	Gun Loaded	$6A5E,31

For the "gun loaded" switch, the corresponding state check function is $6A5E,31, ROM offset 0x46A5E. This function is called with the same set of state check values as previously described for the ball popper switch handler. An analysis of this function shows similar set of calls into this function:

- When ball first enters the cannon, state check 0x07
- When ball is shot from the cannon, state check 0x0A
- When ball is shot from the cannon, state check 0x0D
- When ball is shot from the cannon, state check 0x0B

A more detailed survey of code would be needed to fully understand how all of the switch handler code behaves however it is evident that the game uses this mechanism to allow common code and function callback tables as an effective and neat way to implement the game logic.

For the "gun loaded" switch, as shown in the table above, the function $6A5E,31 gets called for state checks. This function is called with the state indicator number as mentioned above. Examination of the $6A5E,31 function reveals that it only has a handler for two of the states. The others are ignored with no particular work taking place for that state on behalf of the "gun loaded" switch. The handled states are indicated in the table below.

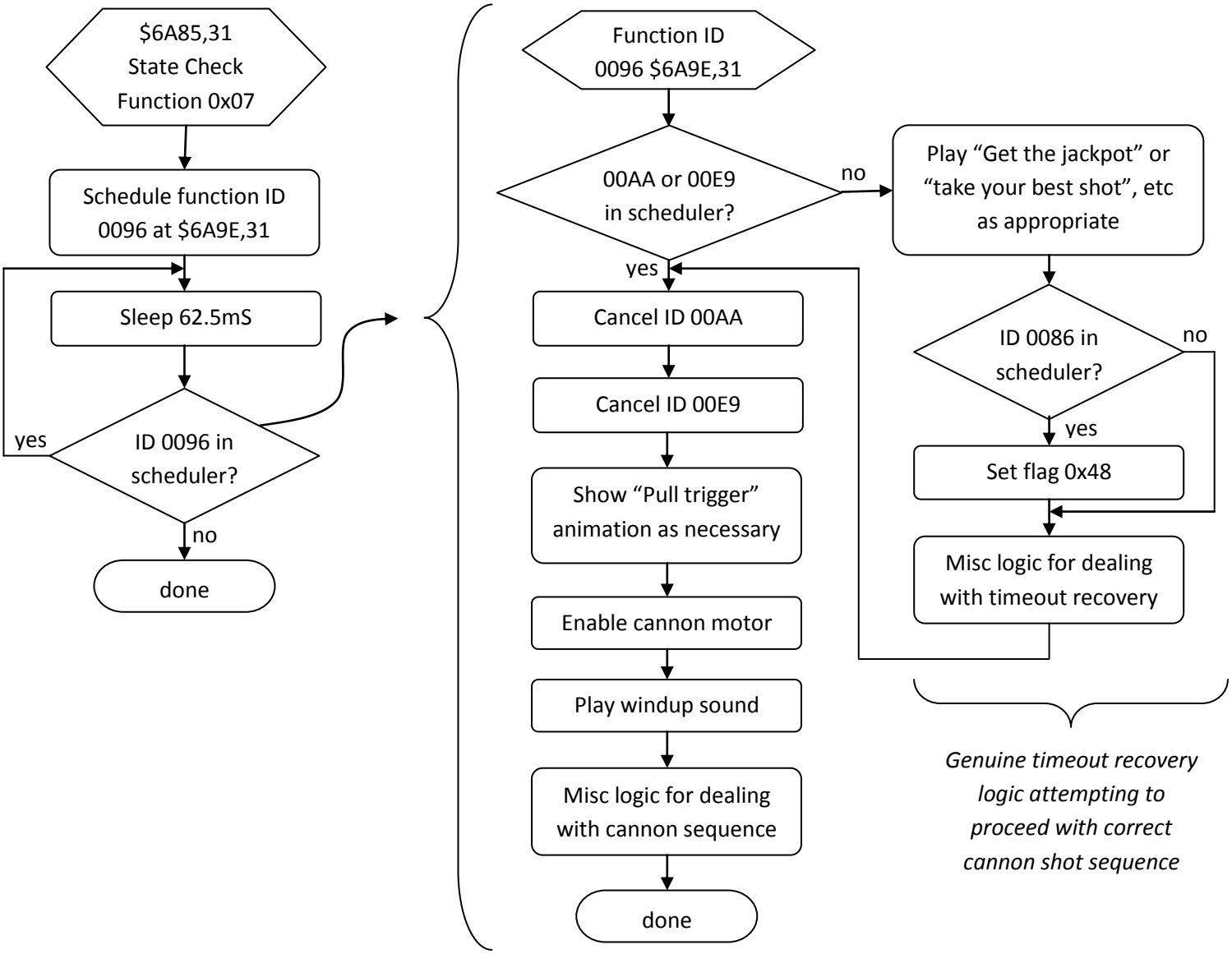
State Check Function for "Gun Loaded" Switch, at $6A5E,31		
State	**Purpose**	**Function**
0x07	Gun loaded switch closed	$6A85,31
0x08	<not used>	<none>
0x05	<not used>	<none>
0x0A	After ball is shot from cannon (1/3)	<none>
0x0D	After ball is shot from cannon (2/3)	<none>
0x0B	After ball is shot from cannon (3/3)	<none>
0x04	Error state handler(?)	$6A67,31

As shown, only two handlers are in use for the "gun loaded" switch. The other states are listed in the table for completeness and for comparison with the similar table previously shown for the "ball popper" switch.

The $6A85,31 function at ROM offset 0x46A85 handles the “gun loaded” switch closure. This function is where the game is certain that the ball is now in the cannon and may therefore engage the gun motor and prepare for pulsing of the cannon solenoid, etc.

Where it is applicable to the PAPA Lost Super Jackpot problem, this function is where the 00AA and 00E9 timeout functions are cancelled, thus preventing the unexpected clearing of the 0x48 flag that would occur if the timeout was allowed to take place *(and what actually happens in the case of the PAPA Lost Super Jackpot bug)*.

Depicted in the flowcharts below is the logic of the $6A85,31 “gun loaded” state 0x07 function:



The state check 0x07 for the “gun loaded” switch, as shown in the flowchart above, performs a lot of logic related to initialization of the cannon sequence and subsequent cannon shot. Related to the PAPA Lost Super Jackpot problem, the key takeaways are:

- The “gun loaded” switch handler cancels the 00AA and 00E9 functions from the scheduler. These two functions are what run 5.859375 second timers which, if not cancelled, would timeout and clear the 0x48 flag which is what leads to the lost super jackpot problem.
- The “gun loaded” switch handler does have some logic for dealing with genuine timeouts. This logic will only be applied if the 00AA and 00E9 functions are NOT currently in the scheduler. The idea is that the functions must have expired and are no longer in the scheduler so, therefore, a genuine timeout took place.
    - Perhaps the ball popper switch is broken and ball-search kicked ball to the cannon.
    - Perhaps the habitrail is blocked or for some other reason the ball very slowly gets from the ball popper to the cannon.

PAPA Lost Super Jackpot Investigation Summary

With all of the logic presented, a summary of events that leads up to a lost super jackpot is as follows:

- Ball popper is hit.
- A 5.859375 second timer is started at state check 0x0A.
- Game is busy playing animations and/or busy handling other switches.
- The 5.859375 second timer expires.
- Timer expiration results in clearing of the 0x48 flag (indicator of ball popper loaded during MB).
- Animations are finished.
- A 5.859375 second timer is started at state check 0x0B.
- Game reports “Shoot for super jackpot”
- Game pulses the ball popper solenoid.
- Ball reaches the cannon.
- Gun loaded switch handler observes the 5.859375 timer is running, no timeout recovery.
- Player pulls trigger to hit a lit 5-bank target.
- Game gives award but, due to lack of 0x48 flag, assumes cannon was loaded outside of multiball.

The logic flow, above, is a high level summary of how the problem can happen. This description is based on observations made with the simulator and observing the actual problem taking place by locking the ball in the popper immediately after scoring a super jackpot. While the super jackpot animation is playing and the ball is in the popper, the first timeout occurs which clears the 0x48 flag which essentially makes the game forget that the ball was loaded during multiball. The subsequent cannon loading takes place during the secondary timer which is still running and then cancelled shortly later during the handling of the “gun loaded” switch closure.

It seems the timeout logic that currently exists is mainly about handling cases of faulty hardware and not faulty software which seems to be the case with faulty logic and/or improperly timed timeout that reaches expiration in some normal game play scenarios.

PAPA Lost Super Jackpot Code Fixes

Some low impact code changes can be added to the code to effectively resolve the PAPA Lost Super Jackpot bug with the following goals:

- Minimal code changes. Retain as much original code as possible.
- Prevent a lengthy animation in progress from contributing to a timeout condition.
- In the event a timeout *does* take place, retain the jackpot or super jackpot attempt.

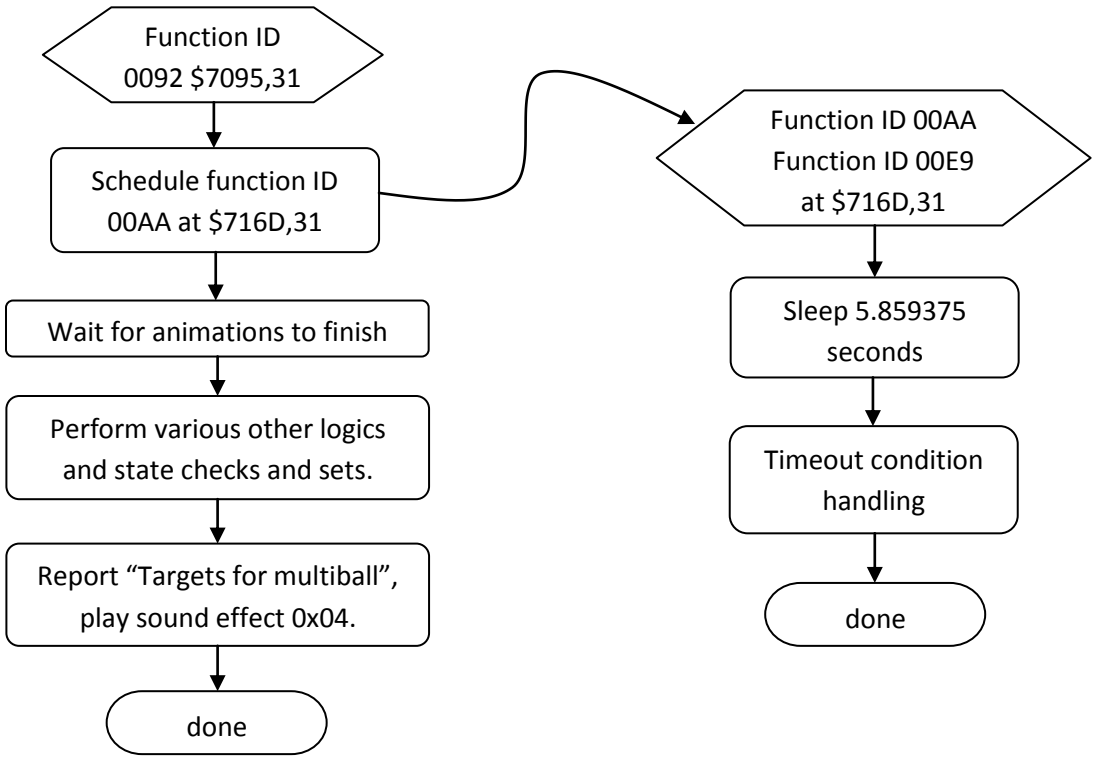
After some code analysis it is found that the solution may be reduced two code changes:

- Have the ball popper state check 0x0A wait for animations to complete before doing its work.
- Prevent the timeout handler from clearing the 0x48 flag if multiball is in progress.

The following sections will go into details about the code changes.

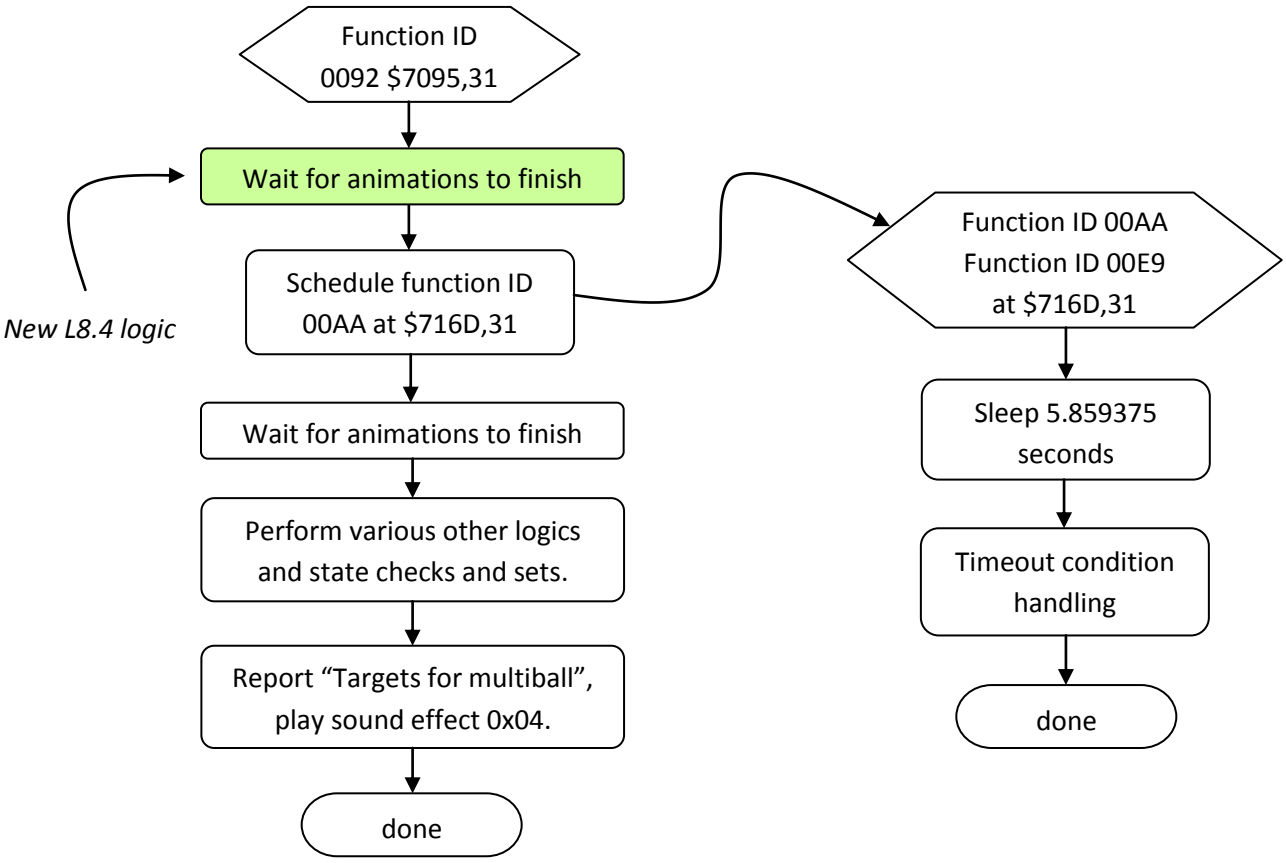
PAPA Lost Super Jackpot Logic Update: State Check 0x0A Animation Check

As shown above, the crux of the problem is that the 0092 function sets a 5.859375 second timer (which appears to be for measuring time from popper to cannon) and then waits for animations that are currently in progress to complete before actually firing the ball from popper to cannon. Shown below are the 0092 and 00AA (reduced) flowcharts from above.



To prevent the animation (that might be in progress) to cut into the 5.859375 second timer period, the updated logic involves simply having the ball popper state check 0x0A helper function 0092 wait for any animations in progress to complete prior to launching the function 00AA which starts the timer. This way, any animations that might be in progress at the time the ball popper is hit will be allowed to play to completion prior to the ball popper switch handler starting any of its own animations or timers. Also, since the extra ball award animation is started by the ball popper state check 0x07, its animation is included in this check so that the extra ball animation runs to completion before the 0090 function proceeds with starting the timer and completing its work.

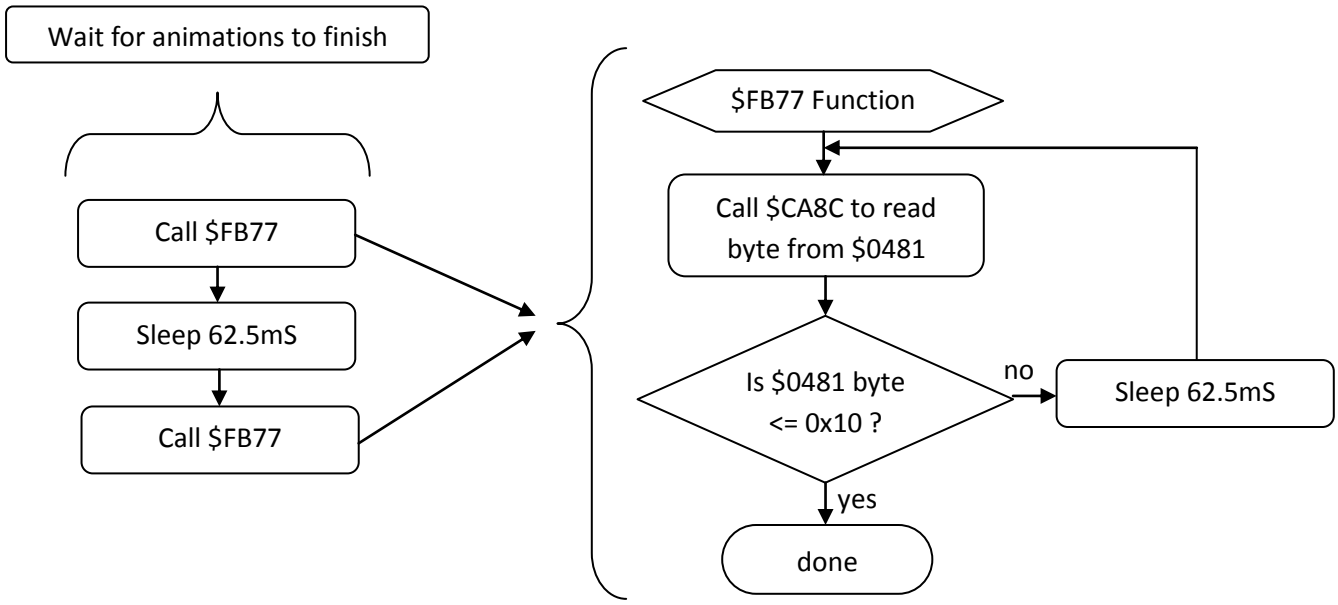
The updated flowchart logic is shown below with the new L8.4 logic highlighted in green.



The insertion of a "Wait for animations to finish" simply makes a function call to the same wait functions that are already called by the same function *after* the 00AA function gets scheduled. By also having the code wait for animations to finish *prior* to the scheduling of 00AA, any existing animation that is in progress (such as extra ball animation) will be allowed to finish in its entirety before the ball popper handling code will proceed. In doing this, the 5.859375 second timer that is started will only apply towards new work that is done in handling the ball popper switch and not towards any existing animations that might have been playing while the ball was first shot into the ball popper *(this includes the extra ball animation that is initiated by the ball-popper switch closed handler)*.

An analysis of exactly what it means to wait for animation to complete is in order.

The existing code that waits for animation to complete is listed as a single element in the previously depicted flowchart. What actually takes place in the 0092 code at $7095,31, is depicted below:



The process of the 0092 function waiting for animation to complete in existing L-8 code is depicted above. The code makes two wait attempts, 62.5mS apart. Each wait attempt consists of read of memory location $0481 and, if its byte value is greater than 0x10, sleep 62.5mS and check again, indefinitely.

A brief examination into the $0481 byte value was done to see that, indeed, it is loaded with a value greater than 0x10 during animation sequences and cleared to 0x00 at the end of the animation. It was also observed with the simulator that this wait in function 0092 is the reason that the ball-popper handling waits before advancing the ball towards the cannon *(and, as such, the reason a timeout can occur and lost jackpot or super jackpot can take place as previously described)*.

For L8.4, the 0092 function will make a single new call to $FB77 prior to scheduling the 00AA timeout function. As depicted above, a single call to the $FB77 function will cause the software to repeatedly poll the value of $0481 until it contains a value less than or equal to 0x10. This will effectively cause the code to wait until any existing animation that is in progress to complete before the ball-popper code proceeds with its usual activities.

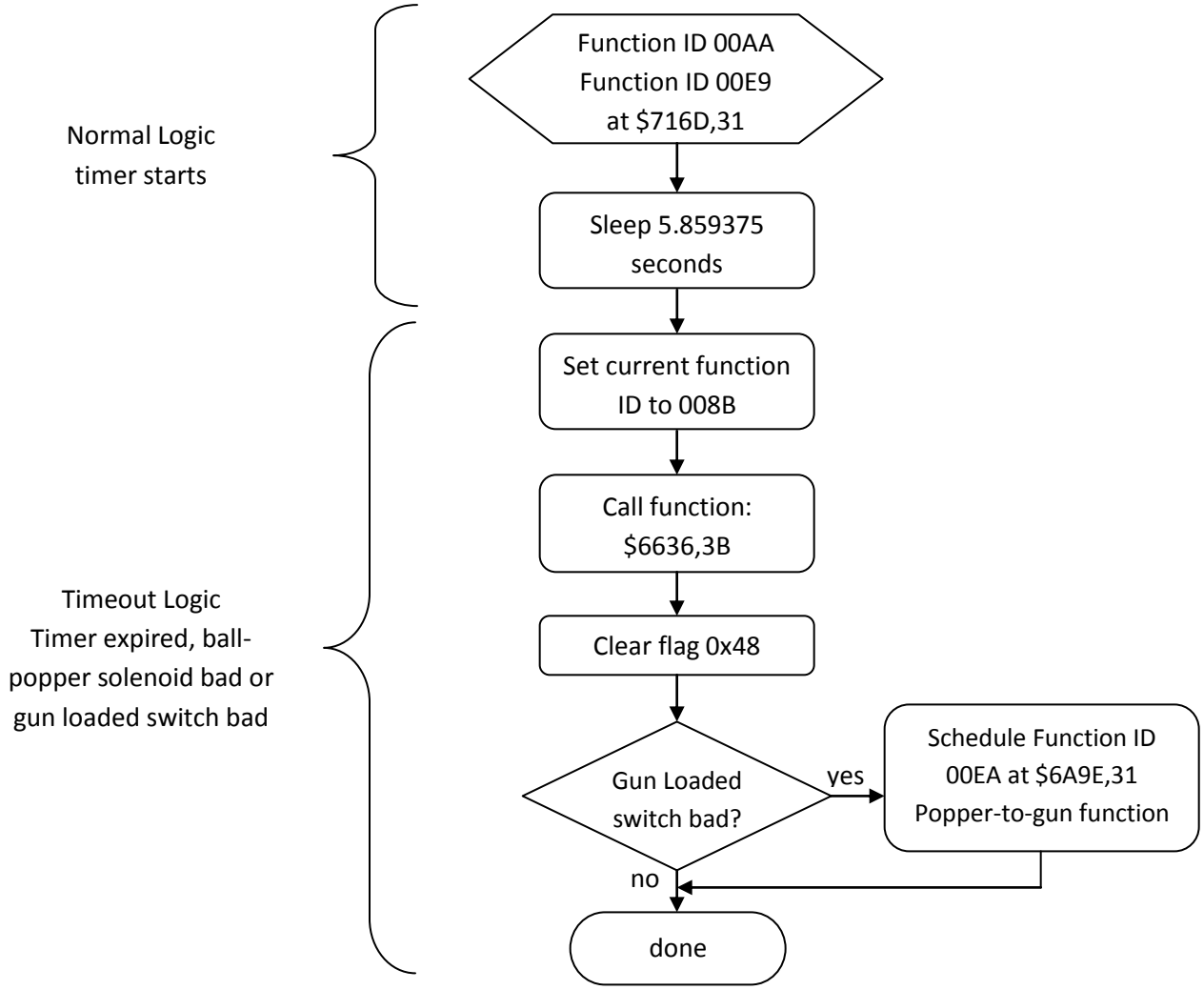
The L8.4 code changes for this fix are described further below.

PAPA Lost Super Jackpot Logic Update: Timeout 0x48 Flag Retention

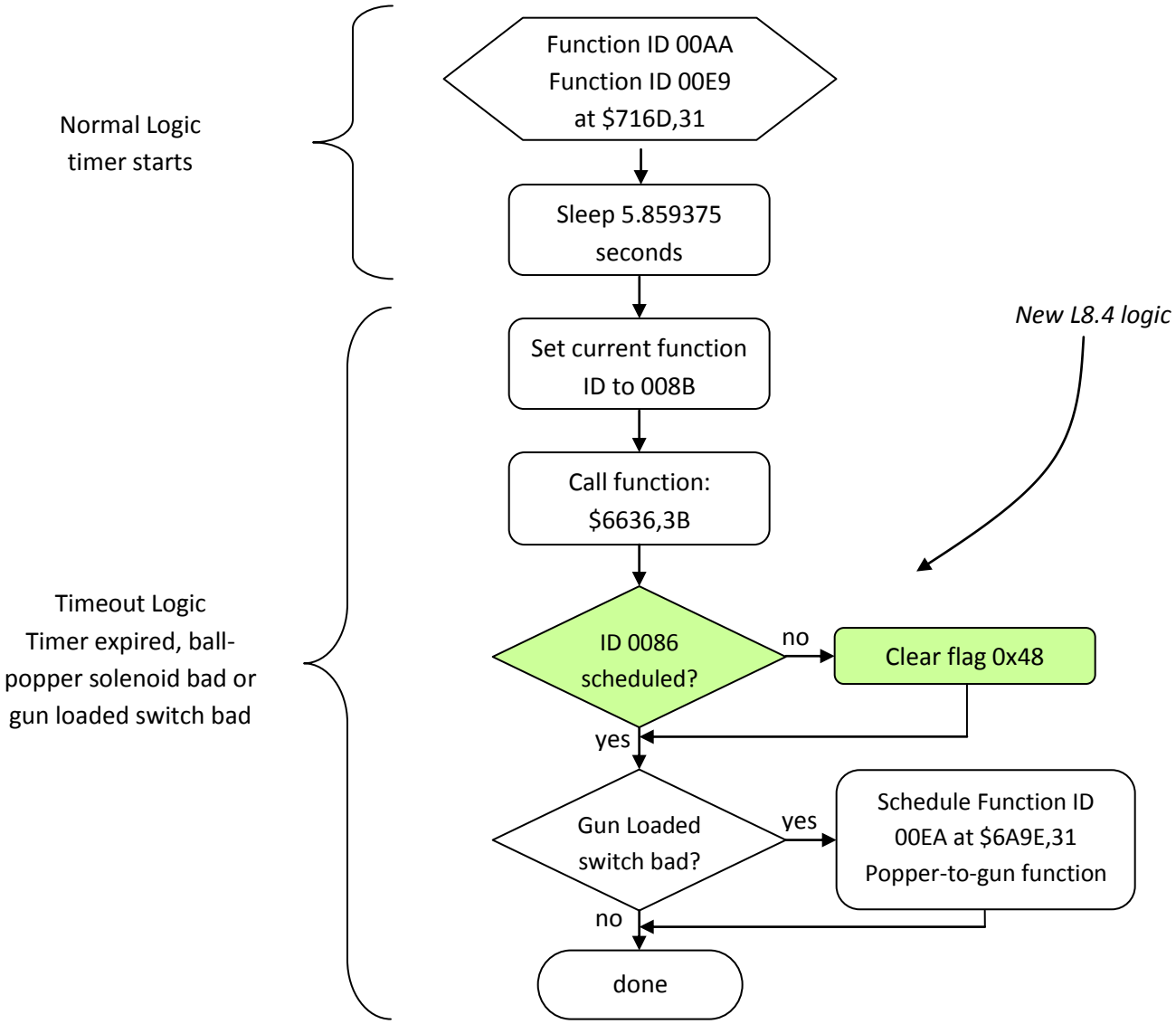
As previously described, the timeout function, after experiencing the 5.859375 second timeout, clears the 0x48 flag as part of its timeout handling. It is evident that the original L-8 design has the "gun loaded" switch handler code re-set the 0x48 flag when it discovers that the timeout function is *not* running (meaning timeout occurred) and multiball is running. It is also evident that the original design flaw is how there are two 5.859375 timeout functions (during state check 0x0A and again during state check 0x0B). In the problem scenario the first timeout occurs which clears the 0x48 flag but the second timeout function is still running when the "gun loaded" switch handler code is invoked. In this case, the multiball is still in progress however the timeout function is still running and, as such, the 0x48 flag does not get reset in the "ball popper" switch handler code (as it appears was the intent of the original flawed design).

For L8.4 the intent is not to try to re-craft the overall design in order to implement new design without the flaw. The goal is to patch the code to overcome the flaw and, therefore, the timeout logic which takes place when the timer expires is being updated to prevent the problem of lost jackpot or lost super jackpot.

Shown below is the flowchart from the timeout function copied from above, for reference:



For L8.4 the logic in the timeout handler is updated as shown in the updated flowchart below.



As highlighted in the flowchart above. The L8.4 will simply add a check if function ID 0086 is running and, if not, then clear the 0x48 flag. The 0086 function is the multiball loop function which existing L-8 code uses as the determining factor for whether to set the 0x48 flag in the first place. This simple update to the logic will prevent a timeout condition from clearing the 0x48 flag when the 0x48 flag state should be retained. By retaining the state of the 0x48 flag, the subsequent cannon shot and 5-bank target hit will be properly treated as a jackpot or super jackpot, as appropriate, in the unlikely situation where a timeout does occur.

Prior to implementing the new logic, proof of concept code has been tested to confirm that the lack of 0x48 flag clearing during the timeout handler will allow the subsequent super jackpot to be awarded where it, previously, would have encountered the PAPA Lost Super Jackpot bug. This new logic, along with the previously described function 0092 animation wait, effectively hardens the code protecting it from experiencing a lost jackpot or lost super jackpot.

Simply put, the L8.4 will update code to reduce the likelihood of hitting the timeout and even if a timeout does happen, it will not result in the lack of jackpot or super jackpot at the subsequent cannon shot that takes place after the ball moves from the ball popper to the cannon.

PAPA Lost Super Jackpot Code Fixes

As depicted above, the code fixes for the PAPA Lost Super Jackpot bug are localized to:

- The 0092 ball popper state check 0x0A function where a new animation wait is added, and
- The timeout function, where a multiball in progress condition is added to the 0x48 flag clearing.

PAPA Lost Super Jackpot Code Fix: Add Animation Wait to 0092 Function

The ball popper state check 0x0A function, as previously described, schedules the ID 0092 function which then schedules the timeout function. As previously described this function is being augmented so it waits for animation completion *prior* to scheduling the timeout function.

The 0092 function is shown below along with the L-8 code removal and L8.4 code addition.

New L8.4 code

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Ball popper state check 0x0A
                                        ; Scheduled Callback Function
                                        ;  ID 0092
                                        ;
7095: BD 68 B6    JSR   $68B6           ;
~~7098: BD F7 59    JSR   $F759           ; ChecksGameMode(), if game in progress, z-bit set~~
~~709B: 7E 70 9E    JMP   $709E           ; <nop>~~
7098: BD FB 77    JSR   $FB77           ; WaitForAnimationToComplete()
709B: BD F7 59    JSR   $F759           ; ChecksGameMode(), if game in progress, z-bit set
709E: 26 50       BNE   $70F0           ;
70A0: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
70A3: 00 AA                             ;  ID 00AA Ball-In-Popper
70A5: 71 6D 31                          ;  Timeout function
70A8: BD 56 9C    JSR   $569C           ;
70AB: 86 0A       LDA   #$0A            ;
70AD: BD CF 4B    JSR   $CF4B           ;
70B0: BD FB 77    JSR   $FB77           ; WaitForAnimationToComplete()
70B3: 7E 70 B6    JMP   $70B6           ; <nop>
70B6: BD 83 46    JSR   $8346           ; Sleep()
70B9: 04                                ;
70BA: BD FB 77    JSR   $FB77           ; WaitForAnimationToComplete()
70BD: 7E 70 C0    JMP   $70C0           ; <nop>
70C0: BD 86 90    JMP   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
70C3: 00 86                             ;  Search for 0x0086, C-clear means multiball is running
70C5: 24 06       BCC   $70CD           ;
70C7: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // c-clear means flag is set
70CA: 48                                ; 0x48 == Huntership hit
70CB: 25 06       BCS   $70D3           ; If C-set, meaning flag not set, skip the following
70CD: BD 84 80    JSR   $8480           ; SetMemoryFlag()
70D0: 48                                ; 0x48 == ball popper hit during multiball
```

```
70D1: 8D 32       BSR   $7105           ;
                                        ;
70D3: BD 86 90    JSR   $8690           ; -\ SearchLinkedListForId()  // c-clear = ID found
70D6: 00 ED                             ;  |
70D8: 25 06       BCS   $70E0           ;  |
70DA: BD 83 46    JSR   $8346           ;  | Sleep()
70DD: 03                                ;  |
70DE: 20 F3       BRA   $70D3           ; -/
                                        ;
70E0: BD 85 B2    JSR   $85B2           ;
70E3: 05                                ;
70E4: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
70E7: 29                                ; "X Target for multiball" or "Get the super jackpot" or
                                        ; whatever message is needed at cannon laod.
70E8: BD 83 46    JSR   $8346           ; Sleep()
70EB: 10                                ; 1/4 second
70EC: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
70EF: 04                                ; 0x04 Waszhwawawawu
70F0: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
70F3: D7                                ;
70F4: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;------------------------------------------------------
```

The highlighted code, above, shows that for L8.4 the code was slightly re-arranged so that the $FB77 function is called prior to a CheckGameMode() function call. This is easy inclusion since the original code consisted of the CheckGameMode() followed by dummy JMP instruction. For L8.4 the dummy JMP is removed and its freed space allows for the inclusion of a JSR to the $FB77 instruction. *The L-8 code has several such dummy JMP instructions that occur after a JMP to a function in non-banked ROM. This appears to be a design approach that can make it easier to move a function from non-banked ROM to somewhere in banked ROM in which case the dummy JMP instruction can then be used to facilitate the call to the function in banked ROM.*

PAPA Lost Super Jackpot Code Fix: Add Multiball Conditional To Timeout Function

The timeout function, as described, is augmented in L8.4 so that it will only perform its clearing of flag 0x48 if the game is not currently in multiball (as determined by lack of 0086 function in the scheduler). The timeout function is shown below with new code for L8.4 highlighted.

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Function ID 00AA via ball-popper state check 0x0A
                                        ; Function ID 00E9 via ball-popper state check 0x0B
                                        ;
                                        ; State Check Timeout Function when ball is in popper
                                        ;
716D: BD 86 79    JSR   $8679           ; SleepPlusURegisterSave()
7170: 01 77                             ; 5.859375 second sleep time
                                        ;
7172: 10 8E 00 8B LDY   #$008B          ; ID 008B
7176: BD 9B 83    JSR   $9B83           ; UpdateCurrentRunningScheduleFunctionIDY()
7179: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
717C: 66 36 3B                          ;
717F: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
7182: 48                                ; Clear 0x48
                                        ; 0x48 bit is 0x08 bit of flags starting at $0328
                                        ;  So this is 0x80 bit of $0328
```

New L8.4 code

```
717F: BD 7F 8F    JSR   $7F8F           ; L84ConditionalFlag48Clear()
7182: 12          NOP                   ;
                                        ;
7183: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte()  C-clr = broken
7186: 19                                ; SwitchTableEntry19, 31, Gun Loaded
                                        ;
7187: 25 08       BCS   $7191           ; if (GunLoaded switch is broken)
                                        ; {
7189: BD 8B C3    JSR   $8BC3           ;    ScheduleFunctionCallback()
718C: 00 EA                             ;
718E: 6A 9E 31                          ;    $6A9E,31 is gun loaded switch handler
                                        ; }
7191: 7E 99 A2    JMP   $99A2           ; Done
                                        ;
----------------------------------------;------------------------------------------------------
```

The timeout function, as shown above, is updated so that upon a timeout, the code that normally unconditionally clears the 0x48 flag now makes a call to function $7F8F,31 to conditionally clear the 0x48 flag. The 0x48 parameter byte is replaced with a NOP instruction to ensure correct code flow.

Complementing the above code change, is the addition of the new function at $7F8F,31, ROM offset 0x47F8F. This is located in a small region of unused ROM space near the end of bank $31.

New L8.4 code

```
----------------------------------------;------------------------------------------------
                                        ;
7F8F: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clr means ID is found
7F92: 00 86                             ;  Search for 0x0086
7F94: 24 04       BCC   $7F9A           ;  C-bit clear means multiball is running so skip over
                                        ;  the 0x48 flag clear.  Skip over next 4 bytes.
                                        ;
7F96: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
7F99: 48                                ; Clear 0x48
                                        ; 0x48 bit is 0x08 bit of flags starting at $0328
                                        ;  So this is 0x80 bit of $0328
7F9A: 39          RTS                   ;
                                        ;
----------------------------------------;------------------------------------------------
```

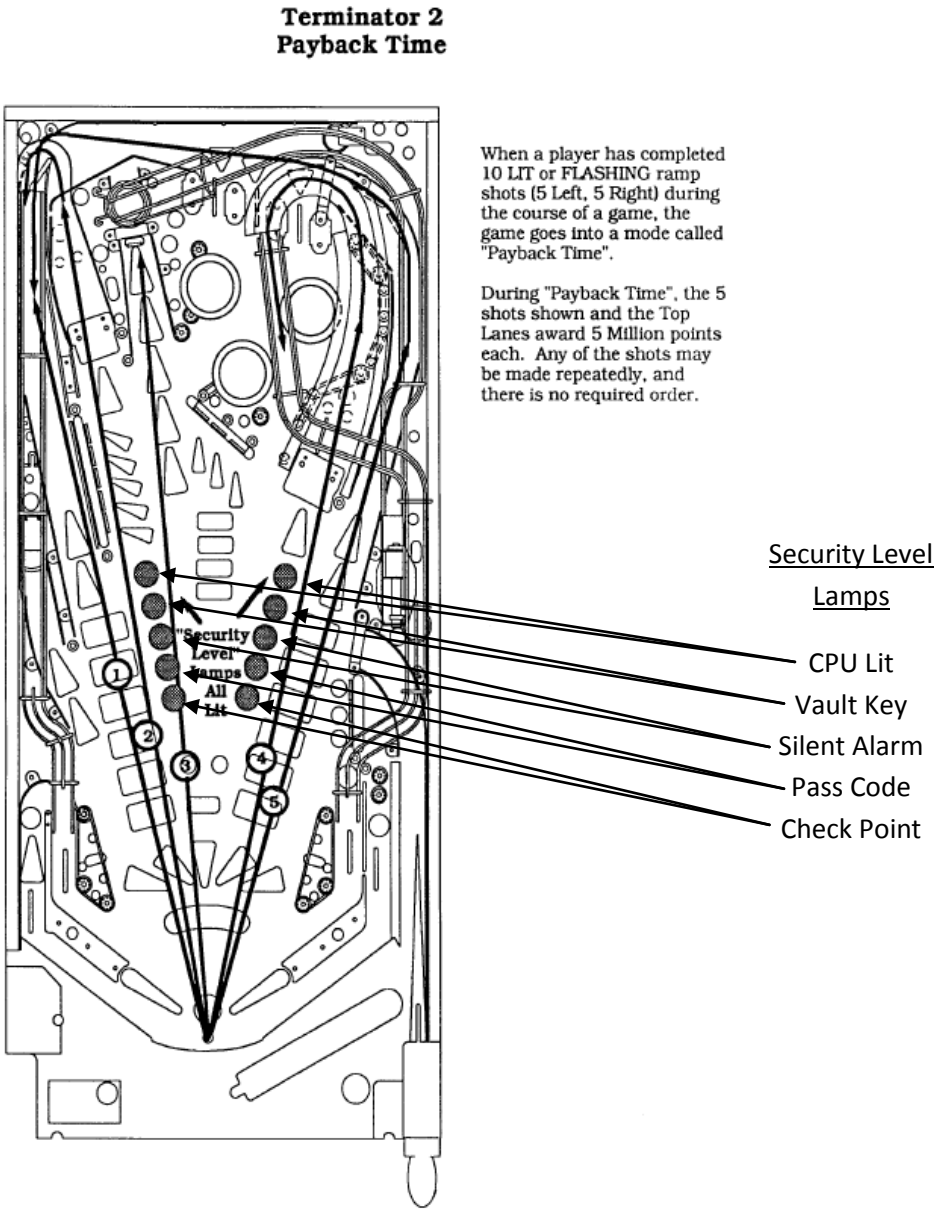
The new function above simply checks if the 0086 function is running and, if so, skips over the clearing of flag 0x48. This simple logic by itself effectively fixes the PAPA Lost Super Jackpot bug. This change, along with the animation wait described previously, effectively ensures that the timeout situation should not happen and even if it were to occur, the timeout won't result in the lost jackpot or lost super jackpot if the imminent cannon shot successfully hits a lit target.

Payback Time Insert Misalignment Bug

For L8.4, an existing L-8 bug related to the playfield insert lamp states for Payback Time is being corrected. A summary of Payback Time and the bug that is being fixed is described below.

Payback Time Summary

The payback time is a short period of game play in which 5 main shots of the game are worth 5 million points each. There is no order or limit on the ways in which the 5 shots may be hit. The augmented image from the T2 manual is shown below for reference:



The way in which the payback time is started is by completing (illuminating) the 10 "Security Level" lamps as depicted in the manual page, above. There are 5 lamps for each ramp and the lamps must be lit by the player in alternating progressive order going from bottom up.

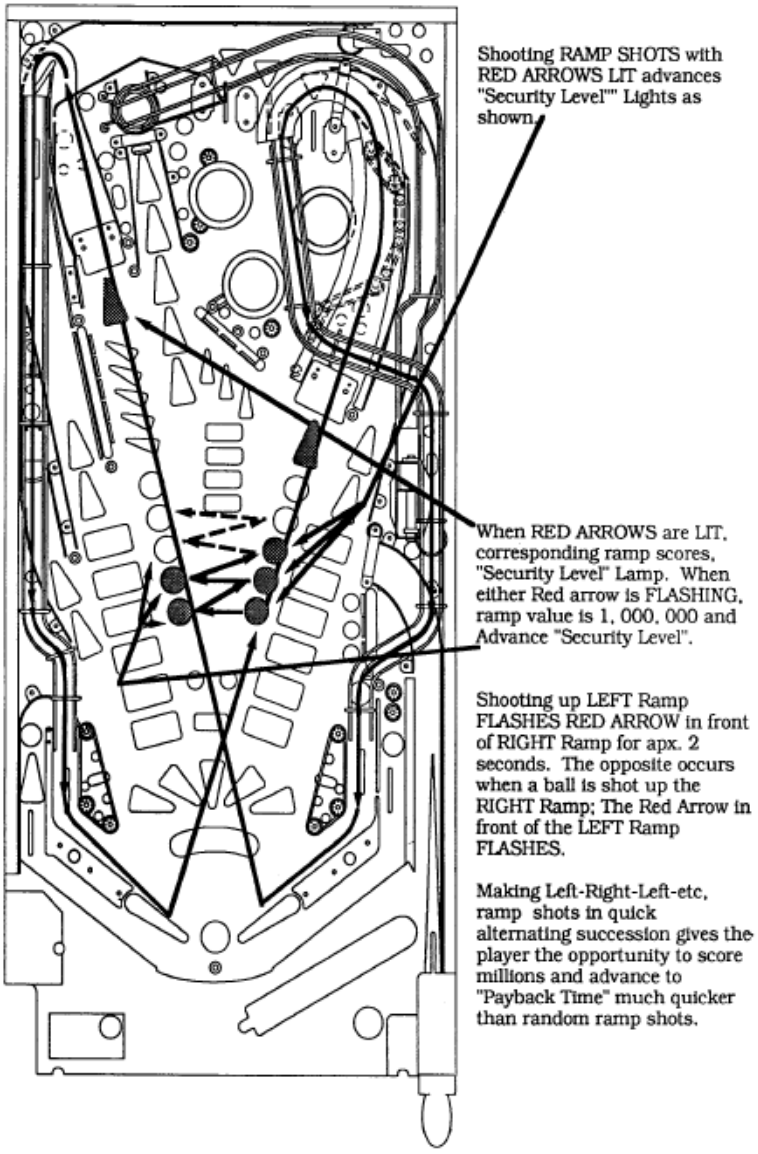
The game is designed so that both "Check Point" lamps are lit before player can light a "Pass Code" lamp. Then both "Pass Code" lamps must be lit before player can light a "Silent Alarm" lamp. This progresses upward until both "CPU Lit" lamps get lit which starts Payback Time.

The game employs some simple rules to ensure the logic is as described above, namely:

- The starting point is zero lit security level lamps and both red arrows are lit, one at each ramp.
- When a ramp with a red arrow is successfully hit:
    - If the ramp's red arrow is blinking, an extra 1 million points is awarded.
    - The next security level lamp for that ramp is illuminated.
        - If this caused "CPU Lit" to illuminate then the other ramp's "CPU Lit" is set to a blinking state.
    - The red arrow for the ramp is turned off.
    - The red arrow for the other ramp is illuminated (if not already lit).
    - The red arrow for the other ramp is also set to blink for the next few seconds. After the blinking period is over, its lamp will then go back to being solidly lit.
- The start of Payback Time will reset the lamps:
    - All 10 of the security level lamps are turned off
    - **Both ramps red arrows are set to be illuminated**
        - These states are briefly overridden during the Payback Time mode but these lamp states will reveal themselves after Payback Time is done.
- The game also offers a "Security Pass" award:
    - May be awarded via Escape Route or Database.
    - Causes the security level to increase by 1 for the left ramp and by 1 for the right ramp.
    - Depending on which ramp has a red arrow lit, the award will either:
        - Advance security level on left then right if the left ramp arrow is lit, or
        - Advance security level on right then left if the left ramp arrow is not lit.
    - Payback Time may initiate after the 1st or after the 2nd security level advancement.
        - If Payback Time started after 1st security level advancement, the second security level advancement will carry over. After Payback Time completes there will be one lit security level lamp (a "Check point" lamp) for one of the ramps.
        - The nature of the bug is because of how **both ramp arrows are lit** in this carry over scenario and described in more detail, below.
- At the start of every ball, if the left and right security levels are identical then both left and right ramp arrows are lit.

The following page from the Terminator 2 manual also describes the Payback Time feature in the manufacturer's words:

Terminator 2
Ramp Shots and Payback Time



As mentioned, in addition to using the ramps to advance the security level lamps for each ramp, there are two additional methods to advance the lit security level lamps:

- Escape Route (left loop) awarding Security Pass, or
- Database (left lock) awarding Security Pass

When the “Security Pass” is awarded from either of these two mechanisms, the game will add a new lit lamp to the left ramp security level lamps and it will add a new lit lamp to the right ramp security level

lamps. This effectively gives the player 2 of the 10 security lamps needed for advancement toward Payback Time. To further describe the "Security Pass" award, some additional details are worth noting:

- If the left and right ramps have the same number of lit security level lamps, the "Security Pass" will advance both left and right ramp security level lamps evenly.
- If the left and right ramps have uneven number of lit security level lamps, the "Security Pass" will maintain such unevenness by adding a new lit security lamp to the left set of lamps and adding a new lit security lamp to the right set of lamps.

It is this "Security Pass" award that is the main contributor to the problem that is being fixed in L8.4. Details on the bug are described next.

Payback Time Security Levels Bug

The problem with the payback time security levels occurs when the "Security Pass" award is given at the period in which only a single remaining security level lamp remains to be lit (to start Payback Time).

When there is only a single security level lamp remaining for Payback Time (which would have to be one of the two "CPU Lit" lamps) and "Security Pass" is awarded the game will start Payback Time as one would expect. Once payback time is finished the game reveals that the "Security Pass" award has rolled over into the next round (the next round of 10 security level lamps that must be lit for another Payback Time). As a reasonable person would expect, there is 1 lit security level lamp (which is one of the two "Check Point" lamps) which is from the "Security Pass" award that was awarded to start the previous Payback Time.

The problem, however, is that although there is a lit security level lamp, the game is showing **both ramps with red arrows**. This, alone, goes against the previously described logic in which the game is supposed to ensure a sequential build-up of the security level lamps. The game is not supposed to allow a lamp to have its arrow lit if it has a security level lamp lit while at the same time the opposite ramp's corresponding security level is unlit.

By having both red arrows lit after the Payback Time in this manner, the player can get the build-up of security level lamps out of sequence. This can result in a confusing start of the next Payback Time as one ramp can have 2 unlit lamps while the other has all 5 security level lamps lit. This can happen if, after the Payback time with both ramps with red arrows, the player shots the ramp which already has the lit "Check Point" lamp. In doing this, that ramp will then light the "Pass Code" lamp even though the opposite ramp has zero lamps lit.

Payback Time Security Levels Bug Recipe

If this description is unclear or confusing, refer to the following recipe to reproduce the problem:

- Start a game, shoot ball onto playfield
- Shoot left ramp, illuminating its "Check Point" lamp
- Shoot right ramp, illuminating its "Check Point" lamp
- Shoot left ramp, illuminating its "Pass Code" lamp
- Shoot right ramp, illuminating its "Pass Code" lamp

- Shoot left ramp, illuminating its “Silent Alarm” lamp
- Shoot right ramp, illuminating its “Silent Alarm” lamp
- Shoot left ramp, illuminating its “Vault Key” lamp
- Shoot right ramp, illuminating its “Vault Key” lamp
- Shoot left ramp, illuminating its “CPU Lit” lamp
- Shoot Escape Route (left loop) to get the “Security Pass” award, starts Payback Time
- Allow Payback Time to complete

  At this time bug reveals itself:
  - The left ramp “Check Point” security level lamp is lit (expected)
  - The right ramp has no security level lamps lit (expected)
  - The left ramp red-arrow **is lit** (unexpected)
  - The right ramp red-arrow is lit (expected)

  Continuing the recipe, the lamps get out of sequence by, in this example, shooting the left ramp next.

- Shoot left ramp, illuminating its “Pass Code” lamp

  After this the right ramp arrow is flashing and left ramp arrow is off, as expected
- Shoot right ramp, illuminating its “Check Point” lamp
- Shoot left ramp, illuminating its “Silent Alarm” lamp
- Shoot right ramp, illuminating its “Pass Code” lamp
- Shoot left ramp, illuminating its “Vault Key” lamp
- Shoot right ramp, illuminating its “Silent Alarm” lamp
- Shoot left ramp, illuminating its “CPU Lit” lamp

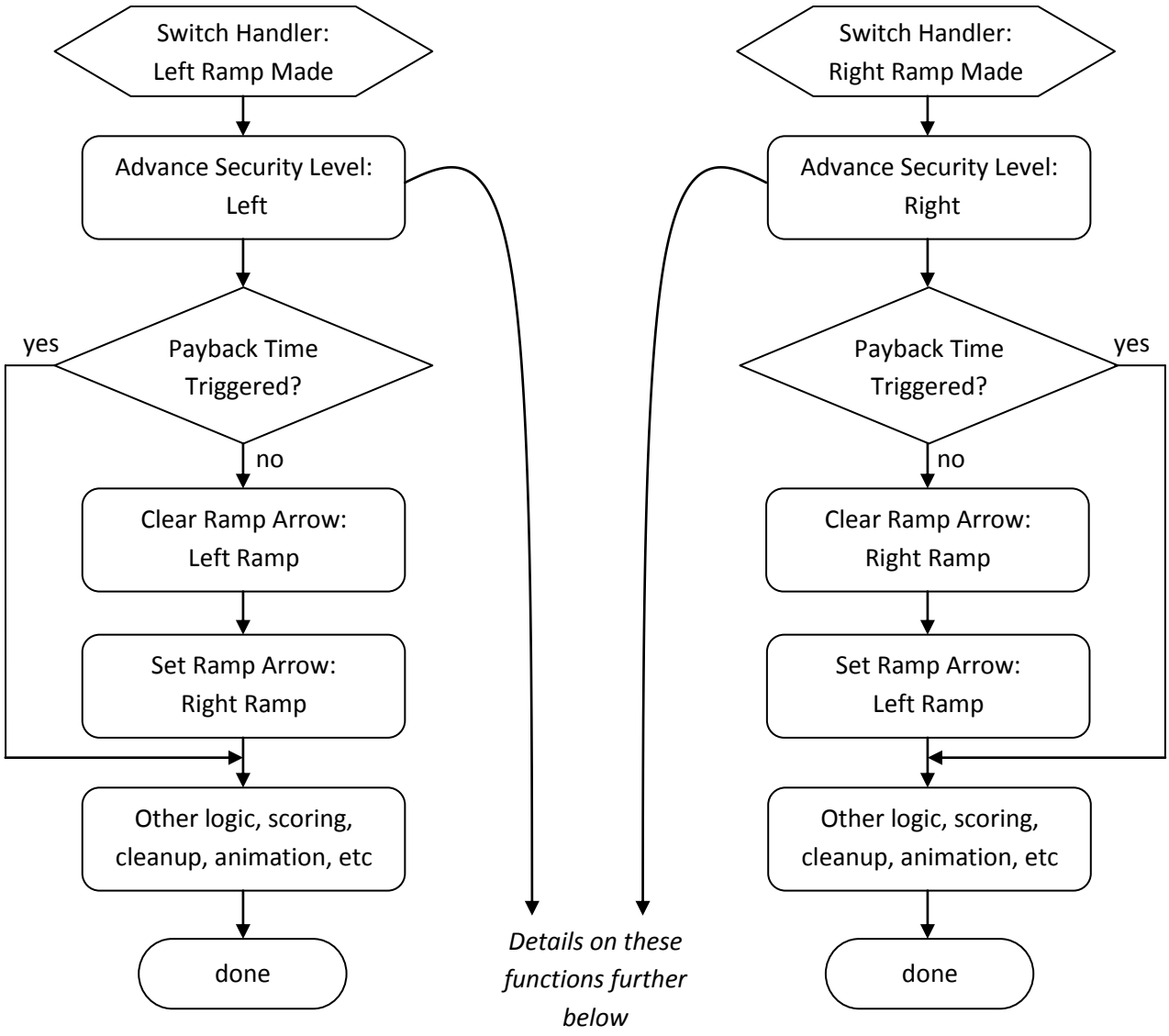
  After this the left ramp has all 5 security level lamps lit while the right ramp has only 3 of the 5 security level lamps solidly lit. It would seem there are 2 more shots needed on right ramp to light the remaining two security level lamps in order to start Payback Time, however we see here this is not the case in the next step.
- Shoot right ramp, Starts Payback Time.
- Allow Payback Time to complete

  At this time, there are no security lamps lit & both ramp red arrows are lit, as expected

Security Levels Code Design

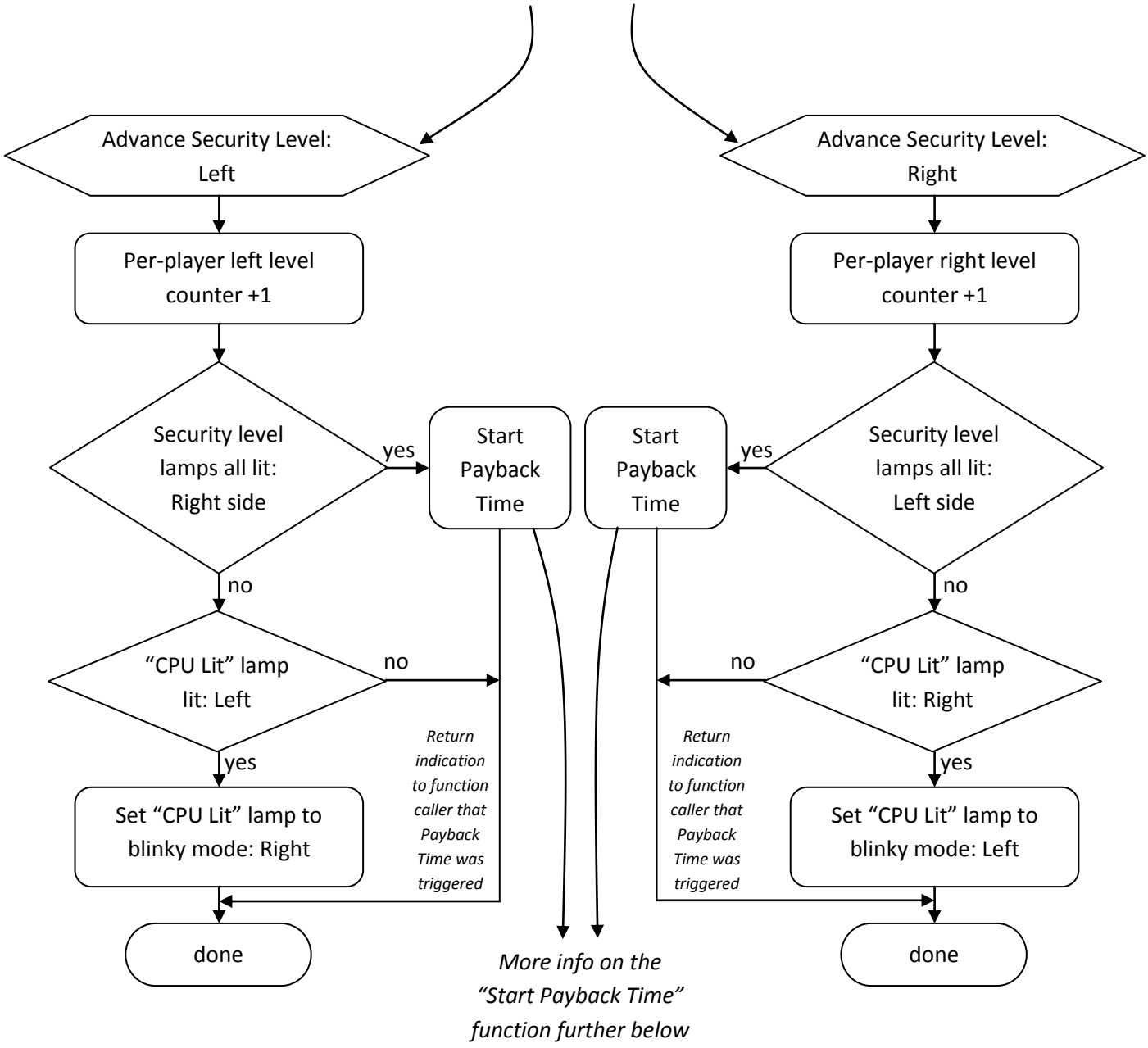
In order to develop the correct fix for the Payback Time Security Levels Bug, the L-8 software design related to the advancement of security level lamps has been investigated and is documented here. The flowcharts, below, depict the various software elements related to the security level lamps and where they get cleared and set during the handling of various playfield switches and awards.

The depicted logic is simplified, depicting code as it relates to the security level lamps. Other code elements are not detailed in the following flowcharts.



The ramp 'made' switch handlers both call an individual function for advancing the security level on behalf of the ramp shot. The details of these 'advance security level' functions will be provided below. The important takeaway from this logic is how, after advancing the ramp security level, a check is done to see if Payback Time had been triggered (as a result of such security level advancement) and, if so, skip the management of the ramp arrows. As will be shown below, the start of payback time will set both ramp arrows on, so this logic is important to ensure the arrows are not turned off immediately afterwards.

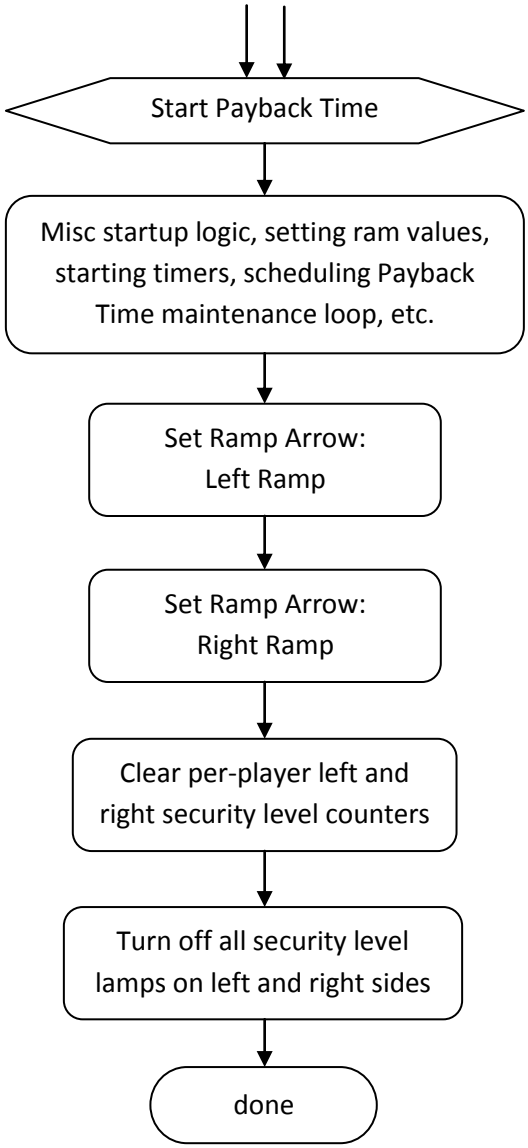
If Payback Time was not triggered by the security level advancement, then this was an ordinary advancement towards payback time. In such case, after advancing the security level then the ramp arrow for the hit ramp is cleared and the ramp arrow for the opposite ramp is lit. *Also as part of the 'other logic' the other ramp is also made to blink for a short period.*



The left and right security level advancement functions are depicted above. Since the code utilizes distinct functions for the left and right security advancements, there two separate flow charts depicted above.

These security advancement functions increment a per-player, per side (left/right) counter for the current security level for the player on each side. After advancing the current side's counter, the opposite side's security level lamps are checked to see if all five of its lamps are lit and, if so, calls function to start Payback Time. If the opposite side's security level lamps are not all lit, then the function will check if the current side's "CPU Lit" lamp is now lit and, if so, causes the opposite side's "CPU Lit" to start blinking.

An abbreviated flowchart is shown below for "Start Payback Time". The important part of the function is how it deals with the lamp arrows and, as such, this is the part depicted in the following flowchart.

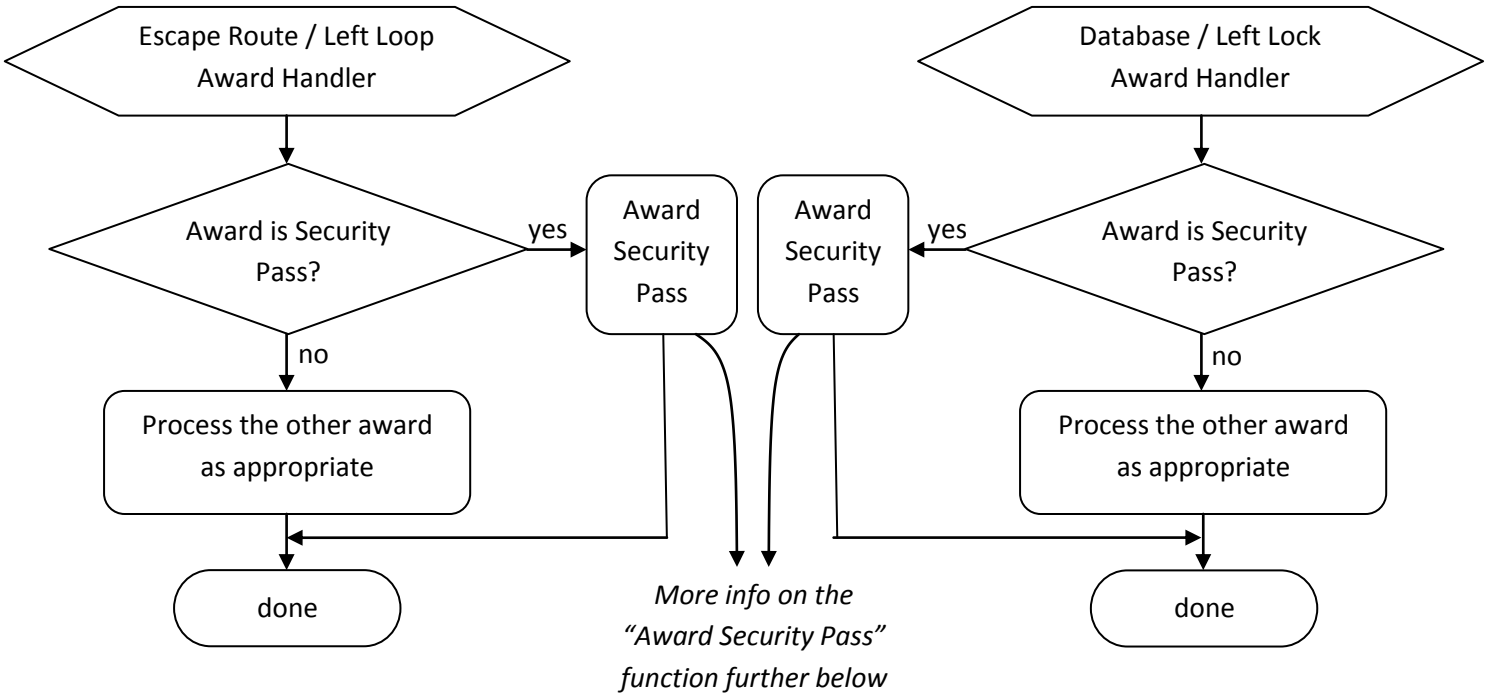


The “Start Payback Time” function, as depicted in the flowchart above, performs some basic housekeeping and schedules a separate function to maintain the Payback Time state. The housekeeping, as shown, includes specifically setting the left and right ramp red arrows as well as extinguishes the left and right security level lamps.

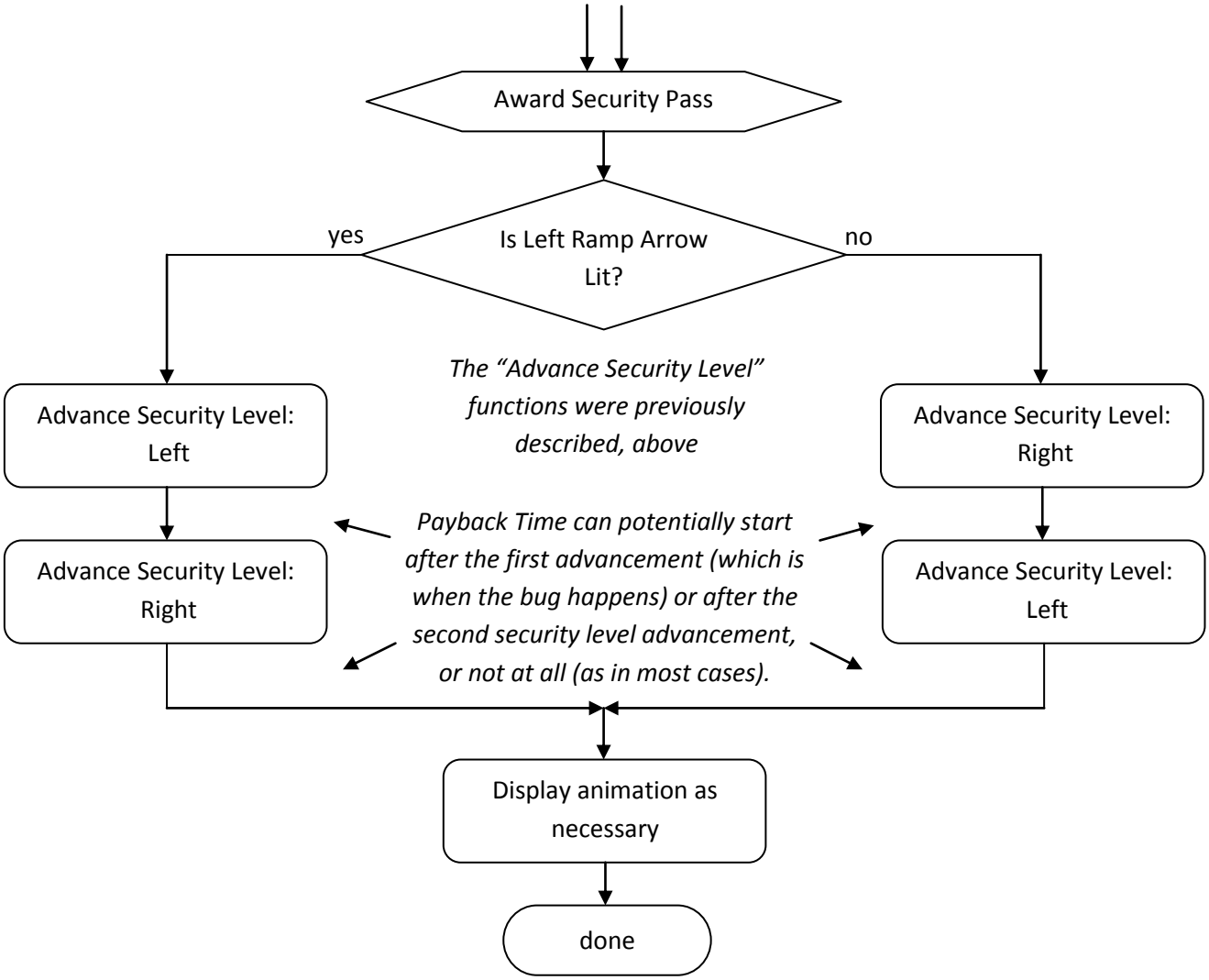
With the logic presented thus far, it is not unreasonable for the “Start Payback Time” function to set the lamps in this way. With assumption that payback time is initiated the instant that the final “CPU Lit” lamp is lit, then it is reasonable for the “Start Payback Time” function to set both lamp red arrows back to an illuminated state in anticipation for the next round of 10 security level completions for the next Payback Time.

Note that although this logic ‘sets’ the ramp arrows as part of Payback Time startup, the actual lit arrows are not revealed on the playfield until after the current Payback Time mode is complete. The Payback Time mode essentially overrides the playfield lamp states for the period of the Payback Time mode. When the mode is complete, the ‘normal’ playfield lamps are restored, including the 2 red ramp arrows as set in this “Start Payback Time” function.

With the above logic established, the “Security Pass” award can now be included into the discussion. The reason the security level lamp mismatch bug occurs will become apparent.



Shown above is the simplified logic for the Escape Route and Database award handlers. As one would expect, there are separate functions that handle the awards however the key takeaway in these flowcharts is the fact that both award handlers end up invoking the same “Award Security Pass” function. The “Award Security Pass” function is described next.

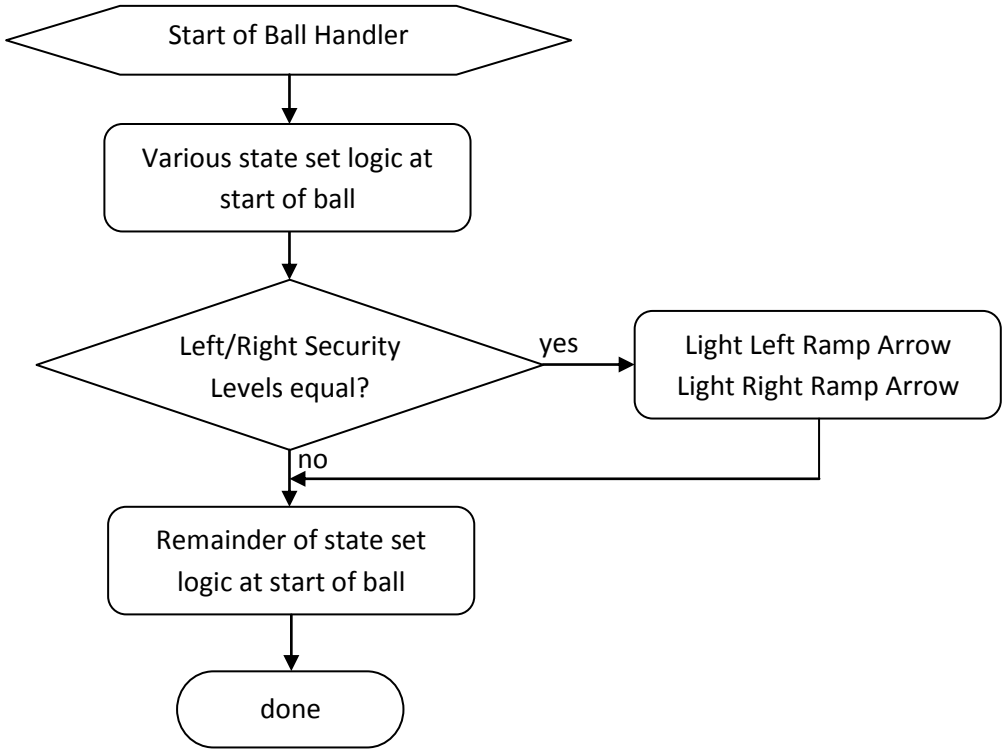


The flowchart for the "Award Security Pass" function is depicted above. This function simply checks if left ramp arrow is lit and uses it to decide whether to advance the security levels in the order of left then right or in the order of right then left. After advancing the security levels, there is some logic to determine whether to display an animation before finishing.

For readers following along, it should become apparent, with all of the logic presented thus far, how the security level lamp mismatch problem can happen. These security level advancements will immediately start Payback Time after either the first advancement or after the second advancement, or as in most cases, the Payback Time is not started at all if both the left and right "CPU Lit" lamps did not get lit as a result of the Security Pass award.

If the Payback Time starts after the first (of the two) advancements, the previous logic has shown how the Payback Time startup will light both left and right ramp arrows. Subsequent to such startup of Payback Time, the secondary advancement of security will take place which will simply increment the per-player left or right security level counter and then set the left or right "Check Point" lamp. In this case, the left and right ramp arrows remain lit as they were set during the "Start Payback Time" function and has not encountered any logic that otherwise clears the left or right ramp arrows.

Before considering the fix for this problem, an additional L-8 game design element should be mentioned. At the start of every ball one of the many things the game does is to check the security level for left and right ramps. If both security levels are equal then the game lights both left and right ramp arrows. Simplified logic for this behavior is as follows:



With the application of all of the logic presented thus far, an implied behavior takes place which will be important to retain with the L8.4 fix. The implied behavior is related to how the Security Pass award, in some cases, should retain both left and right arrows as lit.

In cases where the security levels are even (left and right have same number of lit security pass awards) and the game has both left and right ramp arrows lit, when the player is awarded Security Pass, the expectation is that after the award of left and right security level lamps, the left and right ramp arrows both remain lit. It would be unfair and presumptuous of the security pass award to advance the level evenly but then turn off one of the ramp arrows.

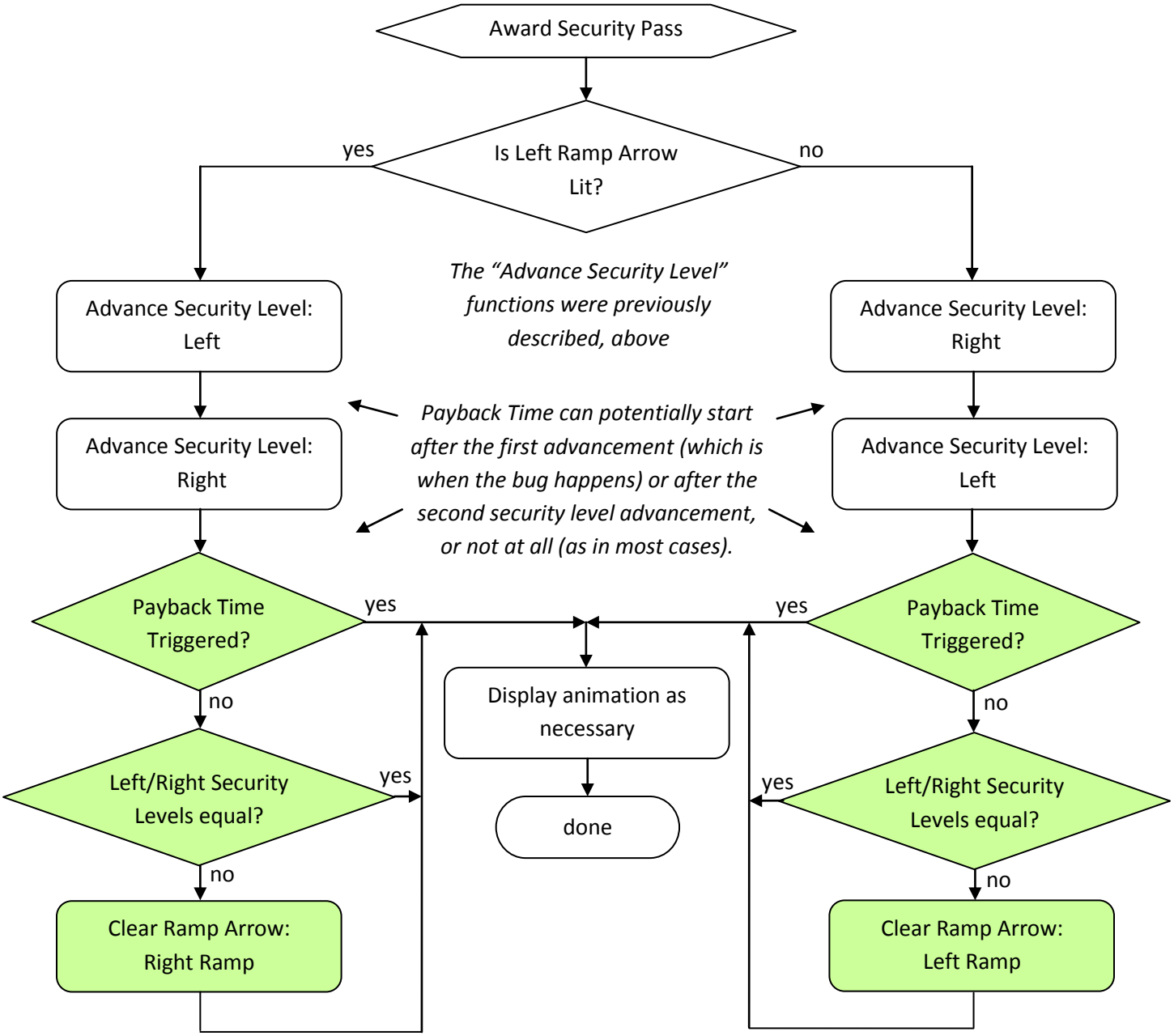
Security Levels Bug Fix

Based on the presented logic, the fix for the security levels lamp mismatch bug appears to be needed in the Security Pass award logic. The existing L-8 software (as shown above) leverages on the left/right ramp "made" switches to clear the lamp arrow after advancing the security level (if Payback Time has not started as a result of such security level advancement).

It is evident that the correct logic would therefore involve having the "Award Security Pass" function also take the extra effort of clearing the lamp arrow for the second (of the two) security level advancements. The existing L-8 "Award Security Pass" code simply makes an *assumption* that the two

security advancements do not require additional consideration about clearing any ramp arrows since the advancement of two security levels should not change the lit ramp arrow state. Clearly this assumption is incorrect since the first of the two security level advancements can cause both ramp arrows to become lit as part of the "Start Payback Time" function.

The updated logic for L8.4 is depicted in the following updated "Award Security Pass" flowchart with new logic highlighted. If the left/right security levels are equal then the new code will not attempt to clear left or right ramp arrows, thus leaving them unchanged to retain L-8 design where an even application of Security Pass award will keep both ramp arrows lit if they were both lit prior to the award of the Security Pass.



Security Levels Bug Fix – Code Changes

The necessary code changes for updating the "Award Security Pass" function require an update to the security pass security level advancement function. The actual implementation of the "Award Security Pass" function, as it is depicted in the flowcharts above, starts at $57DA,31, ROM offset 0x457DA. This is the common function that is invoked from both the Escape Route and the Database Award functions.

```
---------------------------------------;-----------------------------------------------------
                                       ;
                                       ; AwardSecurityPass()
                                       ;
57DA: 86 03       LDA   #$03           ;
57DC: B7 06 0F    STA   $060F          ;
57DF: BD 5E D3    JSR   $5ED3          ; SecurityPassSecurityLevelLampsAdvance()
57E2: 0D C0       TST   $C0            ;
57E4: 27 03       BEQ   $57E9          ;
57E6: BD 52 6D    JSR   $526D          ; WaitForAnimationCompletionE8()
57E9: BD 5E 8F    JSR   $5E8F          ; SecurityPassAnimation()
57EC: BD 71 A3    JSR   $71A3          ; Increment05BDbyPlayerIndexNumber()
57EF: 39          RTS                  ;
                                       ;
---------------------------------------;-----------------------------------------------------
```

The important part of the function is the call to $5ED3,31 which is where the security levels are advanced as part of the Security Pass award. This function is depicted below:

```
---------------------------------------;-----------------------------------------------------
                                       ;
                                       ; SecurityPassSecurityLevelLampsAdvance()
                                       ;
5ED3: BD 84 49    JSR   $8449          ; GetLampLitState() C-clear when lamp is on
5ED6: 30                               ; 30 == Left Ramp
5ED7: 24 06       BCC   $5EDF          ;
5ED9: 8D 47       BSR   $5F22          ; AdvanceSecurityLevelRight(), C-clr = Payback Time
5EDB: 8D 07       BSR   $5EE4          ; AdvanceSecurityLevelLeft(),  C-clr = Payback Time
5EDD: 20 04       BRA   $5EE3          ;
5EDF: 8D 03       BSR   $5EE4          ; AdvanceSecurityLevelLeft(),  C-clr = Payback Time
5EE1: 8D 3F       BSR   $5F22          ; AdvanceSecurityLevelRight(), C-clr = Payback Time
5EE3: 39          RTS                  ;
                                       ;
---------------------------------------;-----------------------------------------------------
```

The security level advancement function above, performs the logic previously depicted in the "Award Security Pass" flowchart. This function in L-8 is compact using BSR instructions to cause the function to be called using only 2 bytes per call. Since the bank $31 has very little available space for new code, the L8.4 code fix will require that the $5ED3,31 function to be replaced with a call to the its new replacement function located in bank $3A where more available ROM space can allow for the full and proper fix.

Moving the function to bank $3A could potentially impose additional time and stack space as the new function in bank $3A is called from bank $31 and it, in turn, calls into functions in bank $31. Such extra time is in the order of, perhaps, a few hundred microseconds and not any different from other normally running code that involves code from one bank calling code in another bank. Initial testing with this code change shows it behaves as expected and the L8.4 code will proceed with the following changes.

The security level advancement function at $5ED3,31 for L8.4 is replaced with the following new content:

```
---------------------------------------;------------------------------------------------------
                                       ;
                                       ; SecurityPassSecurityLevelLampsAdvance()
                                       ; Function replacement for L8.4
                                       ;
5ED3: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
5ED6: 7A F4 3A                         ; L8.4 Security Pass Award Fixup
5ED9: 39          RTS                  ;
5EDA: FF FF FF FF                      ; <unused ROM bytes>
5EDE: FF FF FF FF                      ; <unused ROM bytes>
5EE2: FF FF                            ; <unused ROM bytes>
                                       ;
---------------------------------------;------------------------------------------------------
```

The advancement function is now a jump to the new L8.4 fixup function located at $7AF4,3A, ROM offset 0x6BAF4. At this location in bank $3A is unused ROM bytes which are replaced with the following new function to incorporate the logic fixes described above for the security level lamp mismatch problem. This location in bank $3A is immediately after other L8.4 code additions, thus keeping a lot of the new L8.4 code in the same region of the ROM image.

```
---------------------------------------;------------------------------------------------------
                                       ;
                                       ; SecurityPassSecurityLevelLampsAdvance()
                                       ;
7AF4: BD 84 49    JSR   $8449          ; GetLampLitState() C-clear when lamp is on
7AF7: 30                               ; 30 == Left Ramp
7AF8: 24 18       BCC   $7B12          ;
                                       ; Left ramp lamp is not lit so assume right ramp is lit.
                                       ; Apply the Security Pass award, right then left levels.
                                       ; Then clear the lamp from left ramp lamp unless its
                                       ; security level advancement started Payback Time.
                                       ;
7AFA: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
7AFD: 5F 22 31                         ; AdvanceSecurityLevelRight()
7B00: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
7B03: 5E E4 31                         ; AdvanceSecurityLevelLeft()
7B06: 24 08       BCC   $7B10          ; If Payback Time started, C-clr, skip ramp lamp off
7B08: 8D 1F       BSR   $7F29          ; Check if left/right security levels are equal
7B0A: 27 04       BEQ   $7B10          ; If left/right levels are equal, don't clear ramp lamp
7B0C: BD 84 2B    JSR   $842B          ; ClearSingleLamp()
7B0F: 30                               ; 30 == Left Ramp
7B10: 20 16       BRA   $7F28          ;
                                       ;
                                       ; Left ramp lamp is lit so assume right ramp is not lit.
                                       ; Apply the Security Pass award, left then right levels.
                                       ; Then clear the lamp from right ramp unless its
                                       ; security level advancement started Payback Time.
                                       ;
7B12: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
7B15: 5E E4 31                         ; AdvanceSecurityLevelLeft()
7B18: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
7B1B: 5F 22 31                         ; AdvanceSecurityLevelRight()
7B1E: 24 08       BCC   $7F28          ; If Payback Time started, C-clr, skip ramp lamp off
7B20: 8D 07       BSR   $7F29          ; Check if left/right security levels are equal
7B22: 27 04       BEQ   $7F28          ; If left/right levels are equal, don't clear ramp lamp
7B24: BD 84 2B    JSR   $842B          ; ClearSingleLamp()
7B27: 3A                               ; 3A == Right Ramp
```

```
                                        ;
7F28: 39          RTS                   ;
                                        ;
;---------------------------------------;-------------------------------------------------------
;---------------------------------------;-------------------------------------------------------
                                        ;
                                        ;
7F29: 34 12       PSHS  X,A             ;
7F2B: 8E 05 95    LDX   #$0595          ; 0x0595 RAM storage for per-player left security level
7F2E: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
7F31: A6 84       LDA   ,X              ;
7F33: 8E 05 99    LDX   #$0599          ; 0x0599 RAM storage for per-player right security level
7F36: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
7F39: A1 84       CMPA  ,X              ;
7F3B: 35 92       PULS  A,X,PC          ;
                                        ;
---------------------------------------;-------------------------------------------------------
```

As shown in the code above, the new logic behaves as per the previously depicted flowchart. The left/right security levels are compared and, if not equal, then the ramp lamp gets cleared. This ensures correct logic whereby an even security level advancement of both sides will keep both ramp lamps lit if they were lit prior to the Security Pass award. If the levels are uneven then the ramp lamp is sure to be cleared, thus correcting the bug for L8.4.

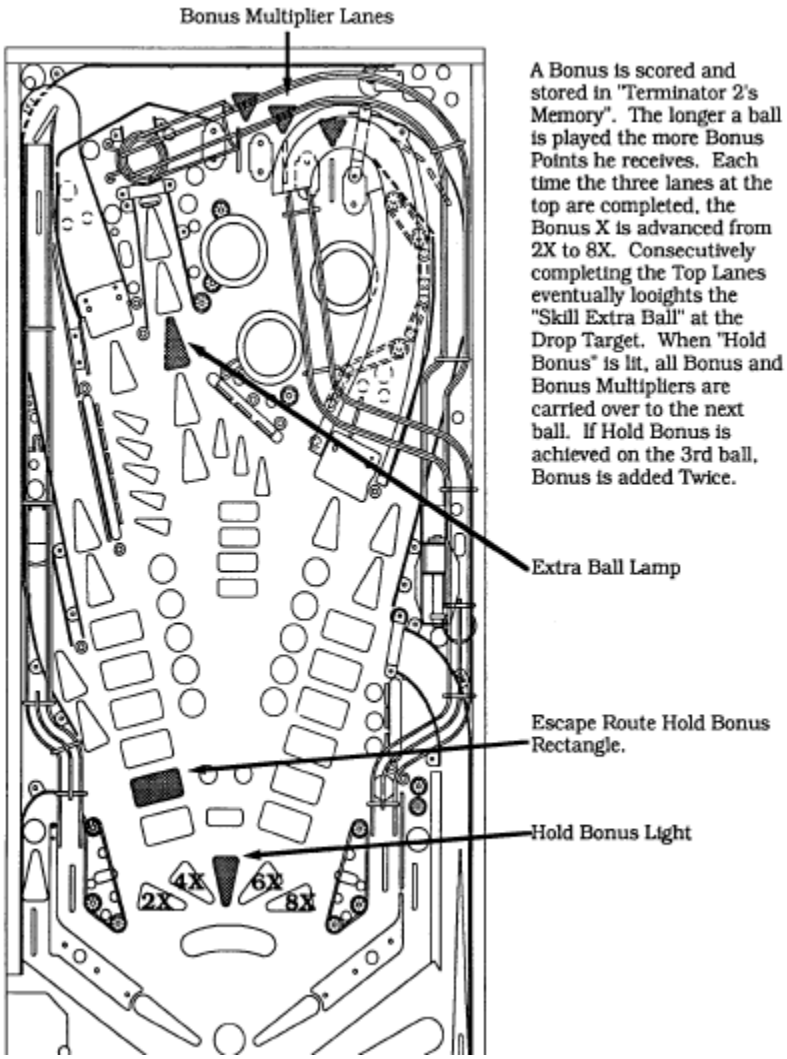
Post-Tilt Bonus Multipliers Lamp Bug

For L8.4, an existing L-8 bug related to the playfield insert lamp states for Bonus-X values is being corrected. The problem is related to the bonus-x lamp states in the next ball in play after the player has experienced a tilt.

A summary of the bonus multipliers logic and the bug is provided, below.

Terminator 2 Bonus and Bonus Multipliers Logic

The bonus multiplier logic in Terminator-2 is fairly straightforward and commonly understood by most pinball players. Below is a portion of the page from the Terminator-2 manual describing the way in which the bonus and bonus multipliers work.



It is worth noting that the manual description for the Database award also indicates that “Bonus X” is a possible award however the actual L-8 code does NOT offer “Bonus X” as a database award. It appears

that the actual game software added the “100,000” (or FUA) in place of the “Bonus X” since all of the other database awards in the game correspond to what the manual lists for database award values.

What, perhaps, is not as well known is how the bonus value is accumulated. Based on some investigation into the L-8 code, it appears the bonus value starts at 0 at each ball, and then increments during game play as the ball hits different targets. Each target’s handler has a specific call into 1 of 4 possible functions to increase the player’s bonus by a fixed set of points. Listed below are the 4 functions, the bonus points it adds, and the number of calls to each function in the L-8 code.

Bonus Point Accumulation Functions, L-8		
Function	**Points**	**Number of calls in L-8**
$FAA8	1,130	12
$FAAE	3,330	12
$FAB4	5,530	5
$FABA	9,930	6

Further details as exactly which switch ends up calling each bonus point accumulator function is left as an exercise to the reader. For readers investigating the code, it is worth noting that the bonus is stored in ram in binary coded decimal (BCD) and points are added up using opcodes that are intended to accumulate BCD values. For reference the functions start at $FAA8, ROM offset 0x7FAA8, and are shown below.

```
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; AddToBonus_1130_points()
FAA8: 34 16       PSHS  X,B,A           ;
FAAA: 86 11       LDA   #$11            ;
FAAC: 20 10       BRA   $FABE           ;
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; AddToBonus_3330_points()
FAAE: 34 16       PSHS  X,B,A           ;
FAB0: 86 33       LDA   #$33            ;
FAB2: 20 0A       BRA   $FABE           ;
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; AddToBonus_5530_points()
FAB4: 34 16       PSHS  X,B,A           ;
FAB6: 86 55       LDA   #$55            ;
FAB8: 20 04       BRA   $FABE           ;
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; AddToBonus_9930_points()
FABA: 34 16       PSHS  X,B,A           ;
FABC: 86 99       LDA   #$99            ;
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; AddToBonusValue()
                                        ; Adds value in A in hundreds plus 30 points.
                                        ;
FABE: BD F7 DE    JSR   $F7DE           ; GetAddressOfCurrentPlayerBonusScoreBcd()
FAC1: 6D 01       TST   $0001,X         ;
FAC3: 27 06       BEQ   $FACB           ;
FAC5: E6 02       LDB   $0002,X         ;
FAC7: C1 15       CMPB  #$15            ;
```

```
FAC9: 22 1E       BHI   $FAE9           ;
FACB: 34 02       PSHS  A               ;
FACD: 86 30       LDA   #$30            ; Adds 30 points to the lowest 2 BCD digits.
FACF: AB 04       ADDA  $0004,X         ;
FAD1: 19          DAA                   ;
FAD2: A7 04       STA   $0004,X         ;
FAD4: 35 02       PULS  A               ;
FAD6: A9 03       ADCA  $0003,X         ; Adds value in A to next 2 BCD digits.
FAD8: 19          DAA                   ;
FAD9: A7 03       STA   $0003,X         ;
FADB: 86 00       LDA   #$00            ;
FADD: A9 02       ADCA  $0002,X         ;
FADF: 19          DAA                   ;
FAE0: A7 02       STA   $0002,X         ;
FAE2: 86 00       LDA   #$00            ;
FAE4: A9 01       ADCA  $0001,X         ;
FAE6: 19          DAA                   ;
FAE7: A7 01       STA   $0001,X         ;
FAE9: 35 96       PULS  A,B,X,PC        ;
                                        ;
----------------------------------------;-------------------------------------------------
```

An additional detail to mention is the “Hold Bonus” feature. When the bonus is held (as an Escape Route award) the player’s bonus value is added to their running score and the bonus score value is simply not cleared at the start of the next ball. The bonus score value in memory continues to accumulate during the next ball in play.

At the end of a ball during bonus add-up, the code will check if the “Hold Bonus” lamp is set and use this as logic to determine how to proceed. This is shown in the portion of code, below. This is code starting at $7776,33, ROM offset 0x4F776.

```
7776: BD 84 49    JSR   $8449           ; GetLampLitState() C-clear when lamp is on
7779: 03                                ; 03 == Hold Bonus
777A: 24 0F       BCC   $778B           ; If lamp is on (C-clr) skip down to $778B
                                        ;
                                        ; This is the normal clearing of bonus lamps at end of
                                        ; ball that takes place when the "Hold Bonus" is not
                                        ; taking place.
                                        ;
777C: BD 87 BE    JSR   $87BE           ; ExtinguishLampGroupParamBytes()
777F: 05 00                             ;  Bonus2x4x6x8x lamps
7781: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
7784: 41                                ;  Bonus Held flag
7785: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
7788: 4A                                ;
7789: 20 38       BRA   $77C3           ;
                                        ;
                                        ; Here handling "Hold Bonus" condition at end of ball.
                                        ;
778B: BD 84 80    JSR   $8480           ; SetMemoryFlag()
778E: 41                                ;  Bonus Held flag
778F: BD 84 2B    JSR   $842B           ; ClearSingleLamp()
7792: 03                                ; 03 == Hold Bonus
7793: BD FB AE    JSR   $FBAE           ; ClearDisplayMemory()
7796: 7E 77 99    JMP   $7799           : <nop>
7799: BD D7 99    JSR   $D799           ; Print string on DMD
779C: 00 0A                             ; String index
779E: 0A                                ; font
779F: 40 17                             ; Coordinates
```

The above logic takes place during bonus collection. The logic will clear the “Hold Bonus” flag, if set, then sets the 0x41 flag before proceeding with a handful of additional work that adds in the bonus (not depicted). This 0x41 flag is set here as a crumb that the subsequent start of next ball will use to determine whether or not to clear the bonus points that have accumulated for the current player.

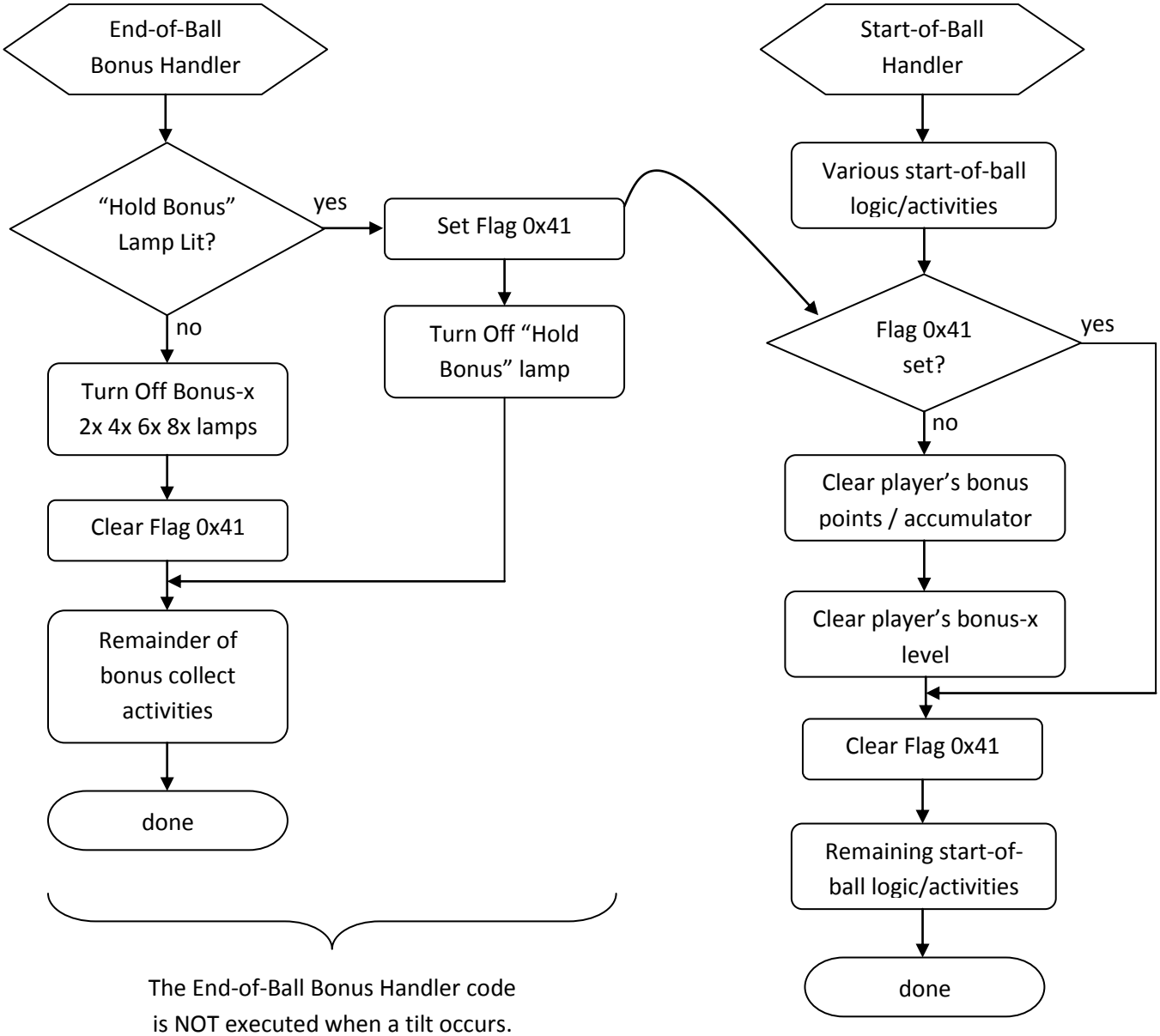
During the new ball-in-play reset function, the software reads flag 0x41 to determine if bonus has been held and, if so, the accumulated bonus points value is not cleared. If the 0x41 flag is set, the clearing of the bonus value and the clearing of the player's bonus-x level is skipped. If the bonus-held flag is not set then the game clears the player's bonus value and the player's bonus-x level. The code where this happens is at $62DE,3B, ROM offset 0x6E2DE, and is depicted below.

```
62DE: BD 84 AD    JSR   $84AD           ; GetMemoryFlag()  // C-bit clear when flag is set
62E1: 41                                ;  Bonus Held flag, it flag is set (c-clear)
62E2: 24 1B       BCC   $62FF           ;   then don't clear the bonus value here
                                        ;
62E4: BD F7 DE    JSR   $F7DE           ; GetAddressOfCurrentPlayerBonusScoreBcd()
62E7: 7E 62 EA    JMP   $62EA           ; <nop>  Next, clear the player's bonus BCD value
62EA: 6F 84       CLR   ,X              ; Clears $05DB:$05DC  (player 1)
62EC: 6F 01       CLR   $0001,X         ; Clears $05DD:$05DE  (player 1)
62EE: 6F 02       CLR   $0002,X         ; Clears $05DF:$05E0  (player 1)
62F0: 6F 03       CLR   $0003,X         ; Clears $05E1:$05E2  (player 1)
62F2: 6F 04       CLR   $0004,X         ; Clears $05E3:$05E4  (player 1)
                                        ;
62F4: 8E 05 91    LDX   #$0591          ; Starting address of per-player bonus-x level
62F7: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
62FA: 7E 62 FD    JMP   $62FD           ; <nop>
62FD: 6F 84       CLR   ,X              ; Clear the player's bonus-x level
                                        ;
62FF: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
6302: 41                                ; Clear bonus-held flag
```

As shown in the code sample above, the game tracks a per-player bonus-x level at $0591. This stores a numeric value representing the current bonus-x level, storing value 0 or 1 through 4 to indicate bonus-2x through 8x, respectively. This code portion reveals that the game tracks this bonus-x level separately from the lit bonus-x lamps. Although the game clears the player's bonus-x level here, the actual bonus-x lamps are cleared as separate part of the code. The actual part of code that clears the lamps is during the bonus add-up that takes place at end of ball, as depicted earlier where the bonus-x lamps were cleared with a call to $87BE with parameter bytes 0x05 and 0x00. This separation in how the lamps are tracked and a separate bonus-x level counter is used, leads to the Terminator-2 Bonus Multipliers Bug, described next.

Terminator 2 Bonus and Bonus Multipliers Logic Flowcharts

The previously depicted code for bonus can be summarized in the following flowcharts. These flowcharts show the end-of-ball bonus logic and the start-of-ball logic where the bonus-x and "hold bonus" lamps are considered.



The End-of-Ball Bonus Handler code is NOT executed when a tilt occurs.

Terminator 2 Bonus and Bonus Multipliers Bug

The bug that occurs in L-8 related to bonus multipliers is simply that, upon a tilt, the game software clears the player's bonus-x level however it does not turn off the corresponding bonus-x lamps that the player might have had accumulated prior to the tilt. The result of this is that during the subsequent ball-in-play, the internally tracked bonus multiplier starts at zero even though bonus-x lamps remain lit. When the next bonus multiplier is achieved, a new lamp is added to the set of bonus-x lamps that are already lit (if all 4 bonus-x lamps are not already lit).

An examination of the code reveals this behavior is simply the result of code that has no consideration for tilt and how it applies to the bonus-x lamps. The bonus-x lamp code appears to be designed with the idea that the game will always award bonus at end of a ball, however this assumption is invalid since a tilt can occur which bypasses bonus. The start-of-ball code that subsequently executes assumes that bonus-x lamps were cleared and proceeds to only clear the internally tracked bonus-x level for the player without doing anything with the bonus-x lamps.

At first it would seem the fix need to simply ensure that the code clears the bonus-x lamps at the same time it resets the player's bonus-x level. However it was also observed that the 'Hold Bonus' lamp remains lit throughout a tilt and the game will, at drain of the subsequent ball, hold the bonus as per the lit lamp (where such lit lamp was retained through the tilt). It is questionable as to whether the original design should have also cleared the "Hold Bonus" lamp during the tilt or if the current behavior is per design intentions. *Further study, below, determines this is likely a bug and that the "Hold Bonus" lamp should also be turned off after a tilt.*

The behavior of a game in how it handles tilt and how it penalizes the player in removal of bonus and/or bonus hold is entirely game specific. There is no requirement or expectation that all games behave identically in this matter.

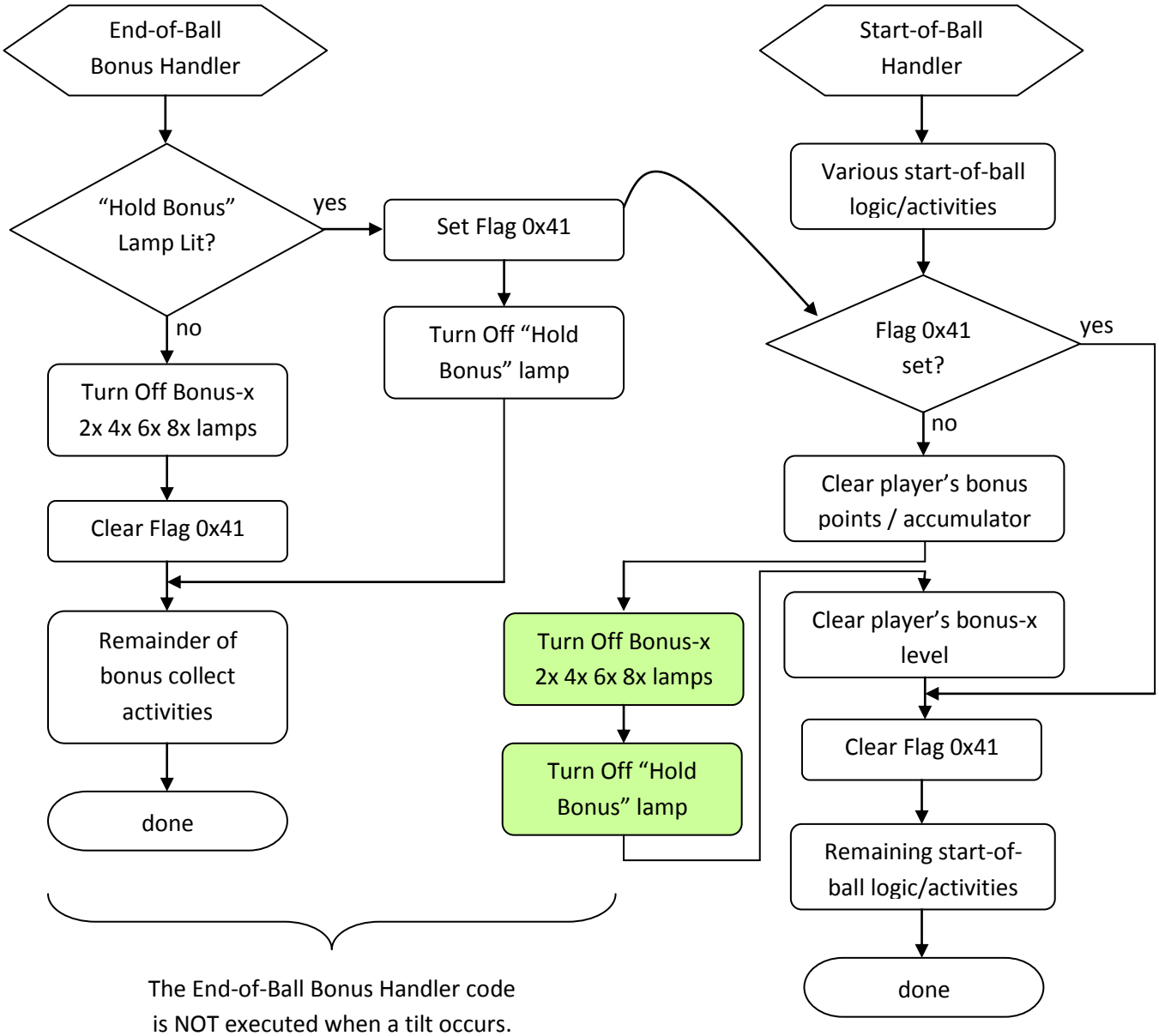
As part of considering what the correct logic should be for Terminator 2, with regard to tilt and bonus-x and bonus hold retention, a summary of existing behaviors is below, along with comparisons with two subsequent games from the same designer, the great Steve Ritchie.

Game	S/W	Multiplier Lamps	Hold Lamp	Hold Duration	Holds points	Holds multiplier	Tilt clears points	Tilt clears multipliers	Tilt clears hold
Terminator 2	L-8	2X 4X 6X 8X	Hold Bonus	Single Ball	Yes	Yes	Yes	Yes*	No**
The Getaway	L-5	2X 4X 6X 8X	Hold Bonus	Single Ball	Yes	Yes	Yes	Yes	Yes
Star Trek: TNG	LX-7	2X 4X 8X 10X	Hold Multiplier	Single Ball	N/A***	Yes	N/A***	No	Yes

*Multipliers are cleared but the lamps remain lit, bug subject to being fixed in L8.4.
** For L8.4 the code is being updated so that tilt will clear the "Hold Bonus".
*** STTNG does not accumulate bonus points in same sense as other games. Achievements = bonus.

Terminator 2 Bonus and Bonus Multipliers Bug Fix

After investigation in the Terminator 2 bonus-x and hold bonus lamp code, it was determined the appropriate fix for the bonus-x at tilt issue is as depicted in the updated flowchart below. New elements are highlighted.



The updated logic will simply clear the bonus-x 2x, 4x, 6x, 8x lamps and the "Hold Bonus" lamp when, at start of ball, it determines that the 0x41 flag is not set (and therefore normally clears only the per-player bonus-x level). This effectively fixes the problem with tilt that was previously resulting in these lamps

remaining lit at the start of the next ball following the tilt. This fix makes the obvious decision to clear the 2x, 4x, 6x and 8x lamps when the bonus-x level is being cleared. Prior to this, code was making a blind assumption that these lamps were already cleared when bonus was collected. This fix also blindly clears the “Hold Bonus” flag as there are no valid scenarios where the “Hold Bonus” lamp is lit at the start of a ball. With these changes in place, the regular game play and post-tilt game play will have correct set of lamps regardless of whether a tilt took place or whether the bonus was held at the end of the previous ball.

The code update that corresponds to this new logic starts in the previously depicted section of code at $62DE,3B, ROM offset 0x6E2DE, and is highlighted below.

```
62DE: BD 84 AD    JSR   $84AD           ; GetMemoryFlag()  // C-bit clear when flag is set
62E1: 41                                 ;  Bonus Held flag, it flag is set (c-clear)
62E2: 24 1B       BCC   $62FF            ;   then don't clear the bonus value here

62E4: BD F7 DE    JSR   $F7DE            ; GetAddressOfCurrentPlayerBonusScoreBcd()
62E7: 7E 62 EA    JMP   $62EA            ; <nop>  Next, clear the player's bonus BCD value
62EA: 6F 84       CLR   ,X               ; Clears $05DB:$05DC  (player 1)
62EC: 6F 01       CLR   $0001,X          ; Clears $05DD:$05DE  (player 1)
62EE: 6F 02       CLR   $0002,X          ; Clears $05DF:$05E0  (player 1)
62F0: 6F 03       CLR   $0003,X          ; Clears $05E1:$05E2  (player 1)
62F2: 6F 04       CLR   $0004,X          ; Clears $05E3:$05E4  (player 1)
                                         ;
62F4: 8E 05 91    LDX   #$0591           ; Starting address of per-player bonus-x level
62F7: BD FB 29    JSR   $FB29            ; IncrementXByPlayerIndexNumber()
62FA: 7E 62 FD    JMP   $62FD            ; <nop>
62FD: 6F 84       CLR   ,X               ; Clear the player’s bonus-x level
                                         ;
62F4: BD 88 F5    JSR   $88F5            ; CallBankedFunction_Param_WPCAddr()
62F7: 7B 3D 3A                           ; $7B3D,3A L8.4 Bonus-x bugfix
62FA: 7E 62 FF    JMP   $62FF            ; Jump down to resume normal code
62FD: 12          NOP                    ; Filler NOP
62FE: 12          NOP                    ; Filler NOP
                                         ;
62FF: BD 84 8F    JSR   $848F            ; ClearMemoryFlag()
6302: 41                                 ; Clear bonus-held flag
```

The start-of-ball code that was clearing the player’s bonus-x level is replaced with a call to a new function that is placed at $7B3D,3A, ROM offset 0x6BB3D. This is in a region of unused ROM space where other L8.4 bug fix functions are also located. The new function is as follows.

```
---------------------------------------;-------------------------------------------------------
                                       ;
7B3D: 34 10       PSHS  X              ;
7B3F: 8E 05 91    LDX   #$0591         ; Starting address of per-player bonux-X level
7B42: BD FB 29    JSR   $FB29          ; IncrementXByPlayerIndexNumber()
7B45: 6F 84       CLR   ,X             ;
7B47: BD 87 BE    JSR   $87BE          ; ExtinguishLampGroupParamBytes()
7B4A: 05 00                            ;  Bonux2x4x6x8x lamps
7B4C: BD 84 2B    JSR   $842B          ; ClearSingleLamp()
7B3F: 03                               ; 03 == Hold Bonus
7B40: 35 90       PULS  X,PC           ;
                                       ;
---------------------------------------;-------------------------------------------------------
```

The new function simply clears the player’s bonus-x level and then extinguishes the bonus-x 2x, 4x, 6x, 8x lamps and then extinguishes the “Hold Bonus” lamp, thus fixing the bonus-x at tilt bug for L8.4.

HSTD Table Update Investigation

One item on the list of L8.4 fixes was investigation into the report of high score data containing unexpected ordering. The report started with a video showing a Terminator 2 machine with attract mode depicting scores in the following order:

```
Grand Champion  : LRS - 191,784,950 <-- unexpected
Top Marksmen 1  : LRS - 324,629,310
Top Marksmen 2  : LRS - 215,750,310
Top Marksmen 3  : LRS - 201,094,490
Top Marksmen 4  :     - 158,320,130
```

The top score #4 contains blank name. The report is that the player walked away and missed their opportunity to enter the 3 initials.

An investigation into the code in the area of HSTD determination and insertion was done with the following observations:

- The game code involves redundant mechanisms when dealing with high score data.
- A player's score is checked multiple times for correct order when being put into the table.
- The Grand Champion and top-4 (Top Marksmen) are 2 separately maintained tables.
- The HSTD data is checked for valid ordering and memory contents.
- When HSTD is found to be invalid it gets reset to factory default content.
- When a score is found to have invalid data (anything other than 0-9 content) it is not used.
- For Initials, spaces are treated same as A-Z. It does not play a role in possible problems.
- Game adjustments allow custom default scores.
- If adjusted top-4 default scores are out of order, game will use factory default scores instead.
- Game will allow default grand champion to be less than top-4.
- Game allows disable of grand champion score from being tracked or shown.
- Game allows disable of all high scores (grand champion and top-4).

There was no coding flaw identified in which the invalid ordering could occur. The invalid ordering was, however, found to be achievable by utilizing the game adjustments such as in the following way:

- Start with factory default high scores.
- Score 191,784,950, Grand Champion.
- Adjust HSTD settings to disable Grand Champion.
- Score 324,629,310, Top Marksmen 1
- Score 215,750,310, Top Marksmen 2
- Score 201,094,490, Top Marksmen 3
- Score 158,320,130, Top Marksmen 4
- Adjust HSTD settings to enable Grand Champion.

After the above sequence, the scores would be shown as indicated in the problem statement, above. This is only serving to report one possible way in which the Grand Champion score could possibly be shown with a lower value than the Top Marksmen 1 score.

For L8.4 the following will go into detail showing the discovered HSTD code and how it operates. Perhaps a keen eye will discover something that was overlooked during this discovery process whereby the ordering of high score data could possibly get into an unexpected ordering.

HSTD Full Problem Statement

The full sequence that was reported in the problem statement is as follows:

Start with a Factory Reset which restores the HSTD table to the following:

```
Grand Champion  : AJA - 150,000,000
Top Marksmen 1  : XAQ -  90,000,000
Top Marksmen 2  : DOC -  70,000,000
Top Marksmen 3  : JAS -  50,000,000
Top Marksmen 4  : JCS -  30,000,000
```

Then simply play normally. Scoring various high scores over a period of time. Game may be shut off and back on between games, as days have been reported to have elapsed before the problem took place. All games are reported to have been single player games.

```
Grand Champion  : LRS - 324,629,310
Top Marksmen 1  : LRS - 215,750,310
Top Marksmen 2  : LRS - 201,094,490
Top Marksmen 3  :     - 158,320,130
Top Marksmen 4  : LSR - ???,???,??? (unknown)
```

The top score #4 would have to be something greater than 150,000,000 and less than or equal to 158,320,130. This is because the Grand Champion was 150,000,000 at the most recent factory settings reset. The entry with no name was due to player walking away without entering their initials and the lack of initials is not suspected of being related to the score ordering problem.

Next a game is played which scores 191,784,950 points. It is reported that the end of game reports that the player is a Grand Champion prior to the entry of initials. After entering the initials the player's score then appears as Grand Champion with the HSTD table then shown as depicted below (and shown in a video):

```
Grand Champion  : LRS - 191,784,950
Top Marksmen 1  : LRS - 324,629,310
Top Marksmen 2  : LRS - 215,750,310
Top Marksmen 3  : LRS - 201,094,490
Top Marksmen 4  :     - 158,320,130
```

As shown, the expectation is that the score with 191,784,950 should have been declared as "Top Marksmen #3" and it should have pushed the 158,320,130 score to position #4.

HSTD Code Walk Through

This section will describe the code flow from the end of game sequence to high-score report and collection of initials from the player. The attract mode sequence for reporting high scores is also shown. The comments and analysis are sometimes a best-guess as to the code intentions and, as always, subject to correction and revised interpretation.

Shown below is code starting in the middle of a function that handles end-of-ball sequence. Code shown below starts at $476C,38, ROM offset 0x6076C.

```
476C: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
476F: 10                                ; 0x10 == EndOfBallBonusAddupSequence()
                                        ;
4770: BD 83 46    JSR   $8346           ;-\ Sleep()
4773: 02                                ; | 31.25mS
4774: BD CA 90    JSR   $CA90           ; |
4777: 81 10       CMPA  #$10            ; |
4779: 27 F5       BEQ   $4770           ;-/
                                        ;
477B: BD 86 90    JSR   $8690           ;-\ SearchLinkedListForId()  // c-bit clear = ID found
477E: 38 13                             ; |
4780: 25 06       BCS   $4788           ; |
4782: BD 83 46    JSR   $8346           ; | Sleep()
4785: 01                                ; | 15.625mS
4786: 20 F3       BRA   $477B           ;-/
                                        ;
4788: 0F 88       CLR   $88             ;
478A: BD 85 E1    JSR   $85E1           ;
478D: 04                                ;
478E: BD 8A 9A    JSR   $8A9A           ; CancelAllCallbacksIdMaskParameterBytes()
4791: 00 00                             ;
4793: 10 00                             ;
4795: BD 4B 39    JSR   $4B39           ;
4798: BD 86 21    JSR   $8621           ; CallFunctionPointerParameterBytes()
479B: 80 31                             ; $8031 has $634E,3D
479D: BD B2 4E    JSR   $B24E           ; Checks if at end of game
47A0: 25 07       BCS   $47A9           ; C-set after last ball bonus was added up
47A2: 4F          CLRA                  ;
47A3: BD 46 9E    JSR   $469E           ; ends up setting lamps for current player at ball start
47A6: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;-------------------------------------------------
```

The code, above shows where the end of ball bonus sequence is triggered followed by some sleep loops which, presumably, wait for the bonus display to complete or dramatic pause at end of ball. After this is some other code that hasn't been investigated followed by the highlighted code where code checks if the last ball has just completed and, if so, branch to $47A9,38 which is the section of code that immediately follows.

The function at $47A9,38, ROM offset 0x607A9, is where the end of game sequences take place and is shown below.

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; Last ball (of all players) bonus was added up.
                                        ; now check for high score
                                        ;
47A9: 1C FE       ANDCC #$00FE          ;
47AB: BD 86 21    JSR   $8621           ; CallFunctionPointerParameterBytes()
47AE: 80 16                             ; $8016 has $634D,3D, Debug hook, RTS
47B0: 25 3A       BCS   $47EC           ; C-set upon error, nothing to do here
47B2: 86 01       LDA   #$01            ;
47B4: 97 87       STA   $87             ; Set $87 to 0x01 to mark end-of-game / game-over
47B6: BD 4D 3A    JSR   $4D3A           ;
47B9: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
47BC: 62 84 39                          ; EndOfGameReplayLevelAdjust()
47BF: BD 86 21    JSR   $8621           ; CallFunctionPointerParameterBytes()
47C2: 80 D0                             ; $80D0 has $6585,3B
47C4: BD 86 AC    JSR   $86AC           ; UpdateCurrentRunningScheduleFunctionIDParameterBytes()
47C7: 00 0C                             ;
47C9: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
47CC: 01                                ; $5D1A,3D table entry [01] has $6AC1,3B
47CD: 8D 20       BSR   $47EF           ; EndOfGameHstdUpdateAndMatchSequence()
                                        ;
                                        ;-------------------------------------------------
                                        ; Get here after any Initials have been entered and
```

```
                                        ; match sequence has finished.
                                        ;-------------------------------------------------------
47CF: BD 4D 5A    JSR   $4D5A           ;
47D2: BD 9E 0A    JSR   $9E0A           ;
47D5: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
47D8: 60 12 39                          ;
47DB: 24 04       BCC   $47E1           ;
47DD: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
47E0: 06                                ; $5D1A,3D table entry [06] has $7E19,30
47E1: BD 4B 6D    JSR   $4B6D           ;
47E4: BD 46 13    JSR   $4613           ;
47E7: BD 86 21    JSR   $8621           ;
47EA: 80 19       SUBA  #$19            ;
47EC: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;-------------------------------------------------------
```

The function, above performs various activities at the end of a game. The highlighted function is what is important for understanding HSTD determination and HSTD table insertion. The function at $47EF,38, ROM offset 0x607EF, handles the HSTD table update and match sequence and is shown below.

```
----------------------------------------;-------------------------------------------------------
                                        ;
                                        ; EndOfGameHstdUpdateAndMatchSequence()
                                        ;
47EF: 34 02       PSHS  A               ;
47F1: 86 03       LDA   #$03            ;
47F3: BD 4A 8D    JSR   $4A8D           ;
47F6: 0F 86       CLR   $86             ;
47F8: A6 9F 81 96 LDA   [$8196]         ; At $8196 is 0xFE90.  At $FE90 is 0x40, put into A here
47FC: 27 03       BEQ   $4801           ; Since $FE90 has 0x40 this never jumps over instruction
47FE: BD 98 94    JSR   $9894           ; Function popualtes video memory at $3A00
4801: BD 48 09    JSR   $4809           ; HighScoreInitialEntryAndAward()
4804: BD 48 DA    JSR   $48DA           ; MatchSequence()
4807: 35 82       PULS  A,PC            ;
                                        ;
----------------------------------------;-------------------------------------------------------
```

The function, above, sets up some video memory and then calls a function that handles the entire HSTD award and then calls a function that handles the entire match sequence. The HSTD function is at $4809,38, ROM offset 0x60809, and is shown below.

```
----------------------------------------;-------------------------------------------------------
                                        ;
                                        ; HighScoreInitialEntryAndAward()
                                        ;
4809: 34 56       PSHS  U,X,B,A         ;
480B: 32 7D       LEAS  $FFFD,S         ; Make room for 3 bytes on the stack.
                                        ;  S[0] Contains bitmap of handled players
                                        ;       0x01=player 1
                                        ;       0x02=player 2
                                        ;       0x04=player 3
                                        ;       0x08=player 4
                                        ;  S[1] contains player indicator.
                                        ;       01=Player 1
                                        ;       02=Player 2
                                        ;       03=Player 3
                                        ;       04=Player 4
                                        ;  S[2] ends up getting HSTD number
                                        ;       00=Grand Champion
                                        ;       01=Top Marksmen 1
                                        ;       02=Top Marksmen 2
                                        ;       03=Top Marksmen 3
                                        ;       04=Top Marksmen 4
480D: 1C FE       ANDCC #$00FE          ;
480F: BD 86 21    JSR   $8621           ; CallFunctionPointerParameterBytes()
```

```
4812: 80 6D                             ; $806D has $637B,3D, Debug hook, just an RTS
                                        ;
4814: 10 25 00 A5 LBCS  $48BD           ; C is always clear, so no branch
                                        ;
4818: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
481B: A2 00                             ; Adj=0xA2 $1B61:$1B62 HstdAdj001, Highest Scores
481D: 10 24 00 9C LBCC  $48BD           ; C-bit set = normal high scores is allowed, no branch
                                        ;
4821: 6F E4       CLR   ,S              ; Start with S[0] = 0x00, no players scores checked yet
                                        ;
4823: 33 E4       LEAU  ,S              ;-\ SP pointer into U, start of 3 bytes on the stack
4825: BD 4A 17    JSR   $4A17           ; | GetNextPlayerHstdAchieved()
4828: 10 25 00 91 LBCS  $48BD           ; | C-set no new high score found.
                                        ; |
                                        ; | Now collect the initials from user.
                                        ; | Example of the three byte buffer in various cases:
                                        ; |
                                        ; |                      Bitmap  PlayerIndex  HSTDEntry
                                        ; | ------------------------------------------------
                                        ; | Player 1 new GC    :   01        01          00
                                        ; | Player 1 new Top #1:   01        01          01
                                        ; | Player 1 new Top #2:   01        01          02
                                        ; | Player 1 new Top #3:   01        01          03
                                        ; | Player 1 new Top #4:   01        01          04
482C: 86 01       LDA   #$01            ; |
482E: B7 04 B1    STA   $04B1           ; |
4831: BD AB A2    JSR   $ABA2           ; | general illumination related
4834: AE 61       LDX   $0001,S         ; | X gets 2 bytes of PlayerIndex and HSTDEntry
4836: BD 86 21    JSR   $8621           ; | CallFunctionPointerParameterBytes()
4839: 80 61                             ; | $8061 has $6377,3D, Debug hook, just and RTS
483B: BD 85 53    JSR   $8553           ; | ShowMonochromeAnimationParameterByte()
483E: 07                                ; | $5D1A,3D [07] $505E,24
                                        ; |  ReportPlayerGrandChamptionOrTopMarksmen()
483F: 24 06       BCC   $4847           ; | C-set when error detected
                                        ; |
                                        ; |-----------------------------------
4841: BD 82 B6    JSR   $82B6           ; | ErrorHandler()
4844: 61                                ; |
4845: 20 76       BRA   $48BD           ; | Branch to the end in error scenario
                                        ; |-----------------------------------
                                        ; |
4847: BD 4A 06    JSR   $4A06           ; | WaitForHstdFunctionToComplete()
484A: AE 61       LDX   $0001,S         ; | X gets PlayerIndex and HSTDEntry
484C: BD 86 21    JSR   $8621           ; | CallFunctionPointerParameterBytes()
484F: 80 64                             ; | $8064 has $6378,3D, RTS, debug hook
4851: BD 85 53    JSR   $8553           ; | ShowMonochromeAnimationParameterByte()
4854: 08                                ; | $5D1A,3D [08] $4DC3,24 HstdCollectPlayerInitials()
4855: 25 EA       BCS   $4841           ; |
4857: BD 4A 06    JSR   $4A06           ; | WaitForHstdFunctionToComplete()
485A: BD 88 F5    JSR   $88F5           ; | CallBankedFunction_Param_WPCAddr()
485D: 5B FF 39                          ; |
                                        ; |
                                        ; | Now figure out how many credits/tickets to award
4860: CE 00 00    LDU   #$0000          ; |
4863: 33 C6       LEAU  A,U             ; |
4865: 86 A5       LDA   #$A5            ; | Adj=0x25 $1B67:$1B68 HstdAdj004, Champion Credits
4867: AB 62       ADDA  $0002,S         ; | Lookup adjustment 0xA5, 0xA6, 0xA7, 0xA8, or 0xA9 to
                                        ; |  determine how many credits or tickets to award
4869: BD 92 DE    JSR   $92DE           ; | Get8BitSettingIntoA()
486C: 25 3A       BCS   $48A8           ; | C-set upon error, if so go to $48A8
486E: BD 86 5B    JSR   $865B           ; | LookupGameAdjParameter1andCheckIfEqualsParam2()
4871: A3 01                             ; | Adj=0xA3 $1B63:$1B64 HstdAdj002, HSTD Award
4873: 25 1C       BCS   $4891           ; | C-set when credit, c-clear when ticket
                                        ; |
                                        ; | Awarding ticket
4875: E6 61       LDB   $0001,S         ; |
4877: BD 88 F5    JSR   $88F5           ; |
487A: 45 B1 3D                          ; |
487D: C6 09       LDB   #$09            ; |
487F: BD 88 F5    JSR   $88F5           ; |
4882: 45 2B 3D                          ; |
```

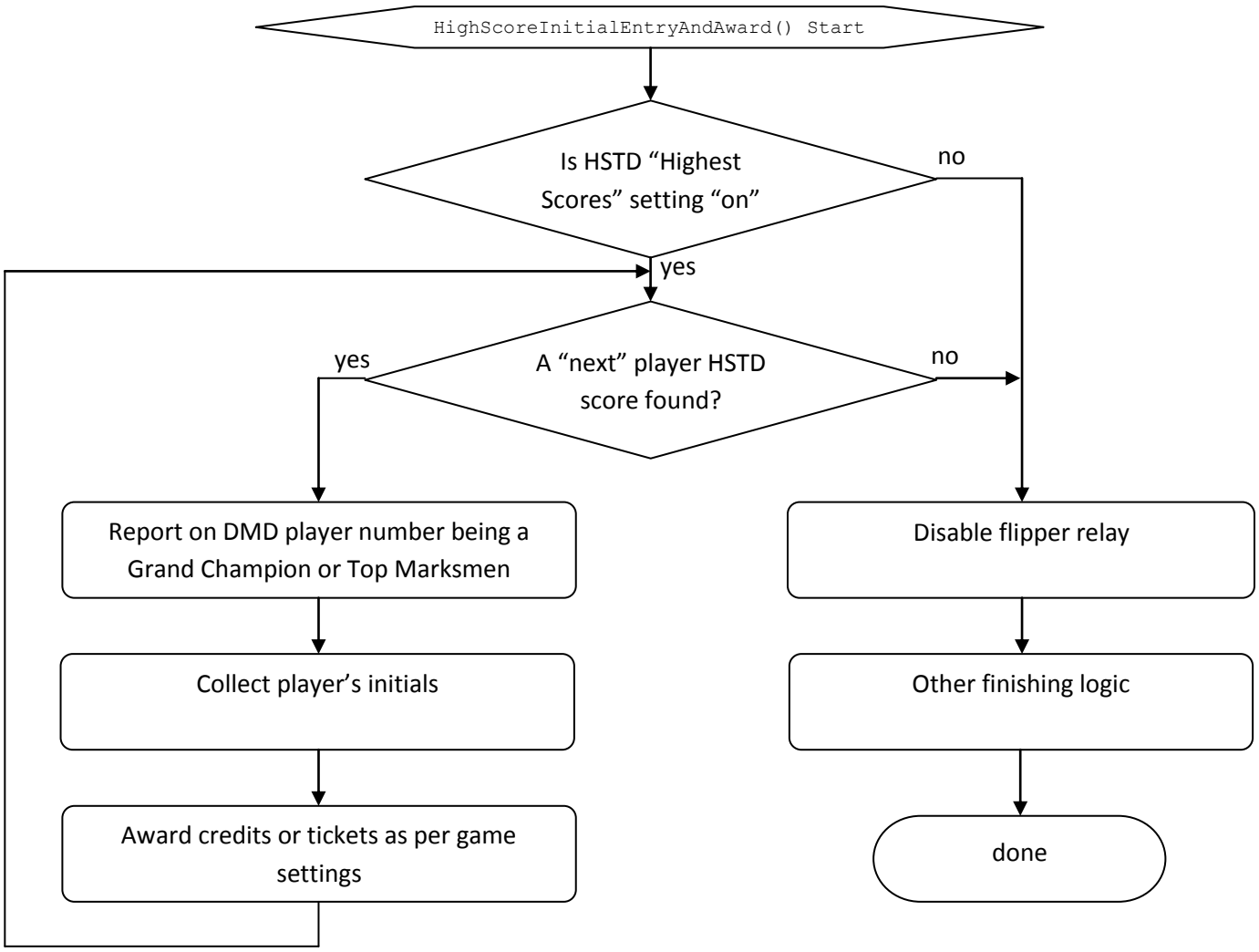
```
4885: 8E 80 19    LDX   #$8019           ; |
4888: BD 88 F5    JSR   $88F5            ; |
488B: 52 5D 39                           ; |
488E: 4F          CLRA                   ; |
488F: 20 17       BRA   $48A8            ; |
                                         ; |
                                         ; | Awarding credit
4891: BD 88 F5    JSR   $88F5            ; |
4894: 45 90 3D                           ; |
4897: C6 04       LDB   #$04             ; |
4899: BD 88 F5    JSR   $88F5            ; |
489C: 45 22 3D                           ; |
489F: 8E 80 19    LDX   #$8019           ; |
48A2: BD 88 F5    JSR   $88F5            ; |
48A5: 52 5D 39                           ; |
                                         ; |
48A8: 1F 89       TFR   A,B              ; |
48AA: AE 61       LDX   $0001,S          ; |
48AC: BD 86 21    JSR   $8621            ; | CallFunctionPointerParameterBytes()
48AF: 80 67                              ; | $8067 == $6379,3D
48B1: BD 85 53    JSR   $8553            ; | ShowMonochromeAnimationParameterByte()
48B4: 09                                 ; | 0x09 == $50B9,24
48B5: 25 8A       BCS   $4841            ; | C-set? goto error handler
48B7: BD 4A 06    JSR   $4A06            ; |
48BA: 7E 48 23    JMP   $4823            ;-/
                                         ;
48BD: B6 04 B1    LDA   $04B1            ;
48C0: 27 0B       BEQ   $48CD            ;
48C2: BD AB 98    JSR   $AB98            ; FlippersRelayDisable()
48C5: 7F 04 B1    CLR   $04B1            ;
48C8: BD 86 21    JSR   $8621            ; CallFunctionPointerParameterBytes()
48CB: 80 6A                              ; $806A has $637A,3D
                                         ;
48CD: 32 63       LEAS  $0003,S          ; Restore stack pointer, done with 3 byte buffer
48CF: 35 D6       PULS  A,B,X,U,PC       ;
                                         ;
-----------------------------------------;-------------------------------------------------
```

The function, above repeatedly calls the GetNextPlayerHstdAchieved() function until it returns C-bit set. Each time the GetNextPlayerHstdAchieved() function returns C-bit clear, it means another player (of the up to 4 players) has been found to have a score that belongs as a new entry in the Grand Champion or top-4 high score tables. Each call to GetNextPlayerHstdAchieved() discovers the best score among all players (and which has not been discovered from a previous call to GetNextPlayerHstdAchieved()) as the loop keeps checking until no more players with score worthy of HSTD are present.

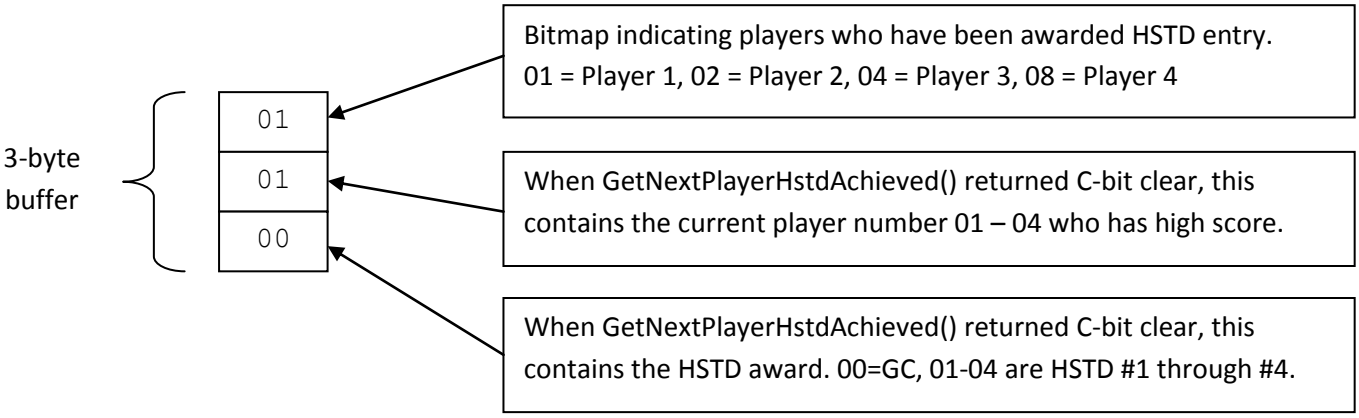
For each player that has been found to to have a score that belongs as a new entry in the GC or top-4 HSTD tables, the code will collect the player's initials and then award credits or tickets, as according to the game settings. After the award, the next pass through loop takes place to check for another player with a high score.

Highlighted in the function, above, are the calls to GetNextPlayerHstdAchieved() where the 'next' player score is discovered for HSTD insertion and the call to HstdCollectPlayerInitials() where the game collects the 3 initials for the player that is currently being added as a new GC or top-4 HSTD entry. Details on these functions will be provided later, below.

Next is a flowchart depicting the above function.



As indicated in the function code, above, the code utilizes a 3-byte buffer as it repeatedly calls the GetNextPlayerHstdAchieved() function. The first byte stores a bitmap that saves which players have been processed so that subsequent calls into GetNextPlayerHstdAchieved() will exclude such players from consideration due to already being recognized as a high score. In the case of a single player game, the GetNextPlayerHstdAchieved() function is only called a single time.



As indicated, above, the GetNextPlayerHstdAchieved() is the first part of code that makes a determination of a player's score as being worthy of being inserted into the GC or top-4 high score tables. *At first one might think the incorrect ordering of players score could happen if the 3-byte buffer somehow contained an incorrect value of 00 in its 3rd byte (which means player gets Grand Champion) however later it will be shown that even if the 3rd byte were to incorrectly contain a 00, subsequent code will refuse to put the players score into the Grand Champion if the existing GC score is larger than the player's score that is being inserted.*

The GetNextPlayerHstdAchieved() is at $4A17,38, ROM offset 0x60A17, and is shown below.

```
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ;
                                        ; GetNextPlayerHstdAchieved()
                                        ;
4A17: 34 76       PSHS  U,Y,X,B,A       ;
4A19: 32 7B       LEAS  $FFFB,S         ; Make room for 5 bytes on the stack
4A1B: CC 00 00    LDD   #$0000          ;
4A1E: A7 61       STA   $0001,S         ;
4A20: A7 62       STA   $0002,S         ;
4A22: ED 63       STD   $0003,S         ;    Counter   Player Idx  Bitmap     Score Ptr
4A24: 86 01       LDA   #$01            ; ------------------------------------------
4A26: A7 E4       STA   ,S              ; SP:  01        00         00         00 00
                                        ;
4A28: C6 01       LDB   #$01            ; Starting bitmap: 01 (player 1)
                                        ;
                                        ; The following loop finds the next best top high score
                                        [ among up to 4 players scores
                                        ; A bitmap is used to exclude already awarded players
                                        ;-----------------------------------------------------
4A2A: EE 6B       LDU   $000B,S         ;-\ U gets address of caller's 3-byte buffer
4A2C: E5 C4       BITB  ,U              ; | Check if high score byte [0] has the bit in B set
4A2E: 26 18       BNE   $4A48           ; | If set, skip, already awarded this player
                                        ; |
                                        ; | If haven't processed this player's score yet
                                        ; | {
4A30: A6 E4       LDA   ,S              ; |    A gets first byte of 5-byte buffer, player index
                                        ; |     this is the 'current player' this pass in loop.
4A32: BD BB 3E    JSR   $BB3E           ; |    GetPlayerScoreIndexAintoU() U points to score
4A35: 10 AE 63    LDY   $0003,S         ; |    Y gets last 2 bytes of 5-byte buffer, cur best
4A38: 27 08       BEQ   $4A42           ; |    If (Y != 0x0000) if we previously found a best
                                        ; |    {
4A3A: BD 88 F5    JSR   $88F5           ; |       CallBankedFunction_Param_WPCAddr()
4A3D: 42 0A 3D                          ; |       $420A,3D  CompareBCDScoreUWithScoreY()
                                        ; |        C-clear if score U is greater than score Y
                                        ; |        Comparing current player score with cur best
4A40: 25 06       BCS   $4A48           ; |       if (c-set) goto $4A48  C-set when current
                                        ; |        player score is not better than the cur best
                                        ; |        score, so skip current player's score.
                                        ; |    }
                                        ; |    ---------------------------------------------
                                        ; |    Getting here means we have a new current best
                                        ; |    Now update the 5-byte buffer
                                        ; |    ---------------------------------------------
4A42: EF 63       STU   $0003,S         ; |    Store address of cur high score in 4th and 5th b
4A44: A7 61       STA   $0001,S         ; |    Store current player index into 2nd byte of 5
4A46: E7 62       STB   $0002,S         ; |    Store current bitmap bit into 3rd byte of 5
                                        ; | }
                                        ; |
4A48: 58          ASLB                  ; | Shift B bitmap left
4A49: 6C E4       INC   ,S              ; | Increment player index number
4A4B: BD B1 9F    JSR   $B19F           ; | GetCurrentGameNumberOfPlayersIntoA()
4A4E: A1 E4       CMPA  ,S              ; | Check if we have more players scores to check
4A50: 24 D8       BCC   $4A2A           ;-/ If so, keep looping
                                        ;
4A52: EE 6B       LDU   $000B,S         ; U gets address of caller's 3-byte buffer
```

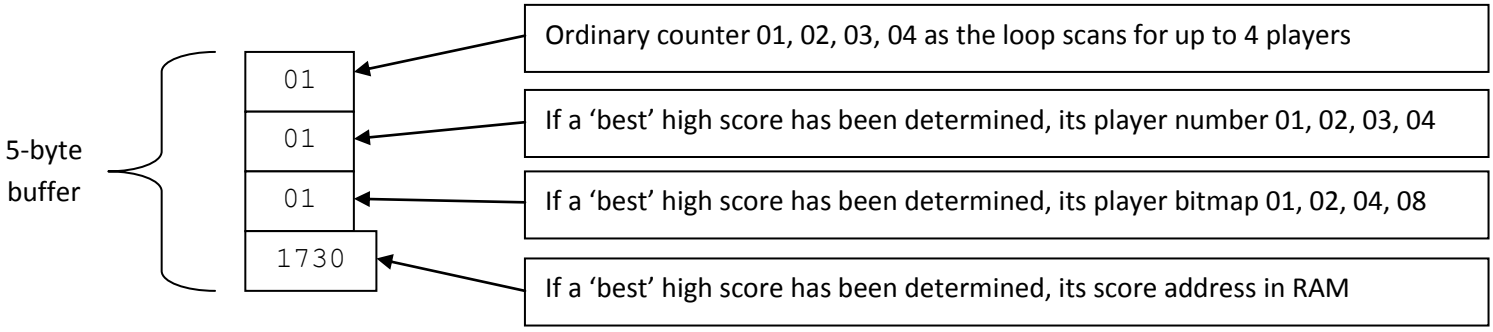
```
4A54: A6 61       LDA   $0001,S          ; Get player index of highest score into A
4A56: 27 2F       BEQ   $4A87            ; If no player index found (0x00) skip to the end
4A58: A7 41       STA   $0001,U          ; Store player index of highest score into 2nd byte of 5
4A5A: E6 62       LDB   $0002,S          ; Get bitmap for high score into B
4A5C: EA C4       ORB   ,U               ; OR it with first byte of callers 3-byte buffer
4A5E: E7 C4       STB   ,U               ; Store result in first byte of callers 3-byte buffer
                                         ;
4A60: 8E 43 45    LDX   #$4345           ; X gets 0x4345 <--- addr in bank $3D for GC HSTD data
4A63: BD 88 F5    JSR   $88F5            ; CallBankedFunction_Param_WPCAddr()
4A66: 43 C0 3D                           ; $43C0,3D, VerifyHstdTableXAllowed()
                                         ;  c-set if not allowed.  c and $04FB cleared if allowed
4A69: 25 0F       BCS   $4A7A            ; C-set? not allowed so skip over the following
                                         ;
                                         ; if (Grand Champion HSTD allowed)
                                         ; {
4A6B: BD 88 F5    JSR   $88F5            ;    CallBankedFunction_Param_WPCAddr()
4A6E: 41 E3 3D                           ;    $41E3,3D, ComparePlayerIndexAScoreWithHSTDTableX()
                                         ;     C-clear if player A score is a new HSTD entry.
                                         ;     A has winning HSTD table index (for GC it is 01)
4A71: 25 07       BCS   $4A7A            ;    C-set? skip the following
                                         ;
                                         ;    if (player score greater than grand champion HSTD)
                                         ;    {
4A73: 4F          CLRA                   ;       A = 0x00 (Grand Champion)
                                         ;
                                         ;saveEntryAndExit:
4A74: A7 42       STA   $0002,U          ;       Store A into 3rd byte of caller's 3-byte buffer.
                                         ;        00 (Grand Champion),
                                         ;        01 - 04 for top marksmen 1 through 4
4A76: 1C FE       ANDCC #$00FE           ;       Clear C-bit
4A78: 20 0F       BRA   $4A89            ;       goto done
                                         ;    }
                                         ; }
                                         ;
4A7A: A6 61       LDA   $0001,S          ; Get player index of highest score into A
4A7C: 8E 43 3E    LDX   #$433E           ; X gets 0x433E <--- addr in bank $3D for Top4 HSTD data
4A7F: BD 88 F5    JSR   $88F5            ; CallBankedFunction_Param_WPCAddr()
4A82: 41 E3 3D                           ; $41E3,3D, ComparePlayerIndexAScoreWithHSTDTableX()
                                         ;  C-clear if hit player A score is a new HSTD entry.
                                         ;  A has winning index 01, 02, 03, 04 HSTD entry number
4A85: 24 ED       BCC   $4A74            ; C-clear? goto saveEntryAndExit, Save the high score
                                         ;  table index into 3rd byte of 3-byte buffer and done
                                         ;
4A87: 1A 01       ORCC  #$0001           ; Set c-bit when no high score found
                                         ;
4A89: 32 65       LEAS  $0005,S          ; Restore stack back to normal, done with 5-byte array
4A8B: 35 F6       PULS  A,B,X,Y,U,PC     ; done
                                         ;
-----------------------------------------;-------------------------------------------------
```

The function, above, samples each player's score, discovering the best top score among all players (excluding any players who have previously been discovered as a best top score) and then updates the 3-byte buffer that the calling function provided with information about the top score.

If a top score is found, the function returns with C-bit clear and sets the callers 3-byte buffer in the following way:

- Byte[0] is the bitmask of awarded players is updated with bit set for the discovered score player. The 0x01 bit is set for player 1, 0x02 bit for player 2, 0x04 bit for player 3, 0x08 bit for player 4.
- Byte[1] gets the player index that is discovered, 01, 02, 03, or 04.
- Byte[2] gets the detected high score slot. 00 for GC, 01, 02, 03, 04 for a Top4 entry.

Prior to updating the caller's 3-byte buffer this function, itself, uses a 5-byte buffer of its own that it uses to discover which of the (up to) 4 player's scores represents the best next high score to declare as a new HSTD table entry.

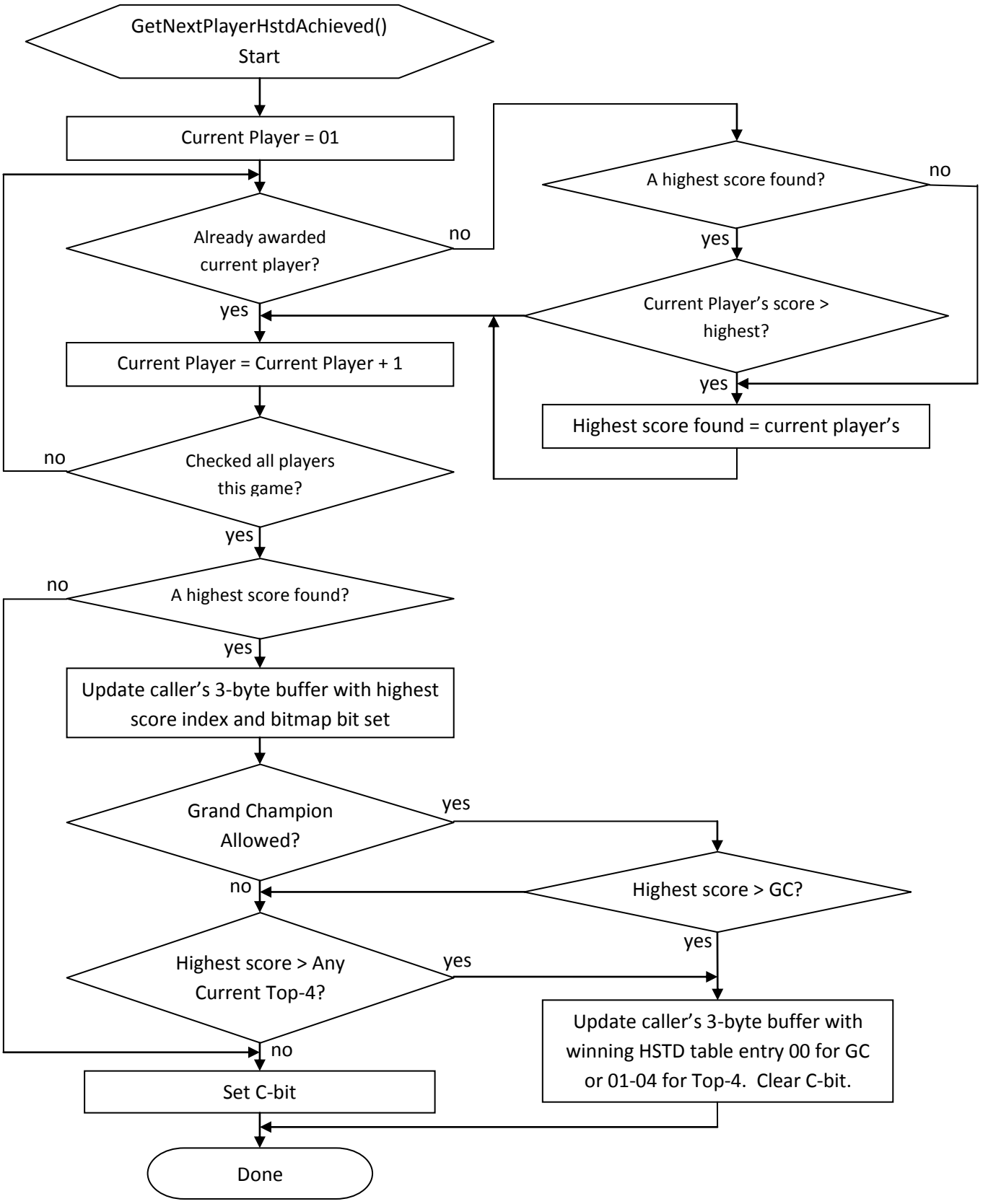


The best score among all players is tracked with this 5-byte buffer. For single player games, the function only needs to loop through a single time, using the player's score as the 'best' score. For multi player games, the function will loop through all players scores and determine the best score among all players (excluding any that have previously been awarded from a prior call through this function as mentioned above).

Once the 'best' score among all players is determined, the caller's 3-byte buffer is updated with the player index and player's bitmap bit set, indicating this player should not be included in a subsequent call through this function (if the current call through the function returns C-bit clear then a follow-up call will take place to look for any other HSTD winners). After updating these 2 bytes of the caller's 3-byte buffer, the current highest score is then checked for possible inclusion into the Grand Champion HSTD table. If the score does not warrant entry in the Grand Champion HSTD table, then it is checked for possible inclusion into the top-4 Top Marksmen HSTD table.

If the score is worthy of inclusion into the Grand Champion or Top Marksmen table then the function updates the caller's 3-byte buffer by setting the 3rd byte to indicate if the top score should be added as a new Grand Champion or a new top-4 score. After this, the function returns C-bit clear to inform the calling function that a new high score has been found and needs to be added into the HSTD table according to the data that has been populated into the caller's 3-byte buffer (for winning player number and the HSTD table/entry into which it should be inserted).

A flowchart is shown below to depict the logic of the GetNextPlayerHstdAchieved() function and to help illustrate its logic.


GetNextPlayerHstdAchieved()
Start
Current Player = 01
Already awarded current player?
no
yes
Current Player = Current Player + 1
Checked all players this game?
no
yes
A highest score found?
no
yes
Update caller's 3-byte buffer with highest score index and bitmap bit set
Grand Champion Allowed?
yes
no
Highest score > GC?
yes
Highest score > Any Current Top-4?
yes
no
Update caller's 3-byte buffer with winning HSTD table entry 00 for GC or 01-04 for Top-4. Clear C-bit.
Set C-bit
Done
A highest score found?
no
yes
Current Player's score > highest?
yes
Highest score found = current player's

The GetNextPlayerHstdAchieved() function, above, determines HSTD worthiness of each players score, returning player index (1 – 4) and HSTD table entry (0 for GC or 1-4 for top-4). These two pieces of information are then delivered to the HstdCollectPlayerInitials() function, as previously highlighted. The HstdCollectPlayerInitials() function is at $4DC3,24, ROM offset 0x10DC3 and shown below.

There are various helper routines that ensure player initials are valid, and for handling the input of player initial selection and display characteristics during the initials-entry process. Such functions are not necessarily depicted as the primary focus is the insertion of the player's initials and score, as highlighted, into the HSTD tables.

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; HstdCollectPlayerInitials()
                                        ;
4DC3: BD 89 48        JSR   $8948       ; ScheduleFunction()
4DC6: 50 53 24                          ; InitialsCollectionTimer()
4DC9: BD FB AE        JSR   $FBAE       ; ClearDisplayMemory()
4DCC: 7E 4D CF        JMP   $4DCF       ; <nop>
4DCF: 1F 10           TFR   X,D         ; Copy X to D. A has Player Index 1-4, B has HSTD slot.
4DD1: B7 05 90        STA   $0590       ; $0590 gets player index 1-4
4DD4: F7 05 8E        STB   $058E       ; $058E gets HSTD slot, 0=GC 1-4=Top4
4DD7: 86 08           LDA   #$08        ; A = 08, Grand Champion music
4DD9: 7D 05 8E        TST   $058E       ; If $058E is not 00 (Grand Champion)
4DDC: 27 02           BEQ   $4DE0       ; {
4DDE: 86 05           LDA   #$05        ;   A = 05, Top Marksmen music
                                        ; }
4DE0: B7 06 08        STA   $0608       ; Save music byte to $0608
4DE3: BD C0 BC        JSR   $C0BC       ; PlayMusicRegisterA() Play initials-entry music
4DE6: BD 85 46        JSR   $8546       ; DoSoundTableParameterByte()
4DE9: 87                                ;  87 = "You are superior"
4DEA: 86 20           LDA   #$20        ; 20 = ' ' space character
4DEC: B7 05 83        STA   $0583       ; $0583, currently selected character is space ' '
4DEF: B7 05 89        STA   $0589       ; $0589, Initial 1 is a space ' '
4DF2: B7 05 8A        STA   $058A       ; $058A, Initial 2 is a space ' '
4DF5: B7 05 8B        STA   $058B       ; $058B, Initial 3 is a space ' '
4DF8: 7F 05 85        CLR   $0585       ; $0585 is 00
4DFB: 7F 05 87        CLR   $0587       ; $0587 is 00, number of initials entered so far
4DFE: 86 08           LDA   #$08        ; A = 8
4E00: B7 05 8D        STA   $058D       ; $058D gets 8
4E03: 86 01           LDA   #$01        ; A = 1
4E05: B7 05 8F        STA   $058F       ; $058F is 01
4E08: BD 84 8F        JSR   $848F       ; ClearMemoryFlag()
4E0B: E1                                ;  E1, start button or gun trigger held-down flag
4E0C: BD 84 8F        JSR   $848F       ; ClearMemoryFlag()
4E0F: E2                                ;  E2, left or right flippper button held-down flag
4E10: BD 4D 83        JSR   $4D83       ; PrintPlayerNumber()
4E13: BD 4F D7        JSR   $4FD7       ;
4E16: BD E2 74        JSR   $E274       ;
                                        ;
4E19: BD 83 29        JSR   $8329       ;-\ GetSwitchClosedState()  C-clear if switch closed
4E1C: 0B                                ; | SwitchTableEntry0B, 13, Start Button
4E1D: 10 24 00 97     LBCC  $4EB8       ; |
                                        ; |
4E21: BD 83 29        JSR   $8329       ; | GetSwitchClosedState()  C-clear if switch closed
4E24: 1C                                ; |  SwitchTableEntry1C, 34, Grip Trigger
4E25: 10 24 00 8F     LBCC  $4EB8       ; |
                                        ; |
4E29: BD 84 8F        JSR   $848F       ; | ClearMemoryFlag()
4E2C: E1                                ; |  E1, start button or gun trigger held-down flag
                                        ; |
4E2D: BD 83 29        JMP   $8329       ; | GetSwitchClosedState()  C-clear if switch closed
4E30: 0A                                ; |  SwitchTableEntry0A, 12, Left Flipper
4E31: 24 4A           BCC   $4E7D       ; |
                                        ; |
4E33: BD 83 29        JSR   $8329       ; | GetSwitchClosedState()  C-clear if switch closed
4E36: 09                                ; |  SwitchTableEntry09, 11, Right Flipper
```

```
4E37: 24 54          BCC   $4E8D       ; |
                                       ; |
                                       ; |----------------------------------------
                                       ; | No switches are closed, do housekeeping
                                       ; |----------------------------------------
4E39: BD 84 8F       JSR   $848F       ; | ClearMemoryFlag()
4E3C: E2                               ; |  E2, left or right flippper button held-down flag
4E3D: 7F 05 8F       CLR   $058F       ; | $058F = 0
4E40: 7C 05 8F       INC   $058F       ; | $058F Increment by 1
4E43: 7A 05 8D       DEC   $058D       ; | If (--$058D == 0)
4E46: 26 2F          BNE   $4E77       ; | {  Reach this every 0.125 seconds, dmd housekeeping
4E48: 86 08          LDA   #$08        ; |    A = 8
4E4A: B7 05 8D       STA   $058D       ; |    reset $058D to 8
4E4D: BD D3 56       JSR   $D356       ; |    display memory related call
4E50: 86 3F          LDA   #$3F        ; |
4E52: B7 05 2D       STA   $052D       ; |    $052D gets 0x3F
4E55: 86 16          LDA   #$16        ; |
4E57: B7 05 2E       STA   $052E       ; |    $052E gets 0x16
4E5A: 86 07          LDA   #$07        ; |
4E5C: B7 05 31       STA   $0531       ; |    $0531 gets 0x07
4E5F: 86 09          LDA   #$09        ; |
4E61: B7 05 32       STA   $0532       ; |    $0532 gets 0x09
4E64: BD EF E1       JSR   $EFE1       ; |
4E67: B6 05 85       LDA   $0585       ; |
4E6A: 88 FF          EORA  #$FF        ; |
4E6C: B7 05 85       STA   $0585       ; |    $0585 gets its bits flipped
4E6F: 27 03          BEQ   $4E74       ; |
4E71: BD 4D 9A       JSR   $4D9A       ; |    EnteredInitialsValidate()
4E74: BD D3 4C       JSR   $D34C       ; |    display related call
                                       ; | }
4E77: BD 83 46       JSR   $8346       ; | Sleep()
4E7A: 01                               ; |  15.625mS
4E7B: 20 9C          BRA   $4E19       ;-/
                                       ;
                                       ;----------------------------
                                       ; Left Flipper Button
                                       ;----------------------------
4E7D: 7A 05 8F       DEC   $058F       ; Decrement flipper-button debounce time count
4E80: 26 F5          BNE   $4E77       ; If debounce count not zero then skip button entry
4E82: B6 05 83       LDA   $0583       ; A gets currently selected characer from $0583
4E85: BD 4F 99       JSR   $4F99       ; InitialsEntryLeftFlipperButtonNewCharIntoA()
4E88: B7 05 83       STA   $0583       ; Store newly selected character back into $0583
4E8B: 20 0E          BRA   $4E9B       ; goto common flipper button handler, below
                                       ;
                                       ;----------------------------
                                       ; Right Flipper Button
                                       ;----------------------------
4E8D: 7A 05 8F       DEC   $058F       ; Decrement flipper-button debounce time count
4E90: 26 E5          BNE   $4E77       ; If debounce count not zero then skip button entry
4E92: B6 05 83       LDA   $0583       ; A gets currently selected characer from $0583
4E95: BD 4F B8       JSR   $4FB8       ; InitialsEntryRightFlipperButtonNewCharIntoA()
4E98: B7 05 83       STA   $0583       ; Store newly selected character back into $0583
                                       ;
                                       ;-----------------------------------------------
                                       ; Common code for left and right flipper buttons
                                       ; Common code for start and gun trigger handling
                                       ;-----------------------------------------------
4E9B: BD 4F 59       JSR   $4F59       ;
4E9E: BD 84 AD       JSR   $84AD       ; GetMemoryFlag() // C-bit clear when flag set
4EA1: E2                     '         ;  E2, left or right flippper button held-down flag
4EA2: 24 0C          BCC   $4EB0       ;
                                       ;-----------------------------------------------------
                                       ; Flipper button now being declared as being held down
                                       ;-----------------------------------------------------
4EA4: BD 84 80       JSR   $8480       ; SetMemoryFlag()
4EA7: E2                               ;  E2, left or right flippper button held-down flag
4EA8: 86 20          LDA   #$20        ; Long debounce count at first 0x20
4EAA: B7 05 8F       STA   $058F       ; $058F gets long debounce count
4EAD: 16 FF 69       LBRA  $4E19       ; Go up to switch scanning
                                       ;
                                       ;-----------------------------------------------------
```

```
                                        ; Flipper button already being declared as held down
                                        ;------------------------------------------------------
4EB0: 86 08           LDA   #$08        ; Short Debounce count 8
4EB2: B7 05 8F        STA   $058F       ; $058F gets debounce count
4EB5: 16 FF 61        LBRA  $4E19       ; Go up to switch scanning
                                        ;
                                        ;---------------------------
                                        ; Start Button & Gun Trigger
                                        ;---------------------------
4EB8: BD 84 AD        JSR   $84AD       ; GetMemoryFlag() // C-bit clear when flag set
4EBB: E1                                ; E1 is start/gun trigger button hold-down indicator
4EBC: 10 24 FF 6D     LBCC  $4E2D       ; If hold-down, go to switch scanning at flipper buttons
                                        ;
                                        ;------------------------------------------------------
                                        ; Start/Gun Trigger now being declared as held down
                                        ;------------------------------------------------------
4EC0: BD 84 80        JSR   $8480       ; SetMemoryFlag()
4EC3: E1                                ; Set the E1 start/gun trigger hold-down indicator
4EC4: F6 05 87        LDB   $0587       ; B gets number of entered initials so far
4EC7: 5C              INCB              ; Locally increment entered initials count in B
4EC8: C1 03           CMPB  #$03        ;
4ECA: 22 CF           BHI   $4E9B       ; If more than 3 initials goto $4E9B, unexpected case
4ECC: B6 05 83        LDA   $0583       ; A gets currently selected character
4ECF: 81 7F           CMPA  #$7F        ; If (Current Character == backspace indicator)
4ED1: 26 19           BNE   $4EEC       ; {
4ED3: C1 01           CMPB  #$01        ;   if (number of entered initials = 1)
4ED5: 27 C4           BEQ   $4E9B       ;     goto $4E9B, unexpected
4ED7: 86 20           LDA   #$20        ;   A gets 0x20, space ' ' character
4ED9: 8E 05 89        LDX   #$0589      ;   X gets 0x0589 (address of first of 3 initials)
4EDC: 5A              DECB              ;   Make B 0-based number
4EDD: 3A              ABX               ;   Increment X so it points to current initial of the 3
4EDE: A7 84           STA   ,X          ;   Now push the space character into current initial
                                        ;   Since backspace is entered the above just wrote
                                        ;   space into the entered position before moving to
                                        ;   position that player wants to go back to and change.
4EE0: 8E 05 89        LDX   #$0589      ;   Reset X to 0x0589 (address of first of 3 initials)
4EE3: 5A              DECB              ;   Decrement to previous initial index (at 1 or 2 now)
4EE4: 3A              ABX               ;   Increment X so it points to previous initial
4EE5: A7 84           STA   ,X          ;   Now push the space character into previous initial
                                        ;   The above overwrites previous position with a space
4EE7: F7 05 87        STB   $0587       ;   Update number of entered initials into $0587
4EEA: 20 AF           BRA   $4E9B       ; }
                                        ;
4EEC: F7 05 87        STB   $0587       ; Update number of entered initials into $0587
4EEF: 8E 05 89        LDX   #$0589      ; X gets address of first of 3 initials
4EF2: 5A              DECB              ; Make current initials count 0-based
4EF3: 3A              ABX               ; Increment X so it points to new entered position
4EF4: B6 05 83        LDA   $0583       ; A gets currently selected character
4EF7: A7 84           STA   ,X          ; Now update the initial in memory w/selected character
4EF9: BD 85 46        JSR   $8546       ; DoSoundTableParameterByte()
4EFC: 61                                ; 0x61 = Smash
4EFD: C1 02           CMPB  #$02        ; Check the 0-based initials count with 2
4EFF: 25 9A           BCS   $4E9B       ; If less than 2 go up and keep collecting initials
4F01: BD 9E 0A        JSR   $9E0A       ; CancelSelf() Cancels timer and self from scheduler.
                                        ; This instance continues to run.
                                        ;
                                        ;------------------------------------------------------
                                        ; All three initials have been entered.
                                        ; Or, branch here when Initials input times out
                                        ;------------------------------------------------------
                                        ;
4F04: BD 4F 59        JSR   $4F59       ;
4F07: BD 83 46        JSR   $8346       ; Sleep()
4F0A: 10                                ; 1/4 second
4F0B: 7D 05 8E        TST   $058E       ; if ($058E is 00) (Grand Champion)
4F0E: 26 2B           BNE   $4F3B       ; {
4F10: C6 01           LDB   #$01        ;   B = 1  (fetching row 1 of Grand Champion table)
4F12: 8E 43 45        LDX   #$4345      ;   X = $4345, Grand Champion data table pointer in $3D
4F15: BD 88 F5        JSR   $88F5       ;   CallBankedFunction_Param_WPCAddr()
4F18: 41 48 3D                          ;   GetHstdTableEntryXIndexB into InitialsUScoreY()
4F1B: 1E 23           EXG   Y,U         ;   Current GC Initials into Y, Score into U
```

```
                                        ;
4F1D: 8E 43 3E        LDX   #$433E      ;   X = $433E, Top4 data table pointer in $3D
4F20: BD 88 F5        JSR   $88F5       ;   CallBankedFunction_Param_WPCAddr()
4F23: 42 1D 3D                          ;   PushHstdTableEntryUpdate() Pushes cur GC into top4
                                        ;
4F26: B6 05 90        LDA   $0590       ;   A gets the cur HSTD player index (1-4) from $0590
4F29: BD BB 3E        JSR   $BB3E       ;   GetPlayerScoreIndexAintoU()  U* = player's score
4F2C: 10 8E 05 89     LDY   #$0589      ;   Y gets 0x0589, pointer to entered initials
4F30: 8E 43 45        LDX   #$4345      ;   X = $4345, Grand Champion data table pointer in $3D
4F33: BD 88 F5        JSR   $88F5       ;   CallBankedFunction Param WPCAddr()
4F36: 42 1D 3D                          ;   PushHstdTableEntryUpdate() Pushes player score to GC
4F39: 20 13           BRA   $4F4E       ; }
                                        ; else
                                        ; {
4F3B: B6 05 90        LDA   $0590       ;   A gets the cur HSTD player index (1-4) from $0590
4F3E: BD BB 3E        JSR   $BB3E       ;   GetPlayerScoreIndexAintoU()  U* = player's score
4F41: 10 8E 05 89     LDY   #$0589      ;   Y gets 0x0589, pointer to entered initials
4F45: 8E 43 3E        LDX   #$433E      ;   X = $433E, Top4 data table pointer in $3D
4F48: BD 88 F5        JSR   $88F5       ;   CallBankedFunction_Param_WPCAddr()
4F4B: 42 1D 3D                          ;   PushHstdTableEntryUpdate() Pushes score to top4
                                        ; }
                                        ;
4F4E: 86 F2           LDA   #$F2        ; A = 0xF2
4F50: BD CA 94        JSR   $CA94       ; sets up time that caller uses to know we are busy
4F53: BD 83 46        JSR   $8346       ;-\ Sleep()
4F56: F0                                ; | 3.75 seconds
4F57: 20 FA           BRA   $4F53       ;-/
                                        ;
----------------------------------------;------------------------------------------------------
```

As shown above, the entry of score into the HSTD table begins at $4F04,24. It is worth mentioning, also, that this function, at $4DC3,24, had started off by launching a background timer that runs for about 88 seconds, as shown below:

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; InitialsCollectionTimer()
                                        ;
5053: BD 86 79        JSR   $8679       ; SleepPlusURegisterSave()
5056: 15 F9                             ; 87.890625 seconds
5058: BD 9E 0A        JSR   $9E0A       ; CancelSelf() Cancels initials entry and self
                                        ; This instance continues to run.
505B: 16 FE A6        LBRA  $4F04       ; Now go to the HSTD table update
                                        ;
----------------------------------------;------------------------------------------------------
```

At timer expiration it then jumps to this same address $4F04,24 whereupon whatever initials that have been entered up to this point, are pushed into the HSTD table. By default, all three characters are space characters, so if no player activity takes place, then the timeout results in 3 space characters being entered as the player's high score. If player entered 1 or 2 characters then they are used as the player's initials when the timer expires.

As shown above, when the HSTD table is updated there are several key pieces of logic:

- If the original determination was that player is the new Grand Champion:
  - The *existing GC score* and initials are inserted into the Top Marksmen table.
  - The current player's score is then inserted into the Grand Champion table.
- If the original determination was that player is a Top Marksmen:
  - The current player's score is inserted into the Top Marksmen table.

All of these logics utilize helper function located in bank $3D and will be shown below. These helper functions perform validation logic to ensure the HSTD table is valid such as:

- When inserting a score into the GC table, the new score must be greater than the current GC score.
- When inserting a score into the Top-4 table, the new score must be greater than at least one of the existing top-4 scores.
- When inserting a score into any table, the score is ensured to be valid (digits 0-9).
- When inserting a score into the top-4 table, the ordering of the table scores are ensured to be correct numerical order.

The entire section of bank $3D that has such HSTD helper functions is pasted below. This starts at $4148,3D, ROM offset 0x74148. Not all portions of this have been annotated. These are the helper functions cited in the code sections depicted above. Presented here for readers who are interested in the full suite of HSTD related code.

```
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; GetHstdTableEntryXIndexB_into_InitialsUScoreY()
4148: 34 16       PSHS  X,B,A          ;
414A: 5D          TSTB                 ;
414B: 27 04       BEQ   $4151          ; If B == 0x00 then ErrorHandler()
414D: E1 03       CMPB  $0003,X        ; Compare the B index w/number of HSTD entries, table X
414F: 23 06       BLS   $4157          ; If winning index B <= HSTD entries then we're good.
                                       ;
4151: BD 82 B6    JSR   $82B6          ; ErrorHandler()
4154: 5E                               ;
4155: C6 01       LDB   #$01           ;
                                       ;
4157: 5A          DECB                 ; Decrement B index, making it zero based
4158: 86 08       LDA   #$08           ; A has 8 bytes per HSTD entry
415A: 3D          MUL                  ; D now has index offset for current HSTD entry
415B: AE 84       LDX   ,X             ; X now is at start of HSTD data
415D: 30 8B       LEAX  D,X            ; Bump X into the correct 8-byte row as indicated in B
415F: 31 03       LEAY  $0003,X        ; Y now points to the corresponding score for HSTD entry
4161: 33 84       LEAU  ,X             ; U now points to the corresponding initials HSTD entry
4163: 35 96       PULS  A,B,X,PC       ;
                                       ;
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; Validates the high score table entry pointed to by X
4165: 8D 08       BSR   $416F          ; ValidateHstdTableX()  C-set if problem
4167: 24 05       BCC   $416E          ;
4169: BD 42 8B    JSR   $428B          ;
416C: 8D 01       BSR   $416F          ;
416E: 39          RTS                  ;
                                       ;
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; ValidateHstdTableX()  C-set if problem
                                       ;
416F: 34 76       PSHS  U,Y,X,B,A      ;
4171: A6 02       LDA   $0002,X        ;
4173: BD 92 3B    JSR   $923B          ; VerifyRAMBlockChecksumIndexA()
4176: 25 24       BCS   $419C          ;
4178: E6 03       LDB   $0003,X        ;
417A: AE 84       LDX   ,X             ;
417C: CE 00 00    LDU   #$0000         ;
                                       ;
417F: 8D 1D       BSR   $419E          ;-\ HstdTableEntryXValidate()
4181: 25 19       BCS   $419C          ; |
4183: 31 C4       LEAY  ,U             ; |
4185: 33 03       LEAU  $0003,X        ; | Advance past the 3 initials
```

```
4187: 10 8C 00 00 CMPY  #$0000          ; |
418B: 27 08       BEQ   $4195           ; |
418D: 8D 7B       BSR   $420A           ; | CompareBCDScoreUWithScoreY()  C-clear if score at
                                        ; |  U is greater than score at Y
418F: 25 04       BCS   $4195           ; |
4191: 1A 01       ORCC  #$0001          ; |
4193: 20 07       BRA   $419C           ; |
4195: 30 08       LEAX  $0008,X         ; |
4197: 5A          DECB                  ; |
4198: 26 E5       BNE   $417F           ;-/
                                        ;
419A: 1C FE       ANDCC #$00FE          ;
419C: 35 F6       PULS  A,B,X,Y,U,PC    ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; HstdTableEntryXValidate()
                                        ;
                                        ;
419E: 34 26       PSHS  Y,B,A           ;
41A0: 31 84       LEAY  ,X              ; Y gets address out of X
41A2: C6 03       LDB   #$03            ;
                                        ;
41A4: A6 A0       LDA   ,Y+             ;-\ A gets next character of HSTD table initials
41A6: 8D 28       BSR   $41D0           ; | HstdTableInitialCharacterValidate()
41A8: 25 07       BCS   $41B1           ; |
41AA: 5A          DECB                  ; |
41AB: 26 F7       BNE   $41A4           ;-/
                                        ;
41AD: 31 03       LEAY  $0003,X         ; Advance past the 3-characters
41AF: 8D 02       BSR   $41B3           ; HighScoreYValidate()
41B1: 35 A6       PULS  A,B,Y,PC        ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; HighScoreYValidate()
                                        ;
41B3: 34 26       PSHS  Y,B,A           ;
41B5: C6 05       LDB   #$05            ;
                                        ;
41B7: A6 A4       LDA   ,Y              ;-\
41B9: 84 0F       ANDA  #$0F            ; |
41BB: 81 09       CMPA  #$09            ; |
41BD: 23 04       BLS   $41C3           ; |
41BF: 1A 01       ORCC  #$0001          ; |
41C1: 20 0B       BRA   $41CE           ; |
41C3: A6 A0       LDA   ,Y+             ; |
41C5: 81 99       CMPA  #$99            ; |
41C7: 22 F6       BHI   $41BF           ; |
41C9: 5A          DECB                  ; |
41CA: 26 EB       BNE   $41B7           ;-/
                                        ;
41CC: 1C FE       ANDCC #$00FE          ;
41CE: 35 A6       PULS  A,B,Y,PC        ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; HstdTableInitialCharacterValidate()
                                        ;
41D0: 81 20       CMPA  #$20            ; Check for ' '
41D2: 27 0E       BEQ   $41E2           ; If found, done
41D4: 81 41       CMPA  #$41            ; Check for 'A'
41D6: 25 08       BCS   $41E0           ;
41D8: 81 5A       CMPA  #$5A            ; Check for 'Z'
41DA: 22 04       BHI   $41E0           ;
41DC: 1C FE       ANDCC #$00FE          ;
41DE: 20 02       BRA   $41E2           ;
41E0: 1A 01       ORCC  #$0001          ;
41E2: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------
```

```
                                        ;
                                        ; ComparePlayerIndexAScoreWithHSTDTableX()
                                        ;  Returns C-clear if hit.
                                        ;  A has winning index 1,2,3,4. (1 when GC)
                                        ;
41E3: 34 40       PSHS  U               ;
41E5: BD BB 3E    JSR   $BB3E           ; GetPlayerScoreIndexAintoU()
41E8: 33 C4       LEAU  ,U              ;
41EA: 8D 02       BSR   $41EE           ; CompareBCDScoreUWithHSTDTableX() C-clear if score
                                        ;  at U is found to be greater than a HSTD table entry.
                                        ;  A has winning index 1,2,3,4. (1 when GC)
41EC: 35 C0       PULS  U,PC            ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; CompareBCDScoreUWithHSTDTableX()
                                        ;  Return C-clear if score U is found to be greater than
                                        ;  a HSTD table entry.
                                        ;  A has winning index 1,2,3,4. (1 when GC)
                                        ;
41EE: 34 74       PSHS  U,Y,X,B         ;
41F0: BD 41 65    JSR   $4165           ; Validates the high score table entry pointed to by X
41F3: 86 01       LDA   #$01            ;
41F5: E6 03       LDB   $0003,X         ; B gets number of high score entries for table entry X
41F7: AE 84       LDX   ,X              ; X gets starting address of high score entries
                                        ;
41F9: 31 03       LEAY  $0003,X         ;-\ Load Y with the score portion of the X table data
41FB: BD 42 0A    JSR   $420A           ; | CompareBCDScoreUWithScoreY()  C-clear if score at U
                                        ; |  is greater than score at Y
41FE: 24 08       BCC   $4208           ; | If C-clear then player's score is greater than the
                                        ; |  current HSTD entry, done
4200: 30 08       LEAX  $0008,X         ; | Advance X to next high score
4202: 4C          INCA                  ; |
4203: 5A          DECB                  ; |
4204: 26 F3       BNE   $41F9           ;-/
                                        ;
4206: 1A 01       ORCC  #$0001          ;
4208: 35 F4       PULS  B,X,Y,U,PC      ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; CompareBCDScoreUWithScoreY()
                                        ;  Returns C-clear if score U is greater than score Y
                                        ;
420A: 34 66       PSHS  U,Y,B,A         ;
420C: C6 05       LDB   #$05            ; 5 bytes per score
                                        ;
420E: A6 C0       LDA   ,U+             ;-\ A gets player score byte
4210: A1 A0       CMPA  ,Y+             ; | Compare player score byte to HSTD score byte
4212: 22 07       BHI   $421B           ; | If player score byte is greater than HSTD score byte
                                        ; |  then we are done, clear C-bit
4214: 25 03       BCS   $4219           ; | If player score byte is less than the HSTD score
                                        ; |  byte then C is set we are done, set C-bit
4216: 5A          DECB                  ; | If player score byte is equal to HSTD score byte
                                        ; | then decrement byte counter
4217: 26 F5       BNE   $420E           ;-/ Keep checking next byte
                                        ;
4219: 1A 01       ORCC  #$0001          ; Return C-bit set to report player score < HSTD score
421B: 35 E6       PULS  A,B,Y,U,PC      ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; PushHstdTableEntryUpdate()
                                        ;
                                        ; Called with U pointing to HSTD entry score
                                        ;  (player's high score)
                                        ; Called with Y pointing to HSTD entry initials
                                        ;  (player's entered initials)
                                        ; Called with X pointing to HSTD table data in bank $3D
                                        ;
421D: 34 36       PSHS  Y,X,B,A         ;
```

```
421F: 32 7E       LEAS  $FFFE,S         ; Make room for 2 bytes on the stack to store addresss
4221: BD 41 65    JSR   $4165           ; Validates the high score table entry pointed to by X
4224: 25 17       BCS   $423D           ; If table invalid, C-bit is set, return
                                        ;
4226: 31 C4       LEAY  ,U              ; Y points to HSTD entry score (player's high score)
4228: BD 41 B3    JSR   $41B3           ; HighScoreYValidate()
422B: 25 4D       BCS   $427A           ; If score invalid, C-bit is set, return
                                        ;
422D: E6 03       LDB   $0003,X         ; B gets number of HSTD entrys in table X
422F: AE 84       LDX   ,X              ; X points to the top of HSTD table data initials
                                        ;
4231: 31 03       LEAY  $0003,X         ;-\ Advance past 3 initials to the score. Y now points
                                        ; |  to the HSTD table data score
                                        ; | U points to player's high score.
                                        ; | Y points to next HSTD Table score
4233: BD 42 0A    JSR   $420A           ; | CompareBCDScoreUWithScoreY()  C-clear if score at U
                                        ; |  is greater than score at Y
                                        ; |
4236: 24 09       BCC   $4241           ; | If HSTD entry score at U is less than or equal to
                                        ; |  current table entry score
                                        ; |
4238: 30 08       LEAX  $0008,X         ; | Advance X to next row in HSTD table, 8 bytes per
                                        ; |  table entry initials/score
423A: 5A          DECB                  ; | Compare player's score to next HSTD table score
423B: 26 F4       BNE   $4231           ;-/
                                        ;
423D: 1A 01       ORCC  #$0001          ; Error, all HSTD table scores are greater than player's
                                        ;  score
423F: 20 39       BRA   $427A           ; Set c-bit and return
                                        ;
                                        ; Jump here when we discovered player's score is greater
                                        ;  than HSTD entry score
                                        ;
                                        ;-------------------------------------------------------
                                        ; The following will make a hole in HSTD table so a new
                                        ; score can be inserted
                                        ;-------------------------------------------------------
4241: AF E4       STX   ,S              ; Save HSTD table entry, score pointer, where we will
                                        ;  overwrite, temporarily onto the stack
4243: AE 64       LDX   $0004,S         ; X=original value from stack, start of HSTD table data
4245: 8D 37       BSR   $427E           ; LoadXWithHstdTableXEndPointer()
4247: BD 92 23    JSR   $9223           ; UnlockRAM()
424A: 86 08       LDA   #$08            ; A gets 0x08, bytes per HSTD table entry,
                                        ;  3 initials 5-bytes of score
424C: 31 18       LEAY  $FFF8,X         ; Y points to tail HSTD table record, at its initials
                                        ;
424E: 30 38       LEAX  $FFF8,Y         ;-\ X gets previous HSTD table record
4250: 10 AC E4    CMPY  ,S              ; | Compare addr of HSTD score we want to update with
                                        ; |  addr of tail HSTD entry
4253: 23 07       BLS   $425C           ; | if (addr of current tail HSTD entry <= addr of HSTD
                                        ; |  entry to update)
                                        ; | {
4255: BD A6 B9    JSR   $A6B9           ; |    MemcpyXtoYbytecountA()
4258: 31 84       LEAY  ,X              ; |    Y gets next previous tail HSTD entry
                                        ; | }
425A: 20 F2       BRA   $424E           ;-/ loop up to check next tail HSTD entry
                                        ;
425C: 30 C4       LEAX  ,U              ; X now points to player's high score in ram
425E: 10 AE E4    LDY   ,S              ; Y now points to the HSTD table entry that will be
                                        ;  updated (initials)
4261: 31 23       LEAY  $0003,Y         ; Y now points to the HSTD table entry that will be
                                        ;  updated (score)
4263: 86 05       LDA   #$05            ; A gets 0x05, 5 bytes of score to copy
4265: BD A6 B9    JSR   $A6B9           ; MemcpyXtoYbytecountA()
4268: 31 3D       LEAY  $FFFD,Y         ; Y now points to the HSTD table entry that will be
                                        ;  updated (initials)
426A: AE 66       LDX   $0006,S         ; X now points to HSTD entry initials (player's entered
                                        ;  initials)
426C: 86 03       LDA   #$03            ; A gets 0x05, 3 bytes of initials to copy
426E: BD A6 B9    JSR   $A6B9           ; MemcpyXtoYbytecountA()
4271: AE 64       LDX   $0004,S         ; X now points to HSTD table data in bank $3D
```

```
4273: A6 02       LDA   $0002,X          ; A gets the HSTD table index value needed in the
                                         ;  checksum update, next
4275: BD 92 65    JSR   $9265            ; ComputeAndWriteChecksumAndLockRAM()
4278: 1C FE       ANDCC #$00FE           ; Clear C-bit
427A: 32 62       LEAS  $0002,S          ; Restore stack pointer
427C: 35 B6       PULS  A,B,X,Y,PC       ;
                                         ;
----------------------------------------;-------------------------------------------------
                                         ;
                                         ; LoadXWithHstdTableXEndPointer()
427E: 34 06       PSHS  B,A              ;
4280: E6 03       LDB   $0003,X          ; B gets number of HSTD table rows
4282: 86 08       LDA   #$08             ; A gets 8, bytes per HSTD table
4284: 3D          MUL                    ; D has total number of bytes in HSTD table
4285: AE 84       LDX   ,X               ; X points to start of HSTD table
4287: 30 8B       LEAX  D,X              ; X points to end of HSTD table
4289: 35 86       PULS  A,B,PC           ;
                                         ;
----------------------------------------;-------------------------------------------------
                                         ;
                                         ; ResetHSTDTableXToFactoryDefault()
                                         ;
428B: 34 76       PSHS  U,Y,X,B,A        ;
428D: 32 78       LEAS  $FFF8,S          ; Make room for 8 bytes on the stack
428F: 33 E4       LEAU  ,S               ;
4291: A6 06       LDA   $0006,X          ; A = Adjustment index byte from last byte of table data
4293: E6 03       LDB   $0003,X          ; Get number of table entries into B
4295: C1 04       CMPB  #$04             ; Must be less or equal to 4
4297: 23 04       BLS   $429D            ;
4299: BD 82 98    JSR   $8298            ; Error handler
429C: 62                                 ;
429D: 8D 5E       BSR   $42FD            ; LoadBackupHstdScoreIntoX()
429F: 33 E4       LEAU  ,S               ;
42A1: 10 AE 84    LDY   ,X               ;
42A4: A6 03       LDA   $0003,X          ;
42A6: AE 04       LDX   $0004,X          ;
42A8: BD 90 C5    JSR   $90C5            ;
42AB: BD 92 23    JSR   $9223            ; UnlockRAM()
                                         ;
42AE: 8D 1F       BSR   $42CF            ;-\
42B0: 8D 28       BSR   $42DA            ; |
42B2: 30 03       LEAX  $0003,X          ; |
42B4: 31 28       LEAY  $0008,Y          ; |
42B6: 33 42       LEAU  $0002,U          ; |
42B8: 4A          DECA                   ; |
42B9: 26 F3       BNE   $42AE            ;-/
                                         ;
42BB: AE 6A       LDX   $000A,S          ;
42BD: A6 02       LDA   $0002,X          ;
42BF: BD 92 65    JSR   $9265            ; ComputeAndWriteChecksumAndLockRAM()
42C2: BD 41 6F    JSR   $416F            ;
42C5: 24 04       BCC   $42CB            ;
42C7: BD 82 B6    JSR   $82B6            ; ErrorHandler()
42CA: 5D                                 ;
42CB: 32 68       LEAS  $0008,S          ; Fix stack pointer, release the 8 bytes on stack
42CD: 35 F6       PULS  A,B,X,Y,U,PC     ;
                                         ;
----------------------------------------;-------------------------------------------------
                                         ;
42CF: 34 22       PSHS  Y,A              ;
42D1: 31 A4       LEAY  ,Y               ;
42D3: 86 03       LDA   #$03             ;
42D5: BD 91 39    JSR   $9139            ;
42D8: 35 A2       PULS  A,Y,PC           ;
                                         ;
----------------------------------------;-------------------------------------------------
                                         ;
42DA: 34 36       PSHS  Y,X,B,A          ;
42DC: 1C FE       ANDCC #$00FE           ;
42DE: BD 86 21    JSR   $8621            ; CallFunctionPointerParameterBytes()
42E1: 80 D9                              ; $80D9 has $63BF,3D
```

```
42E3: 25 16       BCS   $42FB           ;
42E5: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
42E8: 42 53 38                          ;
42EB: 8E 03 B7    LDX   #$03B7          ;
42EE: EC C4       LDD   ,U              ;
42F0: ED 1B       STD   $FFFB,X         ;
42F2: 30 1B       LEAX  $FFFB,X         ;
42F4: 86 05       LDA   #$05            ; 5 bytes of score data
42F6: 31 23       LEAY  $0003,Y         ; Advance Y past 3 bytes of initials to start of 5-bytes
                                        ;  of score
42F8: BD A6 B9    JSR   $A6B9           ; MemcpyXtoYbytecountA()
42FB: 35 B6       PULS  A,B,X,Y,PC      ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; LoadBackupHstdScoreIntoX()
                                        ;
42FD: 34 36       PSHS  Y,X,B,A         ;
42FF: 30 C4       LEAX  ,U              ; X gets U pointer
                                        ;
4301: BD 92 D4    JSR   $92D4           ;-\ Get16BitSettingIntoY() A has adjustment 0xAC or 0xAB
                                        ; |  from HSTD table;
                                        ; |  0xAC, $1B75:$1B76 HstdAdjustment011, Backup HSTD 1
                                        ; |  0xAB, $1B73:$1B74 HstdAdjustment010, Backup Champ'n
                                        ; |
4304: 8D 10       BSR   $4316           ; | ValidateNormalizeBCDScoreBytesY()
                                        ; |  // Fixes Y, any nibble > 9 gets set to 9
4306: 10 AF 81    STY   ,X++            ; | Save 2 bytes from Y into X pointer, advance X
                                        ; |  pointer by 2
4309: 4C          INCA                  ; | Increment A to next adjustment backup score value
                                        ; | (used by top4 hstd table)
430A: 5A          DECB                  ; | Decrement B number of HSTD table entries
                                        ; } (1 for GC, 4 for top4)
430B: 26 F4       BNE   $4301           ;-/
                                        ;
430D: 30 C4       LEAX  ,U              ; X gets U pointer
430F: E6 61       LDB   $0001,S         ; B gets original value of from stack at function start
4311: BD A7 9F    JSR   $A79F           ;
4314: 35 B6       PULS  A,B,X,Y,PC      ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; ValidateNormalizeBCDScoreBytesY()
                                        ;  // Fixes Y, any nibble > 9 gets set to 9
                                        ;
                                        ; Backup scores in config are stored in 16-bit value for
                                        ;  4 nibbles of score
4316: 34 06       PSHS  B,A             ;
4318: 1F 20       TFR   Y,D             ;
431A: 8D 0A       BSR   $4326           ; ValidateNormalizeBCDScoreByteA()
                                        ;  // Fixes A, any nibble > 9 gets set to 9
431C: 1E 89       EXG   A,B             ;
431E: 8D 06       BSR   $4326           ; ValidateNormalizeBCDScoreByteA()
                                        ;  // Fixes A, any nibble > 9 gets set to 9
4320: 1E 89       EXG   A,B             ;
4322: 1F 02       TFR   D,Y             ;
4324: 35 86       PULS  A,B,PC          ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; ValidateNormalizeBCDScoreByteA()
                                        ;  // Fixes A, any nibble > 9 gets set to 9
                                        ;
4326: 34 04       PSHS  B               ;
4328: 1F 89       TFR   A,B             ;
432A: C4 0F       ANDB  #$0F            ;
432C: C1 09       CMPB  #$09            ;
432E: 23 04       BLS   $4334           ;
4330: 84 F0       ANDA  #$F0            ;
4332: 8B 09       ADDA  #$09            ;
4334: 81 99       CMPA  #$99            ;
```

```
4336: 23 04       BLS   $433C           ;
4338: 84 0F       ANDA  #$0F            ;
433A: 8B 90       ADDA  #$90            ;
433C: 35 84       PULS  B,PC            ;
                                        ;
----------------------------------------;------------------------------------------------
                                        ;
                                        ;
433E: 1C 61                             ; Pointer in RAM to where Top Marksman #1 is stored,
                                        ;  starting at initials
4340: 85                                ; RAM Table Index used when checksum is updated
4341: 04                                ; There are 4 scores to check for Top Marksmen table
4342: 81 48                             ; Pointer to default initials for factory reset,
                                        ;  at $8148 is $6432,3B
4344: AC                                ; Adjustment: 0xAC, $1B75:$1B76 HstdAdjustment011,
                                        ;  Backup HSTD 1
                                        ;
4345: 1C 83                             ; Pointer in RAM to where Grand Champion entry is stored
                                        ;  starting at initials
4347: 86                                ; RAM Table Index used when checksum is updated
4348: 01                                ; There is 1 score to check for Grand Champion
4349: 81 4B                             ; Pointer to default initials for factory reset
                                        ;  at $814B is $642F,3B
434B: AB                                ; 0xAB, $1B73:$1B74 HstdAdjustment010, Backup Champion
                                        ;
----------------------------------------;------------------------------------------------
                                        ;
434C: 34 22       PSHS  Y,A
434E: 86 87       LDA   #$87
4350: BD 92 35    JSR   $9235           ; VerifyRAMBlockChecksumIndexA_PreserveAandB()
4353: 24 04       BCC   $4359
4355: 8D 1F       BSR   $4376
4357: 20 1B       BRA   $4374
4359: BD 82 F2    JSR   $82F2
435C: AA 10       ORA   $FFF0,X
435E: BC 1C 8D    CPX   $1C8D
4361: 26 F2       BNE   $4355
4363: 10 8C 00 00 CMPY  #$0000
4367: 27 0B       BEQ   $4374
4369: 10 BE 1C 8F LDY   $1C8F
436D: 26 05       BNE   $4374
436F: 8D 05       BSR   $4376
4371: BD 40 E5    JSR   $40E5
4374: 35 A2       PULS  A,Y,PC ; (PUL? PC=RTS)
                                        ;
----------------------------------------;------------------------------------------------
                                        ;
4376: 34 20       PSHS  Y
4378: BD 82 F2    JSR   $82F2
437B: AA BD 92 23 ORA   [-$FFFF6DDD,Y]
437F: 10 BF 1C 8D STY   $1C8D
4383: 10 BF 1C 8F STY   $1C8F
4387: 8D 2B       BSR   $43B4
4389: 35 A0       PULS  Y,PC ; (PUL? PC=RTS)
                                        ;
----------------------------------------;------------------------------------------------
                                        ;
438B: 34 06       PSHS  B,A
438D: 86 87       LDA   #$87
438F: BD 92 35    JSR   $9235
4392: 24 04       BCC   $4398
4394: 8D E0       BSR   $4376
4396: 20 1A       BRA   $43B2
4398: BD B1 9F    JSR   $B19F
439B: 1F 89       TFR   A,B
439D: 4F          CLRA
439E: 34 06       PSHS  B,A
43A0: FC 1C 8F    LDD   $1C8F
43A3: A3 E1       SUBD  ,S++
43A5: 24 03       BCC   $43AA
43A7: CC 00 00    LDD   #$0000
```

```
43AA: BD 92 23    JSR   $9223
43AD: FD 1C 8F    STD   $1C8F
43B0: 8D 02       BSR   $43B4
43B2: 35 86       PULS  A,B,PC ; (PUL? PC=RTS)
43B4: 34 02       PSHS  A
43B6: BD 92 23    JSR   $9223
43B9: 86 87       LDA   #$87
43BB: BD 92 65    JSR   $9265
43BE: 35 82       PULS  A,PC ; (PUL? PC=RTS)
                                        ;
----------------------------------------;--------------------------------------------------
                                        ;
                                        ; VerifyHstdTableXAllowed()
                                        ;  Returns c-set if not allowed.
                                        ;  C and $04FB cleared if allowed
43C0: 8C 43 45    CMPX  #$4345          ;
43C3: 26 07       BNE   $43CC           ;
43C5: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ; // C-bit set when not-equal
43C8: A4 01                             ; 0xA4, $1B65:$1B66 HstdAdjustment003, Champion Hstd
43CA: 25 0A       BCS   $43D6           ;
43CC: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ; // C-bit set when not-equal
43CF: A2 01                             ; 0xA2, $1B61:$1B62 HstdAdjustment001, Highest Scores
43D1: 25 03       BCS   $43D6           ;
43D3: BD 41 65    JSR   $4165           ; Validates the high score table entry pointed to by X
43D6: 39          RTS                   ;
                                        ;
----------------------------------------;--------------------------------------------------
```

To further describe the redundancy and validations in how HSTD gets updated, consider the checks and balances that take place when a score is being processed.

- At end-of-game, GetNextPlayerHstdAchieved() takes the player's score and calls function $41E3,3D ComparePlayerIndexAScoreWithHSTDTableX() to compare score with the GC and, if necessary, the Top-4 HSTD table. This, in turn, calls CompareBCDScoreUWithHSTDTableX() at $41EE,3D to compare scores. This includes a validation of the HSTD table which will cause table to reset to factory defaults if problem is found. The player's score is compared to the GC or to each of the top-4 scores until the player's score is found to be greater than an existing GC or top-4 score, returning the discovered "winning" HSTD table/entry to caller of GetNextPlayerHstdAchieved().
- The "winning" HSTD table/entry information is passed into $4DC3,24 HstdCollectPlayerInitials(). This starts with 3 blank 'space' characters and lets player select displayable characters. When 3 characters have been entered or after a fixed timeout occurs, code at $4F04,24 is reached which then checks the previously declared "winning" HSTD table/entry. If it was Grand Champion then function $4148,3D is called to get existing GC score and $421D,3D is called to push the existing GC score into the top-4 HSTD table. After this, the player's score is used with $421D,3D to insert score into the GC table. Tracing these functions in bank $3D will show how the code ensures the inserted score is applicable for the table. Scores are compared before table insertion and player score must be greater than the existing GC score for it to be added as the new Grand Champion.
- In order for the reported problem to occur, it appears that multiple failures would need to take place in order for the initial determination of GC and subsequent insertion of the player's score into the Grand Champion table.
- Feedback is welcomed and appreciated from any readers who can point to a path in these function calls whereby the players score might incorrectly be inserted into the Grand Champion score (when it should have been inserted into the Top-4 HSTD table instead).

As an additional consideration, the attract-mode display for HSTD data has also been examined. To rule out possibility that, perhaps, the HSTD was properly updated but an attract-mode problem is displaying data incorrectly, the attract mode HSTD display code is presented here. This function is located with a lot of other attract mode functions in bank $30 at $7C43,30, ROM offset 0x43C43.

```
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; AttractMode_HighScores()
                                       ;
7C43: 8E 43 3E        LDX   #$433E     ; Top-4 HSTD Table pointer in bank $3D
7C46: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7C49: 43 C0 3D                         ; VerifyHstdTableXAllowed() c-set if not allowed
                                       ;
7C4C: 10 25 00 B1     LBCS  $7D01      ; Long branch if C-set to the RTS
                                       ; else proceed with display of high scores
                                       ;
7C50: 8D A3           BSR   $7BF5      ; AttractMode_GrandChampion() Displays "Grand Champion"
                                       ;
7C52: 8E 43 3E        LDX   #$433E     ; Top-4 HSTD Table pointer in bank $3D
7C55: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7C58: 41 65 3D                         ; Validates the high score table entry pointed to by X
                                       ;
7C5B: 10 25 00 A2     LBCS  $7D01      ; Long branch if C-set to the RTS
                                       ;
7C5F: 8E 06 52        LDX   #$0652     ;
7C62: BF 17 99        STX   $1799      ;
7C65: 8E 43 3E        LDX   #$433E     ; Top-4 HSTD Table pointer in bank $3D
                                       ;
7C68: BD D3 60        JSR   $D360      ;
7C6B: BD D7 99        JSR   $D799      ; Print string on DMD
7C6E: 00 95                            ; $4BEB,30 String Table index 0x95 "TOP MARKSMEN"
7C70: 09                               ;
7C71: 40 0B                            ;
7C73: C6 01           LDB   #$01       ; B gets 0x01, HSTD #1
7C75: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7C78: 41 48 3D                         ; GetHstdTableEntryXIndexB_into_InitialsUScoreY()
7C7B: BD D7 7D        JSR   $D77D      ; Load text bitmap using absolute x position
7C7E: 80 30                            ; $4001,3C String Table index 0x30 "%B) %SR3KU"
7C80: 09                               ;
7C81: 00 1A                            ;
7C83: BD D7 B4        JSR   $D7B4      ;
7C86: 80 31                            ; $4001,3C String Table index 0x31 "%OIXY"
7C88: 09                               ;
7C89: 7F 1A                            ;
7C8B: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7C8E: 4A 57 24                         ;
7C91: BD 83 46                         ; Sleep()
7C94: 60                               ; 1.5 seconds
7C95: BD D3 60        JSR   $D360      ;
                                       ;
7C98: 5C              INCB             ; B goes to 0x02, HSTD #2
7C99: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7C9C: 41 48 3D                         ; GetHstdTableEntryXIndexB_into_InitialsUScoreY()
7C9F: BD D7 7D        JSR   $D77D      ; Load text bitmap using absolute x position
7CA2: 80 30                            ; $4001,3C String Table index 0x30 "%B) %SR3KU"
7CA4: 09                               ;
7CA5: 00 1A                            ;
7CA7: BD D7 B4        JSR   $D7B4      ;
7CAA: 80 31                            ; $4001,3C String Table index 0x31 "%OIXY"
7CAC: 09                               ;
7CAD: 7F 1A                            ;
7CAF: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7CB2: 4A C6 24                         ;
7CB5: BD 83 46        JSR   $8346      ; Sleep()
7CB8: 60                               ; 1.5 seconds
7CB9: BD D3 60        JSR   $D360      ;
                                       ;
7CBC: 5C              INCB             ; B goes to 0x03, HSTD #3
7CBD: BD 88 F5        JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
```

```
7CC0: 41 48 3D                          ; GetHstdTableEntryXIndexB_into_InitialsUScoreY()
7CC3: BD D7 7D        JSR   $D77D       ; Load text bitmap using absolute x position
7CC6: 80 30                             ; $4001,3C String Table index 0x30 "%B) %SR3KU"
7CC8: 09                                ;
7CC9: 00 1A                             ;
7CCB: BD D7 B4        JSR   $D7B4       ;
7CCE: 80 31                             ; $4001,3C String Table index 0x31 "%OIXY"
7CD0: 09                                ;
7CD1: 7F 1A                             ;
7CD3: BD 88 F5        JSR   $88F5       ; CallBankedFunction_Param_WPCAddr()
7CD6: 4A 9A 24                          ;
7CD9: BD 83 46        JSR   $8346       ; Sleep()
7CDC: 60                                ; 1.5 seconds
7CDD: BD D3 60        JSR   $DE60       ;
                                        ;
7CE0: 5C              INCB              ; B goes to 0x04, HSTD #4
7CE1: BD 88 F5        JSR   $88F5       ; CallBankedFunction_Param_WPCAddr()
7CE4: 41 48 3D                          ; GetHstdTableEntryXIndexB_into_InitialsUScoreY()
7CE7: BD D7 7D        JSR   $D77D       ; Load text bitmap using absolute x position
7CEA: 80 30                             ; $4001,3C String Table index 0x30 "%B) %SR3KU"
7CEC: 09                                ;
7CED: 00 1A                             ;
7CEF: BD D7 B4        JSR   $D7B4       ;
7CF2: 80 31                             ; $4001,3C String Table index 0x31 "%OIXY"
7CF4: 09                                ;
7CF5: 7F 1A                             ;
7CF7: BD 88 F5        JSR   $88F5       ; CallBankedFunction_Param_WPCAddr()
7CFA: 4A C6 24                          ;
7CFD: BD 83 46        JSR   $8346       ; Sleep()
7D00: 60                                ; 1.5 seconds
                                        ;
7D01: 39              RTS               ;
                                        ;
----------------------------------------;------------------------------------------------
```

The attract mode HSTD function, above will first make a call to 7BF5,30, ROM offset 0x43BF5, as shown below:

```
----------------------------------------;------------------------------------------------
                                        ;
                                        ; AttractMode_GrandChampion()
                                        ;
                                        ; Displays "Grand Champion" score and brief delay
                                        ;
7BF5: 8E 43 45        LDX   #$4345      ; Grand Chamption HSTD Table pointer in bank $3D
7BF8: BD 88 F5        JSR   $88F5       ; CallBankedFunction_Param_WPCAddr()
7BFB: 43 C0 3D                          ; VerifyHstdTableXAllowed() c-set if not allowed
7BFE: 25 42           BCS   $7C42       ; If C-set, HSTD table is not valid branch down to RTS
                                        ;
7C00: 8E 43 45        LDX   #$4345      ; Grand Chamption HSTD Table pointer in bank $3D
7C03: BD 88 F5        JSR   $88F5       ; CallBankedFunction_Param_WPCAddr()
7C06: 41 65 3D                          ; Validates the high score table entry pointed to by X
7C09: 25 37           BCS   $7C42       ;
                                        ;
7C0B: BD FB AE        JSR   $FBAE       ; ClearDisplayMemory()
7C0E: 7E 7C 11        JMP   $7C11       ; <nop>
                                        ;
7C11: 8E 06 52        LDX   #$0652      ;
7C14: BF 17 99        STX   $1799       ;
7C17: BD D3 60        JSR   $D360       ;
7C1A: BD D7 99        JSR   $D799       ; Print string on DMD
7C1D: 80 33                             ; $4001,3C String Table index 0x33 "GRAND CHAMPION"
7C1F: 09                                ;
7C20: 40 0B                             ;
7C22: C6 01           LDB   #$01        ; B gets 0x01, Grand Champion table entry 1 of 1
7C24: 8E 43 45        LDX   #$4345      ; Grand Chamption HSTD Table pointer in bank $3D
7C27: BD 88 F5        JSR   $88F5       ; CallBankedFunction_Param_WPCAddr()
7C2A: 41 48 3D                          ; GetHstdTableEntryXIndexB_into_InitialsUScoreY()
7C2D: BD D7 99        JSR   $D799       ; Print string on DMD
```

```
7C30: 00 97                           ; $4BEB,30 String Table index 0x97 "%SR3KU  %R10OIXY"
7C32: 09                              ;
7C33: 40 1A                           ;
7C35: BD 88 F5        JSR   $88F5     ; CallBankedFunction_Param_WPCAddr()
7C38: 4A 14 24                        ;
7C3B: BD D3 4C        JSR   $D34C     ;
7C3E: BD 83 46        JSR   $8346     ; Sleep()
7C41: 80                              ; 2.0 seconds
7C42: 39              RTS             ;
                                      ;
--------------------------------------;-------------------------------------------------
```

The code for displaying scores, above, is rather uneventful and appears to show the Grand Champion score and then the top-4 scores in a straightforward way. There does not appear to be any particular way where the game would incorrectly display a top-4 score when it is trying to display the Grand Champion score.

Readers are encouraged to study the code, above, and perhaps experiment with pinball simulators to further analyze code flow and, perhaps, come up with any explanation for how high scores could ever be inserted incorrectly. If any flaw can be identified and reproduced, a later update will be made which corrects the flaw.

We are actively collecting any further reports of unexpected HSTD table updates.

Adjustable Autofire Timer

For L8.4 it was requested that the autofire timer is made adjustable so that the game operator can choose how many seconds the game will flash the “Autofire” where, upon ball drain, the game will automatically serve another ball to the player without penalty, allowing the current ball-in-play to continue. This request is specifically for the autofire period that occurs at the start of each ball and not applicable to other moments of game play where the game will return the ball back to the player after a ball drain.

Some investigation into the game code reveals that the original L-8 autofire timer code is a bit more complicated than a simple fixed number of seconds at start of the autofire period. The code for autofire uses various timer values depending on the gameplay element that initiated the autofire timer. A summary of all autofire timer values is as follows:

Autofire Mode	Gameplay Event	Seconds
00	Ball Shooter Lane Exit/5-bank Target hit during Skill Shot	2, 3, or 5
01	Multiball Start, player’s first multiball	7
02	Autofire Awarded (i.e. from Database award)	8
03	Drop Target hit and A.2 17 “Drp Tgt Autofire” adjustment is “On”	5

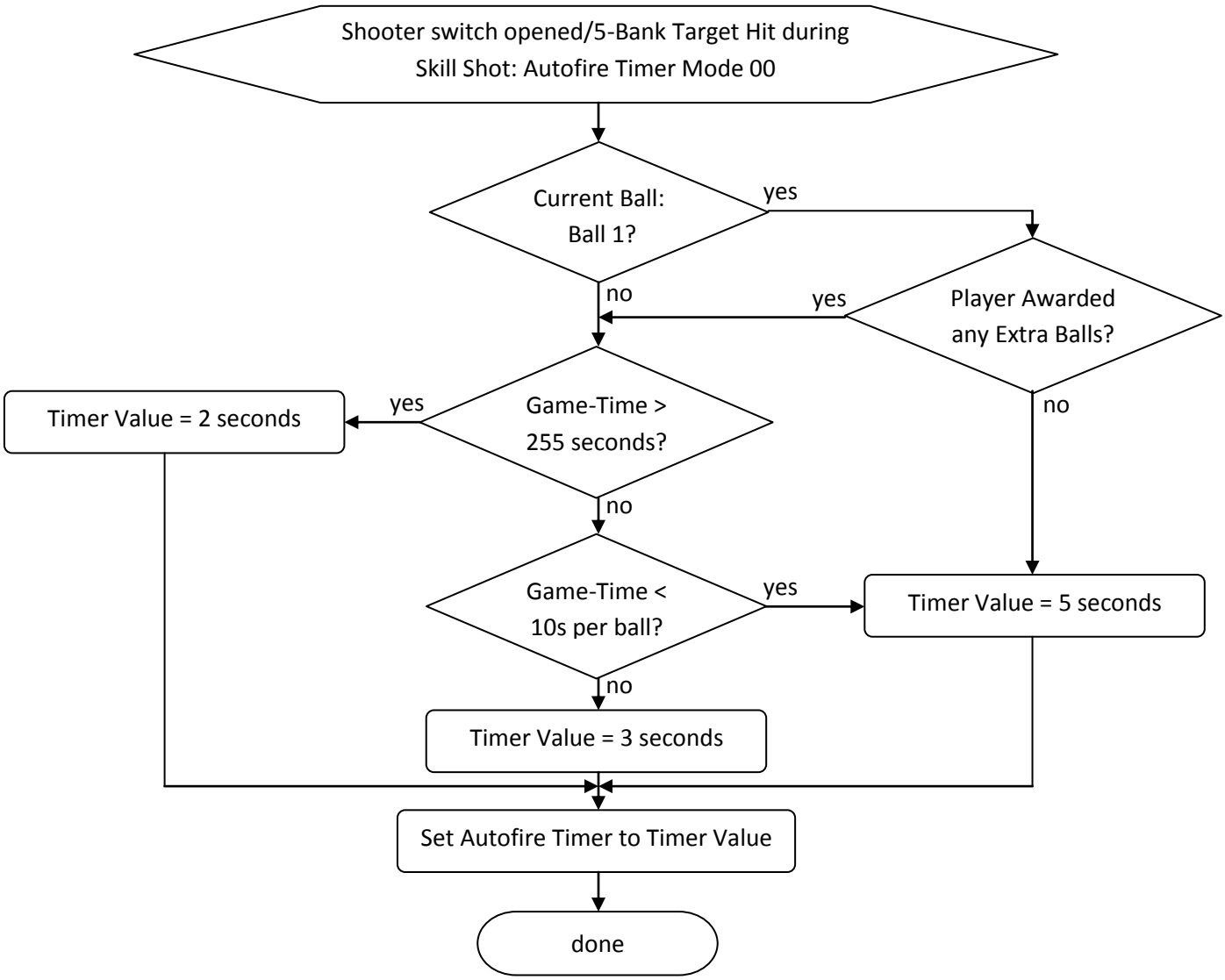
As shown, there are a variety of different autofire timer values, depending on the nature of the gameplay that is causing the autofire to engage. For L8.4, the first row, “Ball Shooter Lane Exit/5-bank Target hit during Skill Shot” is subject of a new adjustment, and it also happens to be the most complicated of all of the autofire timer values.

Autofire Mode 00, Skill Shot

As shown above, there are 3 possible autofire timer values used to start the autofire timer during a skill shot period. The logic for which timer is used boils down to the following:

- If the current ball is “Ball 1” and the player has not been awarded any extra balls then the autofire timer value used is 5 seconds.
- Otherwise, if the player’s game time is greater than 255 seconds then the autofire timer value used is 2 seconds.
- Otherwise, if the player’s game time is less than 10 seconds per ball then the autofire timer value used is 5 seconds.
- Otherwise, the autofire timer value used is 3 seconds.

This logic is depicted in the following flowchart.



Autofire Timer Code

Supporting the described autofire timer behaviors, shown below is the annotated code that handles the autofire timer. This is a function that is called after the B register has been loaded with the desired autofire timer mode, hence the name InitAutoFireModeB(). This function is located at $6DE9,31, ROM offset 0x46DE9. This function is called from a few places in game code whenever the autofire timer is potentially needed. For some modes the logic will perform checks to determine whether autofire is needed such as mode 01 when multiball is starting, only if it is the player's first multiball is when an autofire timer will be started.

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; InitAutoFireModeB()
                                        ;
6DE9: 34 16       PSHS  X,B,A           ;
6DEB: BD 88 F5    JSR   $88F5           ;
6DEE: 4C 7B 38                          ; GetCurrentPlayerIndexIntoAPlayerDataTableIntox()
6DF1: AE 88 22    LDX   $22,X           ; X gets current player's game-time
6DF4: 9F CE       STX   $CE             ; Store it into $CE:$CF
                                        ;
6DF6: C1 00       CMPB  #$00            ; 0x00 == SkillShotFiveBankTargetHit
6DF8: 27 3C       BEQ   $6E36           ;
6DFA: C1 01       CMPB  #$01            ; 0x01 == MultiballStart
6DFC: 27 71       BEQ   $6E6F           ;
6DFE: C1 02       CMPB  #$02            ; 0x02 == AutofireAward
6E00: 27 19       BEQ   $6E1B           ;
6E02: C1 03       CMPB  #$03            ; 0x03 == DropTargetSwitchHdlr
6E04: 27 02       BEQ   $6E08           ;
6E06: 35 96       PULS  A,B,X,PC        ; Should never get here, unhandled B value, unknown case
                                        ;
                                        ;------------------------------------------------------
                                        ;------------------------------------------------------
                                        ;
                                        ;------------------------------------------------------
                                        ; Mode 03 == DropTargetSwitchHdlr --> 5 second autofire
                                        ;------------------------------------------------------
6E08: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
6E0B: 11 01                             ; FeatureAdjustment017, Drt Tgt Autofire     Adj=0x11
6E0D: 25 0A       BCS   $6E19           ;
6E0F: C6 05       LDB   #$05            ; 5 second ball saver
6E11: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
6E14: 00 83                             ;
6E16: 6E 8A 31                          ;
6E19: 20 6D       BRA   $6E88           ;
                                        ;
                                        ;------------------------------------------------------
                                        ; Mode 02 == AutofireAward --> 8 second autofire
                                        ;------------------------------------------------------
6E1B: 8E 05 B1    LDx   #$05B1          ; 0x05B1 == Number of Autofire awards given, per-player
6E1E: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6E21: 7E 6E 24    JMP   $6E24           ; <nop>
6E24: 6C 84       INC   ,X              ; Increment number of autofires given for current player
6E26: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
6E29: 7A                                ; 0x7A == "Autofire"
6E2A: C6 08       LDB   #$08            ; 8 second ball saver
6E2C: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
6E2F: 00 83                             ;
6E31: 6E 8A 31                          ;
6E34: 20 52       BRA   $6E88           ;
```

```
                                        ;
                                        ;-------------------------------------------
                                        ; When B == 0x00 == SkillShotFiveBankTargetHit
                                        ;-------------------------------------------
6E36: 8E 05 AD    LDX   #$05AD          ; 0x05AD == Number of extra balls awarded to player
6E39: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6E3C: 7E 6E 3F    JMP   $6E3F           ; <nop>
6E3F: 6D 84       TST   ,X              ;
6E41: 26 07       BNE   $6E4A           ;
6E43: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
6E46: 81 01       CMPA  #$01            ; if ((extra balls == 0) && (CurrentBallInPlay == 1))
6E48: 27 19       BEQ   $6E63           ;    goto 5 second ball saver
                                        ;
6E4A: 8E 00 CE    LDX   #$00CE          ;
6E4D: 6D 84       TST   ,X              ; If (player game time > 255 seconds)
6E4F: 26 0E       BNE   $6E5F           ;    goto 2 second ball saver
                                        ;
6E51: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
6E54: C6 0A       LDB   #$0A            ;
6E56: 3D          MUL                   ; D = current ball * 10
6E57: E1 01       CMPB  $0001,X         ; if (player game time is less than 10 seconds per ball)
6E59: 22 08       BHI   $6E63           ;    goto 5 second ball saver
                                        ; else
6E5B: C6 03       LDB   #$03            ;    3 second ball saver
6E5D: 20 06       BRA   $6E65           ;
6E5F: C6 02       LDB   #$02            ; 2 second ball saver
6E61: 20 02       BRA   $6E65           ;
                                        ;
6E63: C6 05       LDB   #$05            ; 5 second ball saver
6E65: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
6E68: 00 83                             ;
6E6A: 6E 8A 31                          ;
6E6D: 20 19       BRA   $6E88           ;
                                        ;
                                        ;-------------------------------------------------------
                                        ; Mode 01 == MultiballStart --> 7 second autofire
                                        ;                               if at first multiball
                                        ;-------------------------------------------------------
6E6F: 8E 05 C9    LDX   #$05C9          ; 0x05C9 == Number of multiballs for current player
6E72: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6E75: 7E 6E 78    JMP   $6E78           ; <nop>
6E78: A6 84       LDA   ,X              ;
6E7A: 81 01       CMPA  #$01            ;
6E7C: 26 EF       BNE   $6E6D           ;
6E7E: C6 07       LDB   #$07            ; 7 second ball saver
6E80: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
6E83: 00 83                             ; ID 0083
6E85: 6E 94 31                          ;
6E88: 35 96       PULS  A,B,X,PC        ;
                                        ;
----------------------------------------;-------------------------------------------------------
```

The autofire function refers to a value that is being annotated here as *number of seconds* of game play for the current player. In the simulator this value regularly increases as game play proceeds however actual measurement of *per-second* has not been verified as the actual increment rate, but has been assumed to be the case during code analysis.

For mode 00, the game code has a redundant mechanism to ensure the autofire timer is started. During skillshot, when the ball shooter switch is opened, the timer is started. Also when a 5-bank target is hit during skill shot, it ensures the timer is running if not already started due to shooter switch opening.

Autofire Timer Adjustment

To accommodate the request for adjustable autofire timer, and for it to apply only to skill shot autofire timer, a new adjustment is added in L8.4 to allow a variety of adjustments to fit various needs. The new adjustment will provide the following possible adjustment values:

Adjustment Value	Internal Value	Meaning
Original	0 (0x00)	Retains existing L-8 autofire timer
+1 sec	1 (0x01)	Adds 1 second to existing skill shot autofire timer values
+2 secs	2 (0x02)	Adds 2 seconds to existing skill shot autofire timer values
+3 secs	3 (0x03)	Adds 3 seconds to existing skill shot autofire timer values
+4 secs	4 (0x04)	Adds 4 seconds to existing skill shot autofire timer values
+5 secs	5 (0x05)	Adds 5 seconds to existing skill shot autofire timer values
+6 secs	6 (0x06)	Adds 6 seconds to existing skill shot autofire timer values
+7 secs	7 (0x07)	Adds 7 seconds to existing skill shot autofire timer values
+8 secs	8 (0x08)	Adds 8 seconds to existing skill shot autofire timer values
+9 secs	9 (0x09)	Adds 9 seconds to existing skill shot autofire timer values
+10 secs	10 (0x0A)	Adds 10 seconds to existing skill shot autofire timer values
Off	11 (0x0B)	No skill shot autofire timer
1 sec	12 (0x0C)	Uses 1 second for all skill shot autofire timer values
2 secs	13 (0x0D)	Uses 2 seconds for all skill shot autofire timer values
3 secs	14 (0x0E)	Uses 3 seconds for all skill shot autofire timer values
4 secs	15 (0x0F)	Uses 4 seconds for all skill shot autofire timer values
5 secs	16 (0x10)	Uses 5 seconds for all skill shot autofire timer values
6 secs	17 (0x11)	Uses 6 seconds for all skill shot autofire timer values
7 secs	18 (0x12)	Uses 7 seconds for all skill shot autofire timer values
8 secs	19 (0x13)	Uses 8 seconds for all skill shot autofire timer values
9 secs	20 (0x14)	Uses 9 seconds for all skill shot autofire timer values
10 secs	21 (0x15)	Uses 10 seconds for all skill shot autofire timer values

The set of adjustments provides several options regarding the skill shot autofire timer:

- Retain all original L-8 skill shot autofire timer behavior
- Add 1-10 seconds to the various L-8 skill shot autofire timer values
- Disable the skill shot autofire timer
- Set the skill shot autofire timer to fixed value 1-10 seconds in all scenarios

Autofire Timer Adjustment – L8.4 Code Changes

The code changes for skill shot autofire timer can be divided into two distinct areas:

- Code changes to allow adjustment to be presented with the various values described above.
- Code changes to have the actual autofire timer to be set according to the adjustment value.

Autofire Timer Adjustment – L8.4 Code Changes for Adjustment Management

The adjustment table for L8.4 has extra room to accommodate an extra new adjustment. The table is updated as highlighted below.

Refer to the original text from L8.3 document describing *The L-8 Adjustments Memory Map*. The table with all adjustments is updated to accommodate the new L8-4 adjustment as highlighted in the *table portion*, below:

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

** New adjustments in L8.3 shown for reference.*

*** New L8.4 adjustment (additional new L8.4 adjustments are depicted later in this document)*

The *Feature Adjustments Metadata* table is updated as indicated in the abbreviated table content, below. Below is depicted the increased table size and new content. Additional updates in L8.4 will add additional adjustments causing the resulting L8.4 table size to also increase so 'xx' is depicted here.

```
;-----------------------------------------;-------------------------------------------------------
7000: 00 xx, Incremented by 1               ; Table entries is xx (xx entries), Incremented by 1
7002: 0C                                    ; Bytes per table entry is 0C (12 bytes)
...
70FF: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.21, Drptrgt 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 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.28, Cannon 1 Hit
715F: 00 00 00 00 00 15 00 01 00 7B 52 3A ; Feature Adjustments, A2.29, SS Autofire Time
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>
;---------------------------------------;-------------------------------------------------------
```

The metadata table, above, is located at $7000,3D, ROM offset 0x77000 and is updated with the new adjustment for skillshot autofire timer values. The name of this new adjustment is added to the existing English, German, and French menu string tables as indicated in the table below.

	Table Start	New Pointer Bytes	New String Address	New String Value
English	$6700,3D	$673D,3D	$68F8,3D	"SS AUTOFIRE TIME"
German	$6A00,3D	$6A3D,3D	$6B90,3D	"SS AUTOFIRE ZEIT"
French	$6D00,3D	$6D3D,3D	$6EE1,3D	"SS DUR. AUTOFIRE"

The new adjustment, internally, tracks the various possible values as a simple numeric value that corresponds to the various settings as according to the "Internal Value" previously depicted in the table of possible adjustment values. As the adjustment contains a variety of possible values, a new function is defined in L8.4 to interpret the different values for display in the adjustment menu. To accommodate the display of these unique values, a new function is added at $7B52,3A, ROM offset 0x6BB52 that the adjustment menu will use for displaying the various values. As shown in the updated adjustment table row, above, this function is cited in the last 3 bytes of its 12 byte entry as 0x7B, 0x52, 0x3A.

Since the L8.4 changes are made with efforts to keep changes localized, the adjustment display handler uses its own method of displaying the various strings. This is because the master string table is not being updated for L8.4, so local strings are being referenced which means the common L-8 function for handling display of strings cannot easily be used. Refer to the new adjustment display handler below.

```
---------------------------------------;-------------------------------------------------------
                                       ;
                                       ; Adjustment handler for "SS Autofire Time"
                                       ;
                                       ; A has adjustment index
                                       ; B has code, 0x02 means to write the adjustment value
                                       ;                  ASCII string at pointer Y
                                       ; U has the current value
                                       ; Y has address of where to put ASCII string (when B=02)
                                       ;
                                       ;
7B52: BD 79 E2        JSR   $79E2      ; AdjustmentWriteASCIIValueCclear()
7B55: 25 03           BCS   $7B5A      ; If C-bit is set, RTS now.  Don't write ASCII string.
7B57: BD 7B 6D        JSR   $7B6D      ; Print the adjustment string for value U
7B5A: 39              RTS              ;
                                       ;
---------------------------------------;-------------------------------------------------------
                                       ;
                                       ; Advance X depending on Language Adjustment value
7B5B: 34 06           PSHS  B,A        ;
7B5D: BD 82 FF        JSR   $82FF      ; Get language adjustment value into A
7B60: 95                               ;
7B61: 81 02           CMPA  #$02       ; Check if "French" 0x02
```

```
7B63: 22 04           BHI   $7B69       ; If greater than 0x02, assume error and use English.
7B65: 1F 89           TFR   A,B         ; Need to increment X by value in A*2, store A into B
7B67: 58              ASLB              ; Shift left, to multily B by 2
7B68: 3A              ABX               ; X now gets updated pointer value for language
7B69: AE 84           LDX   ,X          ; De-reference X to get the start address of the string
7B6B: 35 86           PULS  A,B,PC      ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
7B6D: 34 36           PSHS  Y,X,B,A     ;
7B6F: 1F 30           TFR   U,D         ; D now has the adjustment value
                                        ;
7B71: BD 7B C2        JSR   $7BC2       ; Load X with pointer for current adj value D
7B74: BD 7B 5B        JSR   $7B5B       ; Advance X for current language setting
7B77: BD B9 51        JSR   $B951       ; CopyASCIIStringFromXtoYandVerifyLength()
                                        ;
7B7A: 35 B6           PULS  A,B,X,Y,PC  ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
7B7C: 7B 94                             ; English  "ORIGINAL"
7B7E: 7B 94                             ; German   "ORIGINAL"
7B80: 7B 9D                             ; French   "ORIGINALE"
7B82: 7B A7                             ; English  "OFF"
7B84: 7B AB                             ; German   "AUS"
7B86: 7B AF                             ; French   "HORS"
7B88: 7B B4                             ; English  "sec"       English Sigular
7B8A: 7B BD                             ; German   "Sek."      German Singular
7B8C: 7B B4                             ; French   "sec"       French Singular
7B8E: 7B B8                             ; English  "secs"      English Plural
7B90: 7B BD                             ; German   "Sek."      German Plural
7B92: 7B B8                             ; French   "secs"      French Plural
                                        ;
7B94: 4F 52 49 47 49 4E 41 4C 00        ; "ORIGINAL"  (English and German)
7B9D: 4F 52 49 47 49 4E 41 4C 45 00     ; "ORIGINALE" (French "Original")
7BA7: 4F 46 46 00                       ; "OFF"       (English)
7BAB: 41 55 53 00                       ; "AUS"       (German off)
7BAF: 48 4F 52 53 00                    ; "HORS"      (French for OFF)
7BB4: 73 65 63 00                       ; "sec"       (English and French)  Singular
7BB8: 73 65 63 73 00                    ; "secs"      (English and French)  Plural
7BBD: 53 65 6B 2E 00                    ; "Sek."      (German)              Singular or Plural
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
7BC2: C1 00           CMPB  #$00        ; Check if at "ORIGINAL" 0x00
7BC4: 26 05           BNE   $7BCB       ;
7BC6: 8E 7B 7C        LDX   #$7B7C      ; X gets pointer to English: "ORIGINAL"
7BC9: 20 34           BRA   $7BFF       ; Done
                                        ;
7BCB: C1 0B           CMPB  #$0B        ; Check if at "OFF" 0x0B
7BCD: 26 05           BNE   $7BD4       ;
7BCF: 8E 7B 82        LDX   #$7B82      ; X gets pointer to English: "OFF"
7BD2: 20 2B           BRA   $7BFF       ; Done
                                        ;
7BD4: C1 15           CMPB  #$15        ; Check if at "2 secs" through "10 secs" 0x0D - 0x15
7BD6: 22 EE           BHI   $7BC6       ; If higher than "10 secs" error, just show "Original"
                                        ;
7BD8: C1 0A           CMPB  #$0A        ; Check if at "+2 secs" through "+10 secs" 0x02 - 0x0A
7BDA: 23 04           BLS   $7BE0       ; If less then or same as 0x0A then it is "+" seconds
7BDC: C0 0B           SUBB  #$0B        ; Fixup the B value so it now has number of seconds
7BDE: 20 04           BRA   $7BE4       ;
7BE0: 86 2B           LDA   #$2B        ; A gets the '+' character we will print
```

```
7BE2: A7 A0           STA   ,Y+          ; Display now has '+' character prior to seconds value
                                         ;
7BE4: 8E 7B 8E        LDX   #$7B8E       ; X gets pointer to English: "secs" Plural
7BE7: C1 01           CMPB  #$01         ; Check if at "1 sec"
7BE9: 26 03           BNE   $7BEE        ;
7BEB: 8E 7B 88        LDX   #$7B88       ; X gets pointer to English: "sec" Singular
                                         ;
7BEE: C1 09           CMPB  #$09         ; Checking if we are at 1-9 seconds
7BF0: 23 05           BLS   $7BF7        ;
7BF2: 86 31           LDA   #$31         ; A gets the '1' character we will print for the "10"
7BF4: A7 A0           STA   ,Y+          ; Display now has '1' character as part of the "10"
7BF6: 5F              CLRB               ; Clear B so the '0' gets printed next
7BF7: CA 30           ORB   #$30         ; Make the seconds value in B ASCII "1" through "9"
7BF9: E7 A0           STB   ,Y+          ; Now print remaining seconds digit
7BFB: 86 20           LDA   #$20         ; A gets the ' ' character between number and label
7BFD: A7 A0           STA   ,Y+          ; Display now has '1' character as part of the "10"
                                         ;
7BFF: 39              RTS                ;
                                         ;
-----------------------------------------;-------------------------------------------------------
```

The code, above, for displaying the adjustment string value is specific only for this new adjustment, with fixed assumptions that the max value is 10 seconds. The code checks if the adjustment is a 'plus' or 'fixed' number of seconds. Since the number of seconds is allowed to go to 1, there is added logic to handle singular or plural representation of abbreviated seconds indicator. The detailed flow through the above code is left as an exercise to the reader.

Autofire Timer Adjustment – L8.4 Code Changes for Autofire Timer Startup

With the new adjustment in place, the actual autofire timer startup code now needs to be updated to behave in accordance with the new adjustment value. As previously depicted, the function is located at $6DE9,31, ROM offset 0x46DE9. This is where the code will be updated to take the new adjustment value into account.

The applicable portion of the existing code is re-shown below with the L8.4 modification highlighted:

```
                                         ;
                                         ;-------------------------------------------
                                         ; When B == 0x00 == SkillShotFiveBankTargetHit
                                         ;-------------------------------------------
6E36: 8E 05 AD    LDX   #$05AD           ; 0x05AD == Number of extra balls awarded to player
6E39: BD FB 29    JSR   $FB29            ; IncrementXByPlayerIndexNumber()
6E3C: 7E 6E 3F    JMP   $6E3F            ; <nop>
6E3F: 6D 84       TST   ,X               ;
6E41: 26 07       BNE   $6E4A            ;
6E43: BD B1 D1    JSR   $B1D1            ; GameOnLastBallCheckCBitClearCurrentBallInA()
6E46: 81 01       CMPA  #$01             ; if ((extra balls == 0) && (CurrentBallInPlay == 1))
6E48: 27 19       BEQ   $6E63            ;    goto 5 second ball saver
                                         ;
6E4A: 8E 00 CE    LDX   #$00CE           ;
6E4D: 6D 84       TST   ,X               ; If (player game time > 255 seconds)
6E4F: 26 0E       BNE   $6E5F            ;    goto 2 second ball saver
                                         ;
6E51: BD B1 D1    JSR   $B1D1            ; GameOnLastBallCheckCBitClearCurrentBallInA()
6E54: C6 0A       LDB   #$0A             ;
6E56: 3D          MUL                    ; D = current ball * 10
```

```
6E57: E1 01       CMPB  $0001,X          ; if (player game time is less than 10 seconds per ball)
6E59: 22 08       BHI   $6E63           ;    goto 5 second ball saver
                                        ; else
6E5B: C6 03       LDB   #$03            ;    3 second ball saver
6E5D: 20 06       BRA   $6E65           ;
6E5F: C6 02       LDB   #$02            ; 2 second ball saver
6E61: 20 02       BRA   $6E65           ;
                                        ;
6E63: C6 05       LDB   #$05            ; 5 second ball saver
6E65: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
6E68: 00 83                             ;
6E6A: 6E 8A 31                          ;
6E65: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
6E68: 7C 00 3A                          ; New autofire timer startup function $7C00,3A
6E6B: 20 00       BRA   $6E6D           ; <nop> filler for 2 bytes, go to next instruction
6E6D: 20 19       BRA   $6E88           ;
                                        ;
```

The old code used 8 bytes to immediately schedule the function $6E8A,31 with scheduler ID 0x0083 with B register containing the number of seconds for the autofire timer to use. The new code replaces the 8 bytes with a call to a new L8.4 function at $7C00,3A, ROM offset 0x6BC00, which will perform the new logic as shown below.

```
---------------------------------------;-----------------------------------------------------
                                       ;
                                       ; Jump here from $6E65,31 skillshot autofire timer start
                                       ; B has number of seconds the L-8 code determined for
                                       ;  the autofire timer value
                                       ;
7C00: 34 06          PSHS  B,A         ;
                                       ;
7C02: BD 82 FF       JSR   $82FF       ; Get "SS Autofire Time" adjustment value into A
7C05: 1D                               ;
                                       ;
7C06: 81 00          CMPA  #$00        ; Check if at "ORIGINAL" 0x00
7C08: 27 16          BEQ   $7C20       ; If at original, go schedule autofire now using B value
                                       ;
7C0A: 81 0B          CMPA  #$0B        ; Check if at "OFF" 0x0B
7C0C: 27 1A          BEQ   $7C28       ; If at OFF, done, no autofire timer, return
                                       ;
7C0E: 81 15          CMPA  #$15        ; Check if at "2 secs" through "10 secs" 0x0D - 0x15
7C10: 22 0E          BHI   $7C20       ; If higher than "10 secs" error, handle as "Original"
                                       ;
7C12: 81 0A          CMPA  #$0A        ; Check if at "+2 secs" through "+10 secs" 0x02 - 0x0A
7C14: 23 06          BLS   $7C1C       ; If less then or same as 0x0A then it is "+" seconds
7C16: 80 0B          SUBA  #$0B        ; Fixup the A value so it now has number of seconds
7C18: 1F 89          TFR   A,B         ; Overwrite B with the fixed number of seconds in A
7C1A: 20 04          BRA   $7C20       ; Now schedule the timer using fixed number of seconds
                                       ;
7C1C: 34 02          PSHS  A           ;
7C1E: EB E0          ADDB  ,S+         ;
                                       ; Now schedule the autofire timer timer value in B
7C20: BD 8B 77       JSR   $8B77       ; ScheduleFunctionStart()
7C23: 00 83                            ;
7C25: 6E 8A 31                         ;
                                       ;
7C28: 35 86          PULS  A,B,PC      ;
                                       ;
---------------------------------------;-----------------------------------------------------
```

The new skillshot timer logic, above, simply retrieves the new “SS Autofire Time” adjustment value and uses it to determine how to fixup the B value (number of seconds) before scheduling the same original function that was scheduled in original L-8 code. In the event that the new “Off” adjustment value is set, then no timer gets scheduled.

The value of B may be incremented by the adjustment value or set to the specific number of seconds as per the adjustment value. This gives the game operator a lot of flexibility in how they wish to allow their game to run the autofire timer for skillshot attempts. *Note that when the autofire timer is set to “Off” the game will still employ other mechanisms to return the ball to the player in cases such as where no targets have been hit and the ball drains after it had exited the shooter lane.*

Text String Corrections L8.4

The previous L8.3 release covered a large number of text fixups. For L8.4 additional text changes were done and documented here.

Fix German Adjustment Menu Text For Abbreviated Seconds

While implementing the text for Skillshot Autofire Timer feature in L8.4 it was observed that the game menu system uses “secs” for abbreviated “seconds” in English and French while the German text is “sek.” The correct abbreviated term for “seconds” in German should use upper case ‘S’ for this string.

This string is located at $5B01,30, ROM offset 0x41B01 and is as follows:

```
5B01: 25 75 20 73 65 6B 2E 00           ; "%u sek."
```

This string includes “%u” as the game will invoke code to print the string while the desired numeric value is located in the ‘U’ register. For L8.4, the ‘S’ is made upper-case with new code as follows:

```
5B01: 25 75 20 53 65 6B 2E 00           ; "%u Sek."
```

An example of this updated string is as follows:

Super Jackpot Lamp Movement Update for L8.4

For L8.4 it was requested that some enhancement is made to the Super Jackpot lamp movements. The problem is that the original Super Jackpot lamp movement is too predictable. The fact that the lamp movements always begin at the same point in conjunction with the cannon motor, a player can easily hit the super jackpot by simply observing the cannon position alone, and pulling the trigger at the right moment in the cannon swing. Although this is beneficial to the player, it would be better to require the player to observe the cannon and moving target in order to aim for a successful hit.

The goals for the L8.4 Super Jackpot lamp movement changes are as follows:

- Ensure the original L-8 movement can be selected (and is the default adjustment value).
- Provide adjustment(s) to alter the lamp movement difficulty level.
- Ensure that all players in a multi-player game get the same experience.

Super Jackpot Lamp Design Considerations for L8.4

This section generally describes the design ideas and goals defined in the L8.4 Super Jackpot enhancement. In order to define the best solution for the L8.4 design, various ideas were raised and the resulting design ideas and goals are presented below.

Super Jackpot Lamp Movement Possibilities

The current design of L-8 Super Jackpot code makes it fairly easy to update the code to allow for various changes to the lamp movement. Some of the available changes that can be done are as follows:

- Varying starting lamp.
- Varying starting lamp direction (applicable when starting lamp is 2, 3, or 4).
- Faster lamp movements.
- Different lamp movement patterns

These possible changes will be implemented in L8.4 in varying levels of difficulty.

Super Jackpot Levels of Difficulty

In order to give maximum flexibility, it is appropriate to allow for various levels of challenge with the L8.4 enhancement. Game owners and operators can adjust the level of difficulty that best fits their situation. Although the original request was, essentially, to require the player to actually *aim* the cannon (and not just memorize a gun motor position), it seems various settings can be provided to provide a variety of challenge levels which could involve:

- Smaller window of gun position that the player could memorize, or
- Multiple gun positions, depending on starting lamp movements, the player could memorize, or
- Requiring the player to observe the lamp movements and aim (or memorize a lot of patterns).

Super Jackpot Multi-Player Considerations

In order to ensure fairness during multi-player games, it is appropriate that the L8.4 design ensures that all players in a multi-player game get the same experience. The starting lamp and direction and lamp movement pattern cannot be truly random. The goal is to have all players be given the same level of

difficulty in that their Super Jackpot starting lamp and direction and lamp movement pattern should all be the same for all players in the game. The downside of such design goal is that the first player to attempt Super Jackpot is first to encounter the current game's lamp movement while subsequent players are then aware of the Super Jackpot starting lamp movement and direction that they will be presented with when their chance for Super Jackpot occurs.

This particular approach can also involve a different lamp movement for at each subsequent super jackpot attempt during the same game. In this way, all players get same lamp movement at first super jackpot. Then for the 2nd super jackpot attempt a different lamp movement can occur whereby all players going for their 2nd super jackpot will get the same movement for the 2nd super jackpot, and so on.

In order to provide consistent behavior for all players in the multi-player game, the game bookkeeping statistics can be sampled in order to get a numeric value that remains unchanged for the duration of the entire multi-player game (details to follow).

Super Jackpot Variable Lamp Movements

As mentioned above, for L8.4 Super Jackpot lamp movements the design goals are for:

- Unpredictability, or seemingly "random" lamp behaviors (i.e. starting lamp and direction), and
- Same Super Jackpot experiences for all players in a multi-player game.

A survey of game code reveals that a method could be employed whereby game bookkeeping statistics may be used to come up with numbers that are seemingly random but also consistent throughout the entire game (for all players). The bookkeeping values which are only updated at game completion can be used as basis for determining variable lamp movements that apply to the current game in play.

The following bookkeeping values are updated at the end of game. The table below shows the bookkeeping values that can be used for deriving seemingly random Super Jackpot lamp behaviors.

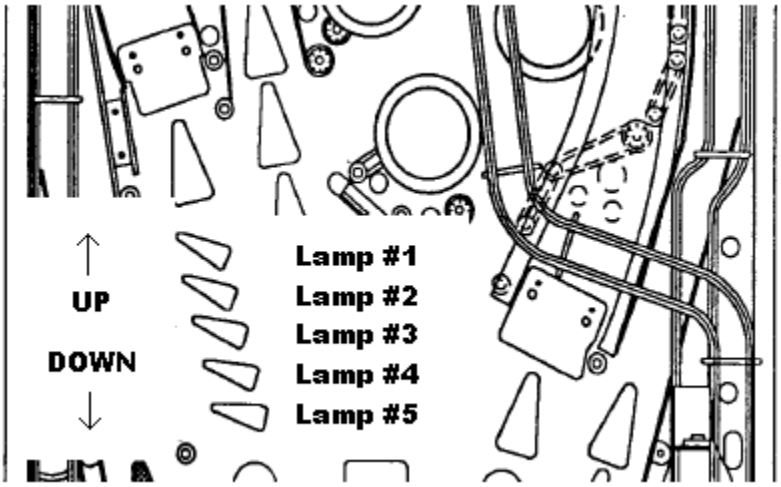
Bookkeeping Entry	ID #	RAM Location
Bookkeeping B.3 08 "MATCH AWARDS"	0x8018	$18A1 $18A2 $18A3
Bookkeeping B.3 28 "1 PLAYER GAMES"	0x8024	$18E9 $18EA $18EB
Bookkeeping B.3 29 "2 PLAYER GAMES"	0x8025	$18EF $18F0 $18F1
Bookkeeping B.3 30 "3 PLAYER GAMES"	0x8026	$18F5 $18F6 $18F7
Bookkeeping B.3 31 "4 PLAYER GAMES"	0x8027	$18FB $18FC $18FD
Bookkeeping B.5 01 "0-1.9 M. SCORE"	0x802D	$191F $1920 $1921
Bookkeeping B.5 02 "2-4.9 M. SCORE"	0x802E	$1925 $1926 $1927
Bookkeeping B.5 03 "3-9.9 M. SCORE"	0x802F	$192B $192C $192D
Bookkeeping B.5 04 "10-19 M. SCORE"	0x8030	$1931 $1932 $1933
Bookkeeping B.5 05 "20-29 M. SCORE"	0x8031	$1937 $1938 $1939
Bookkeeping B.5 06 "30-29 M. SCORE"	0x8032	$193D $193E $193F
Bookkeeping B.5 07 "40-49 M. SCORE"	0x8033	$1943 $1944 $1945
Bookkeeping B.5 08 "50-69 M. SCORE"	0x8034	$1949 $194A $194B
Bookkeeping B.5 09 "70-99 M. SCORE"	0x8035	$194F $1950 $1951
Bookkeeping B.5 10 "100-149 M. SCORE"	0x8036	$1955 $1956 $1957

Bookkeeping B.5 11 "150-199 M. SCORE"	0x8037	$195B $195C $195D
Bookkeeping B.5 12 "200-299 M. SCORE"	0x8038	$1961 $1962 $1963
Bookkeeping B.5 13 "OVER 300 MILLION"	0x8039	$1967 $1968 $1969
Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"	0x803A	$196D $196E $196F
Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"	0x803B	$1973 $1974 $1975
Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"	0x803C	$1979 $197A $197B
Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"	0x803D	$197F $1980 $1981
Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"	0x803E	$1985 $1986 $1987
Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"	0x803F	$198B $198C $198D
Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"	0x8040	$1991 $1992 $1993
Bookkeeping B.5 21 "GAME TIME 4-5 M."	0x8041	$1997 $1998 $1999
Bookkeeping B.5 22 "GAME TIME 5-6 M."	0x8042	$199D $199E $199F
Bookkeeping B.5 23 "GAME TIME 6-8 M."	0x8043	$19A3 $19A4 $19A5
Bookkeeping B.5 24 "GAME TIME 8-10 M."	0x8044	$19A9 $19AA $19AB
Bookkeeping B.5 25 "GAME TIME 10-15 M"	0x8045	$19AF $19B0 $19B1
Bookkeeping B.5 26 "GAME TIME > 15 M"	0x8046	$19B5 $19B6 $19B7

Each of the bookkeeping statistics are cited in game code by the ID value shown in the table, above. Each statistic is stored in memory taking up 3 bytes of RAM to store the statistic value. The design of the L8.4 solution involving these bookkeeping statics will be described in detail, below.

Super Jackpot Lamp Enhancement for L8.4

With the game design considerations and goals in mind (see above), the resulting L8.4 enhancement may now be defined. In order to ensure no ambiguity exists in the following text, refer to the following playfield image for reference of lamp numbers and direction:



The following text will refer to lamp numbers and direction consistent with that shown above. *Some of the depicted code will refer to the 5 lamps with identifiers 0 through 4 corresponding to lamps #1 through #5. In such cases, readers should exercise careful examination when tracing through the code.*

Super Jackpot L8.4 Configuration

For L8.4 a new adjustment is added to allow the game operator to select the level of Super Jackpot difficulty. The adjustment provides the options as described in the table below.

The original L-8 lamp speed is 0.3125 seconds between lamp movements and is referred to as "Slow". For L8.4 a faster speed of 0.250 seconds between lamp movements may also be used and is referred to as "Fast" in the following text.

Setting	Internal Value	Lamp Speed	Starting Lamp	Starting Direction	Lamp Pattern (repeating)	Description
Original (default)	0 (0x00)	Slow	#5	Up	5-4-3-2-1-2-3-4-5	Original L-8 behavior.
Original+1	1 (0x01)	1st: Slow, else Fast	#5	Up	5-4-3-2-1-2-3-4-5	Same as "Original", except first Super Jackpot uses slow, subsequent uses fast lamps.
Original++	2 (0x02)	Fast	#5	Up	5-4-3-2-1-2-3-4-5	Same as "Original", except fast lamp speed for all Super Jackpots.
Medium	3 (0x03)	Slow	Varies	Varies	5-4-3-2-1-2-3-4-5 (variable starting lamp)	Same as "Original" except the starting lamp may be any of the 5 lamps. When starting lamp is #2, #3 or #4, the starting direction could be down or up. When the starting lamp is #5 the Super Jackpot attempt is 100% identical to "Original".
Medium+1	4 (0x04)	1st: Slow, else Fast	Varies	Varies	5-4-3-2-1-2-3-4-5 (variable starting lamp)	Same as "Medium", except first Super Jackpot uses slow, subsequent uses fast lamps.
Medium++	5 (0x05)	Fast	Varies	Varies	5-4-3-2-1-2-3-4-5 (variable starting lamp)	Same as "Medium", except fast lamp speed for all Super Jackpots.
Hard	6 (0x06)	Slow	Varies	Varies	5-4-3-2-1-2-3-4-5, or 1-2-3-4-5-1-2-3-4-5, or 5-4-3-2-1-5-4-3-2-1 (variable starting lamp)	Same as "Medium", except additional lamp patterns may occur. When the lamp pattern is 5-4-3-2-1-2-3-4-5 and the starting lamp is #5 the Super Jackpot attempt is 100% identical to "Original".
Hard+1	7 (0x07)	1st: Slow, else Fast	Varies	Varies	5-4-3-2-1-2-3-4-5, or 1-2-3-4-5-1-2-3-4-5, or 5-4-3-2-1-5-4-3-2-1 (variable starting lamp)	Same as "Hard", except first Super Jackpot uses slow, subsequent uses fast lamps.

Hard++	8 (0x08)	Fast	Varies	Varies	5-4-3-2-1-2-3-4-5, or 1-2-3-4-5-1-2-3-4-5, or 5-4-3-2-1-5-4-3-2-1 (variable starting lamp)	Same as "Hard", except fast lamp speed for all Super Jackpots.
Expert	9 (0x09)	Slow	Varies	Varies	5-4-3-2-1-2-3-4-5, or 1-2-3-4-5-1-2-3-4-5, or 5-4-3-2-1-5-4-3-2-1, or 1-3-5-2-4-1-3-5-2-4, or 4-2-5-3-1-4-2-5-3-1 (variable starting lamp)	Same as "Hard" but additional lamp patterns are added for extra challenge. There are two alternating patterns that a player would need to identify and carefully plan their aim.
Expert+1	10 (0x0A)	1st: Slow, else Fast	Varies	Varies	5-4-3-2-1-2-3-4-5, or 1-2-3-4-5-1-2-3-4-5, or 5-4-3-2-1-5-4-3-2-1, or 1-3-5-2-4-1-3-5-2-4, or 4-2-5-3-1-4-2-5-3-1 (variable starting lamp)	Same as "Expert", except first Super Jackpot uses slow, subsequent uses fast lamps.
Expert++	11 (0x0B)	Fast	Varies	Varies	5-4-3-2-1-2-3-4-5, or 1-2-3-4-5-1-2-3-4-5, or 5-4-3-2-1-5-4-3-2-1, or 1-3-5-2-4-1-3-5-2-4, or 4-2-5-3-1-4-2-5-3-1 (variable starting lamp)	Same as "Expert", except fast lamp speed for all Super Jackpots.

In summary, there are 4 major adjustments, summarized in the following table.

Difficulty Level	Summary
Original	Same as L-8, with fixed starting lamp #5.
Medium	Adds varying starting lamp and varying starting lamp direction (when starting lamp is #2, #3, #4).
Hard	Adds varying lamp patterns.
Expert	Adds additional, more challenging lamp patterns.

Each difficulty level has selectable lamp speeds as denoted by the suffixes summarized in the following table.

Difficulty Level Suffix	Summary
<no suffix>	Time between lamp movements is same as L-8, 0.3125 seconds for each lamp.
"+1"	For the first Super Jackpot (for each player) lamp movements are same as L-8 using 0.3125 seconds for each lamp. Once the first super jackpot is collected, subsequent Super Jackpots will use faster 0.250 seconds for each lamp.
"++"	Time between lamp movements is "fast", using 0.250 seconds for each lamp.

As shown in the tables above, the level of difficulty increases as higher adjustment selections are made. The various selections provide a variety of flexibility in how the Super Jackpot behaves to add extra challenge and excitement to the game play experience.

Super Jackpot Adjustment – L8.4 Code Changes for Adjustment Management

The adjustment table for L8.4 has extra room to accommodate an extra new adjustment. The table is updated as highlighted below.

Refer to the original text from L8.3 document describing *The L-8 Adjustments Memory Map*. The table with all adjustments is updated to accommodate the new L8-4 adjustment as highlighted in the *table portion*, below:

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

** New adjustments in L8.3 shown for reference.*

*** New L8.4 adjustment (additional new L8.4 adjustments are depicted later in this document)*

The *Feature Adjustments Metadata* table is updated as indicated in the abbreviated table content, below. Below is depicted the increased table size and new content. This is the final new adjustment added to L8.4 and, as such, the resulting table size is depicted below, 0x1F.

```
;----------------------------------------;------------------------------------------------------
7000: 00 1F                              ; Table entries is 1F (31 entries)
7002: 0C                                 ; Bytes per table entry is 0C (12 bytes)
...
70FF: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.21, Drptrgt 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 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.28, Cannon 1 Hit
715F: 00 00 00 00 00 15 00 01 00 7B 52 3A ; Feature Adjustments, A2.29, SS Autofire Time
716B: 00 00 00 00 00 0B 00 01 00 7C 2A 3A ; Feature Adjustments, A2.30, Super Jackpot
7177: 00 00 00 00 00 00 00 01 00 8E BC FF ; Feature Adjustments, A2.31, <placeholder>
;----------------------------------------;------------------------------------------------------
```

The metadata table, above, is located at $7000,3D, ROM offset 0x77000 and is updated with the new adjustment for Super Jackpot difficulty levels. The name of this new adjustment is added to the existing English, German, and French menu string tables as indicated in the table below. *Being consistent with L-8, the same English string for "SUPER JACKPOT" is being used for both German and French.*

	Table Start	New Pointer Bytes	New String Address	New String Value
English	$6700,3D	$673F,3D	$6909,3D	"SUPER JACKPOT"
German	$6A00,3D	$6A3F,3D	$6909,3D	"SUPER JACKPOT"
French	$6D00,3D	$6D3F,3D	$6909,3D	"SUPER JACKPOT"

The new adjustment, internally, tracks the various possible values as a simple numeric value that corresponds to the various settings as according to the "Internal Value" previously depicted in the table of Super Jackpot settings. As the adjustment contains a variety of possible values, a new function is defined in L8.4 to interpret the different values for display in the adjustment menu. To accommodate the display of these unique values, a new function is added at $7C2A,3A, ROM offset 0x6BC2A that the adjustment menu will use for displaying the various values. As shown in the updated adjustment table row, above, this function is cited in the last 3 bytes of its 12 byte entry as 0x7C, 0x2A, 0x3A.

Since the L8.4 changes are made with efforts to keep changes localized, the adjustment display handler needs to use its own method of displaying the various strings. This is because the master string table is not being updated for L8.4, so local strings are being referenced which means the common L-8 function for handling display of strings cannot easily be used. Refer to the new adjustment display handler below.

```
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; Adjustment handler for "Super Jackpot"
                                        ;
                                        ; A has adjustment index
                                        ; B has code, 0x02 means write adj string to pointer Y
                                        ; U has the current value
                                        ; Y has addr where to put ASCII string (when B is 0x02)
                                        ;
                                        ;
7C2A: BD 79 E2        JSR   $79E2       ; AdjustmentWriteASCIIValueCclear()
7C2D: 25 02           BCS   $7C31       ; If C-bit is set, RTS now.  Don't write string to Y.
7C2F: 8D 01           BSR   $7C32       ; Print the adjustment string for value U
7C31: 39              RTS               ;
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
7C32: 34 36           PSHS  Y,X,B,A     ;
7C34: 1F 30           TFR   U,D         ; D now has the adjustment value
                                        ;
7C36: 8D 08           BSR   $7C40       ; Load X with pointer for current adj value D
7C38: BD 7B 5B        JSR   $7B5B       ; Advance X for current language setting
7C3B: BD B9 51        JSR   $B951       ; CopyASCIIStringFromXtoYandVerifyLength()
                                        ;
7C3E: 35 B6           PULS  A,B,X,Y,PC  ;
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ; B has adjustment value.  Using it, load X with
                                        ; address of pointer to English string from table, below.
```

```
                                        ;
7C40: C1 0C           CMPB  #$0C        ; Check that B is not out of bounds
7C42: 25 01           BCS   $7C45       ; If out of bounds
7C44: 5F              CLRB              ; then reset to 0 "Original" setting.
                                        ;
7C45: 86 06           LDA   #$06        ; 6 bytes per set of pointers for each adjustment
7C47: 3D              MUL               ; B now has result of multiplication
7C48: 8E 7C 4E        LDX   #$7C4E      ; X gets Base address of pointers starting at "Original"
7C4B: 30 8B           LEAX  D,X         ; X now gets advanced to English pointer for adjustment
7C4D: 39              RTS               ;
                                        ;
----------------------------------------;-------------------------------------------------------
                                        ;
                                        ; The first 3 point to existing string in L8.4 ROM
7C4E: 7B 94                             ; English  "ORIGINAL"  // Original, slow
7C50: 7B 94                             ; German   "ORIGINAL"  // Original, slow
7C52: 7B 9D                             ; French   "ORIGINALE" // Original, slow
7C54: 7C 96                             ; English  "ORIG.+1"   // Original, 1st slow, rest fast
7C56: 7C 96                             ; German   "ORIG.+1"   // Original, 1st slow, rest fast
7C58: 7C 96                             ; French   "ORIG.+1"   // Original, 1st slow, rest fast
7C5A: 7C 9E                             ; English  "ORIG.++"   // Original, fast
7C5C: 7C 9E                             ; German   "ORIG.++"   // Original, fast
7C5E: 7C 9E                             ; French   "ORIG.++"   // Original, fast
7C60: 7C A6                             ; English  "MEDIUM"    // Medium, slow
7C62: 7C AD                             ; German   "MITTEL"    // Medium, slow
7C64: 7C B4                             ; French   "MOYEN"     // Medium, slow
7C66: 7C BA                             ; English  "MEDIUM+1"  // Medium, 1st slow, rest fast
7C68: 7C C3                             ; German   "MITTEL+1"  // Medium, 1st slow, rest fast
7C6A: 7C CC                             ; French   "MOYEN+1"   // Medium, 1st slow, rest fast
7C6C: 7C D4                             ; English  "MEDIUM++"  // Medium, fast
7C6E: 7C DD                             ; German   "MITTEL++"  // Medium, fast
7C70: 7C E6                             ; French   "MOYEN++"   // Medium, fast
7C72: 7C EE                             ; English  "HARD"      // Hard, slow
7C74: 7C F3                             ; German   "SCHWER"    // Hard, slow
7C76: 7C FA                             ; French   "DIFFICILE" // Hard, slow
7C78: 7D 04                             ; English  "HARD+1"    // Hard, 1st slow, rest fast
7C7A: 7D 0B                             ; German   "SCHWER+1"  // Hard, 1st slow, rest fast
7C7C: 7D 14                             ; French   "DIFF/LE+1" // Hard, 1st slow, rest fast
7C7E: 7D 1E                             ; English  "HARD++"    // Hard, fast
7C80: 7D 25                             ; German   "SCHWER++"  // Hard, fast
7C82: 7D 2E                             ; French   "DIFF/LE++" // Hard, fast
7C84: 7D 38                             ; English  "EXPERT"    // Expert, slow
7C86: 7D 3F                             ; German   "EXPERTE"   // Expert, slow
7C88: 7D 3F                             ; French   "EXPERTE"   // Expert, slow
7C8A: 7D 47                             ; English  "EXPERT+1"  // Expert, 1st slow, rest fast
7C8C: 7D 50                             ; German   "EXPERTE+1" // Expert, 1st slow, rest fast
7C8E: 7D 50                             ; French   "EXPERTE+1" // Expert, 1st slow, rest fast
7C90: 7D 5A                             ; English  "EXPERT++"  // Expert, fast
7C92: 7D 63                             ; German   "EXPERTE++" // Expert, fast
7C94: 7D 63                             ; French   "EXPERTE++" // Expert, fast
                                        ;
7C96: 4F 52 49 47 2E 2B 31 00           ; "ORIG.+1"   // Original, 1st slow, rest fast En,Gm,Fr
7C9E: 4F 52 49 47 2E 2B 2B 00           ; "ORIG.++"   // Original, 1st slow, rest fast En,Gm,Fr
7CA6: 4D 45 44 49 55 4D 00              ; "MEDIUM"    // Medium, slow (English)
7CAD: 4D 49 54 54 45 4C 00              ; "MITTEL"    // Medium, slow (German)
7CB4: 4D 4F 59 45 4E 00                 ; "MOYEN"     // Medium, slow (French)
7CBA: 4D 45 44 49 55 4D 2B 31 00        ; "MEDIUM+1"  // Medium, 1st slow, rest fast (English)
7CC3: 4D 49 54 54 45 4C 2B 31 00        ; "MITTEL+1"  // Medium, 1st slow, rest fast (German)
7CCC: 4D 4F 59 45 4E 2B 31 00           ; "MOYEN+1"   // Medium, 1st slow, rest fast (French)
7CD4: 4D 45 44 49 55 4D 2B 2B 00        ; "MEDIUM++"  // Medium, fast (English)
7CDD: 4D 49 54 54 45 4C 2B 2B 00        ; "MITTEL++"  // Medium, fast (German)
7CE6: 4D 4F 59 45 4E 2B 2B 00           ; "MOYEN++"   // Medium, fast (French)
```

```
7CEE: 48 41 52 44 00                  ; "HARD"      // Hard, slow (English)
7CF3: 53 43 48 57 45 52 00            ; "SCHWER"    // Hard, slow (German)
7CFA: 44 49 46 46 49 43 49 4C 45 00   ; "DIFFICILE" // Hard, slow (French)
7D04: 48 41 52 44 2B 31 00            ; "HARD+1"    // Hard, 1st slow, rest fast (English)
7D0B: 53 43 48 57 45 52 2B 31 00      ; "SCHWER+1"  // Hard, 1st slow, rest fast (German)
7D14: 44 49 46 46 2F 4C 45 2B 31 00   ; "DIFF/LE+1" // Hard, 1st slow, rest fast (French)
7D1E: 48 41 52 44 2B 2B 00            ; "HARD++"    // Hard, fast (English)
7D25: 53 43 48 57 45 52 2B 2B 00      ; "SCHWER++"  // Hard, fast (German)
7D2E: 44 49 46 46 2F 4C 45 2B 2B 00   ; "DIFF/LE++" // Hard, fast (French)
7D38: 45 58 50 45 52 54 00            ; "EXPERT"    // Expert, slow (English)
7D3F: 45 58 50 45 52 54 45 00         ; "EXPERTE"   // Expert, slow (German and French)
7D47: 45 58 50 45 52 54 2B 31 00      ; "EXPERT+1"  // Expert, 1st slow, rest fast (English)
7D50: 45 58 50 45 52 54 45 2B 31 00   ; "EXPERTE+1" // Expert, 1st slow, rest fast (German,Fr)
7D5A: 45 58 50 45 52 54 2B 2B 00      ; "EXPERT++"  // Expert, fast (English)
7D63: 45 58 50 45 52 54 45 2B 2B 00   ; "EXPERTE++" // Expert, fast (German,French)
                                      ;
--------------------------------------;-------------------------------------------------------
```

The code, above, for displaying the adjustment string value is specific only for this new adjustment. The detailed flow through the above code is left as an exercise to the reader.

Super Jackpot Lamp Movement – Original L-8 Code

The Super Jackpot lamp movement is managed by scheduled callback function ID 1033 which is launched from $58B7,34 with the following instructions:

```
58B7: BD 89 48    JSR   $8948         ; Scheduler function
58BA: 59 26 34                        ; Schedules function that cycles the arrow
```

This particular scheduler function assigns the ID of the scheduled function to the current function that is invoking the scheduler function. In this case the function ID is 1033 which is the ID used by the Super Jackpot arrow cycler function. The arrow cycler function is at $5926,34, ROM offset 0x51926 and is shown below for reference.

```
--------------------------------------;-------------------------------------------------------
                                      ;
                                      ; ID 1033
                                      ;
                                      ; This function maintains the super-jackpot lit arrow
                                      ;
5926: 86 FF       LDA   #$FF          ; Store 0xFF into $D3
5928: 97 D3       STA   $D3           ; Checked during huntership hit from $4BE3,31
592A: BD 87 15    JSR   $8715         ; LampOnParameterByte1PlaneParameterByte2()
592D: 15                              ; Lamp 15, Target 5 Low
592E: 18                              ; Solid lamps plane (?)
592F: 86 04       LDA   #$04          ; Store 0x04 into $05F9 (5th lamp)
5931: B7 05 F9    STA   $05F9         ; Checked during huntership hit from $4BE9,31
                                      ;
5934: BD 83 46    JSR   $8346         ; --\--\ Sleep()
5937: 14                              ;   |  | 0.3125 seconds
5938: BD FD B4    JSR   $FDB4         ;   |  |
593B: 18 18                           ;   |  | This lights the previous lamp in the 5-bank
593D: 7A 05 F9    DEC   $05F9         ;   |  |
5940: B6 05 F9    LDA   $05F9         ;   |  |
5943: 4D          TSTA                ;   |  |
5944: 26 EE       BNE   $5934         ;   |--/
                                      ;   |
5946: BD 83 46    JSR   $8346         ;   |--\ Sleep()
```

```
5949: 14                                ;   |  | 0.3125 seconds
594A: BD FD A2    JSR   $FDA2           ;   |  |
594D: 18 18                             ;   |  | This lights the next lamp in the 5-bank
594F: 7C 05 F9    INC   $05F9           ;   |  |
5952: B6 05 F9    LDA   $05F9           ;   |  |
5955: 81 04       CMPA  #$04            ;   |  |
5957: 25 ED       BCS   $5946           ;   |--/
                                        ;   |
5959: 20 D9       BRA   $5934           ; --/
                                        ;
----------------------------------------;-------------------------------------------------------
```

The L-8 function, above will be relocated and expanded in L8.4 to accommodate the various adjustment settings in order to move the arrow in the way set in the new adjustment.

Super Jackpot Lamp Movement – Bookkeeping Statistics To Derive Variable Lamp Behaviors

As described earlier, the bookkeeping statistics can be used in L8.4 to determine a starting number that applies to the current game in play (regardless if it is single player or multi-player game). The starting number can then be used to establish a particular variable behavior such as starting lamp or starting lamp direction (applicable when starting lamp is #2, #3 or #4).

The way in which the base values for the variable behaviors is derived, is shown in the table below.

Bookkeeping Entry	**Addr**	**Bookkeeping Statistic Participation**		
		Starting Lamp	Starting Direction*	Lamp Pattern**
Bookkeeping B.3 08 "MATCH AWARDS"	$18A3	✓		
Bookkeeping B.3 28 "1 PLAYER GAMES"	$18EB	✓	✓	✓
Bookkeeping B.3 29 "2 PLAYER GAMES"	$18F1	✓	✓	✓
Bookkeeping B.3 30 "3 PLAYER GAMES"	$18F7	✓	✓	✓
Bookkeeping B.3 31 "4 PLAYER GAMES"	$18FD	✓	✓	✓
Bookkeeping B.5 01 "0-1.9 M. SCORE"	$1921	✓		✓
Bookkeeping B.5 02 "2-4.9 M. SCORE"	$1927		✓	✓
Bookkeeping B.5 03 "3-9.9 M. SCORE"	$192D	✓		
Bookkeeping B.5 04 "10-19 M. SCORE"	$1933		✓	
Bookkeeping B.5 05 "20-29 M. SCORE"	$1939	✓		✓
Bookkeeping B.5 06 "30-29 M. SCORE"	$193F		✓	✓
Bookkeeping B.5 07 "40-49 M. SCORE"	$1945	✓		
Bookkeeping B.5 08 "50-69 M. SCORE"	$194B		✓	
Bookkeeping B.5 09 "70-99 M. SCORE"	$1951	✓		✓
Bookkeeping B.5 10 "100-149 M. SCORE"	$1957		✓	✓
Bookkeeping B.5 11 "150-199 M. SCORE"	$195D	✓		
Bookkeeping B.5 12 "200-299 M. SCORE"	$1963		✓	
Bookkeeping B.5 13 "OVER 300 MILLION"	$1969	✓		✓
Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"	$196F		✓	✓
Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"	$1975	✓		
Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"	$197B		✓	
Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"	$1981	✓		✓

Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"	$1987		✓	✓
Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"	$198D	✓		
Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"	$1993		✓	
Bookkeeping B.5 21 "GAME TIME 4-5 M."	$1999	✓		✓
Bookkeeping B.5 22 "GAME TIME 5-6 M."	$199F		✓	✓
Bookkeeping B.5 23 "GAME TIME 6-8 M."	$19A5	✓		
Bookkeeping B.5 24 "GAME TIME 8-10 M."	$19AB		✓	
Bookkeeping B.5 25 "GAME TIME 10-15 M"	$19B1	✓		✓
Bookkeeping B.5 26 "GAME TIME > 15 M"	$19B7		✓	✓

* Variable starting direction only applies when the starting lamp is #2, #3 or #4

** Variable lamp pattern only applies when adjustment is set to have multiple patterns.

Each statistic is stored in a 3-byte region however only the last of the 3 bytes is needed for purposes of deriving a unique number for the current game in play. The table, above, shows how each statistic is used in deriving the base number for the variable behavior indicated. By using the bookkeeping values in this way, it results in a seemingly random behavior which is difficult for the typical player to predict.

The “seed” value is determined by taking the sum of all bookkeeping statistics for the given item. To add different behavior at subsequent Super Jackpots, the number of current player’s acquired Super Jackpot awards is also added to this sum. To reduce complexity the sum is accumulated into an 8-bit register, discarding any overflow that might take place.

Once the total sum is determined, the resulting 8-bit value is then used as an index into an array of data bytes to acquire a resulting byte value from which the final value will be determined. The array of bytes is simply ROM bytes that normally represents code, however in this case it can be treated as “random” data bytes to serve the purpose of getting a variable value at each game (and after each Super Jackpot awarded to the player).

Offset:	Bytes:	ANSI Text:
0007FC80	80 C6 16 3D C3 0A 60 1F 03 39 34 76 BD EF E8 FE	€Æ┬=Ã ` └94v½ïèþ
0007FC90	0B 10 27 09 10 BE 17 99 BD D3 C4 20 03 BD D3 60	♂┼' ┼¾┤™½ÓÄ └½Ó`
0007FCA0	CE 0A 60 A6 C4 27 0A 10 AE 45 AE 42 EC C4 BD EF	Î ` ¦Ä' ┼®E®BìÄ½ï
0007FCB0	FC 33 C8 16 11 83 0B 10 25 E9 35 F6 FC 2C FC 2E	ü3È┬◄ƒ♂┼%é5öü,ü.
0007FCC0	FC 4A FC 50 FC 5F FC 6A FC 75 FC 7D 34 10 8E 03	üJüPü_üjüuü}4┼Ž└
0007FCD0	86 8D 02 35 90 34 76 1F 13 1F 32 E6 80 27 16 C1	†▯┐5▯4v ‼ 2æ€'┬Á
0007FCE0	20 27 F8 E7 C0 1F 32 E6 80 27 0A E7 C0 C1 20 27	'øçÀ 2æ€' çÀÁ '
0007FCF0	02 1F 32 20 F2 6F A4 35 76 1C FE 6D 84 27 02 1A	┐ 2 òo¤5v þm„'┐→
0007FD00	01 39 34 02 86 FF B7 3F BD 7F 3F F8 35 02 3B 34	┌94┐†ÿ·?½▯?ø5┐;4
0007FD10	16 BE 0C C7 BC 0C C5 27 17 F6 3E 66 C5 02 27 10	┬¾□Ç¼□Å'┤ö>fÅ┐'┼
0007FD20	A6 80 B7 3E 67 8C 0C C1 25 03 8E 0C 71 BF 0C C7	¦€·>gŒ□Á%└Ž□q¿□Ç
0007FD30	F6 3E 66 C5 01 27 18 B6 3E 67 BE 0C C1 A7 80 8C	ö>fÅ┌'↑¶>g¾□Á§€Œ
0007FD40	0C 71 25 03 8E 0C 21 BC 0C C3 27 03 BF 0C C1 20	□q%└Ž□!¼□Ã'└¿□Á
0007FD50	47 34 16 BE 0C C7 BC 0C C5 27 1E F6 3F C0 C5 20	G4┬¾□Ç¼□Å' ö?ÀÅ
0007FD60	26 17 F6 3F C4 C5 01 27 10 A6 80 B7 3F C3 8C 0C	&┤ö?ÄÅ┌'┼¦€·?ÃŒ□
0007FD70	C1 25 03 8E 0C 71 BF 0C C7 B6 3F C4 85 02 27 18	Á%└Ž□q¿□Ç¶?Ä…┐'↑

At WPC address $FD00, ROM offset 0x7FD00, the starting point for the array of data bytes has been arbitrarily chosen for purposes of acquiring the "random" data. By using the signed 8-bit seed value as the index into this data, there are 256 bytes from where the resulting data byte will be used. Being a signed value, this means the 128 bytes prior to the $FD00 and the 127 bytes after $FD00 are used, depending on the seed value. This results in an effective range of data of $FC80 through $FD7F.

The selected byte from the 256 bytes is then subjected to a formula to determine the value that will be used for the given variable feature behavior. The table below shows the formula used for each behavior.

Variable Behavior	Needed Values		Formula to derive value X from byte B (C-like syntax)
	Range	# of values	
Starting Lamp	0..4	5	X = (B & 0x07); X = (X > 4) ? (X – 4) : X
Starting Direction	0..1	2	X = B & 0x01
Lamp Pattern (Hard)	0..2	3	X = (B & 0x03); X = (X > 2) ? (X – 2): X
Lamp Pattern (Expert)	0..4	5	X = ((B >> 4) & 0x07); X = (X > 4) ? (X – 4) : X

The resulting set of numbers/values for each of the variable behaviors is shown in the table below. The information in this table was derived in order to confirm that a seemingly random mix of all possible value are in the pool of numbers that will be used in L8.4.

Variable Behavior	Formula Results	Resulting Behavior
Starting Lamp **Formula Results define which lamp number is start of Super Jackpot:** **#1 Top lamp** **…** **#5 Bottom Lamp**	0 2 2 1 3 2 0 3 3 1 4 2 1 3 0 2	#1 #3 #3 #2 #4 #3 #1 #4 #4 #2 #5 #3 #2 #4 #1 #3
	3 0 3 1 0 2 3 1 1 3 4 0 3 1 3 0	#4 #1 #4 #2 #1 #3 #4 #2 #2 #4 #5 #1 #4 #2 #4 #1
	2 2 0 2 4 3 2 0 2 1 2 2 4 4 1 3	#3 #3 #1 #3 #5 #4 #3 #1 #3 #2 #3 #3 #5 #5 #2 #4
	4 3 0 2 1 3 3 0 1 1 1 2 4 4 4 2	#5 #4 #1 #3 #2 #4 #4 #1 #2 #2 #2 #3 #5 #5 #5 #3
	4 2 4 0 4 3 4 2 4 1 4 1 4 0 2 3	#5 #3 #5 #1 #5 #4 #5 #3 #5 #2 #5 #2 #5 #1 #3 #4
	2 1 2 1 0 4 2 3 3 3 2 2 0 3 2 1	#3 #2 #3 #2 #1 #5 #3 #4 #4 #4 #3 #3 #1 #4 #3 #2
	0 3 0 3 0 3 2 2 0 3 2 3 0 1 0 3	#1 #4 #1 #4 #1 #4 #3 #3 #1 #4 #3 #4 #1 #2 #1 #4
	2 3 2 0 2 3 4 1 2 4 2 1 4 3 2 2	#3 #4 #3 #1 #3 #4 #5 #2 #3 #5 #3 #2 #5 #4 #3 #3
	1 1 4 2 2 3 3 3 1 3 3 0 1 2 3 4	#2 #2 #5 #3 #3 #4 #4 #4 #2 #4 #4 #1 #2 #3 #4 #5
	2 2 4 3 4 4 1 3 3 2 2 2 1 2 3 0	#3 #3 #5 #4 #5 #5 #2 #4 #4 #3 #3 #3 #2 #3 #4 #1
	2 0 3 2 3 4 4 1 1 3 2 4 1 3 4 3	#3 #1 #4 #3 #4 #5 #5 #2 #2 #4 #3 #5 #2 #4 #5 #4
	2 2 2 1 1 3 0 2 2 3 2 4 1 3 0 4	#3 #3 #3 #2 #2 #4 #1 #3 #3 #4 #3 #5 #2 #4 #1 #5
	4 1 1 3 2 4 1 4 4 3 3 3 3 4 1 0	#5 #2 #2 #4 #3 #5 #2 #5 #5 #4 #4 #4 #4 #5 #2 #1
	3 4 2 2 4 3 4 4 1 3 2 2 3 0 1 0	#4 #5 #3 #3 #5 #4 #5 #5 #2 #4 #3 #3 #4 #1 #2 #1
	2 3 2 3 4 1 1 3 0 2 0 3 3 3 4 4	#3 #4 #3 #4 #5 #2 #2 #4 #1 #3 #1 #4 #4 #4 #5 #5
	1 1 3 2 4 1 3 4 3 2 3 4 1 2 3 0	#2 #2 #4 #3 #5 #2 #4 #5 #4 #3 #4 #5 #2 #3 #4 #1
Starting Direction **Formula Results define which direction the lamp will initially move when starting lamp is #2, #3 or #4.**	0 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0	D D D U U D D U U U D D U U D D
	1 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0	U D U U D D U U U U D D U U U D
	0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1	D D D D D U D D D U D D D D U U
	0 1 0 0 1 1 1 0 1 1 1 0 0 0 0 0	D U D D U U U D U U U D D D D D
	0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1	D D D D D U D D D U D U D D D U
	0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1	D U D U D D D U U U D D D U D U
	0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1	D U D U D U D D D U D U D U D U
	0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0	D U D D D U D U D D D U D U D D
	1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 0	U U D D D U U U U U U D U D U D

U = Up **D = Down**	0 0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0	D D D U D D U U U D D D U D U D D D U D U D D U U U D D U U D U D D D U U U D D D U D D U U D D D U U U D D U D D U U U U D U D U D D D D U D D U U D D U D U D D U D U D U U U D D D U U U D D U U U D D U U D U D U D U D U D
Lamp Pattern (Hard) **Formula Results define which lamp pattern will be used during "Hard" difficulty settings.** **#1 = 543212345** **#2 = 1234512345** **#3 = 5432154321** **Starting lamp in the pattern will vary.**	0 2 2 1 1 2 0 1 1 1 0 2 1 1 0 2 1 0 1 1 0 2 1 1 1 1 0 0 1 1 1 0 2 2 0 2 0 1 2 0 2 1 2 2 0 0 1 1 0 1 0 2 1 1 1 0 1 1 1 2 0 0 0 2 0 2 0 0 0 1 0 2 0 1 0 1 0 0 2 1 2 1 2 1 0 0 2 1 1 1 2 2 0 1 2 1 0 1 0 1 0 1 2 2 0 1 2 1 0 1 0 1 2 1 2 0 2 1 0 1 2 0 2 1 0 1 2 2 1 1 0 2 2 1 1 1 1 1 1 0 1 2 1 0 2 2 0 1 0 0 1 1 1 2 2 2 1 2 1 0 2 0 1 2 1 0 0 1 1 1 2 0 1 1 0 1 2 2 2 1 1 1 0 2 2 1 2 0 1 1 0 0 0 1 1 1 2 0 1 0 0 1 1 1 1 0 1 0 1 0 2 2 0 1 0 0 1 1 2 2 1 0 1 0 2 1 2 1 0 1 1 1 0 2 0 1 1 1 0 0 1 1 1 2 0 1 1 0 1 2 1 0 1 2 1 0	#1 #3 #3 #2 #2 #3 #1 #2 #2 #2 #2 #1 #3 #2 #2 #1 #3 #2 #1 #2 #2 #1 #3 #2 #2 #2 #2 #1 #1 #2 #2 #2 #1 #3 #3 #1 #3 #1 #2 #3 #1 #3 #2 #3 #3 #1 #1 #2 #2 #1 #2 #1 #3 #2 #2 #2 #1 #2 #2 #2 #3 #1 #1 #1 #3 #1 #3 #1 #1 #1 #2 #1 #3 #1 #2 #1 #2 #1 #1 #3 #2 #3 #2 #3 #2 #1 #1 #3 #2 #2 #2 #3 #3 #1 #2 #3 #2 #1 #2 #1 #2 #1 #2 #3 #3 #1 #2 #3 #2 #1 #2 #1 #2 #3 #2 #3 #1 #3 #2 #1 #2 #3 #1 #3 #2 #1 #2 #3 #3 #2 #2 #1 #3 #3 #2 #2 #2 #2 #2 #2 #1 #2 #3 #2 #1 #3 #3 #1 #2 #1 #1 #2 #2 #2 #3 #3 #3 #2 #3 #2 #1 #3 #1 #2 #3 #2 #1 #1 #2 #2 #2 #3 #1 #2 #2 #1 #2 #3 #3 #3 #2 #2 #2 #1 #3 #3 #2 #3 #1 #2 #2 #1 #1 #1 #2 #2 #2 #3 #1 #2 #1 #1 #2 #2 #2 #2 #1 #2 #1 #2 #1 #3 #3 #1 #2 #1 #1 #2 #2 #3 #3 #2 #1 #2 #1 #3 #2 #3 #2 #1 #2 #2 #2 #1 #3 #1 #2 #2 #2 #1 #1 #2 #2 #2 #3 #1 #2 #2 #1 #2 #3 #2 #1 #2 #3 #2 #1
Lamp Pattern(Expert) **Formula Results define which lamp pattern will be used during "Expert" difficulty settings.** **#1 = 543212345** **#2 = 1234512345** **#3 = 5432154321** **#4 = 1352413524** **#5 = 4253142531** **Starting lamp in the pattern will vary.**	0 4 1 3 4 0 2 1 0 3 3 3 3 2 2 3 0 1 2 0 1 3 1 1 3 1 4 2 0 3 1 2 4 0 2 2 4 2 0 1 2 4 2 4 2 4 3 2 3 3 4 1 1 0 0 1 2 2 3 3 3 2 3 2 3 4 3 1 3 1 3 2 3 3 3 3 3 1 0 0 0 0 0 3 1 3 3 1 1 1 3 2 0 2 1 4 2 2 3 2 4 1 3 2 0 2 0 2 4 4 2 2 0 1 3 2 3 2 2 3 3 1 3 2 0 2 0 1 0 3 3 0 0 3 3 3 3 3 3 3 3 0 3 3 1 3 0 4 3 0 4 2 1 3 3 2 4 0 2 1 2 0 3 3 2 0 0 4 2 0 0 0 3 3 0 4 3 3 2 4 0 2 1 3 3 2 3 0 4 2 0 0 0 3 2 0 0 0 2 3 0 4 2 0 3 0 4 2 4 3 1 3 0 4 3 0 4 2 1 3 3 4 4 2 2 1 3 3 4 4 0 2 1 2 0 3 3 4 0 0 4 2 0 0 0 3 3 0 4 3 3 4 0 0 2 1	#1 #5 #2 #4 #5 #1 #3 #2 #1 #4 #4 #4 #4 #3 #3 #4 #1 #2 #3 #1 #2 #4 #2 #2 #4 #2 #5 #3 #1 #4 #2 #3 #5 #1 #3 #3 #5 #3 #1 #2 #3 #5 #3 #5 #3 #5 #4 #3 #4 #4 #5 #2 #2 #1 #1 #2 #3 #3 #4 #4 #4 #3 #4 #3 #4 #5 #4 #2 #4 #2 #4 #3 #4 #4 #4 #4 #4 #2 #1 #1 #1 #1 #1 #4 #2 #4 #4 #2 #2 #2 #4 #3 #1 #3 #2 #5 #3 #3 #4 #3 #5 #2 #4 #3 #1 #3 #1 #3 #5 #5 #3 #3 #1 #2 #4 #3 #4 #3 #3 #4 #4 #2 #4 #3 #1 #3 #1 #2 #1 #4 #4 #1 #1 #4 #4 #4 #4 #4 #4 #4 #4 #1 #4 #4 #2 #4 #1 #5 #4 #1 #5 #3 #2 #4 #4 #3 #5 #1 #3 #2 #3 #1 #4 #4 #3 #1 #1 #5 #3 #1 #1 #1 #4 #4 #1 #5 #4 #4 #3 #5 #1 #3 #2 #4 #4 #3 #4 #1 #5 #3 #1 #1 #1 #4 #3 #1 #1 #1 #3 #4 #1 #5 #3 #1 #4 #1 #5 #3 #5 #4 #2 #4 #1 #5 #4 #1 #5 #3 #2 #4 #4 #5 #5 #3 #3 #2 #4 #4 #5 #5 #1 #3 #2 #3 #1 #4 #4 #5 #1 #1 #5 #3 #1 #1 #1 #4 #4 #1 #5 #4 #4 #5 #1 #1 #3 #2

The analysis, above, was done to show there is a variable set of possible values and behaviors that will occur by using this mechanism. Since each "random" value for each variable behavior depends on a different set of game bookkeeping statistics, the actual chosen value is not easily for a player to predict and, as such, results in a seemingly random behavior as the result.

Super Jackpot Code Updates for L8.4

The original code for Super Jackpot code scheduler is updated as follows. At $58B7,34, ROM offset 0x518B7, where L-8 code previously scheduled function $5926,34, it is now updated to schedule the new L8.4 function located at $7E95,3A. The new byte values highlighted below.

```
58B7: BD 89 48    JSR   $8948           ; Scheduler function
58BA: 7E 95 3A                         ; Schedules New L8.4 function for Super Jackpot
```

As shown, the old Super Jackpot vector $5926,34 has been replaced with $7E95,3A. In addition to this jump point change, the original L-8 Super Jackpot code at $5926,34 (ROM offset 0x51926) has been removed with its bytes set to 0xFF, with such ROM region available for other code in the future.

The new L8.4 Super Jackpot code at $7E95,3A (ROM offset 0x6BE95) is the entry point for the L8.4 Super Jackpot code. It is added to unused region of ROM toward the end of bank $3A where a lot of the other new L8.4 code is also located. There are various helper functions and data tables that are also part of this portion of L8.4 code. The entire block of code for Super Jackpot is from $7D6D,3A through $7F47,3A and is depicted below.

The code, below, handles the entirety of L8.4 Super Jackpot, supporting all of the Super Jackpot adjustment settings, fast and slow speed lamp movements, and the different possible lamp patterns. A detailed description of this logic is is being left as an exercise to the reader. For tracing the code, the entry point at $7E95,3A.

```
----------------------------------------;------------------------------------------------------
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Starting Lamp Position
                                        ;------------------------
                                        ; Table of Bookkeeping Storage Locations (L-8)
                                        ;
7D6D: 18 A3                             ; Bookkeeping B.3 08 "MATCH AWARDS"
7D6F: 18 EB                             ; Bookkeeping B.3 28 "1 PLAYER GAMES"
7D71: 18 F1                             ; Bookkeeping B.3 29 "2 PLAYER GAMES"
7D73: 18 F7                             ; Bookkeeping B.3 30 "3 PLAYER GAMES"
7D75: 18 FD                             ; Bookkeeping B.3 31 "4 PLAYER GAMES"
7D77: 19 21                             ; Bookkeeping B.5 01 "0-1.9 M. SCORE"
7D79: 19 2D                             ; Bookkeeping B.5 03 "3-9.9 M. SCORE"
7D7B: 19 39                             ; Bookkeeping B.5 05 "20-29 M. SCORE"
7D7D: 19 45                             ; Bookkeeping B.5 07 "40-49 M. SCORE"
7D7F: 19 51                             ; Bookkeeping B.5 09 "70-99 M. SCORE"
7D81: 19 5D                             ; Bookkeeping B.5 11 "150-199 M. SCORE"
7D83: 19 69                             ; Bookkeeping B.5 13 "OVER 300 MILLION"
7D85: 19 75                             ; Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"
7D87: 19 81                             ; Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"
7D89: 19 8D                             ; Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"
7D8B: 19 99                             ; Bookkeeping B.5 21 "GAME TIME 4-5 M."
7D8D: 19 A5                             ; Bookkeeping B.5 23 "GAME TIME 6-8 M."
7D8F: 19 B1                             ; Bookkeeping B.5 25 "GAME TIME 10-15 M"
7D91: 00 00                             ; <End of data>
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Starting Lamp Direction
                                        ;-------------------------
                                        ; Table of Bookkeeping Storage Locations (L-8)
                                        ;
7D93: 18 EB                             ; Bookkeeping B.3 28 "1 PLAYER GAMES"
7D95: 18 F1                             ; Bookkeeping B.3 29 "2 PLAYER GAMES"
7D97: 18 F7                             ; Bookkeeping B.3 30 "3 PLAYER GAMES"
```

```
7D99: 18 FD                         ; Bookkeeping B.3 31 "4 PLAYER GAMES"
7D9B: 19 27                         ; Bookkeeping B.5 02 "2-4.9 M. SCORE"
7D9D: 19 33                         ; Bookkeeping B.5 04 "10-19 M. SCORE"
7D9F: 19 3F                         ; Bookkeeping B.5 06 "30-29 M. SCORE"
7DA1: 19 4B                         ; Bookkeeping B.5 08 "50-69 M. SCORE"
7DA3: 19 57                         ; Bookkeeping B.5 10 "100-149 M. SCORE"
7DA5: 19 63                         ; Bookkeeping B.5 12 "200-299 M. SCORE"
7DA7: 19 6F                         ; Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"
7DA9: 19 7B                         ; Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"
7DAB: 19 87                         ; Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"
7DAD: 19 93                         ; Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"
7DAF: 19 9F                         ; Bookkeeping B.5 22 "GAME TIME 5-6 M."
7DB1: 19 AB                         ; Bookkeeping B.5 24 "GAME TIME 8-10 M."
7DB3: 19 B7                         ; Bookkeeping B.5 26 "GAME TIME > 15 M"
7DB5: 00 00                         ; <End of data>
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Starting Lamp Pattern
                                    ;-----------------------
                                    ; Table of Bookkeeping Storage Locations (L-8)
                                    ;
7DB7: 18 EB                         ; Bookkeeping B.3 28 "1 PLAYER GAMES"
7DB9: 18 F1                         ; Bookkeeping B.3 29 "2 PLAYER GAMES"
7DBB: 18 F7                         ; Bookkeeping B.3 30 "3 PLAYER GAMES"
7DBD: 18 FD                         ; Bookkeeping B.3 31 "4 PLAYER GAMES"
7DBF: 19 21                         ; Bookkeeping B.5 01 "0-1.9 M. SCORE"
7DC1: 19 27                         ; Bookkeeping B.5 02 "2-4.9 M. SCORE"
7DC3: 19 39                         ; Bookkeeping B.5 05 "20-29 M. SCORE"
7DC5: 19 3F                         ; Bookkeeping B.5 06 "30-29 M. SCORE"
7DC7: 19 51                         ; Bookkeeping B.5 09 "70-99 M. SCORE"
7DC9: 19 57                         ; Bookkeeping B.5 10 "100-149 M. SCORE"
7DCB: 19 69                         ; Bookkeeping B.5 13 "OVER 300 MILLION"
7DCD: 19 6F                         ; Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"
7DCF: 19 81                         ; Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"
7DD1: 19 87                         ; Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"
7DD3: 19 99                         ; Bookkeeping B.5 21 "GAME TIME 4-5 M."
7DD5: 19 9F                         ; Bookkeeping B.5 22 "GAME TIME 5-6 M."
7DD7: 19 B1                         ; Bookkeeping B.5 25 "GAME TIME 10-15 M"
7DD9: 19 B7                         ; Bookkeeping B.5 26 "GAME TIME > 15 M"
7DDB: 00 00                         ; <End of data>
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Get sum of bookkeeping values pointed to by X into A
                                    ;-----------------------------------------------------
7DDD: 4F          CLRA              ; Start the sum at 0x00
                                    ;
7DDE: 10 AE 81    LDY   ,X++        ;-\ Y gets address of RAM of next bookkeeping value
7DE1: 27 04       BEQ   $7DE7       ; | If Y gets 0x0000 then we reached end of the table
7DE3: AB A4       ADDA  ,Y          ; | Increment A with value from Y
7DE5: 20 F7       BRA   $7DDE       ;-/
                                    ;
7DE7: 8D 59       BSR   $7E42       ; Get Player's Number of Super Jackpots Addr into X
7DE9: AB 84       ADDA  ,X          ; Add # of player's super jackpots to the sum
                                    ;
7DEB: 39          RTS               ;
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Get "random" byte from ROM for given data table X
                                    ;--------------------------------------------------
7DEC: 8D EF       BSR   $7DDD       ; Call function that gets bookkeeping values sum into A
7DEE: 8E FD 00    LDX   #$FD00      ; X gets address in ROM from where seed byte is loaded
7DF1: A6 86       LDA   A,X         ; A now gets byte from ROM
7DF3: 39          RTS               ;
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Get variable value 0..4 for starting lamp
                                    ;-------------------------------------------
```

```
7DF4: 8E 7D 6D    LDX   #$7D6D          ; X gets address of the starting-lamp data table
7DF7: 8D F3       BSR   $7DEC           ; Call function that gets "random" byte from ROM into A
7DF9: 84 07       ANDA  #$07            ; Mask off the low 3 bits
7DFB: 81 05       CMPA  #$05            ; Compare to 05 (ie subtract 05 from A, C-set if borrow)
7DFD: 25 02       BCS   $7E01           ; If C-set then skip over the following
7DFF: 80 04       SUBA  #$04            ; A is 5, 6 or 7, so now subtract 4
7E01: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Get "random" byte for lamp pattern determination
                                        ;-------------------------------------------------
7E02: 8E 7D B7    LDX   #$7DB7          ; X gets address of the lamp-pattern data table
7E05: 8D E5       BSR   $7DEC           ; Call function that gets "random" byte from ROM into A
7E07: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Get variable value 0..4 for lamp pattern (Expert)
                                        ;-------------------------------------------------
7E08: 8D F8       BSR   $7E02           ; Get "random" byte for lamp-pattern from ROM into A
7E0A: 46          RORA                  ; Move high nibble into lower nibble
7E0B: 46          RORA                  ;
7E0C: 46          RORA                  ;
7E0D: 46          RORA                  ;
7E0E: 20 E9       BRA   $7DF9           ; branch up into routine for getting 0..4 starting lamp
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Get variable value 0..1 for starting lamp direction
                                        ;-------------------------------------------------
7E10: 8E 7D 93    LDX   #$7D93          ; X gets address of the lamp-direction data table
7E13: 8D D7       BSR   $7DEC           ; Call function that gets "random" byte from ROM into A
7E15: 84 01       ANDA  #$01            ; Mask off the low 1 bit
7E17: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Get variable value 0..2 for lamp pattern (Hard)
                                        ;-------------------------------------------------
7E18: 8D E8       BSR   $7E02           ; Get "random" byte for lamp-pattern from ROM into A
7E1A: 84 03       ANDA  #$03            ; Mask off the low 2 bits
7E1C: 81 03       CMPA  #$03            ; Compare to 03 (ie subtract 03 from A, C-set if borrow)
7E1E: 25 02       BCS   $7E22           ; If C-set then skip over the following
7E20: 80 02       SUBA  #$02            ; A is 3, so now subtract 2
7E22: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Super Jackpot, Illuminate first lamp
                                        ;
7E23: 86 04       LDA   #$04            ; Start with 0x04 (5th lamp), default starting lamp
                                        ;
7E25: C1 03       CMPB  #$03            ; Compare Super Jackpot adjustment with 03 "Medium"
7E27: 25 02       BCS   $7E2B           ; C-set -> adj is < 0x03, use lamp 04 & skip following
                                        ;
7E29: 8D C9       BSR   $7DF4           ; Call function to get variable value 0..4
                                        ;
7E2B: 81 05       CMPA  #$05            ; Check that A is not out of bounds
7E2D: 25 02       BCS   $7E31           ;
7E2F: 86 04       LDA   #$04            ; A was unexpectedly out of range, reset to lamp 04
                                        ; Now the starting lamp has been determined 0..4 in A
7E31: 8D 01       BSR   $7E34           ; Branch to function to set lamp and save it in $05F9
7E33: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Takes in lamp index 0..4 in A & sets lamp in 5-bank
7E34: 34 04       PSHS  B               ;
                                        ;
7E36: B7 05 F9    STA   $05F9           ; Save current/starting lamp 0..4 into $05F9
```

```
7E39: 8B 11       ADDA  #$11            ; Convert lamp idx 0..4 to 11..15 its lamp matrix index
7E3B: C6 18       LDB   #$18            ; 0x18 == solid lamps plane
7E3D: BD 9E 7F    JSR   $9E7F           ; ValidateThenSingleLampSetIndexAPlaneB()
                                        ;
7E40: 35 84       PULS  B,PC            ;
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; X gets address of player's super jackpots counter
                                        ;--------------------------------------------------
7E42: 8E 05 D1    LDX   #$05D1          ; Number of Super Jackpots
7E45: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
7E48: 39          RTS                   ;
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Check if adj B matches any of the "+1" settings
                                        ;------------------------------------------------
7E49: C1 01       CMPB  #$01            ; Check if "Original+1"
7E4B: 27 10       BEQ   $7E5D           ;  If equal, return C-set
7E4D: C1 04       CMPB  #$04            ; Check if "Medium+1"
7E4F: 27 0C       BEQ   $7E5D           ;  If equal, return C-set
7E51: C1 07       CMPB  #$07            ; Check if "Hard+1"
7E53: 27 08       BEQ   $7E5D           ;  If equal, return C-set
7E55: C1 0A       CMPB  #$0A            ; Check if "Expert+1"
7E57: 27 04       BEQ   $7E5D           ;  If equal, return C-set
7E59: 1C FE       ANDCC #$00FE          ; Clear c-bit
7E5B: 20 02       BRA   $7E5F           ;  go to end
7E5D: 1A 01       ORCC  #$0001          ; Set c-bit
7E5F: 39          RTS                   ; return
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Check if adj B matches any of the "++" settings
                                        ;------------------------------------------------
7E60: C1 02       CMPB  #$02            ; Check if "Original++"
7E62: 27 10       BEQ   $7E74           ;  If equal, return C-set
7E64: C1 05       CMPB  #$05            ; Check if "Medium++"
7E66: 27 0C       BEQ   $7E74           ;  If equal, return C-set
7E68: C1 08       CMPB  #$08            ; Check if "Hard++"
7E6A: 27 08       BEQ   $7E74           ;  If equal, return C-set
7E6C: C1 0B       CMPB  #$0B            ; Check if "Expert++"
7E6E: 27 04       BEQ   $7E74           ;  If equal, return C-set
7E70: 1C FE       ANDCC #$00FE          ; Clear c-bit
7E72: 20 02       BRA   $7E76           ;  go to end
7E74: 1A 01       ORCC  #$0001          ; Set c-bit
7E76: 39          RTS                   ; return
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; SleepPerSuperJackpotsAdjustment()
                                        ;
                                        ; Sleep for time based on adjustment value in B
                                        ;----------------------------------------------
7E77: 34 10       PSHS  X               ;
                                        ;
7E79: 8D CE       BSR   $7E49           ; Check if the 'B' adj value is any of the "+1" settings
7E7B: 24 08       BCC   $7E85           ; C-clr means NOT "+1" adj, skip to check "++" setting
                                        ;
7E7D: 8D C3       BSR   $7E42           ; Get Player's Number of Super Jackpots Addr into X
7E7F: A6 84       LDA   ,X              ; A gets # of player's super jackpots
7E81: 27 0C       BEQ   $7E8F           ; If A is 0x00 then go do slow/normal sleep
7E83: 20 04       BRA   $7E89           ; Otherwise not first super jackpot go do fast sleep
                                        ;
7E85: 8D D9       BSR   $7E60           ; Check if the 'B' adj value is any of the "++" settings
7E87: 24 06       BCC   $7E8F           ; C-clr means NOT "++" adj, skip to slow/normal sleep
                                        ;
7E89: BD 83 46    JSR   $8346           ; Sleep()
7E8C: 10                                ; 0.250 seconds   fast sleep period
7E8D: 20 04       BRA   $7E93           ; go to end
                                        ;
```

```
7E8F: BD 83 46    JSR   $8346           ; Sleep()
7E92: 14                                ; 0.3125 seconds  slow/normal sleep period
7E93: 35 90       PULS  X,PC            ; Done, RTS
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; ID 1033
                                        ;
                                        ; Start function for the super-jackpot arrow movement
                                        ;
7E95: 86 FF       LDA   #$FF            ; Store 0xFF into $D3
7E97: 97 D3       STA   $D3             ; Checked during huntership hit from $4BE3,31
                                        ;
7E99: BD 83 0C    JSR   $830C           ; Get8BitSettingIntoBParameterByte()
7E9C: 1E                                ; 1E=SuperJackpot Adjustment
                                        ;
7E9D: 8D 84       BSR   $7E23           ; Call function to illuminate the starting lamp
                                        ;
7E9F: C1 06       CMPB  #$06            ; Compare adjustment to "Hard"
7EA1: 24 02       BCC   $7EA5           ; C-clear means adjustment is 06 or higher, skip down
7EA3: 20 1C       BRA   $7EC1           ; Adj 00, 01, 02, 03, 04, 05, do original lamp movement
                                        ;
7EA5: C1 09       CMPB  #$09            ; Compare adjustment to "Expert"
7EA7: 24 05       BCC   $7EAE           ; C-clr means adj is >=09, skip to random pattern 0..4
                                        ;
                                        ; Adj 06, 07, 08 (Hard) get random pattern 0..2
7EA9: BD 7E 18    JSR   $7E18           ; Get random value 0..2 into A
7EAC: 20 03       BRA   $7EB1           ;
                                        ; Adj 09, 0A, 0B (Expert) get random pattern 0..4
7EAE: BD 7E 08    JSR   $7E08           ; Get random value 0..4 into A
                                        ;
7EB1: 81 04       CMPA  #$04            ; Check for Expert pattern, 4253142531
7EB3: 27 74       BEQ   $7F29           ;  If equal, go to pattern start
7EB5: 81 03       CMPA  #$03            ; Check for Expert pattern, 1352413524
7EB7: 27 6B       BEQ   $7F24           ;  If equal, go to pattern start
7EB9: 81 02       CMPA  #$02            ; Check for Medium/Hard/Expert pattern, 5432154321
7EBB: 27 35       BEQ   $7EF2           ;  If equal, go to pattern start
7EBD: 81 01       CMPA  #$01            ; Check for Medium/Hard/Expert pattern, 1234512345
7EBF: 27 49       BEQ   $7F0A           ;  If equal, go to pattern start
                                        ; Drop down to Original L-8 lamp pattern, 543212345
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Original L-8 lamp movement pattern, 543212345
                                        ;
7EC1: B6 05 F9    LDA   $05F9           ; Get starting lamp 0..4 into A
7EC4: 27 19       BEQ   $7EDF           ; If lamp is 00 then need to start going down
7EC6: 81 04       CMPA  #$04            ; Check if at lamp 4 (or higher for some reason)
7EC8: 24 05       BCC   $7ECF           ; C-clear means lamp is 04 (or higher?), start going up
                                        ;
7ECA: BD 7E 10    JSR   $7E10           ; Get random value 0..1 into A
7ECD: 27 10       BEQ   $7EDF           ; If A is zero (Down) goto loop portion that goes down
                                        ;
7ECF: 8D A6       BSR   $7E77           ; --\--\ SleepPerSuperJackpotsAdjustment()
7ED1: BD FD B4    JSR   $FDB4           ;   |  | Call $6706,3B with the next 2 bytes in D
7ED4: 18 18                             ;   |  | Lights the previously numbered lamp
7ED6: 7A 05 F9    DEC   $05F9           ;   |  |
7ED9: B6 05 F9    LDA   $05F9           ;   |  |
7EDC: 4D          TSTA                  ;   |  |
7EDD: 26 F0       BNE   $7ECF           ;   |--/
                                        ;   |
7EDF: 8D 96       BSR   $7E77           ;   |--\ SleepPerSuperJackpotsAdjustment()
7EE1: BD FD A2    JSR   $FDA2           ;   |  | Call $66F5,3B with the next 2 bytes in D
7EE4: 18 18                             ;   |  | Lights the next numbered lamp
7EE6: 7C 05 F9    INC   $05F9           ;   |  |
7EE9: B6 05 F9    LDA   $05F9           ;   |  |
7EEC: 81 04       CMPA  #$04            ;   |  |
7EEE: 25 EF       BCS   $7EDF           ;   |--/
                                        ;   |
7EF0: 20 DD       BRA   $7ECF           ; --/
                                        ;
```

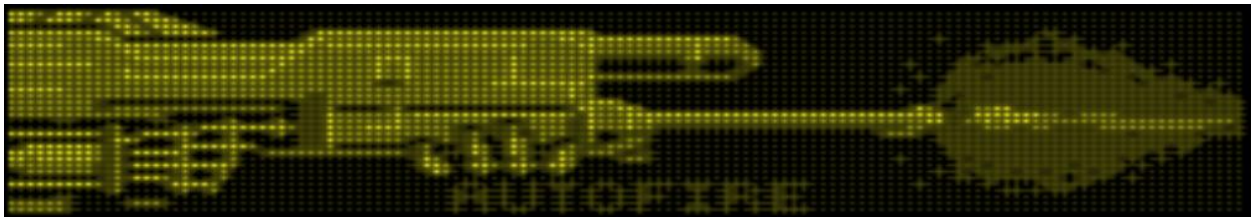
```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Medium/Hard/Expert pattern, 5432154321
                                        ;
7EF2: 8D 83       BSR   $7E77           ; --\ SleepPerSuperJackpotsAdjustment()
7EF4: BD FD B4    JSR   $FDB4           ;   | Call $6706,3B with the next 2 bytes in D
7EF7: 18 18                             ;   | Lights the previously numbered lamp
7EF9: B6 05 F9    LDA   $05F9           ;   | Get currently illuminated lamp 0..4 into A
7EFC: 26 07       BNE   $7F05           ;   | If lamp is 01, 02, 03, 04 go down and decrement
                                        ;   |
7EFE: 86 04       LDA   #$04            ;   | Reset to lamp 04
7F00: B7 05 F9    STA   $05F9           ;   | Save current/starting lamp 0..4 into $05F9
7F03: 20 03       BRA   $7F08           ;   | New next lamp is in $05F9, done with iteration
                                        ;   |
7F05: 7A 05 F9    DEC   $05F9           ;   | Decrement 01, 02, 03, 04 then loop back to sleep
7F08: 20 E8       BRA   $7EF2           ; --/
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Medium/Hard/Expert pattern, 1234512345
                                        ;
7F0A: BD 7E 77    JSR   $7E77           ; --\ SleepPerSuperJackpotsAdjustment()
7F0D: BD FD A2    JSR   $FDA2           ;   | Call $66F5,3B with the next 2 bytes in D
7F10: 18 18                             ;   | This automatically lights the next numbered lamp
7F12: B6 05 F9    LDA   $05F9           ;   | Get currently illuminated lamp 0..4 into A
7F15: 81 04       CMPA  #$04            ;   | Check if at lamp 4 (or higher for some reason)
7F17: 25 06       BCS   $7F1F           ;   | C-set == 00, 01, 02, 03 go down and increment
                                        ;   |
7F19: 4F          CLRA                  ;   | Reset to lamp 00
7F1A: B7 05 F9    STA   $05F9           ;   | Save current/starting lamp 0..4 into $05F9
7F1D: 20 03       BRA   $7F22           ;   | New next lamp is in $05F9, done with iteration
                                        ;   |
7F1F: 7C 05 F9    INC   $05F9           ;   |
7F22: 20 E6       BRA   $7F0A           ; --/
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Expert pattern, 1352413524 (branch point)
                                        ;-------------------------------------------
7F24: 8E 7F 3E    LDX   #$7F3E          ; X gets address of the next-lamp table for 1352413524
7F27: 20 03       BRA   $7F2C           ;
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Expert pattern, 4253142531
                                        ;----------------------------
7F29: 8E 7F 43    LDX   #$7F43          ; X gets address of the next-lamp table for 4253142531
                                        ;
7F2C: BD 7E 77    JSR   $7E77           ; --\ SleepPerSuperJackpotsAdjustment()
7F2F: BD 87 BE    JSR   $87BE           ;   | ExtinguishLampGroupParamBytes()
7F32: 18 18                             ;   | Lamp grp 18 = 5-bank hunter ship, 18 = solid lamps
7F34: B6 05 F9    LDA   $05F9           ;   | A gets current lamp
7F37: A6 86       LDA   A,X             ;   | A gets 'next' lamp from X table
7F39: BD 7E 34    JSR   $7E34           ;   | Push updated lamp idx to $05F9 and illuminate lamp
7F3C: 20 EE       BRA   $7F2C           ; --/
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; 1352413524 'next' table
                                        ;-------------------------
7F3E: 02                                ; 00 --> 02  (for 1 to 3 transition)
7F3F: 03                                ; 01 --> 03  (for 2 to 4 transition)
7F40: 04                                ; 02 --> 04  (for 3 to 5 transition)
7F41: 00                                ; 03 --> 00  (for 4 to 1 transition)
7F42: 01                                ; 04 --> 01  (for 5 to 2 transition)
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; 4253142531 'next' table
                                        ;-------------------------
7F43: 03                                ; 00 --> 03  (for 1 to 4 transition)
```

```
7F44: 04                                ; 01 --> 04  (for 2 to 5 transition)
7F45: 00                                ; 02 --> 00  (for 3 to 1 transition)
7F46: 01                                ; 03 --> 01  (for 4 to 2 transition)
7F47: 02                                ; 04 --> 02  (for 5 to 3 transition)
                                        ;
----------------------------------------;------------------------------------------------------
----------------------------------------;------------------------------------------------------
```

Multiball Auto-Fire Problem Fix L8.4

A bug was discovered in L-8 software regarding the auto-fire ball-saver that takes place during the first multiball that the player achieves during a game. This bug is being investigated during L-8.4 and being addressed as described below.

During the player's first multiball, there is a 7 second autofire timer that is started as part of the multiball startup. Any balls drained during this 7 second period will be returned to the playfield. In most cases, when this happens, there is an accompanying animation as the ball is returned to the playfield.



The problem behavior is when the multiball starts and then:

- Lock one ball into the ball-popper (skull)
- Lock one ball into a lock shot (database or left loop)

Once those 2 balls are locked, the player is given a countdown for locking the remaining ball. The problem behavior is when this remaining ball is drained, and when such ball drain occurs during the 7-second auto-fire timer.

The expected behavior is that the game plays the "Autofire" animation and returns the ball back onto the playfield, allowing the player to continue their efforts at locking this remaining ball. This is the actual behavior if the ball is drained early in the 7-second auto-fire timer (which means the other 2 balls needed to be locked quite soon as multiball started). The problem is when the ball is drained near the end of the 7-second auto-fire timer. When the ball is drained near the end of the 7-second auto-fire timer, what happens is:

- The game behaves as if the ball had drained, and advances the ball in the popper to the gun for jackpot attempt, and
- The game quietly returns the drained ball back onto the playfield without the "Autofire" animation.

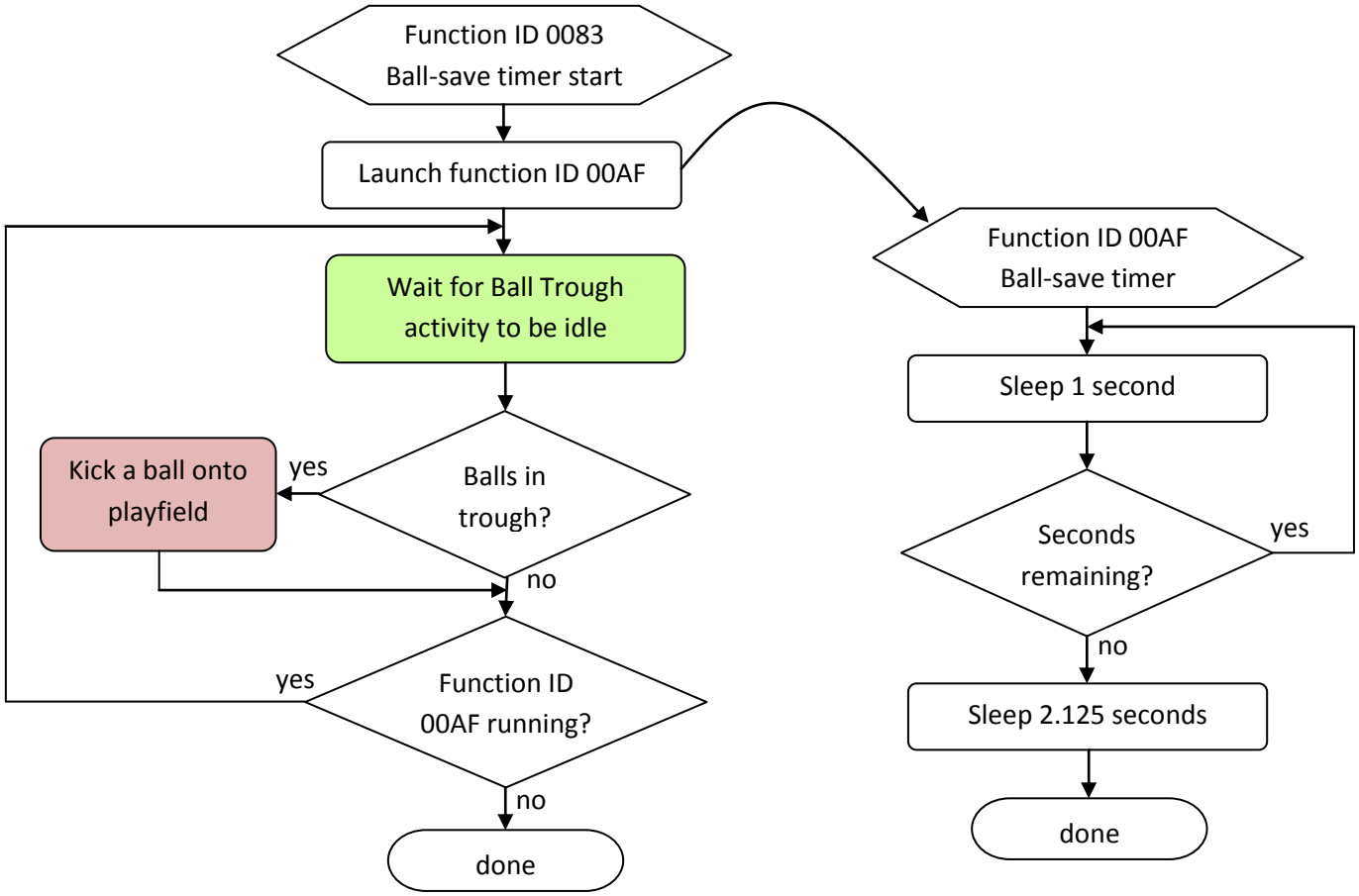
The problematic behavior, as described above, forces the player to handle both the cannon-shot for jackpot along with the returned ball on the playfield.

Multiball Auto-Fire Code Analysis

In order to find the problematic code, the L-8 code was examined for how the auto-fire timer behaves with regard to multiball ball saver. The problem code was identified and is described below in detail.

It does appear a likely coding error is occurring where the game is mistakenly treating the drained ball as if the ball-saver is not running (which causes the cannon to be loaded), while a auto-fire timer secondary mechanism (alternative ball saver code) engages which causes the ball to be returned to the playfield.

Shown below is the basic ball-saver timer logic. This logic includes logic that determines that a ball needs to be returned to the playfield. Such ball-return in this logic is a secondary mechanism to return the ball when other/normal ball save return code didn't do its duty and return the ball to the playfield.



Wait for Ball Trough activity to be idle ⇒ During this step is when the ball is normally detected and returned to the playfield along with the "Autofire" animation. This happens as part of ball-trough switch handling code.

Kick a ball onto playfield ⇒ This is the secondary point in which a ball can be returned to the playfield in the event the normal method fails. This is a redundant code mechanism to ensure ball is returned to playfield when timer is running.

As shown in the logic flowcharts, above, there are 2 separate functions that are running for auto-fire timer:

- Function ID 0083 is a auto-fire timer maintenance function
- Function ID 00AF is simply running the timer countdown

Attentive readers might already be able to spot the possible timing condition that could occur with this logic. The 00AF function could potential reach its end while the ID 0083 function is busy waiting for ball trough activity to complete. Once such ball trough activity is complete (and ID 00AF had finished) the ID 0083 function would then check balls in trough and kick one onto the playfield without care or concern that the timer function 00AF had already completed. After the 0083 function kicks the ball onto the playfield it then checks for ID 00AF running where it finally determines the timer had expired and, itself, also completes. This means there is a potential for a window of time where the 00AF function is not running yet 0083 function will then kick a ball onto the playfield. This behavior, in fact, happens in the L-8 problem where a ball may be returned onto the playfield while the cannon is being loaded.

The partially annotated code for the timer maintenance loop and timer countdown loop is shown below, for reference. This code starts at $6E94,31, ROM offset 0x46E94. This code starts at the function ID 0083 entry point for first-multiball auto-fire timer. This is called with the timer countdown value in the B register. In this case value 7 is in the B register for a seven second ball save timer.

```
----------------------------------------;-------------------------------------------------------
                                        ;
6E94: 4F          CLRA                  ;
6E95: 97 BE       STA   $BE             ; Save 0x00 ball count into $BE
                                        ;-----------------------------------------------
                                        ;-----------------------------------------------
                                        ;
6E97: BD 88 D5    JSR   $88D5           ; IncreaseBookkeepingCounterAddrXBy1()
6E9A: 00 0D                             ;
6E9C: BD 71 94    JSR   $7194           ;
6E9F: BD 89 48    JSR   $8948           ;
6EA2: 6E F7 31                          ;
6EA5: D1 C3       CMPB  $C3             ;
6EA7: 25 02       BCS   $6EAB           ;
6EA9: D7 C3       STB   $C3             ;
                                        ;
6EAB: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
6EAE: 00 AF                             ; 00AF == Ball-saver countdown timer loop
6EB0: 6E E5 31                          ; Countdown Timer Loop
                                        ;
6EB3: BD F7 66    JSR   $F766           ; -\ BallTroughAuditUntilIdle() Trough ball count in A
6EB6: 7E 6E B9    JMP   $6EB9           ;  | <nop>
6EB9: 91 BE       CMPA  $BE             ;  | Compare ball trough count w/desired count in $BE
6EBB: 22 0D       BHI   $6ECA           ;  | Branch if ball trough count is > desired count
6EBD: BD 83 46    JSR   $8346           ;  | Sleep()
6EC0: 04                                ;  | 0.0625 seconds
                                        ;  |
6EC1: BD 86 90    JSR   $8690           ;  | SearchLinkedListForId()  // c-bit clear = ID found
6EC4: 00 AF                             ;  | 00AF == Ball-saver countdown timer loop
6EC6: 25 1A       BCS   $6EE2           ;  | Countdown timer loop has ran to completion?
6EC8: 20 E9       BRA   $6EB3           ; -/
                                        ;
                                        ; Found a ball in the trough that needs kicked out
                                        ;-----------------------------------------------
6ECA: BD 45 0A    JSR   $450A           ;
6ECD: BD 8B C3    JSR   $8BC3           ; ScheduleFunctionCallback()
6ED0: 00 82                             ; 0082 == PutATroughBallOntoPlayfield()
6ED2: 6F 0D 31                          ; PutATroughBallOntoPlayfield()
                                        ;
6ED5: BD 83 46    JSR   $8346           ; -\ Sleep()
```

```
6ED8: 03                                ;  |
6ED9: BD 86 90    JSR   $8690           ;  | SearchLinkedListForId()  // c-bit clear = ID found
6EDC: 00 82                             ;  | 0082 == PutATroughBallOntoPlayfield()
6EDE: 24 F5       BCC   $6ED5           ; -/
                                        ;
6EE0: 20 DF       BRA   $6EC1           ; ^_ goto $6EC1 _^ keep checking for more balls
                                        ;
6EE2: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; ID 00AF  The actual countdown timer loop
                                        ;
6EE5: BD 9B DA    JSR   $9BDA           ;
6EE8: BD 83 46    JSR   $8346           ;--\ Sleep()
6EEB: 40                                ;  | 0x40 == 1 second
6EEC: 0A C3       DEC   $C3             ;  |
6EEE: 26 F8       BNE   $6EE8           ;--/
6EF0: BD 83 46    JSR   $8346           ; Sleep()
6EF3: 88                                ; 2+ seconds  (grace period)
6EF4: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;------------------------------------------------------
```

There is a complex set of code that takes place during the ball drain, which leads up to the point in which the “Autofire” animation is to be shown and ball returned to playfield while retaining the current ball lock countdown. Shown below is a high level breakdown of some of the involved functions that lead up to the important parts of code related to this bug fix. This information is presented here as a starting point for others who may want to eventually develop a more deeper understanding of how this part of the game code operates.

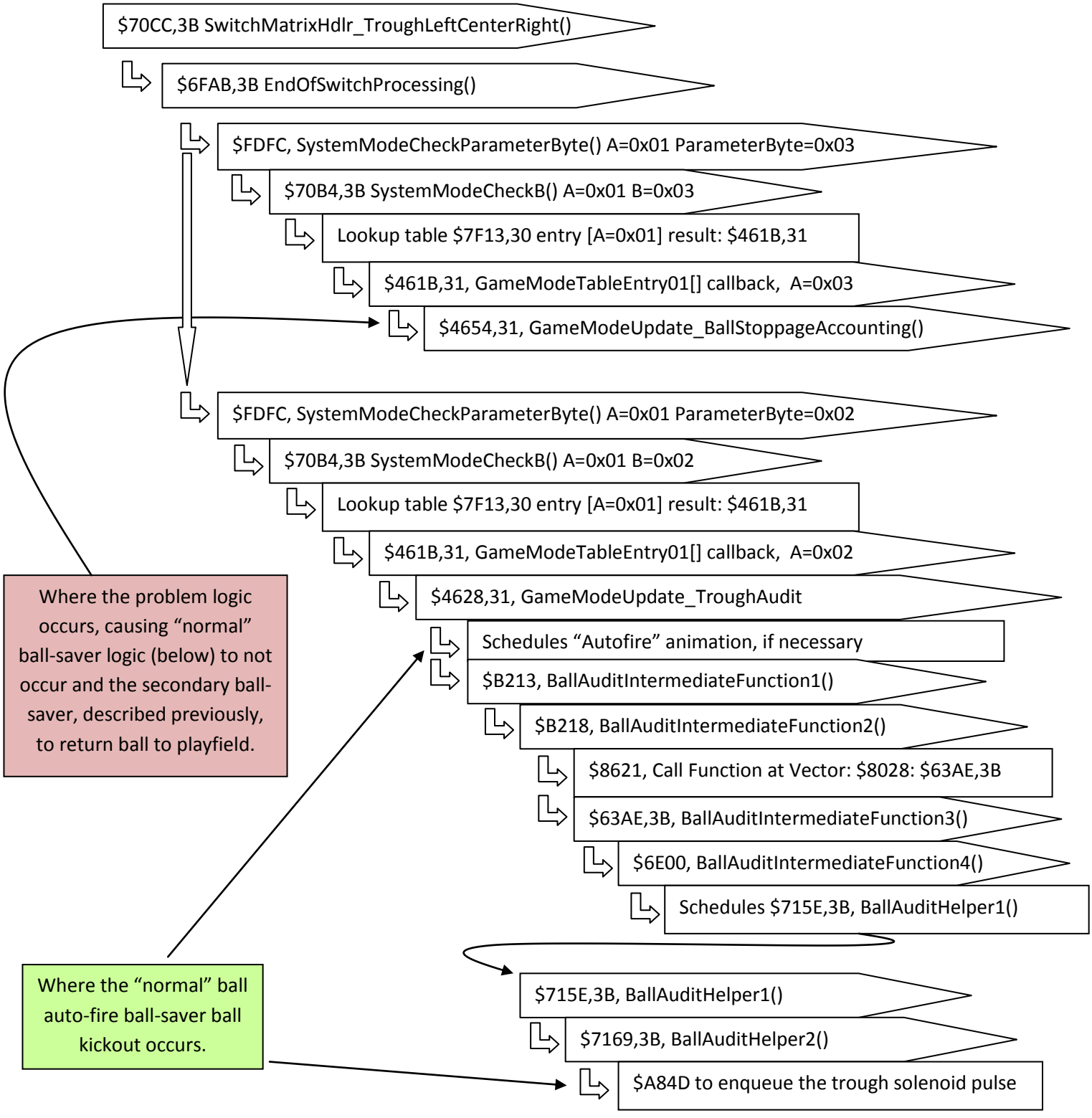
The first part of the process is the ball hitting the outhole switch which causes the outhole solenoid to kick the ball into the ball trough. Focus, below, is on what happens at this point.

The L-8 Playfield Switch Table has been depicted in the L8.3 document. The table starts at $4931,3D, ROM offset 0x74931. In the table, data is provided for each of the playfield switches. For this analysis, the focus is on the 3 ball trough switches:

```
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                          ;
```

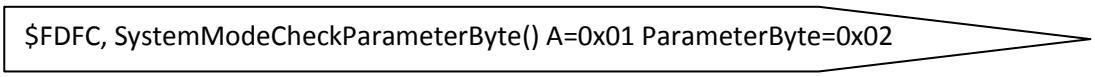
For all three trough switches the same callback function $70CC,3B is invoked. The common handler is designed with the idea that the switches will open and close rapidly as the ball goes through from left to the right in the trough. A lot of the logic is excluded from the following summary. This summary goes through function calls that lead up to the auto-fire ball-save timer problem.

Shown below is the series of function calls involved in the Multiball Auto-Fire ball-saver bug.



Shown above is the high level code flow in how the ball trough switches relate to the autofire ball save timer. Highlighted in green is the "normal" code flow where the ball drain is detected during auto-fire timer resulting in "Autofire" animation and ball being returned to the playfield. Highlighted in red is the problematic code where logic flaw results in the lack of the "normal" flow and, as such, the previously depicted timer code will perform its duty of backup or secondary mechanism to ensure a ball in the trough is returned to the playfield (when the "normal" path, depicted above, fails to do its duty).

For those interested in further details about the failure path, it is worth noting that when the problem code occurs, the entire code flow, starting at the following point, does not occur:



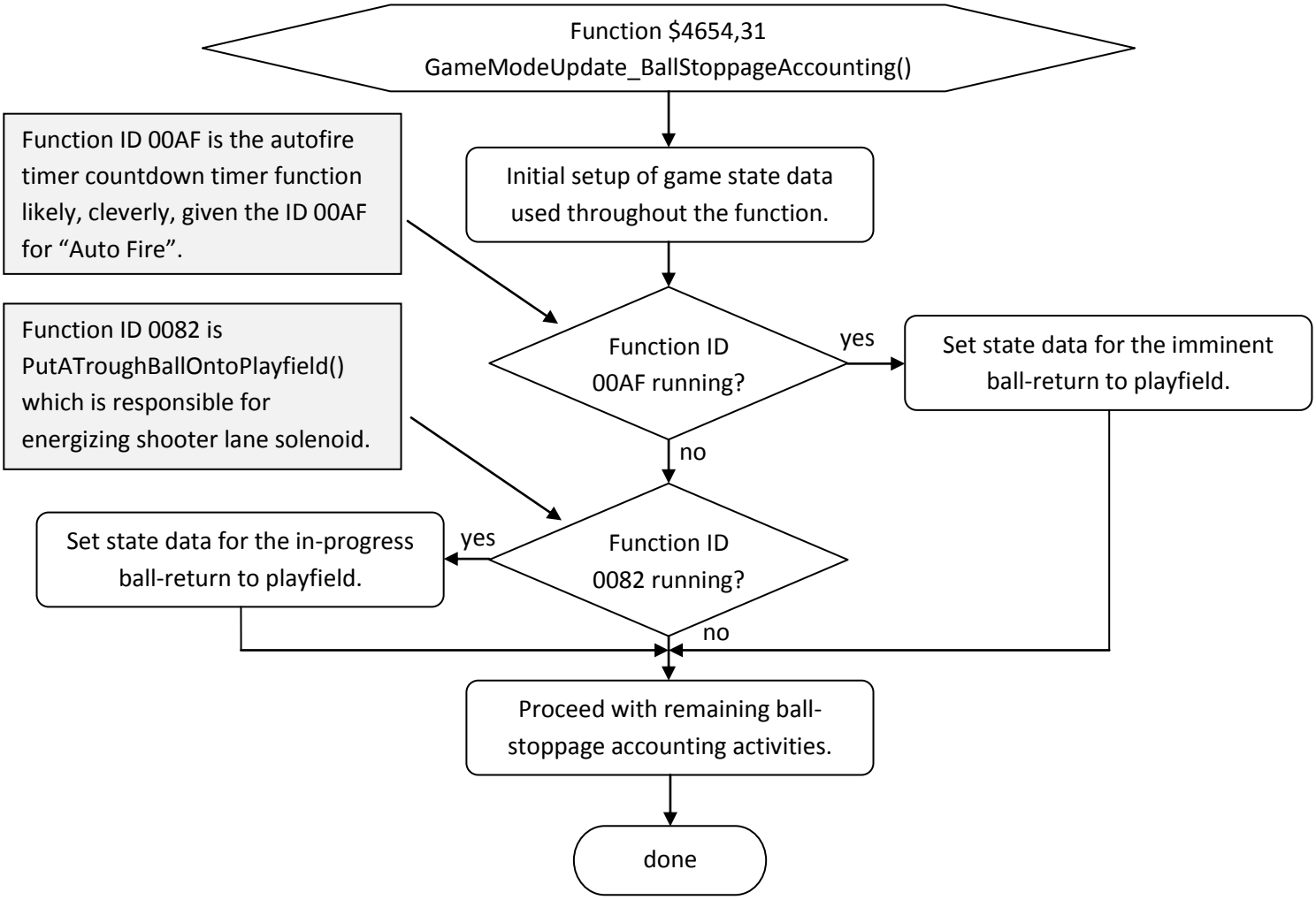
In the problem scenario, the code during "$6FAB,3B EndOfSwitchProcessing()" fails to detect the presence of a scheduled function with ID 0040 <ANDed with> 01F0, and as such, the subsequent call chain starting at the point mentioned above will not take place. This check for ID 0040 <AND> 01F0 occurs at $6FC4,3B. A detailed analysis of why, precisely, the expected function is not running is left as an exercise to the reader.

The problem code, as mentioned, is in this point in the flow:

This accounting function appears to be designed to check for whether or not there is an imminent ball-return to the playfield (due to auto-fire timer running) whereby logic will take a specific path in such case. The specific path taken will result in the subsequent logic to ultimately display the "Autofire" animation and kick the ball back into game play, while allowing the game countdown to continue for the player to lock the 3rd ball towards 3x jackpot.

Not a lot of analysis was done into the details of how, precisely, code ultimately ends up performing the "normal" auto-fire ball save return. Focus here is on the logic used to determine whether or not the ball-save timer is active. The summary flowchart of the problem code is depicted below. This is for the function at $4654,31, ROM offset 0x44654.

This function was given the name GameModeUpdate_BallStoppageAccounting() as it appears to be all about the handling of game state when a ball has been stopped during game play. This function can be called when a ball is stopped at any of the lock shots, ball-popper or ball trough. During this function the game code performs an accounting of various state data, ensuring game flow is as expected.



Readers can correlate the logic, above, with the previously depicted logic for the auto-fire ball save timer logic. The function ID 00AF is launched as a distinct function that counts down the number of remaining seconds while the function ID 0083 is responsible for monitoring the timer, waiting for idle ball trough switch states, and for kicking the ball onto the playfield, as described, as a secondary ball save logic to occur if the ball trough switch handling itself did not perform such ball save.

For readers who are following along, the problem/fix is yet to be revealed and will be described in the next section. The problem and fix, at this point, may be evident to attentive readers.

Shown below is the corresponding code for the function at $4654,31, ROM offset 0x44654. This code is being shown for reference and future investigations. A lot of work is done in this function and, as such, has not been fully analyzed or annotated. Some of the annotations are from various other L8.3 or L8.4 investigations. The annotations were done during various code examinations and may not be entirely correct in what they are describing. Further study is required for full/complete analysis and annotation of this function. The portion of this function applicable to the multiball auto-fire problem is highlighted with red. This section of the code will be the focus of the next section of this document where the corrected code is described.

```
----------------------------------------;-------------------------------------------------
                                       ;
                                       ; GameModeUpdate_BallStoppageAccounting()
                                       ;
4654: 34 76       PSHS  U,Y,X,B,A      ;
4656: BD F7 59    JSR   $F759          ;
4659: 7E 46 5C    JMP   $465C          ; <nop>
465C: 10 26 01 29 LBNE  $4789          ;
4660: 96 BF       LDA   $BF            ;
4662: 97 D1       STA   $D1            ;
4664: 96 C0       LDA   $C0            ;
4666: 97 D0       STA   $D0            ;
4668: BD 88 F5    JSR   $88F5          ;
466B: 70 57 3B                         ; BallTroughAudit() C-set if trough hdlr in progress.
                                       ;  Function returns trough ball count in A
466E: 97 CE       STA   $CE            ;
4670: 0F CF       CLR   $CF            ;
                                       ;
4672: BD 86 90    JSR   $8690          ; SearchLinkedListForId()  // c-clear means ID is found
4675: 00 AF                            ; 00AF == Ball-saver countdown timer loop
4677: 25 06       BCS   $467F          ; if 00AF is NOT running, go check for 0082
                                       ;
4679: 90 BE       SUBA  $BE            ;
467B: 97 CF       STA   $CF            ;
467D: 20 0F       BRA   $468E          ;
                                       ;
467F: BD 86 90    JSR   $8690          ; SearchLinkedListForId()  // c-clear means ID is found
4682: 00 82                            ;  0082 == PutATroughBallOntoPlayfield()
4684: 25 08       BCS   $468E          ; If PutATroughBallOntoPlayfield() is not running, skip
                                       ;
4686: 96 C2       LDA   $C2            ;
4688: 91 CE       CMPA  $CE            ;
468A: 27 02       BEQ   $468E          ;
468C: 0C CF       INC   $CF            ;
                                       ;
468E: BD 88 F5    JSR   $88F5          ;
4691: 70 31 3B                         ;
4694: 9B CF       ADDA  $CF            ;
4696: 97 C0       STA   $C0            ;
4698: BD 88 F5    JSR   $88F5          ;
469B: 75 DF 3B                         ;
469E: 90 CE       SUBA  $CE            ;
46A0: 9B CF       ADDA  $CF            ;
                                       ; Update $BF, # of balls on playfield, tracked during MB
46A2: 97 BF       STA   $BF            ;
46A4: 96 CE       LDA   $CE            ;
46A6: 90 CF       SUBA  $CF            ;
46A8: 2A 01       BPL   $46AB          ;
46AA: 4F          CLRA                 ;
46AB: 97 C1       STA   $C1            ;
46AD: BD 86 90    JSR   $8690          ; SearchLinkedListForId()  // c-clear means ID is found
46B0: 00 86                            ;  Search for 0x0086, C-clr means multiball is running
46B2: 24 18       BCC   $46CC          ;
46B4: BD 86 90    JSR   $8690          ; SearchLinkedListForId()  // c-clear means ID is found
46B7: 00 B8                            ;  This lookup is one that happens at end-of-ball
46B9: 24 11       BCC   $46CC          ;
46BB: BD 47 92    JSR   $4792          ;
46BE: BD 47 A1    JSR   $47A1          ;
46C1: 86 04       LDA   #$04           ;
46C3: BD 88 F5    JSR   $88F5          ;
46C6: 6E 53 3B                         ;
46C9: 16 00 BD    LBRA  $4789          ;
```

```
                                        ;
                                        ; Multiball is running
                                        ;
46CC: 0D C0       TST   $C0             ; Balls in play(?)
46CE: 10 27 00 81 LBEQ  $4753           ; If zero then go to end-of-ball(?)
46D2: 96 BF       LDA   $BF             ; Get number of balls $BF
46D4: 81 03       CMPA  #$03            ;
46D6: 26 0B       BNE   $46E3           ;
46D8: BD 84 AD    JSR   $84AD           ; GetMemoryFlag()
46DB: 42                                ;
46DC: 24 2F       BCC   $470D           ;
46DE: BD 48 88    JSR   $4888           ;
46E1: 20 2A       BRA   $470D           ;
                                        ;
                                        ; Get here after 2 balls drained during MB, 1 remaining
                                        ;
46E3: 96 C0       LDA   $C0             ; Balls in play(?) 01 when 1 remaining during MB
46E5: 81 02       CMPA  #$02            ;
46E7: 27 EF       BEQ   $46D8           ;
                                        ;
                                        ; Get here after 2 balls drained during MB, 1 remaining
46E9: 86 04       LDA   #$04            ;
46EB: BD 88 F5    JSR   $88F5           ;
46EE: 70 41 3B                          ;
46F1: 24 E5       BCC   $46D8           ; When 1 ball remaining during MB, C is set, no branch
46F3: 86 02       LDA   #$02            ;
46F5: BD 88 F5    JSR   $88F5           ;
46F8: 70 41 3B                          ;
46FB: 24 03       BCC   $4700           ; When 1 ball remaining during MB, C is set, no branch
46FD: BD 47 92    JSR   $4792           ;
4700: 86 03       LDA   #$03            ;
4702: BD 88 F5    JSR   $88F5           ;
4705: 70 41 3B                          ;
4708: 24 03       BCC   $470D           ;
470A: BD 47 A1    JSR   $47A1           ;
470D: 96 C0       LDA   $C0             ;
470F: 81 01       CMPA  #$01            ;
4711: 26 31       BNE   $4744           ;
4713: D6 BF       LDB   $BF             ;
4715: D1 D1       CMPB  $D1             ;
4717: 26 04       BNE   $471D           ;
4719: 91 D0       CMPA  $D0             ;
471B: 27 6C       BEQ   $4789           ;
471D: BD 84 AD    JSR   $84AD           ; GetMemoryFlag()
4720: 42                                ;
4721: 10 24 00 64 LBCC  $4789           ;
4725: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4728: 00 AA                             ;  ID 00AA Ball-In-Popper
472A: 24 18       BCC   $4744           ;
472C: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
472F: 00 AB                             ;  ID 00AB Ball-in-gun function is running
4731: 24 11       BCC   $4744           ;
4733: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4736: 00 84                             ;  ID 0084 Ball-gun-to-Target-Period function is running
4738: 24 0A       BCC   $4744           ;
473A: BD 8B 77    JSR   $8B77           ; ScheduleFunctionStart()
473D: 00 A9                             ;  ID 00A9 == "LOAD THE GUN" period where there is only
                                        ;             1 ball remaining at end of multiball
473F: 48 AF 31                          ;
4742: 20 45       BRA   $4789           ;
4744: BD 86 9E    JSR   $869E           ; CancelScheduledCallbackFunction()
4747: 00 A9                             ;  ID 00A9 == "LOAD THE GUN" period where there is only
```

```
                                        ;             1 ball remaining at end of multiball
4749: BD 86 9E    JSR   $869E          ; CancelScheduledCallbackFunction()
474C: 00 B0                           ;
474E: BD 56 9C    JSR   $569C         ;
4751: 20 36       BRA   $4789         ;
4753: 86 04       LDA   #$04          ;
4755: BD 88 F5    JSR   $88F5         ;
4758: 70 41 3B                        ;
475B: 4D          TSTA                ;
475C: 26 08       BNE   $4766         ;
475E: BD 47 BF    JSR   $47BF         ;
4761: BD 47 CE    JSR   $47CE         ;
4764: 20 23       BRA   $4789         ;
4766: BD 86 90    JSR   $8690         ; SearchLinkedListForId()  // c-clear means ID is found
4769: 00 EC                           ;
476B: 24 1C       BCC   $4789         ;
                                      ; Following is called in MB when all 3 balls get locked
                                      ; If MB Init is running then skip show of jackpot values
                                      ; If MB init is running then jackpot values are shown
                                      ;
476D: BD 86 90    JSR   $8690         ; SearchLinkedListForId()  // c-clear means ID is found
4770: 00 B8                           ; ID 00B8 Multiball Init function
4772: 24 15       BCC   $4789         ; If MB init is running then skip all of the following
4774: 86 04       LDA   #$04          ; SolenoidTableEntry04, 04=Trough,      40
4776: BD 88 F5    JSR   $88F5         ; CallBankedFunction_Param_WPCAddr()
4779: 6E 6B 3B                        ;
477C: BD 86 9E    JSR   $869E         ; CancelScheduledCallbackFunction()
477F: 00 A9                           ;  ID 00A9 == "LOAD THE GUN" period where there is only
                                      ;              1 ball remaining at end of multiball
4781: BD 86 9E    JSR   $869E         ; CancelScheduledCallbackFunction()
4784: 00 B0                           ;
4786: BD 56 9C    JSR   $569C         ;
4789: 35 F6       PULS  A,B,X,Y,U,PC  ;
                                      ;
---------------------------------------;-------------------------------------------------
```

Multiball Auto-Fire Code Correction

As shown above, the `GameModeUpdate_BallStoppageAccounting()` function takes particular effort to see if the auto-fire timer is running and, if so, set state information accordingly. This is done by way of checking if the function ID 00AF is currently running. As described in detail, above, it is clearly possible for the function ID 00AF to no longer be running while the auto-fire maintenance function ID 0083 is still in operation.

In fact, the problem behavior is specifically due to the fact that the function ID 00AF is no longer running while the function ID 0083 is still in operation and may still return a ball onto the playfield when it gets the opportunity to do so. Specifically, the ID 0083 function is busy waiting for the ball trough switches to settle after which the alternate/backup mechanism will detect the ball in the trough which needs to be ejected (and in doing so, the problem occurs where ball is served while the cannon is loaded and jackpot attempt is underway). It is worth noting that the ID 0083 function is busy waiting for ball trough switch activity to complete whereby the ball trough is currently active as part of the currently running code, `GameModeUpdate_BallStoppageAccounting()`.

The obvious solution, therefore, is that the `GameModeUpdate_BallStoppageAccounting()` should not be checking if ID 00AF is running, it should instead check if ID 0083 function is running. Checking for ID 0083 will correctly detect if the ball-saver logic is still running and whether a ball-return is imminent. This change is depicted below.

```
----------------------------------------;--------------------------------------------------
                                        ;
                                        ; GameModeUpdate_BallStoppageAccounting()
                                        ;
4654: 34 76       PSHS  U,Y,X,B,A       ;
4656: BD F7 59    JSR   $F759           ;
4659: 7E 46 5C    JMP   $465C           ; <nop>
465C: 10 26 01 29 LBNE  $4789           ;
4660: 96 BF       LDA   $BF             ;
4662: 97 D1       STA   $D1             ;
4664: 96 C0       LDA   $C0             ;
4666: 97 D0       STA   $D0             ;
4668: BD 88 F5    JSR   $88F5           ;
466B: 70 57 3B                          ; BallTroughAudit() C-set if trough hdlr in progress.
                                        ;  Function returns trough ball count in A
466E: 97 CE       STA   $CE             ;
4670: 0F CF       CLR   $CF             ;
                                        ;
4672: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4675: 00 AF                             ; 00AF == Ball-saver countdown timer loop
4677: 25 06       BCS   $467F           ; if 00AF is NOT running, go check for 0082
4672: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
4675: 00 83                             ; 0083 == Ball-saver maintenance function
4677: 25 06       BCS   $467F           ; if 0083 is NOT running, go check for 0082
                                        ;
4679: 90 BE       SUBA  $BE             ;
467B: 97 CF       STA   $CF             ;
467D: 20 0F       BRA   $468E           ;
...
```

By checking for ID 0083, the `GameModeUpdate_BallStoppageAccounting()` function will behave same as it did, previously, when the ID 00AF function was running. This will result in the game showing the "Autofire" animation, ball returned to playfield, and continuance of the timer countdown for player to lock their 3rd ball for a 3x jackpot attempt.

Since the original L-8 code would have returned the ball to the playfield anyway (using the alternate/redundant mechanism in the ID 0083 function itself), this code change results in no negative or undue benefit to the user other than the correct and expected behavior where the ball is returned to the playfield in the same way, regardless if the auto-fire timer had just begun or if it was near its end (which is when the problem would otherwise occur).

Tournament Mode Enhancements, L8.4

The L8.4 update is made to be Tournament Mode friendly. With the fix for the "Lost Super Jackpot" and the enhanced Super Jackpot specifically designed to be identical for all players in the same game, the L8.4 software also includes enhancements to existing game features to make the game play more equal for all players when adjustment A.1 26 "Tournament Play" is enabled in the game adjustments. *Throughout this text the terms "Tournament Play" and "Tournament Mode" are used interchangeably.*

The extra enhancements are described in the following sections, below.

Tournament Mode Enhancement: Database Award, L8.4

In L8.4 when adjustment A.2 26 "Tournament Play" is enabled, the database award will behave in a way that all players in a multi player game get the same database awards. The game will award seemingly random database awards, however the first database award for all players will be the same award. The second database award will also be seemingly random, but all players get the same award for their second database award, and so on.

When A.2 26 "Tournament Play" is disabled, the original L-8 Database Award logic will be used which involves an algorithm to give the player an award to help them if their game is doing poorly.

Database Award Logic L-8

The database award logic occurs in bank $31. Shown below is a partially annotated function starting at $527E,31, ROM offset 0x4527E. This function is shown here for others who are interested to further study and figure out what, precisely, is being done. Wherever the logic calls $FB29 function is where a per-player game statistics is cited as part of the award determination. Others are encouraged to further analyze the code and figure out some of the missing details.

```
----------------------------------------;------------------------------------------------
                                        ;
                                        ; DatabaseAwardInitiate()
                                        ;
527E: BD 71 A3    JSR   $71A3           ; Increment05BDbyPlayerIndexNumber()
5281: BD 88 D5    JSR   $88D5           ; IncreaseBookkeepingCounterAddrXBy1()
5284: 00 05                             ; database awards statistic(?)
                                        ;----------------------------------------
                                        ;  Exclusion Bits
                                        ;-----------------------------------
                                        ; $064F & 0x01 --> exclude "Multiball"
                                        ; $064F & 0x02 --> exclude "Lite Extra Ball"
                                        ; $064F & 0x04 --> exclude "500,000"
                                        ; $064F & 0x08 --> exclude "Video mode"
                                        ; $064F & 0x10 --> exclude "Lite kickback"
                                        ; $064F & 0x20 --> exclude "Special"
                                        ; $064F & 0x40 --> exclude "Lite hurry up"
                                        ; $064F & 0x80 --> exclude "3,000,000"
                                        ; $0650 & 0x01 --> exclude "Chase loop"
                                        ; $0650 & 0x02 --> exclude "Extra ball"
                                        ; $0650 & 0x04 --> exclude "100,000"
                                        ; $0650 & 0x08 --> exclude "Autofire"
```

```
                                    ; $0650 & 0x10 --> exclude "Security pass"
                                    ; $0650 & 0x20 --> exclude "Lite Special"
                                    ; $0650 & 0x40 --> exclude "Hurry up"
                                    ; $0650 & 0x80 --> exclude "1,000,000"
                                    ;-----------------------------------------
5286: 7F 06 4F    CLR   $064F       ; $064F = 0x00 Exclude nothing
5289: 7F 06 50    CLR   $0650       ; $0650 = 0x00 Exclude nothing
528C: 7F 06 51    CLR   $0651       ; $0651 = 0x00 Random number divisor
                                    ;
528F: 86 63       LDA   #$63        ; A = 0x63 (99)
                                    ;
5291: BD A7 5B    JSR   $A75B       ; Get16BitPseudoRandomValueintoA() random number 0..98
5294: 4C          INCA              ; A++ random number now 1..99
                                    ;
5295: 7D 05 D9    TST   $05D9       ; Number of specials awarded this game (for all players)
5298: 26 19       BNE   $52B3       ;
529A: BD 84 49    JSR   $8449       ; GetLampLitState() C-clear when lamp is on
529D: 0A                            ; 0x0A "Special" lamp
529E: 24 13       BCC   $52B3       ; If "Special" lamp is on, then go to $52B3
52A0: C6 01       LDB   #$01        ;
52A2: BD F8 2C    JSR   $F82C       ;
52A5: 7E 52 A8    JMP   $52A8       ; <nop>
52A8: 34 04       PSHS  B           ;
52AA: A1 E0       CMPA  ,S+         ;
52AC: 22 05       BHI   $52B3       ;
52AE: C6 0E       LDB   #$0E        ; Set winning selection: 0x0E "SPECIAL"
52B0: 7E 55 65    JMP   $5565       ; Go award winning selection B
                                    ;
52B3: C6 20       LDB   #$20        ;
52B5: FA 06 4F    ORB   $064F       ;
52B8: F7 06 4F    STB   $064F       ; $064F |= 0x20, exclude "Special"
                                    ;
52BB: 7D 05 D9    TST   $05D9       ; if ($05D9 == 0x00)
52BE: 26 16       BNE   $52D6       ; {
52C0: BD 84 49    JSR   $8449       ;    GetLampLitState() C-clear when lamp is on
52C3: 0A                            ;    0x0A "Special" lamp
52C4: 24 10       BCC   $52D6       ;    if ("Special" lamp == OFF)
                                    ;    {
52C6: C6 06       LDB   #$06        ;
52C8: F1 06 12    CMPB  $0612       ;
52CB: 27 09       BEQ   $52D6       ;
52CD: 81 0C       CMPA  #$0C        ;
52CF: 22 05       BHI   $52D6       ;
52D1: 7C 06 51    INC   $0651       ;
52D4: 20 08       BRA   $52DE       ;       goto $52DE
                                    ;    }
                                    ; }
52D6: C6 20       LDB   #$20        ;
52D8: FA 06 50    ORB   $0650       ;
52DB: F7 06 50    STB   $0650       ; $0650 |= 0x20, exclude "Lite Special"
                                    ;
52DE: C6 02       LDB   #$02        ;
52E0: F1 06 12    CMPB  $0612       ; Check if previous DB award was 0x02, "Extra Ball"
52E3: 27 1B       BEQ   $5300       ;
52E5: BD FA 1E    JSR   $FA1E       ;
52E8: 7E 52 EB    JMP   $52EB       ;
52EB: 25 13       BCS   $5300       ;
                                    ;
52ED: C6 00       LDB   #$00        ;
52EF: BD F8 2C    JSR   $F82C       ;
52F2: 7E 52 F5    JMP   $52F5       ;
52F5: 34 04       PSHS  B           ;
```

```
52F7: A1 E0       CMPA  ,S+             ;
52F9: 22 05       BHI   $5300           ;
                                        ;
52FB: C6 02       LDB   #$02            ; Set winning selection: 0x02 "EXTRA BALL"
52FD: 7E 55 65    JMP   $5565           ; Go award winning selection B
                                        ;
5300: C6 02       LDB   #$02            ;
5302: FA 06 50    ORB   $0650           ;
5305: F7 06 50    STB   $0650           ; $0650 |= 0x02, exclude "Extra ball"
                                        ;
5308: C6 0A       LDB   #$0A            ;
530A: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x0A, "Lite Extra Ball"
530D: 27 11       BEQ   $5320           ;
530F: BD FA 1E    JSR   $FA1E           ;
5312: 7E 53 15    JMP   $5315           ;
5315: 25 09       BCS   $5320           ;
5317: 81 2C       CMPA  #$2C            ;
5319: 22 05       BHI   $5320           ;
531B: 7C 06 51    INC   $0651           ; Increment random number divisor
531E: 20 08       BRA   $5328           ;
                                        ;
5320: C6 02       LDB   #$02            ;
5322: FA 06 4F    ORB   $064F           ;
5325: F7 06 4F    STB   $064F           ; $064F |= 0x02, exclude "Lite Extra Ball"
                                        ;
5328: C6 0C       LDB   #$0C            ;
532A: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x0C, "Video Mode"
532D: 27 2E       BEQ   $535D           ;
532F: 8E 05 B5    LDX   #$05B5          ;
5332: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
5335: 7E 53 38    JMP   $5338           ;
5338: 6D 84       TST   ,X              ;
533A: 26 21       BNE   $535D           ;
                                        ;
533C: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
533F: 00 86                             ;  Search for 0x0086, Multiball Running
5341: 24 1A       BCC   $535D           ;
5343: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
5346: 00 A4                             ;
5348: 24 13       BCC   $535D           ;
534A: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
534D: 00 89                             ;
534F: 24 0C       BCC   $535D           ;
5351: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
5354: 00 A9                             ; ID 00A9 == "LOAD THE GUN" period at end of multiball
5356: 24 05       BCC   $535D           ;
5358: 7C 06 51    INC   $0651           ; Increment random number divisor
535B: 20 08       BRA   $5365           ;
                                        ;
535D: C6 08       LDB   #$08            ;
535F: FA 06 4F    ORB   $064F           ;
5362: F7 06 4F    STB   $064F           ; $064F |= 0x08, exclude "Video mode"
                                        ;
5365: C6 04       LDB   #$04            ;
5367: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x04, "Autofire"
536A: 27 35       BEQ   $53A1           ;
536C: 8E 05 B1    LDX   #$05B1          ; 0x05B1 == Number of Autofire awards given, per-player
536F: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
5372: 7E 53 75    JMP   $5375           ; <nop>
5375: 6D 84       TST   ,X              ;
5377: 26 28       BNE   $53A1           ;
                                        ;
```

```
5379: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
537C: 00 86                             ;  Search for 0x0086, Multiball running
537E: 24 21       BCC   $53A1           ;
5380: BD 88 F5    JSR   $88F5           ;
5383: 4C 7B 38                          ; GetCurrentPlayerIndexIntoAPlayerDataTableIntox()
5386: AE 88 22    LDX   $22,X           ; Getting player data table byte [22]
5389: 9F CE       STX   $CE             ;
538B: 8E 00 CE    LDX   #$00CE          ;
538E: 6D 84       TST   ,X              ;
5390: 26 0F       BNE   $53A1           ;
5392: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
5395: C6 0A       LDB   #$0A            ; B=0x0A
5397: 3D          MUL                   ; D=A*B (Current Ball multiplied by 0x0A)\
5398: E1 01       CMPB  $0001,X         ;
539A: 23 05       BLS   $53A1           ;
539C: 7C 06 51    INC   $0651           ; Increment random number divisor
539F: 20 08       BRA   $53A9           ;
                                        ;
53A1: C6 08       LDB   #$08            ;
53A3: FA 06 50    ORB   $0650           ;
53A6: F7 06 50    STB   $0650           ; $0650 |= 0x08, exclude "Autofire"
                                        ;
53A9: C6 07       LDB   #$07            ;
53AB: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x07, "Hurry up"
53AE: 27 1F       BEQ   $53CF           ;
53B0: 8E 05 A9    LDX   #$05A9          ; X = 0x05A9
53B3: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
53B6: 7E 53 B9    JMP   $53B9           ; <nop>
53B9: 6D 84       TST   ,X              ; Text per-player statistic from 0x05A9
53BB: 26 12       BNE   $53CF           ;
53BD: BD 84 49    JSR   $8449           ; GetLampLitState() C-clear when lamp is on
53C0: 0B                                ; 0x0B = "Left Return Lane" (lite hurry-up)
53C1: 24 0C       BCC   $53CF           ;
                                        ;
53C3: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
53C6: 00 86                             ;  Search for 0x0086, Multiball running
53C8: 24 05       BCC   $53CF           ;
53CA: 7C 06 51    INC   $0651           ; Increment random number divisor
53CD: 20 08       BRA   $53D7           ;
                                        ;
53CF: 86 40       LDA   #$40            ;
53D1: BA 06 50    ORA   $0650           ;
53D4: B7 06 50    STA   $0650           ; $0650 |= 0x40, exclude "Hurry up"
                                        ;
53D7: C6 0F       LDB   #$0F            ;
53D9: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x0F, "Lite hurry up"
53DC: 27 1F       BEQ   $53FD           ;
53DE: 8E 05 A9    LDX   #$05A9          ;
53E1: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
53E4: 7E 53 E7    JMP   $53E7           ;
53E7: 6D 84       TST   ,X              ;
53E9: 26 12       BNE   $53FD           ;
53EB: BD 84 49    JSR   $8449           ; GetLampLitState() C-clear when lamp is on
53EE: 0B                                ; 0x0B = "Left Return Lane" (lite hurry-up)
53EF: 24 0C       BCC   $53FD           ;
                                        ;
53F1: BD 86 90    JSR   $8690           ; SearchLinkedListForId()  // c-clear means ID is found
53F4: 00 86                             ;  Search for 0x0086, Multiball running
53F6: 24 05       BCC   $53FD           ;
53F8: 7C 06 51    INC   $0651           ; Increment random number divisor
53FB: 20 08       BRA   $5405           ;
                                        ;
```

```
53FD: 86 40       LDA   #$40           ;
53FF: BA 06 4F    ORA   $064F          ;
5402: B7 06 4F    STA   $064F          ; $064F |= 0x40, exclude "Lite hurry up"
                                       ;
5405: C6 10       LDB   #$10           ;
5407: F1 06 12    CMPB  $0612          ; Check if previous DB award was 0x10, "3,000,000"
540A: 27 16       BEQ   $5422          ;
540C: 8E 05 C1    LDX   #$05C1         ; X = 0x05C1
540F: BD FB 29    JSR   $FB29          ; IncrementXByPlayerIndexNumber()
5412: 7E 54 15    JMP   $5415          ;
5415: BD B1 D1    JSR   $B1D1          ; GameOnLastBallCheckCBitClearCurrentBallInA()
5418: 48          ASLA                 ; Multiply current ball value in A by 2
5419: A1 84       CMPA  ,X             ;
541B: 25 05       BCS   $5422          ;
541D: 7C 06 51    INC   $0651          ; Increment random number divisor
5420: 20 08       BRA   $542A          ;
                                       ;
5422: 86 80       LDA   #$80           ;
5424: BA 06 4F    ORA   $064F          ;
5427: B7 06 4F    STA   $064F          ; $064F |= 0x80, exclude "3,000,000"
                                       ;
542A: C6 0D       LDB   #$0D           ;
542C: F1 06 12    CMPB  $0612          ; Check if previous DB award was 0x0D, "Like Kickback"
542F: 27 27       BEQ   $5458          ;
5431: BD 84 49    JSR   $8449          ; GetLampLitState() C-clear when lamp is on
5434: 09                               ; 0x09 = Kickback"
5435: 24 21       BCC   $5458          ;
5437: BD 88 F5    JSR   $88F5          ;
543A: 4C 7B 38                         ; GetCurrentPlayerIndexIntoAPlayerDataTableIntox()
543D: AE 88 22    LDX   $22,X          ;
5440: 9F CE       STX   $CE            ;
5442: 8E 00 CE    LDX   #$00CE         ;
5445: 6D 84       TST   ,X             ;
5447: 26 0F       BNE   $5458          ;
5449: BD B1 D1    JSR   $B1D1          ; GameOnLastBallCheckCBitClearCurrentBallInA()
544C: C6 0A       LDB   #$0A           ;
544E: 3D          MUL                  ;
544F: E1 01       CMPB  $0001,X        ;
5451: 23 05       BLS   $5458          ;
5453: 7C 06 51    INC   $0651          ; Increment random number divisor
5456: 20 08       BRA   $5460          ;
                                       ;
5458: 86 10       LDA   #$10           ;
545A: BA 06 4F    ORA   $064F          ;
545D: B7 06 4F    STA   $064F          ; $064F |= 0x10, exclude "Lite kickback"
                                       ;
5460: C6 0B       LDB   #$0B           ;
5462: F1 06 12    CMPB  $0612          ; Check if previous DB award was 0x0B, "500,000"
5465: 27 05       BEQ   $546C          ;
5467: 7C 06 51    INC   $0651          ; Increment random number divisor
546A: 20 08       BRA   $5474          ;
                                       ;
546C: 86 04       LDA   #$04           ;
546E: BA 06 4F    ORA   $064F          ;
5471: B7 06 4F    STA   $064F          ; $064F |= 0x04, exclude "500,000"
                                       ;
5474: C6 03       LDB   #$03           ;
5476: F1 06 12    CMPB  $0612          ;
5479: 27 05       BEQ   $5480          ;
547B: 7C 06 51    INC   $0651          ; Increment random number divisor
547E: 20 08       BRA   $5488          ;
                                       ;
```

```
5480: 86 04       LDA   #$04            ;
5482: BA 06 50    ORA   $0650           ;
5485: B7 06 50    STA   $0650           ; $0650 |= 0x04
                                        ;
5488: C6 01       LDB   #$01            ;
548A: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x01, "Chase loop"
548D: 27 0B       BEQ   $549A           ;
548F: BD 84 49    JSR   $8449           ; GetLampLitState() C-clear when lamp is on
5492: 24                                ; 0x24 = "3,000,000"
5493: 24 05       BCC   $549A           ;
5495: 7C 06 51    INC   $0651           ; Increment random number divisor
5498: 20 08       BRA   $54A2           ;
                                        ;
549A: 86 01       LDA   #$01            ;
549C: BA 06 50    ORA   $0650           ;
549F: B7 06 50    STA   $0650           ; $0650 |= 0x01, exclude "Chase loop"
                                        ;
54A2: C6 08       LDB   #$08            ;
54A4: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x08, "1,000,000"
54A7: 27 17       BEQ   $54C0           ;
54A9: 8E 05 C1    LDX   #$05C1          ; X = 0x05C1
54AC: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
54AF: 7E 54 B2    JMP   $54B2           ;
54B2: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
54B5: 48          ASLA                  ;
54B6: 48          ASLA                  ;
54B7: A1 84       CMPA  ,X              ;
54B9: 25 05       BCS   $54C0           ;
54BB: 7C 06 51    INC   $0651           ; Increment random number divisor
54BE: 20 08       BRA   $54C8           ;
                                        ;
54C0: 86 80       LDA   #$80            ;
54C2: BA 06 50    ORA   $0650           ;
54C5: B7 06 50    STA   $0650           ; $0650 |= 0x80, exclude "1,000,000"
                                        ;
54C8: C6 09       LDB   #$09            ;
54CA: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x09, "Multiball"
54CD: 27 2E       BEQ   $54FD           ;
54CF: 8E 05 C9    LDX   #$05C9          ; X = 0x05C9
54D2: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
54D5: 7E 54 D8    JMP   $54D8           ;
54D8: 6D 84       TST   ,X              ;
54DA: 26 21       BNE   $54FD           ;
54DC: BD 88 F5    JSR   $88F5           ;
54DF: 4C 7B 38                          ; GetCurrentPlayerIndexIntoAPlayerDataTableIntox()
54E2: AE 88 22    LDX   $22,X           ;
54E5: 9F CE       STX   $CE             ;
54E7: 8E 00 CE    LDX   #$00CE          ;
54EA: 6D 84       TST   ,X              ;
54EC: 26 0F       BNE   $54FD           ;
54EE: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
54F1: C6 14       LDB   #$14            ;
54F3: 3D          MUL                   ;
54F4: E1 01       CMPB  $0001,X         ;
54F6: 23 05       BLS   $54FD           ;
54F8: 7C 06 51    INC   $0651           ; Increment random number divisor
54FB: 20 08       BRA   $5505           ;
                                        ;
54FD: 86 01       LDA   #$01            ;
54FF: BA 06 4F    ORA   $064F           ;
5502: B7 06 4F    STA   $064F           ; $064F |= 0x01, exclude "Multiball"
                                        ;
```

```
5505: C6 05       LDB   #$05            ;
5507: F1 06 12    CMPB  $0612           ; Check if previous DB award was 0x05, "Security pass"
550A: 27 1F       BEQ   $552B           ;
550C: 8E 05 95    LDX   #$0595          ; X = 0x0595
550F: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
5512: 7E 55 15    JMP   $5515           ;
5515: A6 84       LDA   ,X              ;
5517: 8E 05 99    LDX   #$0599          ;
551A: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
551D: 7E 55 20    JMP   $5520           ;
5520: AB 84       ADDA  ,X              ;
5522: 81 07       CMPA  #$07            ;
5524: 22 05       BHI   $552B           ;
5526: 7C 06 51    INC   $0651           ; Increment random number divisor
5529: 20 08       BRA   $5533           ;
                                        ;
552B: 86 10       LDA   #$10            ;
552D: BA 06 50    ORA   $0650           ;
5530: B7 06 50    STA   $0650           ; $0650 |= 0x10, exclude "Security pass"
                                        ;
5533: B6 06 51    LDA   $0651           ; Get random number divisor, 0..14 at this point
                                        ;
5536: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA()
5539: 4C          INCA                  ; Increment random number
553A: 97 D1       STA   $D1             ; $D1 = random number
553C: 86 01       LDA   #$01            ; A = 0x01
553E: 97 D0       STA   $D0             ; $D0 = 0x01, Bitmask used to test bits in $0650 & $064F
5540: 5F          CLRB                  ; B = 0x00
                                        ;
                                        ;-----------------------------------------
                                        ;  Exclusion Bits
                                        ;-----------------------------------
                                        ; $0650 & 0x01 --> exclude "Chase loop"
                                        ; $0650 & 0x02 --> exclude "Extra ball"
                                        ; $0650 & 0x04 --> exclude "100,000"
                                        ; $0650 & 0x08 --> exclude "Autofire"
                                        ; $0650 & 0x10 --> exclude "Security pass"
                                        ; $0650 & 0x20 --> exclude "Lite Special"
                                        ; $0650 & 0x40 --> exclude "Hurry up"
                                        ; $0650 & 0x80 --> exclude "1,000,000"
                                        ;-----------------------------------------
5541: B6 06 50    LDA   $0650           ; A = $0650 (byte that got set with various bits, above)
                                        ;
5544: 5C          INCB                  ;--\ B++
5545: 95 D0       BITA  $D0             ;  | Test $0650 byte has the current $D0 bit set
5547: 26 04       BNE   $554D           ;  |
5549: 0A D1       DEC   $D1             ;  | If $0650 bit $D0 is set set then $D1--
554B: 27 18       BEQ   $5565           ;  | If $D1 reached 0x00 go award winning selection B
                                        ;  |
554D: 09 D0       ROL   $D0             ;  | $D0 <<= 1
554F: 24 F3       BCC   $5544           ;--/ Keep looping until the all 8 bits of $0650 tested
                                        ;
5551: 09 D0       ROL   $D0             ; $D0 <<= 1  Reset the $D0 back to 0x01 w/left shift
                                        ;
                                        ;-----------------------------------------
                                        ;  Exclusion Bits
                                        ;-----------------------------------
                                        ; $064F & 0x01 --> exclude "Multiball"
                                        ; $064F & 0x02 --> exclude "Lite Extra Ball"
                                        ; $064F & 0x04 --> exclude "500,000"
                                        ; $064F & 0x08 --> exclude "Video mode"
                                        ; $064F & 0x10 --> exclude "Lite kickback"
```

```
                                        ; $064F & 0x20 --> exclude "Special"
                                        ; $064F & 0x40 --> exclude "Lite hurry up"
                                        ; $064F & 0x80 --> exclude "3,000,000"
                                        ;-----------------------------------------
5553: B6 06 4F    LDA   $064F           ; A = $064F (byte that got set with various bits, above)
                                        ;
5556: 5C          INCB                  ;--\ B++
5557: 95 D0       BITA  $D0             ;  | Test $064F byte has the current $D0 bit set
5559: 26 04       BNE   $555F           ;  |
555B: 0A D1       DEC   $D1             ;  | If $064F bit $D0 is set set then $D1--
555D: 27 06       BEQ   $5565           ;  | If $D1 reached 0x00 go award winning selection B
                                        ;  |
555F: 09 D0       ROL   $D0             ;  | $D0 <<= 1
5561: 24 F3       BCC   $5556           ;--/ Keep looping until all 8 bits of $0650 tested
                                        ;
5563: C6 0B       LDB   #$0B            ; No matches, so set winning selection: 0x0B "500,000"
                                        ;
5565: F7 06 12    STB   $0612           ; Save winning # from B into $0612 for later access.
                                        ;-------------------------------------------------------
                                        ; B has winning value which had also been saved in $0612
                                        ;  0x01 == "CHASE LOOP"
                                        ;  0x02 == "EXTRA BALL"
                                        ;  0x03 == "100,000"
                                        ;  0x04 == "AUTOFIRE"
                                        ;  0x05 == "SECURITY PASS"
                                        ;  0x06 == "LITE SPECIAL"
                                        ;  0x07 == "HURRY UP"
                                        ;  0x08 == "1,000,000"
                                        ;  0x09 == "MULTIBALL"
                                        ;  0x0A == "LITE EXTRA BALL"
                                        ;  0x0B == "500,000"
                                        ;  0x0C == "VIDEO MODE"
                                        ;  0x0D == "LITE KICKBACK"
                                        ;  0x0E == "SPECIAL"
                                        ;  0x0F == "LITE HURRY UP"
                                        ;  0x10 == "3,000,000"
                                        ;-------------------------------------------------------
5568: BD A7 25    JSR   $A725           ; GetPseudoRandomNumberIntoA()
556B: 84 07       ANDA  #$07            ; Getting random number from 0..7 for database award
556D: 4C          INCA                  ; Make it 1-based 1..8 value
556E: 34 02       PSHS  A               ; Save random number 1..8 onto stack
5570: 97 D1       STA   $D1             ; Save random number 1..8 into $D1
5572: 8E 00 0C    LDX   #$000C          ; X gets 0x000C, starting string index at "CHASE LOOP"
5575: D0 D1       SUBB  $D1             ; B gets B - $D1.  Subtract the (1..8) value from it
5577: C1 11       CMPB  #$11            ;
5579: 23 02       BLS   $557D           ; If B is less than or equal to 0x11 then skip over
557B: CB 10       ADDB  #$10            ; If B is greater than 0x0x11 then add 0x10 to it.
557D: 3A          ABX                   ; Add to X
557E: 35 04       PULS  B               ; B gets random number 1..8 from stack
                                        ;
5580: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
5583: 1C                                ; 0x1C == DatabaseSelector() $72B9,33
                                        ;
5584: B6 06 12    LDA   $0612           ;
                                        ;
5587: BD 8B 3D    JSR   $8B3D           ; AddLinkedListEntry()
558A: 00 8C                             ; ID 008C
558C: 55 90 31                          ; $5590,31
                                        ;
558F: 39          RTS                   ;
                                        ;
;---------------------------------------;-----------------------------------------
```

The logic, above, has a complicated set of logic to determine the award which is finally determined when the code reaches $5565,31 where the winning selection is stored in RAM at $0612. Once this winning selection is determined another random number between 1 and 8 is then determined when instruction $556E,31 is reached. This random value establishes which of the 8 shown awards is the winner. For example if the winning selection is determined to be "1,000,000" and the random number 1..8 is determined to be 8, the remaining logic calculates starting string to be shown as "Chase Loop" so that the 8th shown award is the winning award "1,000,000".

Readers are encouraged to review the logic and use a pinball emulator to trace through the code to further determine how the logic behaves, if so inclined.

Per-Player In-Game Statistics Analysis L-8

The L8.4 database award enhancement will require the game to track the number of database awards awarded for each player throughout the game. Such statistic will be used to ensure that all players get the same 1st database award, 2nd database award, and so on. The L-8 game code does not track this statistic so an analysis of per-player in-game statistic memory was done to see if there are any unused regions that the L8.4 can utilize for this purpose.

When accessing the per-player statistics the game code will:

- Load X with address of first statistic byte (for player 1)
- Call function $FB29 which increases the value of X by 0, 1, 2, 3 if current player is 1, 2, 3, 4.

After $FB29 the value of X points to the current player's byte in the 4 bytes of memory for a given in-game statistic.

At start of a game, the per-player statistics are reset at this code from $617C,3B, ROM offset 0x6E17C. This code reveals the starting RAM location is $0591 and it clears out memory through $060B.

```
617C: 8E 05 91    LDX   #$0591          ; $0591 start of per-player in-game statistics
617F: 4F          CLRA                  ;
6180: 8C 06 0C    CMPX  #$060C          ;-\
6183: 27 04       BEQ   $6189           ; |
6185: A7 80       STA   ,X+             ; | Clear out the per-player in-game stats $0591-$060B
6187: 20 F7       BRA   $6180           ;-/
```

A survey of L-8 game ROM was done to determine the RAM locations used for per-player statistics. Existing game code analysis and pinball emulator can be used to see when each statistic is incremented. A preliminary result of the analysis is shown in the table below. This table is based on a search of L-8 for calls to $FB29 and an analysis of the value used in X prior to such call. This table is only a starting point in developing a full understanding in how the game tracks these per-player in-game statistics.

Per-Player RAM for player 1..4	Description
$0591 - $0594	Per-player bonus-X level

$0595 - $0598	Per-player left security level lamps
$0599 - $059C	Per-player right security level lamps
$059D - $05A0	Per-player jackpots awarded
$05A1 - $05A4	Per-player lit 5-bank target hits from cannon
$05A5 - $05A8	Per-player 3-bank standup target completions
$05A9 - $05AC	Per-player hurry-ups achieved
$05AD - $05B0	Per-player extra-balls awarded
$05B1 - $05B4	Per-player autofire awards via database
$05B5 - $05B8	Per-player video-modes
$05B9 - $05BC	Per-player kickback lit
$05BD - $05C0	Per-player lit-shot hit counter (advances at various lit shots being hit)
$05C1 - $05C4	Per-player lit-shot hit counter (advances at various lit shots being hit)
$05C5 - $05C8	*Appears to be unused*
$05C9 - $05CC	Per-player multiballs achieved through any means
$05CD - $05D0	Per-player skill-shots made
$05D1 - $05D4	Per-player number of super jackpots awarded
$05D5 - $05D8	Per-player multiballs achieved with cannon-shots only
$05D9	Number of specials awarded this game for all players, combined
$05DA	Cannon state as set by mark/home switches during its swing
$05DB - $05DF	Player 1 accumulated bonus in binary coded decimal format, 5 bytes, 10 digits
$05E0 - $05E4	Player 2 accumulated bonus in binary coded decimal format, 5 bytes, 10 digits
$05E5 - $05E9	Player 3 accumulated bonus in binary coded decimal format, 5 bytes, 10 digits
$05EA - $05EE	Player 4 accumulated bonus in binary coded decimal format, 5 bytes, 10 digits
$05EF - $05F3	Hurry-up countdown score value in BCD, 10 digits (first 2 digits always 00)
$05F4 - $05F8	At skill shot award this gets the points vale in BCD , 10 digits (first/last ignored)
$05F9	Stores the current super jackpot 5-bank moving lamp index
$05FA	Stores the current skill shot winning lamp index
$05FB	Gets set to 0x00 at 6.5 seconds after right ramp shot, not set anywhere else(?)
$05FC	Right loop “millions” count-up value when repeatedly hit
$05FD	*Appears to be unused*
$05FE	Tracks next jackpot multiplier 01, 02, or 03 that will be attempted
$05FF - $0602	Per-player bitmap of 5-bank targets needed for multiball. 0x01 = Target 1 High
$0603 - $0606	Per-player number of hits remaining for multiball
$0607	Stores bad 5-bank switch count during multiball 5-bank lamp lit logic
$0608	Tracks end-of-game music playing through HSTD, match and end-of-game
$0609	Related to right chase-loop animation sequences to ensure proper display
$060A	Counts pop bumper hits when 3 rollover switches are bad, to award rollover
$060B	Tracks number of millions achieved during Payback Time
$060C	Related to cannon. Set to 0x01 when gun is calibrating.
$060D	Tracks the most recent left-loop award
$060E	Related to extra-ball animation sequence to ensure proper display
$060F	Used for tracking awards to give at ramp left/right ramp shots
$0610 - $0611	Two bytes used during Hurry-up countdown sequence for state tracking
$0612	Stores winning database award number during database award sequence

$0613	Flag used during escape-route left lock to determine if award is to be given
$0614 - $0617	Per-player skill-shot timer reset value
$0618 - $061B	Per-player drop-target reset timer value
$061C - $061F	Per-player drop-target reset timer value (init is same as $0618 - $061B)
$0620 - $0623	Per-player drop-target reset timer value (init is same as $0618 - $061B)
$0624 - $0627	Per-player hurry-up timer reset value
$0628 - $062B	Per-player 3-bank timer reset value
$062C - $062F	*Appears to be unused*
$0630 - $0633	Per-player jackpot timer reset value

As highlighted in the table above, the region from $05C5 - $05C8 is unused in L-8 and can be used in L8.4 for tracking the number of database awards for each player. The determination that this region is unused is based on analysis of the game ROM, finding no opcodes that appear to cite these 4 bytes. Also the emulator was used with a watchpoint set on this 4 byte region. Game was played with all possible awards having been hit multiple times without a single read/write to this region (other than the game-start reset of the entire block of memory).

Database Award Logic Enhancements for L8.4

For the tournament mode Database award fixups in L8.4, the previously described solution used in the Super Jackpot lamp movements will also be used so that a seemingly "random" number can be used while also ensuring all players in a multi-player game have identical opportunities.

The goal is that all players will have the same database award for their first achieved database. The second database award may be different from the first database award and all players will get the same second database award, and so on.

The game bookkeeping statistics can be used to derive a basis value and the per-player database awards value will be used to determine the current "random" numbers that are applicable to the current database award. For each database award there are two such "random" numbers that are used:

- Value 1..16, defining which is the winning database award, and
- Value 1..8, defining where on the list of items the winning award is shown.

The table, below, shows which of the game bookkeeping statistics are used in deriving a "random" number for each of the two needed random numbers at each database award:

Bookkeeping Entry	**Addr**	**Bookkeeping Statistic Participation**	
		Database Award	Award Position
Bookkeeping B.3 08 "MATCH AWARDS"	$18A3	✓	
Bookkeeping B.3 28 "1 PLAYER GAMES"	$18EB	✓	✓
Bookkeeping B.3 29 "2 PLAYER GAMES"	$18F1	✓	✓
Bookkeeping B.3 30 "3 PLAYER GAMES"	$18F7	✓	✓
Bookkeeping B.3 31 "4 PLAYER GAMES"	$18FD	✓	✓
Bookkeeping B.5 01 "0-1.9 M. SCORE"	$1921	✓	
Bookkeeping B.5 02 "2-4.9 M. SCORE"	$1927		✓

Bookkeeping B.5 03 "3-9.9 M. SCORE"	$192D	✓	
Bookkeeping B.5 04 "10-19 M. SCORE"	$1933		✓
Bookkeeping B.5 05 "20-29 M. SCORE"	$1939	✓	
Bookkeeping B.5 06 "30-29 M. SCORE"	$193F		✓
Bookkeeping B.5 07 "40-49 M. SCORE"	$1945	✓	
Bookkeeping B.5 08 "50-69 M. SCORE"	$194B		✓
Bookkeeping B.5 09 "70-99 M. SCORE"	$1951	✓	
Bookkeeping B.5 10 "100-149 M. SCORE"	$1957		✓
Bookkeeping B.5 11 "150-199 M. SCORE"	$195D	✓	
Bookkeeping B.5 12 "200-299 M. SCORE"	$1963		✓
Bookkeeping B.5 13 "OVER 300 MILLION"	$1969	✓	
Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"	$196F		✓
Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"	$1975	✓	
Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"	$197B		✓
Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"	$1981	✓	
Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"	$1987		✓
Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"	$198D	✓	
Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"	$1993		✓
Bookkeeping B.5 21 "GAME TIME 4-5 M."	$1999	✓	
Bookkeeping B.5 22 "GAME TIME 5-6 M."	$199F		✓
Bookkeeping B.5 23 "GAME TIME 6-8 M."	$19A5	✓	
Bookkeeping B.5 24 "GAME TIME 8-10 M."	$19AB		✓
Bookkeeping B.5 25 "GAME TIME 10-15 M"	$19B1	✓	
Bookkeeping B.5 26 "GAME TIME > 15 M"	$19B7		✓

In addition to the above accumulated game statistics, the per-player database award counter is also used when deriving both of the "random" numbers. The 8-bit sum of all of these statistics will be the "seed" used to lookup a seemingly "random" byte from the non-banked region of ROM.

The chosen byte from ROM is passed through an algorithm specific to the value being obtained.

Variable Behavior	**Needed Values**		**Formula to derive value X from byte B (C-like syntax)**
	Range	# of values	
Database Award	1..16	16	X = (B & 0x0F) + 1
Award Position	1..8	8	X = (B & 0x07) + 1

For each of these 2 "random" numbers, the "seed" value is used to lookup a byte from ROM relative to the arbitrarily chosen address $F900 (ROM offset 0x7F900). This region of ROM appears to have a mix of byte values with non-repeating bytes, as shown below:

Offset:	Bytes:	ANSI Text:
0007F880	C1 F8 2F 09 C1 08 2C 0A 1A 01 4C 35 92 1A 01 4F	Á ø / Á ◘ , → ┌ L 5 ′ → ┌ O
0007F890	35 92 1C FE 4C 35 92 34 16 BD F9 09 B1 17 A6 27	5 ′ þ L 5 ′ 4 ┬ ½ ù ± ┤ ¦ '
0007F8A0	03 BD F9 24 C6 02 34 04 5A 8E F8 EA 3A A6 84 BD	└ ½ ù $ Æ ┐ 4 ┘ Z Ž ø ê : ¦ „ ½
0007F8B0	92 DE BD FA 13 AB 01 81 63 23 0E A6 84 81 08 2C	′ Þ ½ ú ‼ « ┌ ▯ c # ♫ ¦ „ ▯ ◘ ,
0007F8C0	06 6F 01 6C 84 20 04 86 63 A7 01 35 04 5A 26 D6	− o ┌ l „ ┘ † c § ┌ 5 ┘ Z & Ö
0007F8D0	7C 17 A5 BD F9 09 B7 17 A6 35 96 34 16 BD F9 09	| ┤ ¥ ½ ù · ┤ ¦ 5 – 4 ┬ ½ ù
0007F8E0	B1 17 A6 27 03 BD F9 24 35 96 02 01 03 00 04 00	± ┤ ¦ ' └ ½ ù $ 5 – ┐ ┌ └ ┘
0007F8F0	34 16 C6 02 34 04 8E F8 EC 5A 3A A6 84 BD FA 13	4 ┬ Æ ┐ 4 ┘ Ž ø ì Z : ¦ „ ½ ú ‼
0007F900	A7 03 35 04 5A 26 ED 35 96 34 14 8E 17 9D C6 02	§ └ 5 ┘ Z & í 5 – 4 ¶ Ž ┤ ▯ Æ ┐
0007F910	B6 17 A5 AB 84 AB 01 AB 02 AB 03 30 04 5A 26 F3	¶ ┤ ¥ « „ « ┌ « ┐ « └ O ┘ Z & ó
0007F920	88 33 35 94 34 16 8E 17 9D C6 02 6F 84 6F 01 6F	ˆ 3 5 ″ 4 ┬ Ž ┤ ▯ Æ ┐ o „ o ┌ o
0007F930	02 30 04 5A 26 F5 7F 17 A5 BD F8 F0 BD F9 09 B7	┐ O ┘ Z & õ ⌂ ┤ ¥ ½ ø ð ½ ù ·
0007F940	17 A6 35 96 34 16 BD F9 09 B1 17 A6 27 03 BD F9	┤ ¦ 5 – 4 ┬ ½ ù ± ┤ ¦ ' └ ½ ù
0007F950	24 BD FA 13 A6 84 81 F8 2F 02 6A 84 A6 02 81 FF	$ ½ ú ‼ ¦ „ ▯ ø / ┐ j „ ¦ ┐ ▯ ÿ
0007F960	27 02 6C 02 BD F9 09 B7 17 A6 35 96 34 10 BD FA	' ┐ l ┐ ½ ù · ┤ ¦ 5 – 4 ┼ ½ ú
0007F970	13 E6 03 35 90 34 16 B6 17 A5 81 F0 24 02 35 96	‼ æ └ 5 ▯ 4 ┬ ¶ ┤ ¥ ▯ ð $ ┐ 5 –

To illustrate the different possible values, the table below shows all of the possible values that this formula will produce.

Variable Behavior

Database Award

Award #	Award Name	Total Occurrances
1	Chase Loop	9
2	Extra Ball	18
3	100,000	16
4	Autofire	16
5	Security Pass	32
6	Lite Special	18
7	Hurry Up	37
8	1,000,000	21
9	Multiball	8
10	Lite Extra Ball	16
11	500,000	17
12	Video Mode	5
13	Lite Kickback	9
14	Special	18
15	Lite Hurry Up	7
16	3,000,000	9

Award Position

#	Column	Row	Total Occurrances

Formula Results

2	9	16	10	2	9	13	11	11	2	13	6	3	11	2	16
6	3	13	15	13	6	3	5	7	14	10	10	2	8	7	8
4	14	10	5	7	3	5	5	11	15	9	11	11	7	5	14
3	15	14	11	4	12	2	2	4	4	15	7	5	2	9	13
7	16	2	13	5	1	5	7	4	8	2	6	5	11	7	7
13	8	6	14	10	10	8	8	7	6	7	5	7	14	10	10
2	8	7	8	4	14	10	5	6	7	3	2	4	1	5	1
5	7	7	3	5	5	15	9	13	11	11	7	5	14	11	4
8	4	6	5	11	7	14	6	7	5	5	15	8	14	7	3
7	8	6	12	5	12	2	12	3	12	4	1	5	11	7	4
9	4	6	5	5	7	15	8	14	7	3	16	5	16	2	16
3	1	5	11	7	6	16	8	6	14	9	1	14	10	10	8
8	7	6	7	5	7	14	10	10	2	8	7	8	4	14	10
5	14	11	4	7	5	2	9	16	3	11	5	7	3	2	16
8	3	13	3	14	10	10	8	8	7	6	7	5	1	14	11
4	7	4	6	1	5	7	7	8	6	2	1	5	3	6	7

#2	#1	#8	#2	#2	#1	#5	#3	#3	#2	#5	#6	#3	#3	#2	#8
#6	#3	#5	#7	#5	#6	#3	#5	#7	#6	#2	#2	#2	#8	#7	#8
#4	#6	#2	#5	#7	#3	#5	#5	#3	#7	#1	#3	#3	#7	#5	#6
#3	#7	#6	#3	#4	#4	#2	#2	#4	#4	#7	#7	#5	#2	#1	#5
#7	#8	#2	#5	#5	#1	#5	#7	#4	#8	#2	#6	#5	#3	#7	#7
#5	#8	#6	#6	#2	#2	#8	#8	#7	#6	#7	#5	#7	#6	#2	#2

#1	Left	1	17	#2 #8 #7 #8 #4 #6 #2 #5 #6 #7 #3 #2 #4 #1 #5 #1
#2	Left	2	34	#5 #7 #7 #3 #5 #5 #7 #1 #5 #3 #3 #7 #5 #6 #3 #4
#3	Left	3	33	#8 #4 #6 #5 #3 #7 #6 #6 #7 #5 #5 #7 #8 #6 #7 #3
#4	Left	4	21	#7 #8 #6 #4 #5 #4 #2 #4 #3 #4 #4 #1 #5 #3 #7 #4
#5	Right	1	41	#1 #4 #6 #5 #5 #7 #7 #8 #6 #7 #3 #8 #5 #8 #2 #8
#6	Right	2	36	#3 #1 #5 #3 #7 #6 #8 #8 #6 #6 #1 #1 #6 #2 #2 #8
#7	Right	3	44	#8 #7 #6 #7 #5 #7 #6 #2 #2 #2 #8 #7 #8 #4 #6 #2
#8	Right	4	30	#5 #6 #3 #4 #7 #5 #2 #1 #8 #3 #3 #5 #7 #3 #2 #8
				#8 #3 #5 #3 #6 #2 #2 #8 #8 #7 #6 #7 #5 #1 #6 #3
				#4 #7 #4 #6 #1 #5 #7 #7 #8 #6 #2 #1 #5 #3 #6 #7

The tables above show a relative mix of “random” values when the game is in Tournament mode and players acquire a database award. The resulting look and feel will be that of random award however all players will be given equal gameplay throughout the current game.

Extra consideration for the following awards is also given:

- Extra Ball and Lite Extra Ball. The L8.4 code will provide additional benefit during Tournament Mode, if the game is also configured for no extra balls (through adjustment A.1 03) then these two awards will not be given. When the algorithm is normally going to award these awards, alternate, fixed, awards can be given instead (refer to L8.4 code, below). Although such logic could have been moved to where all database awards are given (regardless of Tournament Mode) the decision is to have this only in the L8.4 addition so as to retain existing L-8 behavior when Tournament Mode is off. *Later it was noted that the L-8 Database award logic already excludes Extra Ball and Lite Extra Ball when player is ineligible for EB. This is done by use of the $FA1E function in the original L-8 code depicted earlier.*
- Special and Lite Special. Although a lot of tournaments have no benefit for players achieving these awards, the L8.4 will allow these to be awarded in the event some tournaments might assign certain meanings for when players achieve a special during game play. Since the L8.4 will give all players equal database awards, the award will be evenly achievable by all players and equally ignorable in tournaments where special has no particular benefit.

Database Award Logic Code Update for L8.4

The code changes for L8.4 will insert a function call into the existing L-8 database award logic at the moment where original L-8 code is determining the two items, Award and Position. The code will jump to new L8.4 code which will perform the following high-level logic:

- If Tournament Mode is off, resume original L-8 logic, else
- If Tournament Mode is on, determine new “random” values for:
  - The Database Award for the current player, and
  - The position on the display where the chosen award will be shown, and
  - Return back to normal L-8 logic using the overridden database award data.

The new L8.4 Database award logic is placed in bank $34 where plenty of unused ROM space is available.

As shown in full above, the database award function reaches this moment when the winning award has been determined and the award position is determined:

```
                                        ;
5563: C6 0B       LDB   #$0B           ; No matches, so set winning selection: 0x0B "500,000"
                                        ;
5565: F7 06 12    STB   $0612          ; Save winning # from B into $0612 for later access.
                                        ;-------------------------------------------------------
                                        ; B has winning value which had also been saved in $0612
                                        ;  0x01 == "CHASE LOOP"
                                        ;  0x02 == "EXTRA BALL"
                                        ;  0x03 == "100,000"
                                        ;  0x04 == "AUTOFIRE"
                                        ;  0x05 == "SECURITY PASS"
                                        ;  0x06 == "LITE SPECIAL"
                                        ;  0x07 == "HURRY UP"
                                        ;  0x08 == "1,000,000"
                                        ;  0x09 == "MULTIBALL"
                                        ;  0x0A == "LITE EXTRA BALL"
                                        ;  0x0B == "500,000"
                                        ;  0x0C == "VIDEO MODE"
                                        ;  0x0D == "LITE KICKBACK"
                                        ;  0x0E == "SPECIAL"
                                        ;  0x0F == "LITE HURRY UP"
                                        ;  0x10 == "3,000,000"
                                        ;-------------------------------------------------------
                                        ;
5568: BD A7 25    JSR   $A725          ; GetPseudoRandomNumberIntoA()
556B: 84 07       ANDA  #$07           ; Getting random number from 0..7 for database award
556D: 4C          INCA                 ; Make it 1-based 1..8 value
```

The updated code for L8.4 is as follows:

```
5563: C6 0B       LDB   #$0B           ; No matches, so set winning selection: 0x0B "500,000"
                                        ;
~~5565: F7 06 12    STB   $0612          ; Save winning # from B into $0612 for later access.~~
~~5568: BD A7 25    JSR   $A725          ; GetPseudoRandomNumberIntoA()~~
                                        ;
5565: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
5568: 6B C0 34                         ; L84DatabaseTournamentModeEnhancement()
                                        ;
556B: 84 07       ANDA  #$07           ; Getting random number from 0..7 for database award
556D: 4C          INCA                 ; Make it 1-based 1..8 value
```

The 6 bytes where the winning award is saved to $0612 and random number is derived for display placement is replaced with a 6-byte call to new function in bank $34. The new L8.4 code will perform the tasks of updating $0612 and getting the random number into A before returning.

The new code in bank $34 is shown below. Note the logic for adding up bookkeeping data is copied from previous L8.4 work done for Super Jackpot lamp movements. Rather than modify the already established and tested Super Jackpot code, it is simply duplicated and enhanced here.

As indicated in the updated code, above, the entry point into the new code is at $6BC0,34 (ROM offset 0x52BC0). The entire set of new code is inserted into unused ROM region starting at $6B1D,34 (ROM offset 0x52B1D).

```
---------------------------------------;------------------------------------------------------
                                       ;
                                       ; "Random" Seed Determinator 1
                                       ;------------------------------
                                       ; Table of Bookkeeping Storage Locations (L-8)
                                       ;
6B1D: 18 A3                            ; Bookkeeping B.3 08 "MATCH AWARDS"
6B1F: 18 EB                            ; Bookkeeping B.3 28 "1 PLAYER GAMES"
6B21: 18 F1                            ; Bookkeeping B.3 29 "2 PLAYER GAMES"
6B23: 18 F7                            ; Bookkeeping B.3 30 "3 PLAYER GAMES"
6B25: 18 FD                            ; Bookkeeping B.3 31 "4 PLAYER GAMES"
6B27: 19 21                            ; Bookkeeping B.5 01 "0-1.9 M. SCORE"
6B29: 19 2D                            ; Bookkeeping B.5 03 "3-9.9 M. SCORE"
6B2B: 19 39                            ; Bookkeeping B.5 05 "20-29 M. SCORE"
6B2D: 19 45                            ; Bookkeeping B.5 07 "40-49 M. SCORE"
6B2F: 19 51                            ; Bookkeeping B.5 09 "70-99 M. SCORE"
6B31: 19 5D                            ; Bookkeeping B.5 11 "150-199 M. SCORE"
6B33: 19 69                            ; Bookkeeping B.5 13 "OVER 300 MILLION"
6B35: 19 75                            ; Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"
6B37: 19 81                            ; Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"
6B39: 19 8D                            ; Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"
6B3B: 19 99                            ; Bookkeeping B.5 21 "GAME TIME 4-5 M."
6B3D: 19 A5                            ; Bookkeeping B.5 23 "GAME TIME 6-8 M."
6B3F: 19 B1                            ; Bookkeeping B.5 25 "GAME TIME 10-15 M"
6B41: 00 00                            ; <End of data>
                                       ;
---------------------------------------;------------------------------------------------------
                                       ;
                                       ; "Random" Seed Determinator 2
                                       ;------------------------------
                                       ; Table of Bookkeeping Storage Locations (L-8)
                                       ;
6B43: 18 EB                            ; Bookkeeping B.3 28 "1 PLAYER GAMES"
6B45: 18 F1                            ; Bookkeeping B.3 29 "2 PLAYER GAMES"
6B47: 18 F7                            ; Bookkeeping B.3 30 "3 PLAYER GAMES"
6B49: 18 FD                            ; Bookkeeping B.3 31 "4 PLAYER GAMES"
6B4B: 19 27                            ; Bookkeeping B.5 02 "2-4.9 M. SCORE"
6B4D: 19 33                            ; Bookkeeping B.5 04 "10-19 M. SCORE"
6B4F: 19 3F                            ; Bookkeeping B.5 06 "30-29 M. SCORE"
6B51: 19 4B                            ; Bookkeeping B.5 08 "50-69 M. SCORE"
6B53: 19 57                            ; Bookkeeping B.5 10 "100-149 M. SCORE"
6B55: 19 63                            ; Bookkeeping B.5 12 "200-299 M. SCORE"
6B57: 19 6F                            ; Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"
6B59: 19 7B                            ; Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"
6B5B: 19 87                            ; Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"
6B5D: 19 93                            ; Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"
6B5F: 19 9F                            ; Bookkeeping B.5 22 "GAME TIME 5-6 M."
6B61: 19 AB                            ; Bookkeeping B.5 24 "GAME TIME 8-10 M."
6B63: 19 B7                            ; Bookkeeping B.5 26 "GAME TIME > 15 M"
6B65: 00 00                            ; <End of data>
                                       ;
---------------------------------------;------------------------------------------------------
                                       ;
                                       ; "Random" Seed Determinator 3
                                       ;------------------------------
                                       ; Table of Bookkeeping Storage Locations (L-8)
```

```
                                    ;
6B67: 18 EB                         ; Bookkeeping B.3 28 "1 PLAYER GAMES"
6B69: 18 F1                         ; Bookkeeping B.3 29 "2 PLAYER GAMES"
6B6B: 18 F7                         ; Bookkeeping B.3 30 "3 PLAYER GAMES"
6B6D: 18 FD                         ; Bookkeeping B.3 31 "4 PLAYER GAMES"
6B6F: 19 21                         ; Bookkeeping B.5 01 "0-1.9 M. SCORE"
6B71: 19 27                         ; Bookkeeping B.5 02 "2-4.9 M. SCORE"
6B73: 19 39                         ; Bookkeeping B.5 05 "20-29 M. SCORE"
6B75: 19 3F                         ; Bookkeeping B.5 06 "30-29 M. SCORE"
6B77: 19 51                         ; Bookkeeping B.5 09 "70-99 M. SCORE"
6B79: 19 57                         ; Bookkeeping B.5 10 "100-149 M. SCORE"
6B7B: 19 69                         ; Bookkeeping B.5 13 "OVER 300 MILLION"
6B7D: 19 6F                         ; Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"
6B7F: 19 81                         ; Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"
6B81: 19 87                         ; Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"
6B83: 19 99                         ; Bookkeeping B.5 21 "GAME TIME 4-5 M."
6B85: 19 9F                         ; Bookkeeping B.5 22 "GAME TIME 5-6 M."
6B87: 19 B1                         ; Bookkeeping B.5 25 "GAME TIME 10-15 M"
6B89: 19 B7                         ; Bookkeeping B.5 26 "GAME TIME > 15 M"
6B8B: 00 00                         ; <End of data>
                                    ;
-------------------------------------;-------------------------------------------------------
                                    ;
                                    ; Get sum of bookkeeping values pointed to by X into A
                                    ;-----------------------------------------------------
6B8D: 10 AE 81    LDY   ,X++        ;-\ Y gets address of RAM of next bookkeeping value
6B90: 27 04       BEQ   $6B96       ; | If Y gets 0x0000 then reached end of the table, done
6B92: AB A4       ADDA  ,Y          ; | Increment A with value from Y
6B94: 20 F7       BRA   $6B8D       ;-/
                                    ;
6B96: 39          RTS               ;
                                    ;
-------------------------------------;-------------------------------------------------------
                                    ;
                                    ; Get "random" byte from ROM for given data table X
                                    ;-----------------------------------------------
6B97: 8D F4       BSR   $6B8D       ; Call function to get sum of bookkeeping values into A
6B99: 8E F9 00    LDX   #$F900      ; X gets addr in ROM from where arbitary byte is loaded
6B9C: A6 86       LDA   A,X         ; A now gets byte from ROM
6B9E: 39          RTS               ;
                                    ;
-------------------------------------;-------------------------------------------------------
                                    ;
                                    ; Get "random" byte for use in database awards
                                    ;---------------------------------------------
6B9F: 1F 12       TFR   X,Y         ; Save addr table start into Y
6BA1: 8D 16       BSR   $6BB9       ; Get into X address of per-player DB awards
6BA3: A6 84       LDA   ,X          ; A gets # of player's database awards, starting seed
6BA5: 1F 21       TFR   Y,X         ; Get addr table start back into X
6BA7: 8D EE       BSR   $6B97       ; Call function that gets "random" byte from ROM into A
6BA9: 39          RTS               ;
                                    ;
-------------------------------------;-------------------------------------------------------
                                    ;
                                    ; Get variable value 1..16 for database award
                                    ;---------------------------------------------
6BAA: 8E 6B 1D    LDX   #$6B1D      ; X gets addr of adjustment addrs, database award seed
6BAD: 8D F0       BSR   $6B9F       ; Call function to get random number into A
6BAF: 84 0F       ANDA  #$0F        ; Mask off the low 4 bits
6BB1: 4C          INCA              ; Make it 1..16
6BB2: 39          RTS               ;
                                    ;
```

```
----------------------------------------;-------------------------------------------------------
                                        ;
                                        ; For getting variable value 0..7 for database award
                                        ; The DB code will AND value with 0x07 to get 0..7
                                        ;-------------------------------------------------------
6BB3: 8E 6B 43    LDX   #$6B43          ; X gets addr of adjustment addrs, award position seed
6BB6: 8D E7       BSR   $6B9F           ; Call function to get random number into A
6BB8: 39          RTS                   ;
                                        ;
----------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Get into X the per-player DB award statistic address
                                        ;-------------------------------------------------------
6BB9: 8E 05 C5    LDX   #$05C5          ; Number of Database Awards
6BBC: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6BBF: 39          RTS                   ;
                                        ;
----------------------------------------;-------------------------------------------------------
                                        ;
6BC0: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
6BC3: 9A 01                             ; StandardAdjustment026, Tournament Play Adj=0x9A
6BC5: 24 07       BCC   $6BCE           ; C-clr, tournament mode is on, branch to L8.4 fixups
                                        ;
                                        ; c-set, here tournament mode is off, ordinary L-8 code
6BC7: F7 06 12    STB   $0612           ; Save winning # from B into $0612 for later access.
6BCA: BD A7 25    JSR   $A725           ; GetPseudoRandomNumberIntoA()
6BCD: 39          RTS                   ;
                                        ;
                                        ; Do L8.4 tournament mode database award
6BCE: 34 20       PSHS  Y               ; Preserve Y, caller needs A and B.  X is fine.
6BD0: 8D D8       BSR   $6BAA           ; Get random number 1..16 into A for database award
                                        ;
6BD2: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
6BD5: 83 00                             ; StandardAdjustment003, Max E.B. Count Adj=0x83
6BD7: 25 0C       BCS   $6BE5           ; C-set, extra ball is allowed, no fixup to DB award
                                        ;
                                        ; Extra-ball not allowed
6BD9: 81 02       CMPA  #$02            ; Is award "Extra Ball" ?
6BDB: 26 02       BNE   $6BDF           ;
6BDD: 86 0D       LDA   #$0D            ; Replace it with 0x0D, 13 "Lite Kickback"
                                        ;
6BDF: 81 0A       CMPA  #$0A            ; Is award "Lite Extra Ball" ?
6BE1: 26 02       BNE   $6BE5           ;
6BE3: 86 0F       LDA   #$0F            ; Replace it with 0x0F, 15, "Lite Hurry Up"
                                        ;
6BE5: 1F 89       TFR   A,B             ; Put award number into B a calling function expects
6BE7: F7 06 12    STB   $0612           ; Save winning # from B into $0612 for later access.
6BEA: 8D C7       BSR   $6BB3           ; Get random number 0..7 into A for award position
                                        ;
6BEC: 8D CB       BSR   $6BB9           ; Get into X address of per-player DB awards
6BEE: 6C 84       INC   ,X              ; Increment per-player DB awards
                                        ;
6BF0: 35 A0       PULS  Y,PC            ;
                                        ;
----------------------------------------;-------------------------------------------------------
```

Readers are encouraged to trace through the code starting at $6BC0,34 to see how the new L8.4 code will derive the same “random” database award for all players when “Tournament Play” is enabled.

Tournament Mode Enhancement: Multiball Lamps, L8.4

To further improve the L8.4 during Tournament Mode, the 5-bank lamps that are needed for starting Multiball can be made to be the same pattern for all players in a multi-player game. In doing so, this removes any actual or perceived unfair advantage one player might get over another when the lit targets for multiball are easier for one player while another player gets a more challenging set of lit targets.

One example of such unfair advantage is the case where some games have a more difficult to hit top-target (due to cannon switch adjustments or other mechanical reasons). Another example is the case where one player gets adjacent lit targets (which could both be hit with a single cannon shot) while another player gets a non-adjacent targets lit for their attempt to start multiball. With this L8.4 enhancement, when Tournament Mode is enabled, all players get the same opportunity whether it is an easier or more difficult set of lamps lit to start multiball.

Multiball Lamps Logic Enhancements for L8.4

The code that determines the 5-bank lamp patterns for starting multiball employs a mechanism to pick specific (non-random) targets when there are 1 or more of the 5-bank targets flagged as bad. The game flags a target as bad if it has not been hit in the past 60 balls. A bad target will cause credit dot and test report for the target.

When all 5 targets are good, then the logic will allow the game to pick a random pattern, one of 5 possible choices for each of the cases when 2, 3, or 4 targets are needed for starting multiball. The L8.4 update will remove this randomness when Tournament Mode is enabled so that all players get the same experience for 2, 3 and 4 targets remaining for starting multiball. For L8.4 the logic will pick a seemingly "random" choice of the 5 patterns, however all players in the multi-player game will get the same such "random" choice.

Multiball Lamps, Existing Code L-8

The existing L-8 logic is located in bank $31. Shown below is the full set of code that is used when the game selects the set of lamps for starting multiball. This begins at $68B6,31, ROM offset 0x468B6.

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; FiveBankTargetLampBitmapGet()
                                        ; Loads per-player $05FF with bitmap
                                        ;
68B6: 34 16       PSHS  X,B,A           ;
68B8: 8E 06 03    LDX   #$0603          ; #$0603 is Hunter ship hits remaining for multiball
68BB: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
68BE: 7E 68 C1    JMP   $68C1           ; <nop>
68C1: 6D 84       TST   ,X              ;
68C3: 10 26 01 51 LBNE  $6A18           ; If, somehow, zero hits remain, skip to the end
                                        ;
68C7: 34 10       PSHS  X               ; Save per-player hits-remaining addr on stack
                                        ;
68C9: 8E 5F 5B    LDX   #$5F5B          ; X=0x5F5B <-- addr in $3D the following fn will access
                                        ;
```

```
68CC: BD 88 F5    JSR   $88F5           ;
68CF: 5F 29 3D                         ; CheckBrokenSwitches()
                                       ; A gets bad switch count, C-clr if all switches bad
                                       ;
68D2: 25 08       BCS   $68DC          ; If C-set then not all switches are bad go to $68DC
                                       ;
                                       ;-------------------------------------------------------
                                       ; All 5-bank targets bad, per-player hits-remaining = 1
                                       ;-------------------------------------------------------
68D4: 86 01       LDA   #$01           ; A = 0x01
68D6: 35 10       PULS  X              ;
68D8: A7 84       STA   ,X             ; Force hits-remaining to value 1 on stack
68DA: 20 31       BRA   $690D          ;
                                       ;
                                       ;-------------------------------------------------------
                                       ; NOT all 5-bank targets bad, calculate hits-remaining
                                       ;-------------------------------------------------------
68DC: B7 06 07    STA   $0607          ; Save bad switch count into $0607
68DF: 1F 89       TFR   A,B            ; Save bad switch count into B
68E1: 86 05       LDA   #$05           ; A = 0x05
68E3: 34 04       PSHS  B              ; Save bad switch count onto stack
68E5: A0 E0       SUBA  ,S+            ; Subtract bad sw # from A, result is # of good sw in A
68E7: 1F 89       TFR   A,B            ; Save number of good switches into B
                                       ;
68E9: 8E 05 D5    LDX   #$05D5         ; X=0x05D5
68EC: BD FB 29    JSR   $FB29          ; IncrementXByPlayerIndexNumber()
68EF: 7E 68 F2    JMP   $68F2          ; <nop>
68F2: A6 84       LDA   ,X             ; A gets per-player value from $05D5
68F4: 4C          INCA                 ; Add 1 to per-player value from $05D5
68F5: 34 04       PSHS  B              ; Save updated per-player value into B
68F7: A1 E0       CMPA  ,S+            ; Compare per-player value w/# of good switches on stack
68F9: 23 02       BLS   $68FD          ;
68FB: 1F 98       TFR   B,A            ;
68FD: 35 10       PULS  X              ; Pull addr of hits-remaining off stack to update
68FF: 81 05       CMPA  #$05           ; A = 0x05
6901: 22 06       BHI   $6909          ;
6903: 4D          TSTA                 ;
6904: 26 05       BNE   $690B          ;
6906: 4C          INCA                 ;
6907: 20 02       BRA   $690B          ;
6909: 86 05       LDA   #$05           ;
                                       ;
690B: A7 84       STA   ,X             ; Store new hits-remaining # it per-player storage
                                       ;
690D: 81 01       CMPA  #$01           ;
                                       ;-------------------------------------------------------
                                       ; Only 1 target remaining for multiball
                                       ;-------------------------------------------------------
690F: 26 37       BNE   $6948          ;
6911: 86 04       LDA   #$04           ; 5-bank bitmap: --0--
6913: BD 83 39    JSR   $8339          ; BrokenSwitchCheckParameterByte() C-clr = switch broken
6916: 3B                               ; 0x3B = SwitchTableEntry3B, 73, Target 3
6917: 10 25 00 F2 LBCS  $6A0D          ;
                                       ;-------------------------------------------------------
                                       ; Only 1 target remaining for multiball, Target 3 is bad
                                       ;-------------------------------------------------------
691B: 86 08       LDA   #$08           ; 5-bank bitmap: -0---
691D: BD 83 39    JSR   $8339          ; BrokenSwitchCheckParameterByte() C-clr = switch broken
6920: 3C                               ; 0x3C = SwitchTableEntry3C, 74, Target 4
6921: 10 25 00 E8 LBCS  $6A0D          ;
                                       ;-------------------------------------------------------
                                       ; Only 1T remaining for multiball, T3 and T4 are bad
```

```
                                    ;-------------------------------------------------------
6925: 86 02       LDA   #$02        ; 5-bank bitmap: ---0-
6927: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
692A: 3A                            ; 0x3A = SwitchTableEntry3A, 72, Target 2
692B: 10 25 00 DE LBCS  $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; Only 1T remaining for multiball, T3 T4 T2 are bad
                                    ;-------------------------------------------------------
692F: 86 10       LDA   #$10        ; 5-bank bitmap: 0----
6931: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
6934: 3D                            ; 0x3D = SwitchTableEntry3D, 75, Target 5 Low
6935: 10 25 00 D4 LBCS  $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; Only 1T remaining for multiball, T3 T4 T2 T5 are bad
                                    ;-------------------------------------------------------
6939: 86 01       LDA   #$01        ; 5-bank bitmap: ----0
693B: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
693E: 39                            ; 0x39 = SwitchTableEntry39, 71, Target 1 High
693F: 10 25 00 CA LBCS  $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; Only 1T remaining for multiball, T3 T4 T2 T5 T1 bad
                                    ;-------------------------------------------------------
6943: 86 04       LDA   #$04        ; 5-bank bitmap: --0--  (use center T, all Ts bad)
6945: 7E 6A 0D    JMP   $6A0D       ;
                                    ;
                                    ;-------------------------------------------------------
                                    ; More than 1 target remaining for multiball
                                    ;-------------------------------------------------------
6948: 81 05       CMPA  #$05        ;
694A: 26 05       BNE   $6951       ;
                                    ;-------------------------------------------------------
                                    ; All 5 targets remaining for multiball
                                    ;-------------------------------------------------------
694C: 86 1F       LDA   #$1F        ; 5-bank bitmap: 00000
694E: 7E 6A 0D    JMP   $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; There are 2 or 3 or 4 targets remaining for multiball
                                    ;-------------------------------------------------------
6951: 81 04       CMPA  #$04        ;
6953: 26 10       BNE   $6965       ;
                                    ;-------------------------------------------------------
                                    ; There are 4 targets remaining for multiball
                                    ;-------------------------------------------------------
6955: 7D 06 07    TST   $0607       ; Check bad switch count into $0607
6958: 26 4A       BNE   $69A4       ;
                                    ;-------------------------------------------------------
                                    ; There are 4Ts remaining for multiball, no bad targets
                                    ;-------------------------------------------------------
695A: 86 05       LDA   #$05        ; A = 0x05
695C: 8E 6A 24    LDX   #$6A24      ; X = 0x6A24
695F: BD 6A 29    JSR   $6A29       ; GetBitmapIndexFromMappingX()
6962: 7E 6A 0D    JMP   $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; There are 2 or 3 targets remaining for multiball
                                    ;-------------------------------------------------------
6965: 81 03       CMPA  #$03        ;
6967: 26 40       BNE   $69A9       ;
                                    ;-------------------------------------------------------
                                    ; There are 3 targets remaining for multiball
                                    ;-------------------------------------------------------
6969: B6 06 07    LDA   $0607       ;
696C: 27 2C       BEQ   $699A       ;
```

```
                                    ;-------------------------------------------------------
                                    ; There are 3Ts remaining for MB, 1 or more bad targets
                                    ;-------------------------------------------------------
696E: 81 01       CMPA  #$01        ;
6970: 26 32       BNE   $69A4       ;
                                    ;-------------------------------------------------------
                                    ; There are 3Ts remaining for multiball, 1 bad target
                                    ;-------------------------------------------------------
6972: 86 1A       LDA   #$1A        ; 5-bank bitmap: 00-0-
6974: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
6977: 3B                            ; 0x3B = SwitchTableEntry3B, 73, Target 3
6978: 10 24 00 91 LBCC  $6A0D       ; The one bad T is not in the 00-0- bitmap, use it
                                    ; Else we need to come up with a different bitmap
                                    ;
697C: 86 07       LDA   #$07        ; 5-bank bitmap: --000
697E: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
6981: 3C                            ; 0x3C = SwitchTableEntry3C, 74, Target 4
6982: 10 24 00 87 LBCC  $6A0D       ; The one bad T is not one of the --000 bitmap, use it
                                    ; Else we need to come up with a different bitmap
                                    ;
6986: 86 1C       LDA   #$1C        ; 5-bank bitmap: 000--
6988: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
698B: 3A                            ; 0x3A = SwitchTableEntry3A, 72, Target 2
698C: 24 7F       BCC   $6A0D       ; The one bad T is not one of the 000-- bitmap, use it
                                    ; Else we need to come up with a different bitmap
                                    ;
698E: 86 0E       LDA   #$0E        ; 5-bank bitmap: -000-
6990: BD 83 39    JSR   $8339       ; BrokenSwitchCheckParameterByte() C-clr = switch broken
6993: 3D                            ; 0x3D = SwitchTableEntry3D, 75, Target 5 Low
6994: 24 77       BCC   $6A0D       ; The one bad T is not one of the -000- bitmap, use it
                                    ; Else the bad T must be T1 High, use bitmap to suit
                                    ;
6996: 86 1C       LDA   #$1C        ; 5-bank bitmap: 000--
6998: 20 73       BRA   $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; There are 3Ts remaining for MB and no bad targets
                                    ;-------------------------------------------------------
699A: 86 05       LDA   #$05        ; A = 0x05
699C: 8E 6A 1F    LDX   #$6A1F      ; X = 0x6A1F
699F: BD 6A 29    JSR   $6A29       ; GetBitmapIndexFromMappingX()
69A2: 20 69       BRA   $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; There are 4Ts remaining for MB & 1 or more bad targets
                                    ; There are 3Ts remaining for MB & 2 or more bad target
                                    ; There are 2Ts remaining for MB & 3 bad targets
                                    ;-------------------------------------------------------
69A4: BD 6A 35    JSR   $6A35       ; GetBitmapOf5BankWorkingSwitchesIntoA()
                                    ;  get bitmap of all working targets
69A7: 20 64       BRA   $6A0D       ;
                                    ;-------------------------------------------------------
                                    ; There are 2 targets remaining for multiball
                                    ;-------------------------------------------------------
69A9: B6 06 07    LDA   $0607       ;
69AC: 27 57       BEQ   $6A05       ;
                                    ;-------------------------------------------------------
                                    ; There are 2Ts remaining for MB & 1 or more bad targets
                                    ;-------------------------------------------------------
69AE: 81 03       CMPA  #$03        ;
69B0: 27 F2       BEQ   $69A4       ;
                                    ;-------------------------------------------------------
                                    ; There are 2Ts for MB, 1 or more bad, but not 3 bad Ts
                                    ;-------------------------------------------------------
```

```
69B2: 81 01       CMPA  #$01            ;
69B4: 27 2B       BEQ   $69E1           ;
                                        ;------------------------------------------------------
                                        ; There are 2Ts remaining for MB and 2, 4, or 5 bad Ts
                                        ;------------------------------------------------------
69B6: C6 1B       LDB   #$1B            ; 5-bank bitmap: 00-00
69B8: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69BB: 3B                                ; 0x3B = SwitchTableEntry3B, 73, Target 3
69BC: 25 1A       BCS   $69D8           ;
                                        ; Target 3 is bad
69BE: C6 17       LDB   #$17            ; 5-bank bitmap: 0-000
69C0: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69C3: 3C                                ; 0x3C = SwitchTableEntry3C, 74, Target 4
69C4: 25 12       BCS   $69D8           ;
                                        ; Target 4 is bad
69C6: C6 1D       LDB   #$1D            ; 5-bank bitmap: 000-0
69C8: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69CB: 3A                                ; 0x3A = SwitchTableEntry3A, 72, Target 2
69CC: 25 0A       BCS   $69D8           ;
                                        ; Target 2 is bad
69CE: C6 0F       LDB   #$0F            ; 5-bank bitmap: -0000
69D0: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69D3: 3D                                ; 0x3D = SwitchTableEntry3D, 75, Target 5 Low
69D4: 25 02       BCS   $69D8           ;
                                        ; Target 5 is bad
69D6: C6 1E       LDB   #$1E            ; 5-bank bitmap: 0000-
                                        ;
                                        ; We get here with a bitmap value in B that we can use
69D8: BD 6A 35    JSR   $6A35           ; GetBitmapOf5BankWorkingSwitchesIntoA()
69DB: 34 04       PSHS  B               ; Push bitmap onto stack
69DD: A4 E0       ANDA  ,S+             ; AND the retrieved bitmap with the 'good' bitmap
69DF: 20 2C       BRA   $6A0D           ;
                                        ;------------------------------------------------------
                                        ; There are 2Ts remaining for multiball and 1 bad target
                                        ;------------------------------------------------------
69E1: 86 0A       LDA   #$0A            ; 5-bank bitmap: -0-0-
69E3: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69E6: 3B                                ; 0x3B = SwitchTableEntry3B, 73, Target 3
69E7: 24 24       BCC   $6A0D           ; Bitmap is good even with Target 3 bad, so branch
                                        ;
69E9: 86 06       LDA   #$06            ; 5-bank bitmap: --00-
69EB: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69EE: 3C                                ; 0x3C = SwitchTableEntry3C, 74, Target 4
69EF: 24 1C       BCC   $6A0D           ; Bitmap is good even with Target 4 bad, so branch
                                        ;
69F1: 86 18       LDA   #$18            ; 5-bank bitmap: 00---
69F3: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
69F6: 3A                                ; 0x3A = SwitchTableEntry3A, 72, Target 2
69F7: 24 14       BCC   $6A0D           ; Bitmap is good even with Target 2 bad, so branch
                                        ;
69F9: 86 03       LDA   #$03            ; 5-bank bitmap: ---00
69FB: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-set = switch broken
69FE: 3D                                ; 0x3D = SwitchTableEntry3D, 75, Target 5 Low
69FF: 24 0C       BCC   $6A0D           ; Bitmap is good even with Target 5 Low bad, so branch
                                        ;
                                        ; Assume T1 High is the bad T, use appropriate bitmap
6A01: 86 06       LDA   #$06            ; 5-bank bitmap: --00-
6A03: 20 08       BRA   $6A0D           ;
                                        ;------------------------------------------------------
                                        ; There are 2Ts remaining for MB and no bad targets
                                        ;------------------------------------------------------
6A05: 86 05       LDA   #$05            ; A = 0x05
```

```
6A07: 8E 6A 1A    LDX   #$6A1A          ; X = 0x6A1A
6A0A: BD 6A 29    JSR   $6A29           ;
                                        ;
                                        ; All paths get here with the desired 5-bank bitmap in A
6A0D: 8E 05 FF    LDX   #$05FF          ;
6A10: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6A13: 7E 6A 16    JMP   $6A16           ; <nop>
6A16: A7 84       STA   ,X              ; Store the bitmap into per-player memory at $05FF
6A18: 35 96       PULS  A,B,X,PC        ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; Addresses to these bytes are loaded into X prior
                                        ; to calling $6A29 in the code above
                                        ;
                                        ; $6A1A There are 2 targets for MB and no bad targets
                                        ;-------------------------------------------------------
6A1A: 0A                                ; 5-bank bitmap: -0-0-
6A1B: 18                                ; 5-bank bitmap: 00---
6A1C: 0C                                ; 5-bank bitmap: -00--
6A1D: 06                                ; 5-bank bitmap: --00-
6A1E: 03                                ; 5-bank bitmap: ---00
                                        ;
                                        ; $6A1F There are 3 targets for MB and no bad targets
                                        ;-------------------------------------------------------
6A1F: 1C                                ; 5-bank bitmap: 000--
6A20: 0E                                ; 5-bank bitmap: -000-
6A21: 07                                ; 5-bank bitmap: --000
6A22: 0B                                ; 5-bank bitmap: -0-00
6A23: 1A                                ; 5-bank bitmap: 00-0-
                                        ;
                                        ; $6A24 There are 4 targets for MB and no bad targets
                                        ;-------------------------------------------------------
6A24: 1B                                ; 5-bank bitmap: 00-00
6A25: 1E                                ; 5-bank bitmap: 0000-
6A26: 0F                                ; 5-bank bitmap: -0000
6A27: 1D                                ; 5-bank bitmap: 000-0
6A28: 17                                ; 5-bank bitmap: 0-000
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; Called from above with X pointing to one of the above
                                        ; 5-byte tables and A containing 0x05.  This is used to
                                        ; come up with 5-bank bitmap to use when no bad targets
                                        ; have been flagged.
                                        ;
6A29: 34 14       PSHS  X,B             ;
6A2B: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA() A gets random 0..4
6A2E: 1F 89       TFR   A,B             ; Put random 0..4 value into B
6A30: 3A          ABX                   ; Advance X by random value
6A31: A6 84       LDA   ,X              ; Get the random bimap into A
6A33: 35 94       PULS  B,X,PC          ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; GetBitmapOf5BankWorkingSwitchesIntoA()
                                        ;
6A35: 34 04       PSHS  B               ;
6A37: C6 1F       LDB   #$1F            ; 5-bank bitmap: 00000
6A39: 34 04       PSHS  B               ; Push bitmap onto stack
6A3B: C6 05       LDB   #$05            ; B = 0x05
6A3D: 34 04       PSHS  B               ; Push 0x05 onto stack
```

```
6A3F: 86 FE       LDA   #$FE            ; A = 0xFE, clear bit is low bit
6A41: C6 39       LDB   #$39            ; 0x39 = SwitchTableEntry39, 71, Target 1 High
                                        ;
6A43: BD 96 C3    JSR   $96C3           ;--\ BrokenSwitchCheckB() // C-clr if switch is bad
6A46: 25 08       BCS   $6A50           ;  |
                                        ;  | Current Target B is bad
                                        ;  |-------------------------
6A48: 34 02       PSHS  A               ;  | Save mask onto stack
6A4A: A4 62       ANDA  $0002,S         ;  | Clear bit in the 0x1F mask in the stack
6A4C: A7 62       STA   $0002,S         ;  | Update the 0x1F mask in the stack with cleared bit
6A4E: 35 02       PULS  A               ;  | Pull mask off stack
                                        ;  |
6A50: 1A 01       ORCC  #$0001          ;  | Set C-bit
6A52: 49          ROLA                  ;  | Rotate left the clear bit mask
6A53: 5C          INCB                  ;  | Increment the to next 5-bank target
6A54: 6A E4       DEC   ,S              ;  | Decrement the 5-switch switch counter on the stack
6A56: 26 EB       BNE   $6A43           ;--/ Keep going until all 5 switches have been checked
                                        ;
6A58: 35 04       PULS  B               ; Pull the 5-sw count from stack into B (should be 0x00)
6A5A: 35 02       PULS  A               ; Pull the updated 0x1F mask off stack into A.
                                        ;  A has mask of all valid/working switches.
6A5C: 35 84       PULS  B,PC            ;
                                        ;
----------------------------------------;-------------------------------------------------
```

Readers are encouraged to go through the code above for better understanding of how the game works especially when faced with bad 5-bank targets as compared to when all 5-bank targets are good.

Multiball Lamps, New Logic for L8.4

The same "random" value determinator previously described for the database award will be utilized for deriving a seemingly random value for multiball lamp patterns. The goal is to come up with a seemingly "random" value for where the game currently picks a random value to pick from one of 5 possible patterns for 2, 3, or 4 lamps to light. This new logic will only occur when the game adjustments are found to have the "Tournament Mode" enabled.

The table below shows the game bookkeeping statistics that are used for deriving the random number for lit lamps for multiball:

Bookkeeping Entry	Addr	Bookkeeping Statistic Participation
		Lamp Pattern for 2, 3, 4 hits remaining for MB start
Bookkeeping B.3 08 "MATCH AWARDS"	$18A3	✓
Bookkeeping B.3 28 "1 PLAYER GAMES"	$18EB	✓
Bookkeeping B.3 29 "2 PLAYER GAMES"	$18F1	✓
Bookkeeping B.3 30 "3 PLAYER GAMES"	$18F7	✓
Bookkeeping B.3 31 "4 PLAYER GAMES"	$18FD	✓
Bookkeeping B.5 01 "0-1.9 M. SCORE"	$1921	✓
Bookkeeping B.5 02 "2-4.9 M. SCORE"	$1927	
Bookkeeping B.5 03 "3-9.9 M. SCORE"	$192D	✓
Bookkeeping B.5 04 "10-19 M. SCORE"	$1933	

Bookkeeping B.5 05 "20-29 M. SCORE"	$1939	✓
Bookkeeping B.5 06 "30-29 M. SCORE"	$193F	
Bookkeeping B.5 07 "40-49 M. SCORE"	$1945	✓
Bookkeeping B.5 08 "50-69 M. SCORE"	$194B	
Bookkeeping B.5 09 "70-99 M. SCORE"	$1951	✓
Bookkeeping B.5 10 "100-149 M. SCORE"	$1957	
Bookkeeping B.5 11 "150-199 M. SCORE"	$195D	✓
Bookkeeping B.5 12 "200-299 M. SCORE"	$1963	
Bookkeeping B.5 13 "OVER 300 MILLION"	$1969	✓
Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"	$196F	
Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"	$1975	✓
Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"	$197B	
Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"	$1981	✓
Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"	$1987	
Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"	$198D	✓
Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"	$1993	
Bookkeeping B.5 21 "GAME TIME 4-5 M."	$1999	✓
Bookkeeping B.5 22 "GAME TIME 5-6 M."	$199F	
Bookkeeping B.5 23 "GAME TIME 6-8 M."	$19A5	✓
Bookkeeping B.5 24 "GAME TIME 8-10 M."	$19AB	
Bookkeeping B.5 25 "GAME TIME 10-15 M"	$19B1	✓
Bookkeeping B.5 26 "GAME TIME > 15 M"	$19B7	

The 8-bit sum of each of these statistics along with the current player's number of jackpots achieved are used to then lookup a byte of ROM from starting point $F900 (ROM offset 0x7F900) in a similar way as what is done for the L8.4 Tournament Mode Database award logic.

The resulting byte from ROM is then applied to a formula to come up with the random number 0..4, as indicated in the table below.

Variable Behavior	**Needed Values**		**Formula to derive value X from byte B (C-like syntax)**
	Range	# of values	
5-Bank 0..4 Selector	0..4	5	X = ((B & 0x0F) % 5)

The formula is to take the ROM byte, low nibble modulo 5 which results in a value 0..4. A survey of all possible values results in the following results:

Variable Behavior		**Formula Results**
5-Bank 0..4 Selector		1 3 0 4 1 3 2 0 0 1 2 0 2 0 1 0 0 2 2 4 2 0 2 4 1 3 4 4 1 2 1 2 3 3 4 4 1 2 4 4 0 4 3 0 0 1 4 3 2 4 3 0 3 1 1 1 3 3 4 1 4 1 3 2 1 0 1 2 4 0 4 1 3 2 1 0 4 0 1 1 2 2 0 3 4 4 2 2 1 0 1 4 1 3 4 4
0..4	**Total Occurrances**	

0 1 2 3 4	53 60 46 42 55	1 2 1 2 3 3 4 4 0 1 2 1 3 0 4 0 4 1 1 2 4 4 4 3 2 0 0 1 4 3 0 3 2 3 0 4 0 1 3 0 1 4 4 4 2 3 1 2 1 2 0 1 4 1 1 1 2 1 3 0 4 0 1 3 3 3 0 4 4 1 4 2 3 1 2 0 4 0 1 0 2 0 4 0 1 0 0 2 0 3 3 0 3 4 4 2 2 1 0 1 4 1 3 4 4 1 2 1 2 3 3 4 4 3 0 3 1 4 1 3 0 2 0 4 1 2 1 0 2 2 2 2 3 4 4 2 2 1 0 1 4 0 3 0 3 1 3 0 0 4 1 1 2 0 1 0 4 2 0 1

As shown, there is a reasonable amount of seemingly random values 0..4 that can occur using this formula.

Multiball Lamps, Updated Code for L8.4

The portion of lamp code applicable to the L8.4 fixup is shown below with modified code for L8.4 highlighted. This is the logic used to pick a random lamp pattern from a group of 5 possible patterns for the cases of 2, 3 or 4 lamps to light for multiball. This code starts at $6A1A,31, ROM offset 0x46A1A.

```
----------------------------------------;------------------------------------------------
                                        ;
                                        ; Addresses to these bytes are loaded into X prior
                                        ; to calling $6A29 in the code above.  This is only
                                        ; used when no bad targets are currently flagged.
                                        ;
                                        ; $6A1A There are 2 targets remaining for multiball
                                        ;------------------------------------------------
6A1A: 0A                                ; 5-bank bitmap: -0-0-
6A1B: 18                                ; 5-bank bitmap: 00---
6A1C: 0C                                ; 5-bank bitmap: -00--
6A1D: 06                                ; 5-bank bitmap: --00-
6A1E: 03                                ; 5-bank bitmap: ---00
                                        ;
                                        ; $6A1F There are 3 targets remaining for multiball
                                        ;------------------------------------------------
6A1F: 1C                                ; 5-bank bitmap: 000--
6A20: 0E                                ; 5-bank bitmap: -000-
6A21: 07                                ; 5-bank bitmap: --000
6A22: 0B                                ; 5-bank bitmap: -0-00
6A23: 1A                                ; 5-bank bitmap: 00-0-
                                        ;
                                        ; $6A24 There are 4 targets remaining for multiball
                                        ;------------------------------------------------
6A24: 1B                                ; 5-bank bitmap: 00-00
6A25: 1E                                ; 5-bank bitmap: 0000-
6A26: 0F                                ; 5-bank bitmap: -0000
6A27: 1D                                ; 5-bank bitmap: 000-0
6A28: 17                                ; 5-bank bitmap: 0-000
                                        ;
----------------------------------------;------------------------------------------------
                                        ;
                                        ;
                                        ;
                                        ; Called from above with X pointing to one of the above
                                        ; 5-byte tables and A containing 0x05.  This is used to
                                        ; come up with 5-bank bitmap to use when no bad targets
```

```
                                        ; have been flagged.
                                        ;
6A29: 34 14       PSHS  X,B             ;
                                        ;
6A2B: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA() 0..4 into A
6A2E: 1F 89       TFR   A,B             ; Put random 0..4 value into B
6A30: 3A          ABX                   ; Advance X by random value
                                        ;
6A2B: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
6A2E: 6C 0C 34                          ; L84MultiballLampsTournamentModeEnhancement()
                                        ;
6A31: A6 84       LDA   ,X              ; Get the random bimap into A
6A33: 35 94       PULS  B,X,PC          ;
                                        ;
----------------------------------------;-------------------------------------------------
```

The old L-8 code is replaced with a jump to the new L8.4 code located in bank $34. The new L8.4 code checks for Tournament Mode and performs either the original L-8 logic or performs new fixup logic as described.

The new code entry point is at $6C0C,34. The full set of new code added to bank $34 is as follows.

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; Get variable value 0..4 for multiball lamp pattern
                                        ;-------------------------------------------------
6BF2: 34 30       PSHS  Y,X             ; Preserve Y and X
6BF4: 8E 05 9D    LDX   #$059D          ; Number of Jackpots
6BF7: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6BFA: A6 84       LDA   ,X              ; A gets # of player's jackpots, starting seed value
6BFC: AE E4       LDX   ,S              ; Get X original value from the stack
6BFE: 8D 97       BSR   $6B97           ; Call function that gets "random" byte from ROM into A
                                        ;
6C00: 84 0F       ANDA  #$0F            ; Just mask off the low 4 bits
6C02: 81 04       CMPA  #$04            ;
6C04: 23 04       BLS   $6C0A           ; If A is less then or equal to 4 then we are done
6C06: 80 05       SUBA  #$05            ;
6C08: 20 F8       BRA   $6C02           ; Keep looping until A is 0..4
6C0A: 35 B0       PULS  X,Y,PC          ;
                                        ;
----------------------------------------;-------------------------------------------------
                                        ;
6C0C: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
6C0F: 9A 01                             ; StandardAdjustment026, Tournament Play Adj=0x9A
6C11: 24 05       BCC   $6C18           ; C-clr, tournament mode is on, branch to L8.4 fixups
                                        ;
                                        ; c-set, here tournament mode is off, do L-8 code
6C13: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA() // gets random 0..4
6C16: 20 09       BRA   $6C21           ; Go put 0..4 into B and advance X by random value
                                        ;
                                        ; Do L8.4 tournament mode multiball lamps
6C18: 34 10       PSHS  X               ; Save x as we need to return updated X value to caller
6C1A: 8E 6B 1D    LDX   #$6B1D          ; X gets addr of adjustment addresses for MB award seed
6C1D: 8D D3       BSR   $6BF2           ; Get random number 0..4 into A
6C1F: 35 10       PULS  X               ;
                                        ;
6C21: 1F 89       TFR   A,B             ; Put random 0..4 value into B
6C23: 3A          ABX                   ; Advance X by random value
```

```
6C24: 39          RTS                 ;
                                      ;
---------------------------------------;-------------------------------------------------------
```

The code for multiball tournament mode chooses a “random” value 0..4 based on the game bookkeeping 8-bit sum with lookup into the ROM at $F900, then proceeds with the algorithm previously described. The result is a value 0..4 that the calling code can use to pick one of 5 lamp patterns when there are 2, 3, or 4 hits remaining for starting multiball. The same 8-bit sum is calculated for all players in the same tournament mode game which results in the same 2-lamp, 3-lamp and 4-lamp patterns for all players in the tournament mode game.

Tournament Mode Enhancement: Jackpot Lamp, L8.4

To further improve the L8.4 fixups for Tournament Mode, the jackpot lamp selector is updated in L8.4 so that all players are given the same “random” lamp for each successive jackpot. This refers to the single 5-bank lamp the game will illuminate when the player is going for 1X, 2X or 3X jackpot during multiball. Normally each player gets a randomly lit target, however for L8.4 when Tournament Mode is enabled, the game will ensure all players have the same lamp for their first jackpot. For the second jackpot a different “random” lamp may be chosen while all players get the same such “random” lamp for their second jackpot, and so on.

The per-player number of jackpots statistic is used in the determination of the “random” number in order to get the expected behavior for L8.4.

Jackpot Lamp, Existing Code L-8

The game uses a set of logic to check for failed 5-bank target switches in order to pick a jackpot target lamp that is not associated with a failed target switch. When all 5 targets are good, then the game picks a random value 0..4 to pick one of the 5 targets that will be used for the jackpot attempt.

```
---------------------------------------;-------------------------------------------------
                                      ;
5876: BD 86 90    JSR   $8690         ; SearchLinkedListForId()  // c-clear means ID is found
5879: 00 86                           ; ID 0086 is Multiball running
587B: 24 34       BCC   $58B1         ; MB is running go to $58B1 for jackpot attempt
                                      ;
587D: BD 84 AD    JSR   $84AD         ; GetMemoryFlag() // C-bit clear when flag set
5880: 48                              ;
5881: 24 2E       BCC   $58B1         ;
5883: BD 87 BE    JSR   $87BE         ; ExtinguishLampGroupParamBytes()
5886: 18 10                           ;
5888: 0F D3       CLR   $D3           ; Clears the super-jackpot flag $D3
588A: 8E 05 FF    LDX   #$05FF        ; Per-player 5-bank lamps bitmap for MB, 01 = top lamp
588D: BD FB 29    JSR   $FB29         ; IncrementXByPlayerIndexNumber()
5890: 7E 58 93    JMP   $5893         ; <nop>
5893: E6 84       LDB   ,X            ;
5895: 34 04       PSHS  B             ;
5897: 86 11       LDA   #$11          ;
5899: 34 02       PSHS  A             ; --\
589B: A6 61       LDA   $0001,S       ;   |
589D: 85 01       BITA  #$01          ;   |
589F: 35 02       PULS  A             ;   |
```

```
58A1: 27 05       BEQ   $58A8           ;   |
58A3: C6 40       LDB   #$40            ;   |
58A5: BD 9E 7F    JSR   $9E7F           ;   | ValidateThenSingleLampSetIndexAPlaneB()
58A8: 4C          INCA                  ;   |
58A9: 64 E4       LSR   ,S              ;   |
58AB: 26 EC       BNE   $5899           ; --/
                                        ;
58AD: 35 04       PULS  B               ;
58AF: 20 5C       BRA   $590D           ;
                                        ;
58B1: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
58B4: 42                                ; Flag 0x42 is RAM $0328 bit 0x02
58B5: 25 08       BCS   $58BF           ; C-set then not in super jackpot, skip to $58BF
58B7: BD 89 48    JSR   $8948           ; Schedules function that sets super-jackpot mode and
58BA: 59 26 34                          ;  cycles the arrow
58BD: 20 4E       BRA   $590D           ;
                                        ;
58BF: BD 87 BE    JSR   $87BE           ; ExtinguishLampGroupParamBytes()
58C2: 18 10                             ; 5-bank target lamps
                                        ;
58C4: 7D 06 07    TST   $0607           ; Check bad 5-bank switch count, set at $68DC,31
58C7: 27 32       BEQ   $58FB           ; If no bad switches jump over the bad sw checks
                                        ;
58C9: 86 02       LDA   #$02            ; A=0x02 for lamp 3
58CB: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
58CE: 3B                                ; 0x3B = SwitchTableEntry3B, 73, Target 3
58CF: 10 25 00 2D LBCS  $5900           ; Target 3 is active, use target 3
                                        ;
58D3: 86 03       LDA   #$03            ; A=0x03 for lamp 4
58D5: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
58D8: 3C                                ; 0x3C = SwitchTableEntry3C, 74, Target 4
58D9: 10 25 00 23 LBCS  $5900           ; Target 4 is active, use target 4
                                        ;
58DD: 86 01       LDA   #$01            ; A=0x01 for lamp 2
58DF: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
58E2: 3A                                ; 0x3A = SwitchTableEntry3A, 72, Target 2
58E3: 10 25 00 19 LBCS  $5900           ; Target 2 is active, use target 2
                                        ;
58E7: 86 04       LDA   #$04            ; A=0x04 for lamp 5
58E9: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
58EC: 3D                                ; 0x3D = SwitchTableEntry3D, 75, Target 5 Low
58ED: 10 25 00 0F LBCS  $5900           ; Target 5 is active, use target 5
                                        ;
58F1: 86 00       LDA   #$00            ; A=0x00 for lamp 1
58F3: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte() C-clr = switch broken
58F6: 39                                ; 0x39 = SwitchTableEntry39, 71, Target 1 High
58F7: 10 25 00 05 LBCS  $5900           ; Target 1 is active, use target 1
                                        ;
58FB: 86 05       LDA   #$05            ; A=0x05
58FD: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA() // gets random 0..4
                                        ;
5900: C6 3D       LDB   #$3D            ; Starting lamp Target 1 High
5902: 34 04       PSHS  B               ;
5904: AB E0       ADDA  ,S+             ; A gets set to $3D..$41 to correspond to value 0..4
5906: 97 D3       STA   $D3             ;
5908: C6 40       LDB   #$40            ; Lamp plane 0x40 (blinky lamps)
590A: BD AD F9    JSR   $ADF9           ; Light the single 5-bank lamp for jackpot
                                        ;
590D: BD 83 92    JSR   $8392           ; -\
5910: 06                                ;  |
5911: BD 83 46    JSR   $8346           ;  | Sleep()
5914: 04                                ;  |
```

```
5915: BD 86 90    JSR   $8690           ;  | SearchLinkedListForId()  // c-clr means ID is found
5918: 00 AB                             ;  |
591A: 24 F1       BCC   $590D           ;  |
591C: BD 86 90    JSR   $8690           ;  | SearchLinkedListForId()  // c-clr means ID is found
591F: 00 84                             ;  |
5921: 24 EA       BCC   $590D           ; -/
                                        ;
5923: 7E C6 5F    JMP   $C65F           ;
                                        ;
----------------------------------------;-------------------------------------------------------
```

The function, above contains checks for whether multiball is running and, if so, whether to go for Super Jackpot or for ordinary jackpot. When going for ordinary jackpot the game then picks a random value 0..4 to use as the lit lamp, converting it to lamp index $3D..$41.

Multiball Lamps, New Logic for L8.4

For L8.4 the logic for choosing random value 0..4 is same as that used for the Multiball hits lamp patterns except a different set of bookkeeping statistics are used to create the seed value used to look up a “random” byte from ROM.

The table below shows game bookkeeping statistics that are used for deriving the random number for jackpot lamp. This is same set of statistics used for the hits-remaining random number.

Bookkeeping Entry	Addr	Bookkeeping Statistic Participation
		Jackpot Lamp
Bookkeeping B.3 08 "MATCH AWARDS"	$18A3	
Bookkeeping B.3 28 "1 PLAYER GAMES"	$18EB	✓
Bookkeeping B.3 29 "2 PLAYER GAMES"	$18F1	✓
Bookkeeping B.3 30 "3 PLAYER GAMES"	$18F7	✓
Bookkeeping B.3 31 "4 PLAYER GAMES"	$18FD	✓
Bookkeeping B.5 01 "0-1.9 M. SCORE"	$1921	✓
Bookkeeping B.5 02 "2-4.9 M. SCORE"	$1927	✓
Bookkeeping B.5 03 "3-9.9 M. SCORE"	$192D	
Bookkeeping B.5 04 "10-19 M. SCORE"	$1933	
Bookkeeping B.5 05 "20-29 M. SCORE"	$1939	✓
Bookkeeping B.5 06 "30-29 M. SCORE"	$193F	✓
Bookkeeping B.5 07 "40-49 M. SCORE"	$1945	
Bookkeeping B.5 08 "50-69 M. SCORE"	$194B	
Bookkeeping B.5 09 "70-99 M. SCORE"	$1951	✓
Bookkeeping B.5 10 "100-149 M. SCORE"	$1957	✓
Bookkeeping B.5 11 "150-199 M. SCORE"	$195D	
Bookkeeping B.5 12 "200-299 M. SCORE"	$1963	
Bookkeeping B.5 13 "OVER 300 MILLION"	$1969	✓
Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"	$196F	✓
Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"	$1975	
Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"	$197B	
Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"	$1981	✓

Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"	$1987	✓
Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"	$198D	
Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"	$1993	
Bookkeeping B.5 21 "GAME TIME 4-5 M."	$1999	✓
Bookkeeping B.5 22 "GAME TIME 5-6 M."	$199F	✓
Bookkeeping B.5 23 "GAME TIME 6-8 M."	$19A5	
Bookkeeping B.5 24 "GAME TIME 8-10 M."	$19AB	
Bookkeeping B.5 25 "GAME TIME 10-15 M"	$19B1	✓
Bookkeeping B.5 26 "GAME TIME > 15 M"	$19B7	✓

The 8-bit sum of each of these statistics are then added to the per-player jackpots counter and the resulting sum is used to lookup a byte of ROM from starting point $F900 (ROM offset 0x7F900).

The resulting byte from ROM is then applied to a formula to come up with the random number 0..4, as indicated in the table below.

Variable Behavior	**Needed Values**		**Formula to derive value X from byte B (C-like syntax)**
	Range	# of values	
Jackpot 0..4 Selector	0..4	5	X = ((B & 0x0F) % 5)

The resulting value is simply the low nibble modulo 5 which results in a value of 0..5 for any value. This formula results in a relatively even number of values 0..4 from ROM $F900 that can be obtained as shown in the table of possible values below.

Variable Behavior	**Formula Results**
Jackpot 0..4 Selector **0 = Target 1 High** **1 = Target 2** **2 = Target 3** **3 = Target 4** **4 = Target 5 Low**	1 3 0 4 1 3 2 0 0 1 2 0 2 0 1 0 0 2 2 4 2 0 2 4 1 3 4 4 1 2 1 2 3 3 4 4 1 2 4 4 0 4 3 0 0 1 4 3 2 4 3 0 3 1 1 1 3 3 4 1 4 1 3 2 1 0 1 2 4 0 4 1 3 2 1 0 4 0 1 1 2 2 0 3 4 4 2 2 1 0 1 4 1 3 4 4 1 2 1 2 3 3 4 4 0 1 2 1 3 0 4 0 4 1 1 2 4 4 4 3 2 0 0 1 4 3 0 3 2 3 0 4 0 1 3 0 1 4 4 4 2 3 1 2 1 2 0 1 4 1 1 1 2 1 3 0 4 0 1 3 3 3 0 4 4 1 4 2 3 1 2 0 4 0 1 0 2 0 4 0 1 0 0 2 0 3 3 0 3 4 4 2 2 1 0 1 4 1 3 4 4 1 2 1 2 3 3 4 4 3 0 3 1 4 1 3 0 2 0 4 1 2 1 0 2 2 2 2 3 4 4 2 2 1 0 1 4 0 3 0 3 1 3 0 0 4 1 1 2 0 1 0 4 2 0 1

0..4	**Total Occurrances**
0	53
1	60
2	46
3	42
4	55

As shown, a mix of possible values are possible with a mix of different values with a mix of repeating and mismatched values. This will give the players in Tournament Mode a fair mix of "random" behavior while giving all players in the same tournament mode game the same experience.

Jackpot Lamp, Updated Code for L8.4

The applicable portion of the jackpot lamp code is shown below with modification for L8.4. This depicted code starts at $58F1,34, ROM offset 0x518F1.

```
58F1: 86 00       LDA   #$00            ; A=0x00 for lamp 1
58F3: BD 83 39    JSR   $8339           ; BrokenSwitchCheckParameterByte()  C-set if switch
                                        ; active.  C-clr is switch broken.
58F6: 39                                ; 0x39 = SwitchTableEntry39, 71, Target 1 High
58F7: 10 25 00 05 LBCS  $5900           ; Target 1 is active, use target 1
                                        ;
58FB: 86 05       LDA   #$05            ; A=0x05
58FD: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA() // gets 0..4 in A
58FD: BD 6C 25    JSR   $6C25           ; L84JackpotLampJumpPoint()
                                        ;
5900: C6 3D       LDB   #$3D            ; Starting lamp Target 1 High
5902: 34 04       PSHS  B               ;
5904: AB E0       ADDA  ,S+             ; A gets set to $3D..$40 to correspond to 0..4
5906: 97 D3       STA   $D3             ;
5908: C6 40       LDB   #$40            ;
590A: BD AD F9    JSR   $ADF9           ; Light the single 5-bank lamp for jackpot
```

The L8.4 code jumps to new code in the same bank $34 which performs original L-8 code if Tournament Mode is not set. If Tournament Mode is set, then a "random" value 0..4 is chosen using the described formula.

The new function at $6C25,34, ROM offset 0x52C25, is as follows:

```
---------------------------------------;-------------------------------------------------------
                                       ;
6C25: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
6C28: 9A 01                             ; StandardAdjustment026, Tournament Play Adj=0x9A
6C2A: 24 05       BCC   $6C31           ; C-clr, tournament mode is on, branch to L8.4 fixups
                                        ;
                                        ; c-set, here tournament mode is off, do L-8 code
6C2C: BD A7 5B    JSR   $A75B           ; Get16BitPseudoRandomValueintoA() // gets random 0..4
6C2F: 20 09       BRA   $6C3A           ; done
                                        ;
                                        ; Do L8.4 tournament mode jackpot lamp
6C31: 34 10       PSHS  X               ; Save x in case caller needs it unmodified
6C33: 8E 6B 67    LDX   #$6B67          ; X gets addr of adjustment addresses for jackpot seed
6C36: 8D BA       BSR   $6BF2           ; Get random number 0..4 into A
6C38: 35 10       PULS  X               ;
                                        ;
6C3A: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
```

The jackpot lamp function for L8.4 checks for tournament mode. If tournament mode is not set, then the original L-8 code proceeds where a genuine random value is obtained. If tournament mode is set then a "random" value is chosen based on the logic previously described.

Tournament Mode Enhancement: Video Mode, L8.4

The L8.4, out of completeness and fairness to players in a tournament, enhances the Video Mode when Tournament Mode is enabled so that all players in a multiplayer game get the same video mode experience. The L-8 video mode utilizes 4 different random numbers to determine various characteristics of the video mode experience:

- Maximum number of terminator kills before video mode ends
- Number of terminator kills before hunter ship appears
- Number of terminator kills before head popup appears
- Whether or not the head popup will occur
- Number of terminator kills before EB award appears
- Direction of next terminator appearing from left or right
- Timing of next terminator how it steps before firing

For L8.4 the Tournament Mode will use a similar mechanism described previously to ensure that all players in a tournament mode game will have identical video mode opportunities thereby removing the actual or perceived unfairness that one player might otherwise have over another player when their video mode is easier to acquire more points.

Video Mode, Existing Code L-8

Below is video mode code from bank $2B. This code starts at $42E8,2B, ROM offset 0x2C2E8. The code is partially annotated and serves as a starting point for readers interested in how the T2 Video Mode works.

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; VideoMode()
                                        ;
42E8: BD FB AE    JSR   $FBAE           ; ClearDisplayMemory()
42EB: 7E 42 EE    JMP   $42EE           ; <nop>
                                        ;
42EE: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
42F1: AB                                ; 0xAB = "Video mode activated"
                                        ;
42F2: BD 88 D5    JSR   $88D5           ; IncreaseBookkeepingCounterAddrXBy1ParamBytes()
42F5: 00 22                             ; Video Mode Started
                                        ;
42F7: BD D7 99    JSR   $D799           ; Print string on DMD
42FA: 00 C2                             ; 0xC2 "VIDEO MODE"
42FC: 09                                ; font
42FD: 40 0A                             ; coordinates
42FF: BD D3 B7    JSR   $D3B7           ;
4302: BD D7 99    JSR   $D799           ; Print string on DMD
4305: 00 C3                             ; 0xC3 "USE FLIPPER BUTTONS"
4307: 01                                ; font
4308: 40 13                             ; coordiates
430A: BD D7 99    JSR   $D799           ; Print string on DMD
430D: 00 C4                             ; 0xC4 "TO MOVE SIGHT"
430F: 01                                ; font
4310: 40 1D                             ; coordinates
```

```
4312: C6 14       LDB   #$14            ;
                                        ;
4314: BD E2 74    JSR   $E274           ;--\ dmd update related
                                        ;  |
4317: BD 83 46    JSR   $8346           ;  |
431A: 06                                ;  | 0.09375 seconds
421B: 5A          DECB                  ;  |
431C: 26 F6       BNE   $4314           ;--/
                                        ;
431E: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
4321: B9                                ; 0xB9 = "Destroy everything"
                                        ;
4322: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
4325: 49 58 2A                          ;
                                        ;
4328: 7F 06 44    CLR   $0644           ;
432B: 7F 06 43    CLR   $0643           ;
432E: 7F 06 45    CLR   $0645           ;
4331: 7F 06 48    CLR   $0648           ;
4334: 8E 06 4E    LDX   #$064E          ;
4337: CC 00 30    LDD   #$0030          ;
433A: A7 84       STA   ,X              ;
433C: ED 01       STD   $0001,X         ;
433E: A7 03       STA   $0003,X         ;
4340: A7 04       STA   $0004,X         ;
                                        ;
4342: BD 89 48    JSR   $8948           ;
4345: 44 AE 2B                          ;
                                        ;
4348: 8E 00 31    LDX   #$0031          ;
434B: 10 8E 00 1C LDY   #$001C          ;
434F: C6 11       LDB   #$11            ;
                                        ;
4351: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
4354: 49 6C 2A                          ;
                                        ;
4357: 86 33       LDA   #$33            ;
4359: A7 C8 15    STA   $15,U           ;
435C: 6F 4C       CLR   $000C,U         ;
435E: FF 06 38    STU   $0638           ;
                                        ;
4361: BD 89 48    JSR   $8948           ;
4364: 42 5B 2B                          ;
                                        ;
4367: BD 89 48    JSR   $8948           ;
436A: 42 BF 2B                          ;
                                        ;
436D: BD A7 25    JSR   $A725           ; GetPseudoRandomNumberIntoA()
4370: 84 07       ANDA  #$07            ; Make it 0..7 random number basis for:
                                        ;    max number of terminator kills (0..7 plus 0x14)
                                        ;    number of terminator kills before ship appears
4372: 34 02       PSHS  A               ;
                                        ;
4374: 8B 14       ADDA  #$14            ;
4376: B7 06 46    STA   $0646           ; $0646 gets max num of kills 14 + 0..7 random number
                                        ;
4379: 48          ASLA                  ;
437A: B7 06 4B    STA   $064B           ; $064B gets 2x max number of terminator kills allowed
                                        ;
437D: A6 E0       LDA   ,S+             ;
437F: 8B 05       ADDA  #$05            ;
4381: B7 06 47    STA   $0647           ; $0647 gets num of kills before flying ship appears
```

```
                                        ;
4384: 34 02       PSHS  A               ;
4386: BD A7 25    JSR   $A725           ; GetPseudoRandomNumberIntoA()
4389: 84 03       ANDA  #$03            ; Make it 0..3 random number basis for:
                                        ;    number of terminator kills before head popup
                                        ;    whether or not head-popup will appear
                                        ;    number of terminator kills before EB appears
438B: 34 02       PSHS  A               ;
438D: 8B 02       ADDA  #$02            ;
438F: AB 61       ADDA  $0001,S         ;
4391: B7 06 49    STA   $0649           ; $0649 gets number of kills before head popup
                                        ;
4394: 84 07       ANDA  #$07            ;
4396: B7 06 4A    STA   $064A           ; Random number 0..7, when 0 no head popup will happen
                                        ;
4399: A6 E1       LDA   ,S++            ;
439B: 8B 02       ADDA  #$02            ;
439D: B7 06 4D    STA   $064D           ; $064D gets number of kills before EB shown
                                        ;
43A0: 4F          CLRA                  ;
43A1: BD FA 1E    JSR   $FA1E           ; Checking if Video Mode should include Extra Ball
43A4: 7E 43 A7    JMP   $43A7           ; <nop>
43A7: 25 01       BCS   $43AA           ;
43A9: 4C          INCA                  ;
43AA: B7 06 4C    STA   $064C           ; $064C non-zero if extra ball should be included
                                        ;
43AD: BD FB AE    JSR   $FBAE           ; ClearDisplayMemory()
43B0: 7E 43 B3    JMP   $43B3           ; <nop>
                                        ;
43B3: B6 06 44    LDA   $0644           ;--\ $0644 = kills,  $0646 max kills
43B6: B1 06 46    CMPA  $0646           ;  | If $0644 is <= $0646 video mode keeps running
43B9: 25 05       BCS   $43C0           ;  |
43BB: BD 9E 0A    JSR   $9E0A           ;  | CancelSelf()
43BE: 20 54       BRA   $4414           ;  | Brach to End-of-Video-mode, with award
                                        ;  |
43C0: 86 03       LDA   #$03            ;  |
43C2: B1 06 43    CMPA  $0643           ;  | if ($0643 == 0x03) True = cracked glass
43C5: 26 1B       BNE   $43E2           ;  | {
43C7: BD 9E 0A    JSR   $9E0A           ;  |    CancelSelf()
43CA: 7F 06 43    CLR   $0643           ;  |    Reset $0643 to 0x03
43CD: BE 06 38    LDX   $0638           ;  |
43D0: 86 12       LDA   #$12            ;  |
43D2: A7 01       STA   $0001,X         ;  |
43D4: 86 01       LDA   #$01            ;  |
43D6: A7 84       STA   ,X              ;  |
43D8: A7 0C       STA   $000C,X         ;  |
43DA: 6F 02       CLR   $0002,X         ;  |
43DC: 6F 03       CLR   $0003,X         ;  |
43DE: BD 85 46    JSR   $8546           ;  |    DoSoundTableParameterByte()
43E1: B6                                ;  |    0xB6=Smashey
                                        ;  | }
                                        ;  |
43E2: BE 06 38    LDX   $0638           ;  | if (player got shot, ending video mode)
43E5: A6 01       LDA   $0001,X         ;  | {
43E7: 81 12       CMPA  #$12            ;  |
43E9: 26 06       BNE   $43F1           ;  |
43EB: A6 84       LDA   ,X              ;  |
43ED: 81 08       CMPA  #$08            ;  |
43EF: 27 7A       BEQ   $446B           ;  |    Goto End-of-Video mode, no award
                                        ;  | }
                                        ;  |
43F1: BD D3 60    JSR   $D360           ;  | Clear512BytesFrom1799Pointer()
```

```
43F4: BD 88 F5    JSR   $88F5           ;  | CallBankedFunction_Param_WPCAddr()
43F7: 49 B2 2A                         ;  |
                                       ;  |
43FA: B6 06 43    LDA   $0643          ;  |
43FD: B1 06 45    CMPA  $0645          ;  | Checking if current crosshairs is on a target?
4400: 27 09       BEQ   $440B          ;  | {
4402: BD FB CB    JSR   $FBCB          ;  |    getting here flashes display and kills robot
4405: 7E 44 08    JMP   $4408          ;  |    <nop>
4408: B7 06 45    STA   $0645          ;  |
                                       ;  | }
440B: BD E2 74    JSR   $E274          ;  | display related
440E: BD 83 46    JSR   $8346          ;  | Sleep()
4411: 04                               ;  | 0.0625 seconds
4412: 20 9F       BRA   $43B3          ;--/
                                       ;
4414: BD FB AE    JSR   $FBAE          ; ClearDisplayMemory()
4417: 7E 44 1A    JMP   $441A          ; <nop>
441A: BD 85 46    JSR   $8546          ; DoSoundTableParameterByte()
441D: BA                               ; 0xBA = “Well done”
441E: BD 88 D5    JSR   $88D5          ; IncreaseBookkeepingCounterAddrXBy1ParamBytes()
4421: 00 23                            ; Video Mode End
4423: BD D7 99    JSR   $D799          ; Print string on DMD
4426: 00 C1                            ; 0xC1 "YOU WIN"
4428: 10                               ;
4429: 40 0E                            ;
442B: BD D7 99    JSR   $D799          ; Print string on DMD
442E: 00 C6                            ; 0xC6 "5\,000\,000"
4430: 10                               ;
4431: 40 1F                            ;
4433: BD 85 46    JMP   $8546          ; DoSoundTableParameterByte()
4436: B3                               ; 0xB3 = Dat dat dat do woo
4437: 8E 06 4E    LDX   #$064E         ;
443A: BD F7 82    JSR   $F782          ; Copy BCD score bytes from X to $0577
443D: 7E 44 40    JMP   $4440          ; <nop>
4440: 8E 05 7C    LDX   #$057C         ;
4443: CC 00 05    LDD   #$0005         ;
4446: ED 84       STD   ,X             ;
4448: A7 02       STA   $0002,X        ;
444A: A7 03       STA   $0003,X        ;
444C: A7 04       STA   $0004,X        ;
444E: BD F7 A7    JSR   $F7A7          ;
4451: 7E 44 54    JMP   $4454          ; <nop>
4454: 8E 06 4E    LDX   #$064E         ;
4457: BD F7 CD    JSR   $F7CD          ; <Puts skill shot score in X> during ss call to this fn
445A: 7E 44 5D    JMP   $445D          ; <nop>
445D: C6 0B       LDB   #$0B           ;
                                       ;
445F: BD E2 74    JSR   $E274          ;--\ display related
4462: BD 83 46    JSR   $8346          ;  | Sleep()
4465: 08                               ;  | 0.125 seconds
4466: 5A          DECB                 ;  |
4467: 26 F6       BNE   $445F          ;--/
                                       ;
4469: 20 04       BRA   $446F          ;
446B: BD 83 46    JSR   $8346          ; Sleep()
446E: 20                               ; 1/2 second
                                       ;
446F: BD FB AE    JSR   $FBAE          ; ClearDisplayMemory()
4472: 7E 44 75    JMP   $4475          ; <nop>
4475: F6 06 44    LDB   $0644          ; Get number of kills into B
4478: BD D7 99    JSR   $D799          ; Print string on DMD
447B: 00 C5                            ; 0xC5 "%b TERMINATORS"
```

```
447D: 09                                ; font
447E: 40 0C                             ; coordinates
4480: 10 8E 06 4E LDY   #$064E          ;
                                        ;
4484: BD D7 99    JSR   $D799           ; Print string on DMD
4487: 00 26                             ; 0x26 "%ixy"
4489: 10                                ; font
448A: 40 1D                             ; coordinates
                                        ;
448C: BD D3 B7    JSR   $D3B7           ; display related
448F: BD FB CB    JSR   $FBCB           ;
4492: 7E 44 95    JMP   $4495           ; <nop>
4495: C6 3C       LDB   #$3C            ;
                                        ;
4497: BD E2 74    JSR   $E274           ;--\ display text related
449A: BD 83 46    JSR   $8346           ;  | Sleep()
449D: 03                                ;  | 0.046875 seconds
449E: 5A          DECB                  ;  |
449F: 26 F6       BNE   $4497           ;--/
                                        ;
44A1: 86 05       LDA   #$05            ;
44A3: C6 01       LDB   #$01            ;
44A5: 8E 06 4E    LDX   #$064E          ;
44A8: BD BA CD    JSR   $BACD           ;
44AB: 7E C9 52    JMP   $C952           ; <nop>
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Loop that periodically puts things on the display
                                        ; during video mode
                                        ;
44AE: 7F 06 3A    CLR   $063A           ;
                                        ;
44B1: B6 06 44    LDA   $0644           ;--\
44B4: BB 06 3A    ADDA  $063A           ;  |
44B7: B1 06 46    CMPA  $0646           ;  | $0646 is max number of terminator kills
44BA: 25 03       BCS   $44BF           ;  |
44BC: 7E 99 A2    JMP   $99A2           ;  |
                                        ;  |
44BF: B6 06 3A    LDA   $063A           ;  |
44C2: 81 06       CMPA  #$06            ;  |
44C4: 10 24 01 42 LBCC  $460A           ;  | C-clear? Done with this pass
44C8: B6 06 44    LDA   $0644           ;  | A gets $0644, number of terminators hit
44CB: B1 06 47    CMPA  $0647           ;  | Compare kills w/$0647 (hits until flying ship)
44CE: 25 32       BCS   $4502           ;  | If kills not at $0647 limit, skip over this section
                                        ;  |
                                        ;  | Kills reached $0647 limit, now show flying ship
44D0: 8E 00 80    LDX   #$0080          ;  |
44D3: 10 8E 00 1F LDY   #$001F          ;  |
44D7: C6 13       LDB   #$13            ;  |
44D9: BD 88 F5    JSR   $88F5           ;  |
44DC: 49          ROLA                  ;  |
44DD: 6C 2A       INC   $000A,Y         ;  |
44DF: 10 25 01 27 LBCS  $460A           ;  | C-set? Done with this pass
44E3: 86 FF       LDA   #$FF            ;  |
44E5: B7 06 47    STA   $0647           ;  | Load 0xFF into $0647 so no more ships will appear
44E8: BD 85 46    JSR   $8546           ;  |
44EB: B1 86 17    CMPA  $8617           ;  |
44EE: A7 4C       STA   $000C,U         ;  |
44F0: 86 FC       LDA   #$FC            ;  |
44F2: A7 48       STA   $0008,U         ;  |
44F4: 8E 47 03    LDX   #$4703          ;  |
```

```
44F7: C6 2B       LDB   #$2B            ;  |
44F9: AF C8 11    STX   $11,U           ;  |
44FC: E7 C8 13    STB   $13,U           ;  |
44FF: 16 00 FB    LBRA  $45FD           ;  |
                                        ;  |
4502: B1 06 49    CMPA  $0649           ;  | Compare kills w/$0649 limit (hits until head popup)
4505: 25 37       BCS   $453E           ;  |
                                        ;  |
                                        ;  | Kills reached $0649 limit, now show head popup
4507: 7D 06 4A    TST   $064A           ;  | Check random number 0..7 in $064A
450A: 27 32       BEQ   $453E           ;  | If random number is 00 then NO head this time
450C: 8E 00 60    LDX   #$0060          ;  |
450F: 10 8E 00 3F LDY   #$003F          ;  |
4513: C6 15       LDB   #$15            ;  |
4515: BD 88 F5    JSR   $88F5           ;  |
4518: 49 6C 2A                          ;  |
451B: 10 25 00 EB LBCS  $460A           ;  | C-set? Done with this pass
451F: 86 FF       LDA   #$FF            ;  |
4521: B7 06 49    STA   $0649           ;  | Load 0xFF into $0649 so no more head will appear
4524: BD 85 46    JSR   $8546           ;  | DoSoundTableParameterByte()
4527: B4                                ;  | 0xB4 = Elevating wribble
4528: 86 02       LDA   #$02            ;  |
452A: A7 4C       STA   $000C,U         ;  |
452C: 86 F8       LDA   #$F8            ;  |
452E: A7 4A       STA   $000A,U         ;  | U[A] re dir and pos on the DMD of the animation
4530: 8E 46 91    LDX   #$4691          ;  | $4691,2B handles head popup animation
4533: C6 2B       LDB   #$2B            ;  |
4535: AF C8 11    STX   $11,U           ;  |
4538: E7 C8 13    STB   $13,U           ;  |
453B: 16 00 BF    LBRA  $45FD           ;  |
                                        ;  |
453E: B1 06 4D    CMPA  $064D           ;  | Compare kills w/$064D limit (hits until EB)
4541: 25 46       BCS   $4589           ;  |
                                        ;  |
4543: 7D 06 4C    TST   $064C           ;  | Test $064C to see if EB should be in video mode
4546: 27 41       BEQ   $4589           ;  |
4548: 10 BE 06 38 LDY   $0638           ;  | Adding "EB" to video mode
454C: 6D 22       TST   $0002,Y         ;  |
454E: 26 0B       BNE   $455B           ;  |
4550: A6 23       LDA   $0003,Y         ;  |
4552: 81 40       CMPA  #$40            ;  |
4554: 25 05       BCS   $455B           ;  |
4556: 8E 00 10    LDX   #$0010          ;  |
4559: 20 03       BRA   $455E           ;  |
455B: 8E 00 60    LDX   #$0060          ;  |
455E: 10 8E 00 00 LDY   #$0000          ;  |
4562: C6 14       LDB   #$14            ;  |
4564: BD 88 F5    JSR   $88F5           ;  |
4567: 49 6C 2A                          ;  |
456A: 10 25 00 9C LBCS  $460A           ;  | C-set? Done with this pass
456E: 86 FF       LDA   #$FF            ;  |
4570: B7 06 4D    STA   $064D           ;  | Load 0xFF into $064D so no more EB will appear
4573: 86 04       LDA   #$04            ;  |
4575: A7 4C       STA   $000C,U         ;  |
4577: 86 04       LDA   #$04            ;  |
4579: A7 4A       STA   $000A,U         ;  |
457B: 8E 47 03    LDX   #$4703          ;  |
457E: C6 2B       LDB   #$2B            ;  |
4580: AF C8 11    STX   $11,U           ;  |
4583: E7 C8 13    STB   $13,U           ;  |
4586: 16 00 74    LBRA  $45FD           ;  |
                                        ;  |
```

```
4589: 8E 46 11    LDX   #$4611          ;  |
458C: 10 8E FF E6 LDY   #$FFE6          ;  |
4590: BD A7 25    JSR   $A725           ;  | GetPseudoRandomNumberIntoA()
4593: 84 01       ANDA  #$01            ;  | Make it 0..1,  0=Robot enters from left.  1=right
4595: F6 06 49    LDB   $0649           ;  |
4598: C1 F0       CMPB  #$F0            ;  |
459A: 23 01       BLS   $459D           ;  |
459C: 4F          CLRA                  ;  |
459D: B7 06 3B    STA   $063B           ;  |
45A0: 27 07       BEQ   $45A9           ;  |
45A2: 8E 46 51    LDX   #$4651          ;  |
45A5: 10 8E 00 80 LDY   #$0080          ;  |
45A9: BD A7 25    JSR   $A725           ;  | GetPseudoRandomNumberIntoA()
45AC: 34 02       PSHS  A               ;  |
45AE: B6 06 44    LDA   $0644           ;  |
45B1: B1 06 4B    CMPA  $064B           ;  | $064B is 2x max number of terminator kills allowed
45B4: 23 08       BLS   $45BE           ;  |
45B6: 35 04       PULS  B               ;  |
45B8: C4 07       ANDB  #$07            ;  | Make it 0..7, robot distance, 0=shorter. 7=farther
45BA: CB 08       ADDB  #$08            ;  |
45BC: 20 04       BRA   $45C2           ;  |
45BE: 35 04       PULS  B               ;  |
45C0: C4 0F       ANDB  #$0F            ;  |
45C2: 3A          ABX                   ;  |
45C3: 3A          ABX                   ;  |
45C4: 3A          ABX                   ;  |
45C5: 3A          ABX                   ;  |
45C6: EC 02       LDD   $0002,X         ;  |
45C8: B7 06 3C    STA   $063C           ;  |
45CB: F7 06 3D    STB   $063D           ;  |
45CE: AE 84       LDX   ,X              ;  |
45D0: 1E 12       EXG   X,Y             ;  |
45D2: 7D 06 3B    TST   $063B           ;  |
45D5: 27 04       BEQ   $45DB           ;  |
45D7: C6 0D       LDB   #$0D            ;  |
45D9: 20 02       BRA   $45DD           ;  |
45DB: C6 0E       LDB   #$0E            ;  |
45DD: BD 88 F5    JSR   $88F5           ;  |
45E0: 49 6C 2A                          ;  |
45E3: 25 25       BCS   $460A           ;  | C-set? Done with this pass
45E5: B6 06 3C    LDA   $063C           ;  |
45E8: A7 4C       STA   $000C,U         ;  |
45EA: B6 06 3D    LDA   $063D           ;  |
45ED: A7 48       STA   $0008,U         ;  |
45EF: E7 C8 14    STB   $14,U           ;  |
45F2: 8E 46 C5    LDX   #$46C5          ;  |
45F5: C6 2B       LDB   #$2B            ;  |
45F7: AF C8 11    STX   $11,U           ;  |
45FA: E7 C8 13    STB   $13,U           ;  |
                                        ;  |
45FD: 8E 47 07    LDX   #$4707          ;  |
4600: C6 2B       LDB   #$2B            ;  |
4602: AF 4E       STX   $000E,U         ;  |
4604: E7 C8 10    STB   $10,U           ;  |
4607: 7C 06 3A    INC   $063A           ;  | Increment the loop counter
460A: BD 83 46    JSR   $8346           ;  | Sleep()
460D: 1D                                ;  | 0.453125 seconds
460E: 16 FE A0    LBRA  $44B1           ;--/
                                        ;
----------------------------------------;--------------------------------------------------
```

The video mode code, above, has a fair amount of code not yet annotated. Readers are encouraged to follow through the code and with a pinball emulator to discover more about how the code works.

Highlighted in green is where the L-8 code retrieves 4 random values while performing the video mode. These highlighted instructions are subject of the L8.4 fix and will be described in detail below.

Video Mode, New Logic for L8.4

For L8.4 the 4 random number retrievals that L-8 code performs will be replaced with calls to new L8.4 function that gets either:

- The normal L-8 random number if Tournament Mode is not enabled, or
- A "random" number that is same for all players when Tournament Mode is enabled.

In a manner similar to the previously described L8.4 enhancements, game bookkeeping statistics are used to derive a "seed' value which is then added to the per-player statistic tracking the number of video modes played. This "seed" value is then used to lookup a byte in ROM which is then applied to a formula in order to derive a seemingly "random" number in the same range as the original L-8 code expects.

The table below shows which of the various game bookkeeping values are used for each of the 4 random numbers needed during video mode. Note the random number for robot timing is masked with 0x07 or 0x0F depending on game conditions so both possibilities are shown.

Bookkeeping Entry	Addr	Bookkeeping Statistic Participation			
		Mode Init 0..7	Mode Init 0..3	Robot Left/Right 0..1	Robot Timing 0..7/0..15
Bookkeeping B.3 08 "MATCH AWARDS"	$18A3	✓		✓	
Bookkeeping B.3 28 "1 PLAYER GAMES"	$18EB	✓	✓	✓	✓
Bookkeeping B.3 29 "2 PLAYER GAMES"	$18F1	✓	✓	✓	✓
Bookkeeping B.3 30 "3 PLAYER GAMES"	$18F7	✓	✓	✓	✓
Bookkeeping B.3 31 "4 PLAYER GAMES"	$18FD	✓	✓	✓	✓
Bookkeeping B.5 01 "0-1.9 M. SCORE"	$1921	✓		✓	
Bookkeeping B.5 02 "2-4.9 M. SCORE"	$1927		✓	✓	
Bookkeeping B.5 03 "3-9.9 M. SCORE"	$192D	✓			✓
Bookkeeping B.5 04 "10-19 M. SCORE"	$1933		✓		✓
Bookkeeping B.5 05 "20-29 M. SCORE"	$1939	✓		✓	
Bookkeeping B.5 06 "30-29 M. SCORE"	$193F		✓	✓	
Bookkeeping B.5 07 "40-49 M. SCORE"	$1945	✓			✓
Bookkeeping B.5 08 "50-69 M. SCORE"	$194B		✓		✓
Bookkeeping B.5 09 "70-99 M. SCORE"	$1951	✓		✓	
Bookkeeping B.5 10 "100-149 M. SCORE"	$1957		✓	✓	
Bookkeeping B.5 11 "150-199 M. SCORE"	$195D	✓			✓
Bookkeeping B.5 12 "200-299 M. SCORE"	$1963		✓		✓
Bookkeeping B.5 13 "OVER 300 MILLION"	$1969	✓		✓	
Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"	$196F		✓	✓	

Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"	$1975	✓			✓
Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"	$197B		✓		✓
Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"	$1981	✓		✓	
Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"	$1987		✓	✓	
Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"	$198D	✓			✓
Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"	$1993		✓		✓
Bookkeeping B.5 21 "GAME TIME 4-5 M."	$1999	✓		✓	
Bookkeeping B.5 22 "GAME TIME 5-6 M."	$199F		✓	✓	
Bookkeeping B.5 23 "GAME TIME 6-8 M."	$19A5	✓			✓
Bookkeeping B.5 24 "GAME TIME 8-10 M."	$19AB		✓		✓
Bookkeeping B.5 25 "GAME TIME 10-15 M"	$19B1	✓		✓	
Bookkeeping B.5 26 "GAME TIME > 15 M"	$19B7		✓	✓	

Different sets of bookkeeping values are used for each “random” number seed to add to the uncertainty of each of the four random numbers. As shown in the L-8 code, above, the L-8 game code uses bitwise AND to reduce the resulting random number to 0..7, 0..3, 0..1 or 0..15 range.

For each of these 4 “random” numbers, the “seed” value is used to lookup a byte from ROM relative to the arbitrarily chosen address $F900 (ROM offset 0x7F900). This is same region of ROM used by other L8.4 “random” number logic described previously. This region of ROM appears to have a mix of byte values with non-repeating bytes, as shown below:

Offset:	Bytes:	ANSI Text:
0007F880	C1 F8 2F 09 C1 08 2C 0A 1A 01 4C 35 92 1A 01 4F	Á ø / Á ◘ , → ┌ L 5 ′ → ┌ O
0007F890	35 92 1C FE 4C 35 92 34 16 BD F9 09 B1 17 A6 27	5 ′ þ L 5 ′ 4 ┬ ½ ù ± ┤ ¦ '
0007F8A0	03 BD F9 24 C6 02 34 04 5A 8E F8 EA 3A A6 84 BD	└ ½ ù $ Æ ┐ 4 ┘ Z Ž ø ê : ¦ „ ½
0007F8B0	92 DE BD FA 13 AB 01 81 63 23 0E A6 84 81 08 2C	′ Þ ½ ú ‼ « ┌ ▯ c # ♫ ¦ „ ▯ ◘ ,
0007F8C0	06 6F 01 6C 84 20 04 86 63 A7 01 35 04 5A 26 D6	− o ┌ l „ ┘ † c § ┌ 5 ┘ Z & Ö
0007F8D0	7C 17 A5 BD F9 09 B7 17 A6 35 96 34 16 BD F9 09	| ┤ ¥ ½ ù · ┤ ¦ 5 − 4 ┬ ½ ù
0007F8E0	B1 17 A6 27 03 BD F9 24 35 96 02 01 03 00 04 00	± ┤ ¦ ' └ ½ ù $ 5 − ┐ ┌ └ ┘
0007F8F0	34 16 C6 02 34 04 8E F8 EC 5A 3A A6 84 BD FA 13	4 ┬ Æ ┐ 4 ┘ Ž ø ì Z : ¦ „ ½ ú ‼
0007F900	A7 03 35 04 5A 26 ED 35 96 34 14 8E 17 9D C6 02	§ └ 5 ┘ Z & í 5 − 4 ¶ Ž ┤ ▯ Æ ┐
0007F910	B6 17 A5 AB 84 AB 01 AB 02 AB 03 30 04 5A 26 F3	¶ ┤ ¥ « „ « ┌ « ┐ « └ O ┘ Z & ó
0007F920	88 33 35 94 34 16 8E 17 9D C6 02 6F 84 6F 01 6F	ˆ 3 5 ″ 4 ┬ Ž ┤ ▯ Æ ┐ o „ o ┌ o
0007F930	02 30 04 5A 26 F5 7F 17 A5 BD F8 F0 BD F9 09 B7	┐ O ┘ Z & õ ⌂ ┤ ¥ ½ ø ð ½ ù ·
0007F940	17 A6 35 96 34 16 BD F9 09 B1 17 A6 27 03 BD F9	┤ ¦ 5 − 4 ┬ ½ ù ± ┤ ¦ ' └ ½ ù
0007F950	24 BD FA 13 A6 84 81 F8 2F 02 6A 84 A6 02 81 FF	$ ½ ú ‼ ¦ „ ▯ ø / ┐ j „ ¦ ┐ ▯ ÿ
0007F960	27 02 6C 02 BD F9 09 B7 17 A6 35 96 34 10 BD FA	' ┐ l ┐ ½ ù · ┤ ¦ 5 − 4 ╪ ½ ú
0007F970	13 E6 03 35 90 34 16 B6 17 A5 81 F0 24 02 35 96	‼ æ └ 5 ▯ 4 ┬ ¶ ┤ ¥ ▯ ð $ ┐ 5 −

Given this set of 256 bytes and using simple lookup of each byte allowing the existing L-8 video mode code to simply mask off the low bits of the “random” byte, the table below reports the possible values that can be derived and the number of each possible “random” value that will be used.

The chosen byte from ROM is passed through an algorithm specific to the value being obtained.

Variable Behavior	Needed Values		Formula to derive value X from byte B (C-like syntax)
	Range	# of values	
Mode Init 0..7	0..7	8	X = (B & 0x07)
Mode Init 0..3	0..3	4	X = (B & 0x03)
Robot Left/Right 0..1	0..1	2	X = (B – (++Increment) & 0x01)
Robot Timing 0..7	0..7	8	X = (B + (++Increment) & 0x07)
Robot Timing 0..15	0..15	16	X = (B + (++Increment) & 0x0F)

The final L8.4 code will NOT perform the masking of the resulting value with 0x07, 0x03, 0x01 or 0x0F since the original L-8 code takes the "random" number and performs such masking itself. The masking is depicted here for completeness as it is conceptually taking place as part of the L8.4 logic but is being deferred to the original L-8 code.

The first two "Mode Init" random values are only derived a single time at start of video mode. The random numbers related to robot movement are retrieved for each robot appearance. Each robot appearance is associated with a random number retrieval for left/right and for timing. In order to have variance between each robot movement, the use of "Increment" item is used to cause each "random" byte lookup into the 256-byte ROM region select a different byte. The "Increment" is a byte in RAM that increments each time it is read. This resulting "Increment" byte is then *subtracted* from the sum of bookkeeping values for robot left/right. The resulting "Increment" byte is *added* to the sum of bookkeeping values for robot timing. This method results in unpredictable and variable robot movements while also giving each player in a multi-player tournament game the same video mode experience since the "Increment" value is set to 0 at the start of each video mode.

In order to safely use a byte in RAM for purposes of this "Increment" counter, the existing video mode code was surveyed and it was discovered that the current video mode utilizes a single 8-bit RAM byte to track whether or not the video mode will present the "EB" award. This byte gets set at video mode start based on how many extra balls the player currently has and game adjustments defining whether the player can be awarded a new extra ball. For L8.4, this byte in RAM is being repurposed so that only its high bit 0x80 is used to flag whether or not the player can be presented with "EB" during the video mode while the lower 7-bits will track a 7-bit sum that represents the "Increment" value for robot movements. The code changes, depicted below, will show how the game code is updated for this.

An analysis of all possible values, using the previously depicted formulas, was done with results shown below.

Variable Behavior	Formula Results
Mode Init 0..7	1 0 7 1 1 0 4 2 2 1 4 5 2 2 1 7 5 2 4 6 4 5 2 4 6 5 1 1 1 7 6 7 3 5 1 4 6 2 4 4 2 6 0 2 2 6 4 5

0..7	Total Occurrances
0	17
1	34
2	33
3	21
4	41
5	36
6	44
7	30

2 6 5 2 3 3 1 1 3 3 6 6 4 1 0 4
6 7 1 4 4 0 4 6 3 7 1 5 4 2 6 6
4 7 5 5 1 1 7 7 6 5 6 4 6 5 1 1
1 7 6 7 3 5 1 4 5 6 2 1 3 0 4 0
4 6 6 2 4 4 6 0 4 2 2 6 4 5 2 3
7 3 5 4 2 6 5 5 6 4 4 6 7 5 6 2
6 7 5 3 4 3 1 3 2 3 3 0 4 2 6 3
0 3 5 4 4 6 6 7 5 6 2 7 4 7 1 7
2 0 4 2 6 5 7 7 5 5 0 0 5 1 1 7
7 6 5 6 4 6 5 1 1 1 7 6 7 3 5 1
4 5 2 3 6 4 1 0 7 2 2 4 6 2 1 7
7 2 4 2 5 1 1 7 7 6 5 6 4 0 5 2
3 6 3 5 0 4 6 6 7 5 1 0 4 2 5 6

Mode Init 0..3

0..3	Total Occurrances
0	58
1	70
2	77
3	51

1 0 3 1 1 0 0 2 2 1 0 1 2 2 1 3
1 2 0 2 0 1 2 0 2 1 1 1 1 3 2 3
3 1 1 0 2 2 0 0 2 2 0 2 2 2 0 1
2 2 1 2 3 3 1 1 3 3 2 2 0 1 0 0
2 3 1 0 0 0 0 2 3 3 1 1 0 2 2 2
0 3 1 1 1 1 3 3 2 1 2 0 2 1 1 1
1 3 2 3 3 1 1 0 1 2 2 1 3 0 0 0
0 2 2 2 0 0 2 0 0 2 2 2 0 1 2 3
3 3 1 0 2 2 1 1 2 0 0 2 3 1 2 2
2 3 1 3 0 3 1 3 2 3 3 0 0 2 2 3
0 3 1 0 0 2 2 3 1 2 2 3 0 3 1 3
2 0 0 2 2 1 3 3 1 1 0 0 1 1 1 3
3 2 1 2 0 2 1 1 1 1 3 2 3 3 1 1
0 1 2 3 2 0 1 0 3 2 2 0 2 2 1 3
3 2 0 2 1 1 1 3 3 2 1 2 0 0 1 2
3 2 3 1 0 0 2 2 3 1 1 0 0 2 1 2

Robot Left/Right 0..1

0 = Robot enters from Left
1 = Robot enters from Right

0..1	Total Occurrances
0	135
1	121

1 0 1 1 1 0 0 0 0 1 0 1 0 0 1 1
1 0 0 0 0 1 0 0 0 1 1 1 1 1 0 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 0
0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0
0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1
1 1 0 1 1 1 1 0 1 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
1 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0
0 1 1 1 0 1 1 1 0 1 1 0 0 0 0 1
0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 1
0 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1
1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1
0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1
1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 0
1 0 1 1 0 0 0 0 1 1 1 0 0 0 1 0

Robot Timing 0..7

0 = Slowest
7 = Fastest

0..7	Total Occurrances
0	30
1	17
2	34
3	33
4	21
5	41
6	36
7	44

```
2 1 0 2 2 1 5 3 3 2 5 6 3 3 2 0
6 3 5 7 5 6 3 5 7 6 2 2 2 0 7 0
4 6 2 5 7 3 5 5 3 7 1 3 3 7 5 6
3 7 6 3 4 4 2 2 4 4 7 7 5 2 1 5
7 0 2 5 5 1 5 7 4 0 2 6 5 3 7 7
5 0 6 6 2 2 0 0 7 6 7 5 7 6 2 2
2 0 7 0 4 6 2 5 6 7 3 2 4 1 5 1
5 7 7 3 5 5 7 1 5 3 3 7 5 6 3 4
0 4 6 5 3 7 6 6 7 5 5 7 0 6 7 3
7 0 6 4 5 4 2 4 3 4 4 1 5 3 7 4
1 4 6 5 5 7 7 0 6 7 3 0 5 0 2 0
3 1 5 3 7 6 0 0 6 6 1 1 6 2 2 0
0 7 6 7 5 7 6 2 2 2 0 7 0 4 6 2
5 6 3 4 7 5 2 1 0 3 3 5 7 3 2 0
0 3 5 3 6 2 2 0 0 7 6 7 5 1 6 3
4 7 4 6 1 5 7 7 0 6 2 1 5 3 6 7
```

Depending on game conditions, video mode may use random number 0..7 or 0..15 for robot timing. This random number affects how far the robot will move and how close it appears to the next or previous robot. Further code analysis is needed to describe this with more detail.

Robot Timing 0..15

0 = Slowest
15 = Fastest

0..15	Total Occurrances
0	9
1	18
2	16
3	16
4	32
5	18
6	37
7	21
8	8
9	16
10	17
11	5
12	9
13	18
14	7
15	9

```
 1  8 15  9  1  8 12 10 10  1 12  5  2 10  1 15
 5  2 12 14 12  5  2  4  6 13  9  9  1  7  6  7
 3 13  9  4  6  2  4  4 10 14  8 10 10  6  4 13
 2 14 13 10  3 11  1  1  3  3 14  6  4  1  8 12
 6 15  1 12  4  0  4  6  3  7  1  5  4 10  6  6
12  7  5 13  9  9  7  7  6  5  6  4  6 13  9  9
 1  7  6  7  3 13  9  4  5  6  2  1  3  0  4  0
 4  6  6  2  4  4 14  8 12 10 10  6  4 13 10  3
 7  3  5  4 10  6 13  5  6  4  4 14  7 13  6  2
 6  7  5 11  4 11  1 11  2 11  3  0  4 10  6  3
 8  3  5  4  4  6 14  7 13  6  2 15  4 15  1 15
 2  0  4 10  6  5 15  7  5 13  8  0 13  9  9  7
 7  6  5  6  4  6 13  9  9  1  7  6  7  3 13  9
 4 13 10  3  6  4  1  8 15  2 10  4  6  2  1 15
 7  2 12  2 13  9  9  7  7  6  5  6  4  0 13 10
 3  6  3  5  0  4  6  6  7  5  1  0  4  2  5  6
```

Video Mode, Updated Code for L8.4

The logic described above for L8.4 Video Mode is implemented in code changes to the L-8 video mode code in bank $2B. The code changes consist of these overall changes:

- Replace the 4 “random” number retrievals with 4 calls into new L8.4 code.
- Update the “EB” award byte init code so it sets byte to 0x80 when EB should be shown.
- Replace the “EB” award byte check with call to new L8.4 code to check if EB should be shown.
- Insert new L8.4 code that provides the 4 “random” numbers or original L-8 random number.
- Insert new L8.4 code that reads the “EB” byte to indicate if EB should be shown.

Video Mode, Updated 4 “random” number calls with 4 new calls into L8.4 code

The four random number retrievals are replaced in L8.4 as shown below. The old code is in red coloring and the new code is in green. These changes are at $436D,2B (ROM offset 0x2C36D), $4386,2B (ROM offset 0x2C386), $4590,2B (ROM offset 0x2C590), and $45A9,2B (ROM offset 0x2C5A9).

```
~~436D: BD A7 25    JSR   $A725          ; GetPseudoRandomNumberIntoA()~~
436D: BD 49 E7    JSR   $49E7           ; L84TournamentModeVideoModeInit07()

~~4386: BD A7 25    JSR   $A725          ; GetPseudoRandomNumberIntoA()~~
4386: BD 49 EE    JSR   $49EE           ; L84TournamentModeVideoModeInit03()

~~4590: BD A7 25    JSR   $A725          ; GetPseudoRandomNumberIntoA()~~
4590: BD 4A 1E    JSR   $4A1E           ; L84TournamentModeVideoModeRobotDir()

~~45A9: BD A7 25    JSR   $A725          ; GetPseudoRandomNumberIntoA()~~
45A9: BD 4A 25    JSR   $4A25           ; L84TournamentModeVideoModeRobotTiming()
```

Video Mode, Updated “EB” award byte init code

As described, a RAM byte is needed to track an incrementing value for the L8.4 “random” robot movements. The RAM byte at $064C is used by L-8 to track if “EB” award should be shown during the video mode. For L8.4 this byte is repurposed whereby the low 7-bits are an incrementing sum used to get different “random” bytes from ROM for each robot random number retrieval while the high 0x80 bit is still used to indicate whether an “EB” award should be shown during the video mode.

The initialization of the $064C byte is updated so that the byte is set to 0x80 if an EB should be shown or 0x00 if no EB should be shown during the video mode. Below is the original L-8 code starting at $43A0,2B, ROM offset 0x2C3A0:

```
43A0: 4F          CLRA                ;
43A1: BD FA 1E    JSR   $FA1E         ; Checking if Video Mode should include Extra Ball
43A4: 7E 43 A7    JMP   $43A7         ; <nop>
43A7: 25 01       BCS   $43AA         ;
43A9: 4C          INCA                ; Set A to non-zero to flag that EB is allowed
43AA: B7 06 4C    STA   $064C         ; $064C non-zero if extra ball should be included
```

Below is the new L8.4 code:

```
43A0: 4F          CLRA                ;
43A1: BD FA 1E    JSR   $FA1E         ; Checking if Video Mode should include Extra Ball
43A4: 25 04       BCS   $43AA         ;
43A6: 8A 80       ORA   #$80          ; Set 0x80 bit to flag EB is allowed during video mode
```

```
43A8: 20 00       BRA   $43AA           ; <nop>
43AA: B7 06 4C    STA   $064C           ; $064C high-bit set if extra ball should be included
```

Video Mode, Replace the "EB" byte check with new L8.4 function to check the byte

As shown, above, the $064C byte changed from L-8 where it was zero or non-zero for "EB" to be shown to L8.4 where the 0x80 bit needs to be set for "EB to be shown or cleared for "EB" not to be shown. This logic change requires a modification to the original L-8 code so it now checks for 0x80 bit of $064C when it is determining whether or not to show "EB" during video mode.

The original L-8 code does a TST instruction on $064C when determining if "EB" should be shown. This code is at $4543,2B, ROM offset 0x2C543:

```
4543: 7D 06 4C    TST   $064C           ;  Test $064C to see if EB should be in video mode
4546: 27 41       BEQ   $4589           ;
```

The replacement L8.4 code, below, replaces the TST instruction with a jump to new L8.4 code (depicted later) that looks at the 0x80 bit of $064C and returning, thus allowing the follow-up BEQ instruction to proceed based on the 0x80 bit set/clear status.

```
4543: BD 4A 2C    JSR   $4A2C           ;  Call L8.4 function to see if EB is to be allowed
4546: 27 41       BEQ   $4589           ;
```

Video Mode, Insert new L8.4 code to provide Tournament mode or original L-8 random numbers

As shown above, there are 4 separate calls into new L8.4 code that each will either return the original L-8 random number (when Tournament Mode is not set) or return a L8.4 "random" number for all players to have same video mode (if Tournament Mode is set). This block of code is added in unused ROM space in bank $2B starting at $491E,2B, ROM offset 0x2C91E, as shown below.

```
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; Video Mode Init 0..7
                                       ;----------------------
                                       ; Table of Bookkeeping Storage Locations (L-8)
                                       ;
491E: 18 A3                            ; Bookkeeping B.3 08 "MATCH AWARDS"
4920: 18 EB                            ; Bookkeeping B.3 28 "1 PLAYER GAMES"
4922: 18 F1                            ; Bookkeeping B.3 29 "2 PLAYER GAMES"
4924: 18 F7                            ; Bookkeeping B.3 30 "3 PLAYER GAMES"
4926: 18 FD                            ; Bookkeeping B.3 31 "4 PLAYER GAMES"
4928: 19 21                            ; Bookkeeping B.5 01 "0-1.9 M. SCORE"
492A: 19 2D                            ; Bookkeeping B.5 03 "3-9.9 M. SCORE"
492C: 19 39                            ; Bookkeeping B.5 05 "20-29 M. SCORE"
492E: 19 45                            ; Bookkeeping B.5 07 "40-49 M. SCORE"
4930: 19 51                            ; Bookkeeping B.5 09 "70-99 M. SCORE"
4932: 19 5D                            ; Bookkeeping B.5 11 "150-199 M. SCORE"
4934: 19 69                            ; Bookkeeping B.5 13 "OVER 300 MILLION"
4936: 19 75                            ; Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"
4938: 19 81                            ; Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"
493A: 19 8D                            ; Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"
493C: 19 99                            ; Bookkeeping B.5 21 "GAME TIME 4-5 M."
493E: 19 A5                            ; Bookkeeping B.5 23 "GAME TIME 6-8 M."
4940: 19 B1                            ; Bookkeeping B.5 25 "GAME TIME 10-15 M"
4942: 00 00                            ; <End of data>
```

```
                                    ;
-----------------------------------;-------------------------------------------------------
                                    ;
                                    ; Video Mode Init 0..3
                                    ;---------------------
                                    ; Table of Bookkeeping Storage Locations (L-8)
                                    ;
4944: 18 EB                         ; Bookkeeping B.3 28 "1 PLAYER GAMES"
4946: 18 F1                         ; Bookkeeping B.3 29 "2 PLAYER GAMES"
4948: 18 F7                         ; Bookkeeping B.3 30 "3 PLAYER GAMES"
494A: 18 FD                         ; Bookkeeping B.3 31 "4 PLAYER GAMES"
494C: 19 27                         ; Bookkeeping B.5 02 "2-4.9 M. SCORE"
494E: 19 33                         ; Bookkeeping B.5 04 "10-19 M. SCORE"
4950: 19 3F                         ; Bookkeeping B.5 06 "30-29 M. SCORE"
4952: 19 4B                         ; Bookkeeping B.5 08 "50-69 M. SCORE"
4954: 19 57                         ; Bookkeeping B.5 10 "100-149 M. SCORE"
4956: 19 63                         ; Bookkeeping B.5 12 "200-299 M. SCORE"
4958: 19 6F                         ; Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"
495A: 19 7B                         ; Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"
495C: 19 87                         ; Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"
495E: 19 93                         ; Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"
4960: 19 9F                         ; Bookkeeping B.5 22 "GAME TIME 5-6 M."
4962: 19 AB                         ; Bookkeeping B.5 24 "GAME TIME 8-10 M."
4964: 19 B7                         ; Bookkeeping B.5 26 "GAME TIME > 15 M"
4966: 00 00                         ; <End of data>
                                    ;
-----------------------------------;-------------------------------------------------------
                                    ;
                                    ; Robot Left/Right 0..1
                                    ;----------------------
                                    ; Table of Bookkeeping Storage Locations (L-8)
                                    ;
4968: 18 A3                         ; Bookkeeping B.3 08 "MATCH AWARDS"
496A: 18 EB                         ; Bookkeeping B.3 28 "1 PLAYER GAMES"
496C: 18 F1                         ; Bookkeeping B.3 29 "2 PLAYER GAMES"
496E: 18 F7                         ; Bookkeeping B.3 30 "3 PLAYER GAMES"
4970: 18 FD                         ; Bookkeeping B.3 31 "4 PLAYER GAMES"
4972: 19 21                         ; Bookkeeping B.5 01 "0-1.9 M. SCORE"
4974: 19 27                         ; Bookkeeping B.5 02 "2-4.9 M. SCORE"
4976: 19 39                         ; Bookkeeping B.5 05 "20-29 M. SCORE"
4978: 19 3F                         ; Bookkeeping B.5 06 "30-29 M. SCORE"
497A: 19 51                         ; Bookkeeping B.5 09 "70-99 M. SCORE"
497C: 19 57                         ; Bookkeeping B.5 10 "100-149 M. SCORE"
497E: 19 69                         ; Bookkeeping B.5 13 "OVER 300 MILLION"
4980: 19 6F                         ; Bookkeeping B.5 14 "GAME TIME 0.0-1.0M"
4982: 19 81                         ; Bookkeeping B.5 17 "GAME TIME 2.0-2.5M"
4984: 19 87                         ; Bookkeeping B.5 18 "GAME TIME 2.5-3.0M"
4986: 19 99                         ; Bookkeeping B.5 21 "GAME TIME 4-5 M."
4988: 19 9F                         ; Bookkeeping B.5 22 "GAME TIME 5-6 M."
498A: 19 B1                         ; Bookkeeping B.5 25 "GAME TIME 10-15 M"
498C: 19 B7                         ; Bookkeeping B.5 26 "GAME TIME > 15 M"
498E: 00 00                         ; <End of data>
                                    ;
-----------------------------------;-------------------------------------------------------
                                    ;
                                    ; Robot Speed 0..7/0..15
                                    ;------------------------
                                    ; Table of Bookkeeping Storage Locations (L-8)
                                    ;
4990: 18 EB                         ; Bookkeeping B.3 28 "1 PLAYER GAMES"
4992: 18 F1                         ; Bookkeeping B.3 29 "2 PLAYER GAMES"
4994: 18 F7                         ; Bookkeeping B.3 30 "3 PLAYER GAMES"
```

```
4996: 18 FD                         ; Bookkeeping B.3 31 "4 PLAYER GAMES"
4998: 19 2D                         ; Bookkeeping B.5 03 "3-9.9 M. SCORE"
499A: 19 33                         ; Bookkeeping B.5 04 "10-19 M. SCORE"
499C: 19 45                         ; Bookkeeping B.5 07 "40-49 M. SCORE"
499E: 19 4B                         ; Bookkeeping B.5 08 "50-69 M. SCORE"
49A0: 19 5D                         ; Bookkeeping B.5 11 "150-199 M. SCORE"
49A2: 19 63                         ; Bookkeeping B.5 12 "200-299 M. SCORE"
49A4: 19 75                         ; Bookkeeping B.5 15 "GAME TIME 1.0-1.5M"
49A6: 19 7B                         ; Bookkeeping B.5 16 "GAME TIME 1.5-2.0M"
49A8: 19 8D                         ; Bookkeeping B.5 19 "GAME TIME 3.0-3.5M"
49AA: 19 93                         ; Bookkeeping B.5 20 "GAME TIME 3.5-4.0M"
49AC: 19 A5                         ; Bookkeeping B.5 23 "GAME TIME 6-8 M."
49AE: 19 AB                         ; Bookkeeping B.5 24 "GAME TIME 8-10 M."
49B0: 00 00                         ; <End of data>
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Get sum of bookkeeping values pointed to by X into A
                                    ;-----------------------------------------------------
49B2: 10 AE 81    LDY   ,X++        ;-\ Y gets address of RAM of next bookkeeping value
49B5: 27 04       BEQ   $49BB       ; | If Y gets 0x0000 then we reached table end, done
49B7: AB A4       ADDA  ,Y          ; | Increment A with value from Y
49B9: 20 F7       BRA   $49B2       ;-/
                                    ;
49BB: 39          RTS               ;
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Get "random" byte from ROM for given data table X
                                    ;-----------------------------------------------------
49BC: 8D F4       BSR   $49B2       ; Call function that gets bookkeeping sum into A
49BE: 8E F9 00    LDX   #$F900      ; X gets addr in ROM from where arbitary byte is loaded
49C1: A6 86       LDA   A,X         ; A now gets byte from ROM
49C3: 39          RTS               ;
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ; Get into X the per-player Video Mode statistic address
                                    ;-----------------------------------------------------
49C4: 34 10       PSHS  X           ;
49C6: 8E 05 B5    LDX   #$05B5      ; Number of per-player Video Modes
49C9: BD FB 29    JSR   $FB29       ; IncrementXByPlayerIndexNumber()
49CC: A6 84       LDA   ,X          ; Init A with gets per-player video modes stat
49CE: 35 90       PULS  X,PC        ;
                                    ;
------------------------------------;-----------------------------------------------------
                                    ;
                                    ;-----------------------------------------------------
                                    ; Get Video Mode Random Number: Common handler, all
                                    ;-----------------------------------------------------
49D0: BD 86 5B    JSR   $865B       ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                    ;  C-bit set when not-equal
49D3: 9A 01                         ; StandardAdjustment026, Tournament Play, Adj=0x9A
49D5: 24 07       BCC   $49DE       ; If c-clr then tournament mode do new L8.4 code
                                    ;
                                    ; Do L-8 original code
49D7: BD A7 25    JSR   $A725       ; GetPseudoRandomNumberIntoA()
49DA: 32 62       LEAS  $0002,S     ; Fixup stack so we don't RTS back to L8.4 caller, below
49DC: 35 30       PULS  X,Y         ; Restore X and Y before returning to L-8 code
                                    ;
49DE: 39          RTS               ; done, return to L-8 or L8.4 code
                                    ;
```

```
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ;-----------------------------------------------------
                                        ; Get Video Mode Random Number: Mode Init Common handler
                                        ;-----------------------------------------------------
49DF: 8D EF       BSR   $49D0           ; Check if tournament mode. If return here then do L8.4.
                                        ;
                                        ; Do L8.4 tournament mode "random" number
49E1: 8D E1       BSR   $49C4           ; Init A with per-player number of video modes
49E3: 8D D7       BSR   $49BC           ; Call function to get random number into A from ROM
49E5: 35 B0       PULS  X,Y,PC          ; Return random number, L-8 code ANDs A with  mask
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ;---------------------------------------------
                                        ; Get Video Mode Random Number: Mode Init 0..7
                                        ;---------------------------------------------
49E7: 34 30       PSHS  Y,X             ;
49E9: 8E 49 1E    LDX   #$491E          ; X gets addr for bookkeeping addrs for 0..7
49EC: 20 F1       BRA   $49DF           ; Go to common handler for geting random number
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ;---------------------------------------------
                                        ; Get Video Mode Random Number: Mode Init 0..3
                                        ;---------------------------------------------
49EE: 34 30       PSHS  Y,X             ;
49F0: 8E 49 44    LDX   #$4944          ; X gets addr for bookkeeping addrs for 0..3
49F3: 20 EA       BRA   $49DF           ; Go to common handler for geting random number
                                        ;
----------------------------------------;-----------------------------------------------------
                                        ;
                                        ;-----------------------------------------------------
                                        ; Get Video Mode Random Number: Robot Common handler
                                        ;-----------------------------------------------------
49F5: 8D D9       BSR   $49D0           ; Check if tournament mode. If returns here then do L8.4.
                                        ;
                                        ; Do L8.4 tournament mode "random" number
                                        ;
49F7: B6 06 4C    LDA   $064C           ; Get saved incrementing value from $064C
49FA: 85 80       BITA  #$80            ; check if EB flag 0x80 is set
49FC: 27 05       BEQ   $4A03           ;
49FE: 4C          INCA                  ;
49FF: 8A 80       ORA   #$80            ; Make sure 80 bit is still set as it should be
4A01: 20 03       BRA   $4A06           ;
4A03: 4C          INCA                  ;
4A04: 84 7F       ANDA  #$7F            ; Make sure 80 bit is cleared as it should be
4A06: B7 06 4C    STA   $064C           ; Store updated incremented value back into $064C
                                        ;
4A09: 84 7F       ANDA  #$7F            ; Make sure 80 bit is cleared so we just have 7-bit sum
4A0B: 34 02       PSHS  A               ; Store the 7-bit $064C sum value onto stack
                                        ;
4A0D: 8D B5       BSR   $49C4           ; Load A with per-player number of video modes
                                        ;
4A0F: 8C 49 68    CMPX  #$4968          ; Robot left/right?
4A12: 27 04       BEQ   $4A18           ;
                                        ;
4A14: AB E0       ADDA  ,S+             ; For robot timing, add 7-bit sum to video modes count
4A16: 20 02       BRA   $4A1A           ;
                                        ;
4A18: A0 E0       SUBA  ,S+             ; For robot left/right, subtract 7-bit sum
                                        ;
```

```
4A1A: 8D A0       BSR   $49BC           ; Call function to get random number into A from ROM
4A1C: 35 B0       PULS  X,Y,PC          ; Return random number, L-8 code ANDs A with mask
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ;-------------------------------------------------
                                        ; Get Video Mode Random Number: Robot Left/Right 0..1
                                        ;-------------------------------------------------
4A1E: 34 30       PSHS  Y,X             ;
4A20: 8E 49 68    LDX   #$4968          ; X gets addr for bookkeeping addrs for robot left/right
4A23: 20 D0       BRA   $49F5           ; Go to common handler for geting random number
                                        ;
----------------------------------------;------------------------------------------------------
                                        ;
                                        ;-------------------------------------------------
                                        ; Get Video Mode Random Number: Robot Timing 0..7/0..15
                                        ;-------------------------------------------------
4A25: 34 30       PSHS  Y,X             ;
4A27: 8E 49 90    LDX   #$4990          ; X gets addr for bookkeeping addrs for robot timing
4A2A: 20 C9       BRA   $49F5           ; Go to common handler for geting random number
                                        ;
----------------------------------------;------------------------------------------------------
```

Readers are encouraged to trace through the L8.4 code, above for developing a full understanding of how the L8.4 code behaves. The four entry points are shown above along with all of the subsequent code as described for the L8.4 video mode tournament mode logic.

Video Mode, Insert new L8.4 code to read the EB byte to see if EB should be shown

Lastly, the new function that the L8.4 code now calls is added in unused ROM space immediately after the previously depicted new L8.4 code. This function checks the 0x80 bit of $064C so the normally running video mode code knows whether or not it should show the “EB” during video mode. This code starts at $4A2C,2B, ROM offset 0x2CA2C.

```
----------------------------------------;------------------------------------------------------
                                        ;
                                        ; Function to check if EB can be shown during video mode
4A2C: 34 02       PSHS  A               ;
4A2E: B6 06 4C    LDA   $064C           ; Get saved incrementing value from $064C
4A31: 85 80       BITA  #$80            ; Check if 80 bit is set
4A33: 35 82       PULS  A,PC            ;
                                        ;
----------------------------------------;------------------------------------------------------
```

The original L-8 code, as shown previously, performs a BEQ instruction after calling the L8.4 function. The BEQ will branch depending on the Z-bit having been set or cleared based on the BITA instruction that was performed in this new function.

Extra Ball Award Improvements for L8.4

The T2 software has several adjustments regarding how extra ball is handled. These adjustments allow the game to restrict the amount of extra balls the player can have accumulated or win on a per-ball-in-play basis.

Originally while going over L8.4 it was assumed that A.1 03 adjustment limited the number of extra balls that a player can win per game however it was later understood that A.1 03 adjustment actually adjusts the maximum number of extra balls a player can have accumulated (remaining to play out) at any given moment.

In addition to the above adjustments for limiting the number of extra balls, the game also offers an adjustment that allows the game to award an extra ball when the replay score has been reached, as depicted below:

For L8.4, with its several enhancements related to Tournament Mode, some consideration was given to how the game is typically used in a tournament setting. As a lot of tournaments have restrictions on extra ball awards such as requiring them to be drained and not played. Some improvements were made in L8.4 for how the game behaves when the A.1 03 adjustment is set to disallow all extra balls, as depicted below:

With an assumption that people using tournament mode might also want to set A.1 03 to “NO EX BALL”, some changes were put in place to make the game behave better when the game is set to disallow extra balls in this way.

The goal of the L8.4 changes are to keep the changes simple, and not to alter any game rules. The L8.4 changes are mainly about preventing the player from having to see the extra ball animation when the A.1 03 adjustment is set to “NO EX BALL”. The changes for L8.4 have been carefully considered and the following assumptions are being made:

- It is unlikely that A.1 03 is set to a very low number, robbing players of extra balls.
- It is unlikely that A.1 04 is set to a very low number, robbing players of extra balls.
- It is unlikely that A.1 14 is set to award EB at replay and A.1 03/04 set to limit extra balls.

Because of these assumptions, no specific L8.4 changes are in place to deal with odd game behaviors such as when the A.1 03/04 are set to *restrict* extra balls and player is still allowed to light the EB lamp (in some cases the player can still collect such lit EB at a subsequent ball-in-play). It is also not worth addressing issues when A.1 14 is set to award EB at replay score but player is then blocked from receiving such EB due to restrictive settings of A.1 03/04.

It would seem that somebody restricting the EB awards with A.1 03/04 is either unfairly robbing the player of their extra balls, or it may be that such setting of the game is commonly known by its players and they would, therefore, not be surprised by the game not giving them their extra ball. In some ways players may even work such restrictive settings into their game play strategy.

The L8.4 changes are only focused on improving the game behavior when A.1 03 is set to “NO EX BALL” and, as such, the following issues are being addressed:

- When player is ineligible for receiving an extra ball and the skull shot is hit (drop target or ball popper), the L-8/L8.3 software will extinguish the EB lamp, play the EB animation, and NOT award the EB to the player. For L8.4, the changes will simply exclude the EB animation from being shown in this case. *This change will apply no matter what the settings are for A.1 03/04.*
- When player is ineligible for receiving an extra ball and the left-loop shot is hit to award Extra Ball, the L-8/L8.3 software will show the EB animation and NOT award the EB to the player. For L8.4, the changes will simply exclude the EB animation from being shown in this case. *This change will apply no matter what the settings are for A.1 03/04. The same code change from previous bullet item will automatically apply to this item, fixing both with single code change.*

- When player is ineligible for receiving an extra ball and the bonus-x multipliers are advanced to the point of where EB-lit normally occurs, the L-8/L8.3 software will light the EB lamp. For L8.4 in this case only if A.1 03 is set to "NO EX BALL" then the EB lamp will not be lit. For all other adjustment values the EB will still be allowed to be lit. As previously stated, in some scenarios the player could still collect the lit EB at a subsequent ball-in-play.
- When player is ineligible for receiving an extra ball, the flipper-button status-report will report the bonus-x value for lighting the EB is shown. For L8.4 in this case only if A.1 03 is set to "NO EX BALL" then the status report will exclude the bonus-x level for lighting extra ball. For all other adjustment values the status report will still show the bonus-x needed for lighting the extra ball.
- When player is ineligible for receiving an extra ball and the player is doing poorly at the start of the last ball (usually ball 3), the L-8/L8.3 software will give a consolidation "Extra Ball is Lit" even though subsequent skull-shot will not award the extra ball. For L8.4 in this case only if A.1 03 is set to "NO EX BALL" then the game will not give a consolidation "Extra Ball is Lit".

These simple set of changes are designed to prevent the game from doing anything related to extra ball when the A.1 03 is set to "NO EX BALL" since it makes no sense for the game, in such case, to imply to the player that it is possible to win an extra ball while it will never actually be awarded. To be clear, in L-8/L8.3 the game already prevents the player from being awarded the extra ball in these cases, however the L-8/L8.3 software would annoyingly play the extra ball award animation while not giving the player any such extra ball (when player is ineligible for an extra ball due to A.1 03/04 settings). The L8.4 changes also prevent the bonus-x rollovers from unnecessarily lighting the EB since the EB will not actually be awarded. For completeness it makes sense to prevent the flipper-button status-report from reporting the bonus-x value to light extra ball since the L8.4 will no longer light extra ball and also since the EB will never be awarded (again, due to A.1 03 being set to "NO EX BALL"). Lastly, the consolidation "Extra Ball Lit" at last ball will not be given since is makes no sense to do so when the game is set to "NO EX BALL" (this is also referred to as a Lit Consolidation Ball).

For cases where the A.1 03 is set to anything other than "NO EX BALL" the game will still allow the extra ball to be lit even though subsequent skull shot may not actually award the extra ball. However in this case the changes, described above, will prevent the skull shot from playing the EB animation if the player is not eligible for receiving the extra ball. In such cases the skull shot will result in the EB lamp being extinguished and the game will proceed without the extra ball animation (since the player is being prevented from receiving the EB).

It is worth noting that the L-8/L8.3 software already has the following provisions:

- The DB award will not give extra ball or lite extra ball if player is ineligible for EB, and
- The Video Mode will not present EB award if player is ineligible for extra ball.

The L8.4 does not need any new code to exclude the extra ball from database award or video mode.

Extra Ball Award Improvement: No Extra Ball Animation

If the skull shot is hit while it has the extra ball lamp lit or left-loop award for extra ball has been reached, the L8.4 code will prevent the EB animation from playing when the player is ineligible for extra ball. The L-8/L8.3 code, in this case, will play the EB animation but not actually award the player the extra ball, so this change in L8.4 will make the code behave in a slightly better way although the player would still have the sense of being robbed of the EB since it was lit when they hit the skull shot (drop target or ball-popper).

As previously described, for L8.4 the focus is on improving the game when adjustment A.1 03 is set to "NO EX BALL" where the goal of L8.4 is to remove all game-play references to extra ball. In other cases where the game A.1 03/04 *limit* the extra ball awards, this L8.4 code change will also be used to prevent the game from displaying the extra ball animation while not actually awarding the extra ball. In such cases the result is simply that the extra ball lamp is extinguished and game play resumes.

The applicable code is shown below. It is used when the EB is being awarded for hits to the drop-target, ball-popper, left-loop, or from the database award. Since the database award code already prevents the player from getting Extra Ball or Lite Extra Ball, it is not expected that this code will be used in a way where the Database award will have to inhibit the EB animation. When the drop-target or ball-popper reaches this code, there is a chance that the player may not be eligible for EB award and, as such, the code as shown below will result in animation being played while the player actually is not given the EB.

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; AwardExtraBall()
                                        ;
                                        ; Called when awarding extra ball at ball-popper shot
                                        ; Called when awarding extra ball at database award
                                        ; Called when awarding extra ball at left-loop shot
                                        ;
573B: 34 10       PSHS  X               ;
573D: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
5740: D8                                ;
5741: 20 06       BRA   $5749           ;
                                        ;
5743: 34 10       PSHS  X               ; Called here from drop-target switch handler
5745: BD 84 80    JSR   $8480           ; SetMemoryFlag()
5748: D8                                ;
                                        ;
5749: BD B3 83    JSR   $B383           ; TestCurrentPlayerAllowedExtraBallAndAwardEb()
                                        ;  c-set if extra ball not allowed
574C: 8E 05 AD    LDX   #$05AD          ; 0x05AD == Extra balls awarded to current player
574F: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
5752: 7E 57 55    JMP   $5755           ; <nop>
5755: 6C 84       INC   ,X              ;
                                        ;
5757: BD 52 6D    JSR   $526D           ; WaitForAnimationCompletionE8()
575A: BD 6C DD    JSR   $6CDD           ; WaitForBallStoppageCompletion()
575D: 24 1E       BCC   $577D           ;
575F: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
5762: D8                                ;
5763: 24 10       BCC   $5775           ; If 0xD8 flag is set, drop-target-down caused the EB
                                        ; Two different EB effects, differ in playfield lighting
5765: 96 C0       LDA   $C0             ;
```

```
5767: 81 01       CMPA  #$01            ;
5769: 22 0A       BHI   $5775           ;
576B: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
576E: 6D                                ; 0x6D = "Extra ball"
576F: BD 85 B2    JSR   $85B2           ; ScheduleFnFrom5A0ABank34LookupTableParameterByte()
5772: 33                                ; 0x33: $5F4E,34 ExtraBallAwardB() ball-popper / db EB
5773: 20 08       BRA   $577D           ;
                                        ;
5775: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
5778: 6D                                ; 0x6D = "Extra ball"
5779: BD 85 B2    JSR   $85B2           ; ScheduleFnFrom5A0ABank3DLookupTableParameterByte()
577C: 32                                ; 0x32: $5F52,34 ExtraBallAwardA() drop-target-down EB
                                        ;
577D: BD 71 94    JSR   $7194           ; Increment per-player stat $05C1
5780: BD 71 94    JSR   $7194           ; Increment per-player stat $05C1
5783: BD 71 94    JSR   $7194           ; Increment per-player stat $05C1
5786: BD 71 94    JSR   $7194           ; Increment per-player stat $05C1
5789: 35 90       PULS  X,PC            ;
                                        ;
----------------------------------------;------------------------------------------------------
```

The highlighted portion of code shows where the problem behavior is taking place in L-8. The call to $B383 is to a function that ultimately gives the player the EB or not, depending on player eligibility. In the case that the $B383 function did not give the player their EB, the function returns with C-bit set to indicate as such. The code here, however, pays no attention to the state of the C-bit and proceeds with the further code to display the Extra Ball animation. This means the EB animation is shown even if the player was not awarded an extra ball. There are two possible EB animation effects, depending on whether the ball is stopped on in the playfield (database award, left-loop, or ball-popper) or whether the ball is still in play such as after knocking the drop-target down to get the extra ball. The different EB effects differ in how they control playfield lighting during the EB animation.

For L8.4 the code is updated so it checks the C-bit and goes to end of function if C-bit is set. Since the nearby code has a dummy JMP instruction, the 3 bytes of such dummy instruction are utilized for this new L8.4 logic as follows.

The original code is as follows:

```
5749: BD B3 83    JSR   $B383           ; TestCurrentPlayerAllowedExtraBallAndAwardEb()
                                        ;  c-set if extra ball not allowed
574C: 8E 05 AD    LDX   #$05AD          ; 0x05AD == Extra balls awarded to current player
574F: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
5752: 7E 57 55    JMP   $5755           ; <nop>
5755: 6C 84       INC   ,X              ;
```

Replacement L8.4 code is as follows:

```
5749: BD B3 83    JSR   $B383           ; TestCurrentPlayerAllowedExtraBallAndAwardEb()
                                        ;  c-set if extra ball not allowed
574C: 25 3B       BCS   $5789           ; If c-set branch to end of function, skip animation
574E: 12          NOP                   ;
574F: 8E 05 AD    LDX   #$05AD          ; 0x05AD == Extra balls awarded to current player
5752: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
5755: 6C 84       INC   ,X              ;
```

With the above change in place, the EB animation is not shown when the $B383 function returns C-bit set which occurs when $B383 bypassed the EB award due to player not being eligible for EB due to the game adjustments and with checking how many EBs the player has accumulated or achieved in the current current ball in play. Instead of announcing the Extra Ball award with animation (and then not awarding the EB to the player) the L8.4 will silently proceed without any extra ball fanfare when it was not going to give the player the extra ball anyway (due to player exceeding thresholds established by the A.1 03/04 adjustments).

Extra Ball Award Improvement: No EB Lit at Bonus-X

As mentioned, one of the L8.4 EB improvements is to fix so that the EB will not be lit when player gets bonus-x multiplier normally needed to light extra ball while the game adjustment A.1 03 is set to "NO EX BALL". The goal of this code change is to simply improve the game play experience when the game adjustments are set to disallow all extra balls by removing mention of "extra ball" to the player at all times. There is no benefit to the player by allowing the extra ball to become lit in this case since the player will never be awarded the extra ball due to A.1 03 adjustment.

The applicable portion of code for bonus-x multiplier advancement regarding "lite extra ball" is shown below. This is the original L-8/L8.3 code located at $5B97,31, ROM offset 0x45B97:

```
5B97: 8D 3E       BSR   $5BD7          ; Loads B with 3-bank completions for bonus-x lite-EB
5B99: 34 04       PSHS  B              ;
5B9B: A1 E0       CMPA  ,S+            ; Compare completions with num needed to lite EB
5B9D: 35 02       PULS  A              ;
5B9F: 26 0B       BNE   $5BAC          ;
5BA1: BD 84 80    JSR   $8480          ; SetMemoryFlag()
5BA4: 49                               ; 0x49 flag player reached bonus-x lite-EB this game
5BA5: BD 84 80    JSR   $8480          ; SetMemoryFlag()
5BA8: 4A                               ; 0x4A bonus-x lite-EB flag, cleared at eob when
                                       ;  "Hold Bonus" lamp is not lit
5BA9: BD 57 2E    JSR   $572E          ; LiteExtraBall()
5BAC: BD 85 53    JSR   $8553          ; ShowMonochromeAnimationParameterByte()
5BAF: 2A                               ; 0x2A bonus-x advancement animation
```

This code is exercised when the 3-bank lamps have been completed. The shown code will determine how many 3-bank completions are needed to lite the extra ball and proceeds accordingly. When the EB is to be lit then 2 flags are set and the LiteExtraBall() function is called. The two flags are:

- 0x49, flag that is set to indicate player has achieved at least 1 "lite extra ball" using bonus-x advancement. This flag allows subsequent 3-bank completions to require the next higher level of bonus-x to be reached for a subsequent "lite extra ball" during the current game for the player.
- 0x4A, flag that is set to indicate that the player has already reached "lite extra ball" on the current set of bonus-x advancement. This prevents multiple "lite extra balls" from happening as the player goes from 2x, 4x, 6x, 8x. At the end of ball, as long as "Hold Bonus" is not lit, this flag gets cleared so player can reach "lite extra ball" again through the bonus-x advancement.

To keep the L8.4 code change simple, the code in this region is replaced with a call to a new function that will determine if the “lite extra ball” should be called depending on the A.1 03 adjustment value. This code change will retain the current way in which the game sets flags 0x49 and 0x4A which should be of zero consequence since the game uses these flags in determining whether or not to award player “lite extra ball” which, with this change will never happen anyway when the A.1 03 is set to “NO EX BALL”. Due to lack of available space in bank $31, the new code is located in bank $3B alongside other new L8.4 code. This requires using the $88F5 function to call the new function which requires additional 3 bytes for the JMP instruction. The new function in bank $2B will perform the flag set function calls which were replaced by the JMP instruction. The new function simply returns C-bit set if it is okay to light the extra ball. The new function returns C-bit clear if the bonus-x should NOT light the extra ball.

For this change the updated code is as follows, updated code is highlighted:

```
5B97: 8D 3E       BSR   $5BD7           ; Loads B with 3-bank completions for bonus-x lite-EB
5B99: 34 04       PSHS  B               ;
5B9B: A1 E0       CMPA  ,S+             ; Compare completions with 0x04 or 0x00
5B9D: 35 02       PULS  A               ;
5B9F: 26 0B       BNE   $5BAC           ;
5BA1: BD 84 80    JSR   $8480           ; SetMemoryFlag()
5BA4: 49                                ; 0x49 flag player reached bonus-x lite-EB this game
5BA5: BD 84 80    JSR   $8480           ; SetMemoryFlag()
5BA8: 4A                                ; 0x4A bonus-x lite-EB flag, cleared at eob when
                                        ;  "Hold Bonus" lamp is not lit
5BA1: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
5BA4: 4A 35 2B                          ; BonusXLiteExtraBallDeterminator() // c-set if EB okay
5BA7: 24 03       BCC   $5BAC           ; If C-clear then skip the LiteExtraBall()
5BA9: BD 57 2E    JSR   $572E           ; LiteExtraBall()
5BAC: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
5BAF: 2A                                ; 0x2A bonus-x advancement animation at $7203,35
```

The above change will result in new code located at $4A35,2B being called which will set the flags and then check if the A.1 03 adjustment is set to “NO EX BALL”, returning c-bit clear if set as such. If adjustment A.1 03 is set to anything else then C-bit is set. Upon return, the c-bit is analyzed and if set, the LiteExtraBall() function is called to illuminate the EB lamp, otherwise it is skipped so the EB lamp is not illuminated.

The new function is at $4A35,2B, ROM offset 0x2CA35 in previously unused ROM space, as shown below:

```
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; BonusXLiteExtraBallDeterminator()
                                       ;
                                       ; Returns C-set if okay to lite extra ball
                                       ; Returns C-clr if no lite extra ball
                                       ;
4A35: BD 84 80    JSR   $8480          ; SetMemoryFlag()
4A38: 49                               ; 0x49 flag player reached bonus-x lite-EB this game
4A39: BD 84 80    JSR   $8480          ; SetMemoryFlag()
4A3C: 4A                               ; 0x4A bonus-x lite-EB flag, cleared at eob when
                                       ;  "Hold Bonus" lamp is not lit
4A3D: BD 86 5B    JSR   $865B          ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                       ;  C-bit set when not-equal
4A40: 83 00                            ; StandardAdjustment003, Max E.B. Count, Adj=0x83
```

```
4A42: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
```

As shown, the new function simply checks the “Max E.B. Count” adjustment checking for value 0x00 which is “NO EX BALL”. The result of the adjustment lookup is that C-bit is set when the adjustment is anything other than 0x00. The resulting C-bit state is then retained as the code returns back to the calling function depicted previously.

As it turns out, the above change effectively ensures the Extra Ball lamp is not illuminated when bonus-x reaches the “lite extra ball” level while adjustment A.1 03 is set to “NO EX BALL”, however after the display shows the new bonus-x animation, it will proceed and display “Extra ball is lit” on the display. A further survey of the game code reveals additional changes are needed to properly ensure the “Extra ball is lit” text is not shown in this case.

Depicted above, when the bonus-x is advanced, the following function is invoked:

```
5BAC: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
5BAF: 2A                                ; 0x2A bonus-x advancement animation at $7203,35
```

The above 0x2A index is an index to cause code to call function at $7203,35, offset 0x57203. This function displays the bonus-x animation and follows it up with the “Extra ball is lit” display when it determines the new bonus-x is at the level needed to light the extra ball lamp.

Towards the end of the $7203,35 function is the following code where the decision is made to display “Extra ball is lit”.

```
72AB: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
72AE: 5B D7 31                          ; Audit 3-rollover to see if extra ball should be lit
72B1: 34 04       PSHS  B               ;
72B3: A1 E0       CMPA  ,S+             ; See if 3-rollover completions matches the value in A
72B5: 26 06       BNE   $72BD           ; If no match, skip to the end
72B7: 86 37       LDA   #$37            ; If match, A gets 0x37
72B9: 97 B4       STA   $B4             ; Store 0x37 into $B4
72BB: 20 6D       BRA   $732A           ;
72BD: 7E C9 52    JMP   $C952           ;
```

Readers following along may notice this code calls the $5BD7,31 function to check if extra ball should be lit. This is the same function depicted earlier in code that lead up the illumination (or lack thereof) of the extra ball lamp. In this case, logic proceeds to compare the extra ball lit level with the current/new bonus-x level to determine if “Extra ball is lit” text should be shown.

When the level deserves the display of “Extra ball is lit” code at $72B7,35 proceeds and sets up for the display of the text by loading $B4 with 0x37 and then branching to $732A to further proceed with the display of “Extra ball is lit” along with audio dialog “Get the extra ball”.

Note in this code path, the code ends up jumping to $C952 when done. Note that code simply branches to $732A and doesn’t expect to return. For completeness, the $732A,35 function is shown below:

```
----------------------------------------;-------------------------------------------------
                                        ;
732A: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
732D: 6B                                ; 0x6B == "Get the extra ball"
732E: 8E 00 23    LDX   #$0023          ;
7331: BD FB AE    JSR   $FBAE           ; ClearDisplayMemory()
7334: 7E 73 37    JMP   $7337           ; <nop>
7337: C6 10       LDB   #$10            ;
7339: 10 8E 40 0F LDY   #$400F          ;
733D: BD D7 DB    JSR   $D7DB           ;
7340: BD D7 99    JSR   $D799           ; Prints string on display
7343: 00 24                             ; String index for "Extra ball is lit"
7345: 10                                ; Font
7346: 40 1F                             ; Coordinates
7348: C6 0A       LDB   #$0A            ; Play splat sound 10 times
734A: C5 01       BITB  #$01            ;-\
734C: 26 04       BNE   $7352           ; |
734E: BD 85 46    JSR   $8546           ; | DoSoundTableParameterByte()
7351: 49                                ; | 0x49 == Computer splat
7352: BD E2 74    JSR   $E274           ; |
7355: BD 83 46    JSR   $8346           ; | Sleep
7358: 08                                ; | 0.125 seconds
7359: 5A          DECB                  ; |
735A: 26 EE       BNE   $734A           ;-/
735C: 7E C9 52    JMP   $C952           ;
                                        ;
----------------------------------------;-------------------------------------------------
```

For L8.4 to remove the "Extra ball is lit" portion of bonus-x during A.1 03 "NO EX BALL", the previously shown code is updated as shown below, with changes highlighted.

```
72AB: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
72AE: 5B D7 31                          ; Audit 3-rollover to see if extra ball should be lit
72B1: 34 04       PSHS  B               ;
72B3: A1 E0       CMPA  ,S+             ; See if 3-rollover completions matches the value in A
72B5: 26 06       BNE   $72BD           ; If no match, skip to the end
72B7: 86 37       LDA   #$37            ; If match, A gets 0x37
72B9: 97 B4       STA   $B4             ; Store 0x37 into $B4
72B7: 7E 7E E8    JMP   $7EE8           ; Go to L8.4 intermediate function
72BA: 12          NOP                   ; filler instruction
72BB: 20 6D       BRA   $732A           ;
72BD: 7E C9 52    JMP   $C952           ;
```

The code will jump to new L8.4 code at $7EE8,35, ROM offset 0x57EE8. This happens to be code that was made available/unused as part of L8.3 code changes. In L8.3 the fan-club code was completely removed and made into unused ROM bytes. With limited amount of free space in bank $35, the L8.4 is able to take advantage of this available ROM space for dealing with this "Extra ball is lit" code change.

The new code at $7EE8,35 is as follows:

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; New L8.4 code shown here for reference
                                        ;
7EE8: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
7EEB: 83 00                             ; StandardAdjustment003, Max E.B. Count, Adj=0x83
7EED: 25 03       BCS   $7EF2           ; If C-set then proceed to normal L-8 code
```

```
7EEF: 7E C9 52    JMP   $C952           ; C-clr then "NO EX BALL", done, go to $C952
7EF2: 86 37       LDA   #$37            ; If match, A gets 0x37
7EF4: 97 B4       STA   $B4             ; Store 0x37 into $B4
7EF6: 7E 73 2A    JMP   $732A           ; Continue with regular "Extra ball is lit" code
                                        ;
----------------------------------------;------------------------------------------------
```

With the above code in place, when the adjustment A.1 03 is set to “NO EX BALL”, the bonus-x will no longer award extra ball or report that extra ball is lit.

Extra Ball Award Improvement: No EB Info in Flipper-Button Status-Report

To complement the changes shown above, the L8.4 will also ensure that when the A.1 03 is set to “NO EX BALL” that the flipper-button status-report will no longer give a report of the bonus-x level needed to achieve the lit extra ball. This refers to the following screen shown in the status report that is shown when a flipper button is held down during game play:

When adjustment A.1 03 is set to “NO EX BALL” it makes no sense to report this to the player especially considering that the previous set of L8.4 changes will prevent the game from reporting that the extra ball is lit when the game adjustment is set this way.

Code has been surveyed to determine that the applicable portion of code for the flipper button status report is located at $539B,24, ROM offset 0x1139B, as depicted below:

```
539B: BD 88 F5        JSR   $88F5         ; CallBankedFunction_Param_WPCAddr()
539E: 5B D7 31                            ; Audit 3-rollover to see if extra ball should be lit
53A1: 58              ASLB                ;
53A2: 34 04           PSHS  B             ;
53A4: A1 E0           CMPA  ,S+           ;
53A6: 24 1A           BCC   $53C2         ;
                                          ;
53A8: BD D3 60        JSR   $D360         ; Clear512BytesFrom1799Pointer()
53AB: BD D7 99        JSR   $D799         ; Prints string on DMD
53AE: 00 F4                               ; String index 0xF4: "%bX   LITES"
53B0: 09                                  ; Font
53B1: 40 0B                               ; Coordinates
                                          ;
53B3: BD D7 99        JSR   $D799         ; Prints string on DMD
53B6: 00 23                               ; String index 0x23: "EXTRA BALL"
53B8: 09                                  ; Font
53B9: 40 1A                               ; Coordinates
                                          ;
53BB: BD E2 74        JSR   $E274         ;
53BE: BD 83 46        JSR   $8346         ; Sleep
53C1: 80                                  ; 2.0 seconds
53C2: 35 F6           PULS  A,B,X,Y,U,PC  ;
```

For L8.4 a small change is added to this code to get into new L8.4 code that will check if the adjustment A.1 03 is set to “NO EX BALL” and, if so, finish without displaying this status report message. The updated code is highlighted below:

```
539B: BD 88 F5        JSR   $88F5        ; CallBankedFunction_Param_WPCAddr()
539E: 5B D7 31                           ; Audit 3-rollover to see if extra ball should be lit
53A1: 58              ASLB               ;
53A2: 34 04           PSHS  B            ;
53A4: A1 E0           CMPA  ,S+          ;
53A6: 24 1A           BCC   $53C2        ;
                                         ;
53A8: BD D3 60        JSR   $D360        ; Clear512BytesFrom1799Pointer()
53A8: 7E 57 57        JMP   $5757        ; Go to L8.4 code to check for "NO EX BALL" setting
53AB: BD D7 99        JSR   $D799        ; Prints string on DMD
53AE: 00 F4                              ; String index 0xF4: "%bX   LITES"
53B0: 09                                 ; Font
53B1: 40 0B                              ; Coordinates
                                         ;
53B3: BD D7 99        JSR   $D799        ; Prints string on DMD
53B6: 00 23                              ; String index 0x23: "EXTRA BALL"
53B8: 09                                 ; Font
53B9: 40 1A                              ; Coordinates
                                         ;
53BB: BD E2 74        JSR   $E274        ;
53BE: BD 83 46        JSR   $8346        ; Sleep
53C1: 80                                 ; 2.0 seconds
53C2: 35 F6           PULS  A,B,X,Y,U,PC ;
```

The updated L8.4 code will jump into new code at $5757,24 which is unused ROM space in L-8. The new L8.4 function is shown below.

```
----------------------------------------;--------------------------------------------------
                                        ;
                                        ; The L8.4 $5757 code is shown here
                                        ;
5757: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
575A: 83 00                             ; StandardAdjustment003, Max E.B. Count, Adj=0x83
575C: 25 03       BCS   $5761           ; If C-set then proceed to normal L-8 code
575E: 7E 53 C2    JMP   $53C2           ; C-clr then "NO EX BALL", done, go to $53C2 to finish
5761: BD D3 60    JSR   $D360           ; Clear512BytesFrom1799Pointer()
5764: 7E 53 AB    JMP   $53AB           ; Go back and finish displaying the bonus-x report
                                        ;
----------------------------------------;--------------------------------------------------
```

An examination of the code will show how the code will simply end the attempt to show the flipper-button bonus-x report when the adjustment A.1 03 is set to “NO EX BALL” (value 0x00). When the adjustment is set to anything else, then the original L-8 code is performed. This logic is utilizing the JMP instruction to allow the new L8.4 function to wedge itself into the logic.

Extra Ball Award Improvement: No Lit Consolidation Ball

For L8.4 when the adjustment A.1 03 is set to “NO EX BALL” the code is updated so that players experiencing a poor game are not teased with the “Extra Ball is Lit” message when the final ball in play begins. In such case, in L-8, the player is given such message however they will not actually be awarded

the extra ball when adjustment A.1 03 is set to “NO EX BALL”. To improve the player experience, the L8.4 code is updated to ensure that no such lit extra ball will take place in this situation.

The applicable portion of code is shown below. This is a piece of code that is executed at the start of every ball. Code has a handful of checks that is performs in order to determine whether or not to give the player a “Extra ball is lit” as a consolidation for poor performance. This code starts at $625B,3B, ROM offset 0x6E25B:

```
----------------------------------------;--------------------------------------------------
                                        ;
                                        ; StartOfBallCurrentPlayer_ResetPlayfieldState()
625B: 34 16       PSHS  X,B,A           ;
625D: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
6260: D0                                ; 0xD0 == First Ball Started
6261: 24 04       BCC   $6267           ; If already started first ball skip "Los Angeles" msg
6263: BD 85 53    JSR   $8553           ; ShowMonochromeAnimationParameterByte()
6266: 34                                ; 0x34 == Show "Los Angeles July 11, 2029"
6267: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
626A: D1                                ; 0xD1 == Lit Consolidation Ball
626B: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;    C-bit set when not-equal
626E: 04 00                             ; FeatureAdjustment004, Consolation Ball, Adj=0x04
6270: 24 49       BCC   $62BB           ; If Consolidation ball is off then branch to $62BB
                                        ;
                                        ;------------------------------------------------------
                                        ; Consolidation ball enabled, check if EB should be lit
                                        ;------------------------------------------------------
6272: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
                                        ;    C-clear if on last ball
6275: 25 44       BCS   $62BB           ; No consolidation ball for player, skip to $62BB
6277: BD B3 CA    JSR   $B3CA           ; Get extra balls remaining to play
627A: 4D          TSTA                  ; Check if any extra balls remaining to play
627B: 26 3E       BNE   $62BB           ; If there are extra balls, then no consolidation ball
627D: 8E 05 AD    LDX   #$05AD          ; 0x05AD == Extra balls awarded to current player
6280: BD FB 29    JSR   $FB29           ; IncrementXByPlayerIndexNumber()
6283: 7E 62 86    JMP   $6286           ; <nop>
6286: 6D 84       TST   ,X              ; Check if player won any extra balls this game
6288: 26 31       BNE   $62BB           ; If player won any EBs then no consolidation ball
628A: BD B1 AB    JSR   $B1AB           ; GetCurrentPlayerIndexIntoA()
628D: BD BB 3E    JSR   $BB3E           ; GetPlayerScoreIndexAintoU()
6290: 6D C4       TST   ,U              ; i.e. U --> 00 02 08 02 70 == 2,080,270 points.
6292: 26 27       BNE   $62BB           ; If score fills 1st score byte, no consolidation ball
6294: A6 41       LDA   $0001,U         ; Score is >= 100,000,000 points, no consolidation ball
6296: 81 03       CMPA  #$03            ; If score next two digits >= 03, no consolidation ball
6298: 24 21       BCC   $62BB           ; Score is >= 3,000,000 points, no consolidation ball
629A: BD 88 F5    JSR   $88F5           ;
629D: 4C 7B 38                          ; Gets X per-player game state data
62A0: 6D 88 22    TST   $22,X           ; game-time high-byte non-zero, no consolidation ball
62A3: 26 16       BNE   $62BB           ; player game-time is >= 256 seconds, 4m16s
62A5: EC 88 22    LDD   $22,X           ; Check low byte of player game-time
62A8: C1 1E       CMPB  #$1E            ; Compare player time with 30 seconds
62AA: 22 0F       BHI   $62BB           ; If higher than 30 then no consolidation ball
62AC: BD 88 D5    JSR   $88D5           ; IncreaseBookkeepingCounterAddrXBy1ParamBytes()
62AF: 00 1F                             ; 31. Lit consolidation ball counter
62B1: BD 84 80    JSR   $8480           ; SetMemoryFlag()
62B4: D1                                ; 0xD1 == Lit Consolidation Ball
62B5: BD 88 F5    JSR   $88F5           ;
62B8: 57 23 31                          ; LiteExtraBallWithExtraBallIsLitMessage()
                                        ;
```

```
62BB: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;    C-bit set when not-equal
62BE: 03 01                           ; FeatureAdjustment003, Extra Ball Memory, Adj=0x03
```

As can be seen in the above logic, the game will only present the player with a free “Extra Ball is Lit” at ball start when all of the following criteria are true:

- Game adjustment A.2 04 Consolidation Ball is set to “On”, and
- It is start of last ball in play for the current game, and
- There are zero extra balls for the current player to play out, and
- The player has won zero extra balls in the current game, and
- The player’s score is less than 3,000,000 points, and
- The player’s game time is 30 seconds of less

A simple code change is added where the a new function will be called that will check if game adjustment A.1 03 is set to “NO EX BALL” so that the consolidation ball can be skipped if this is the case. The altered code is shown, highlighted, below:

```
626B: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;    C-bit set when not-equal
626E: 04 00                           ; FeatureAdjustment004, Consolation Ball, Adj=0x04
6270: 24 49       BCC   $62BB           ; If Consolidation ball is off then branch to $62BB
                                        ;
                                        ;-------------------------------------------------------
                                        ; Consolidation ball enabled, check if EB should be lit
                                        ;-------------------------------------------------------
6272: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
6272: BD 78 DA    JSR   $78DA           ; ConsolidationBallCheckL84()
                                        ;    C-clear if consolidation ball is allowed
6275: 25 44       BCS   $62BB           ; No consolidation ball for player, skip to $62BB
6277: BD B3 CA    JSR   $B3CA           ; Get extra balls remaining to play
```

The L8.4 fix simply calls a new L8.4 function which will check if lit consolidation ball may occur. The new function will include the check for last ball since the original such check was replaced with the call to the new function. The new function is as shown below:

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; ConsolidationBallCheckL84()
                                        ;   returns C-clr if lit consolidation ball is allowed
                                        ;   returns C-set if lit consolidation ball disallowed
                                        ;
78DA: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;  C-bit set when not-equal
78DD: 83 00                           ; StandardAdjustment003, Max E.B. Count, Adj=0x83
78DF: 25 04       BCS   $78E5           ; If c-bit is set, consolidation ball may take place
78E1: 1A 01       ORCC  #$0001          ; C-clr, no consolidation ball, set the C-bit and return
78E3: 20 03       BRA   $78E8           ;
78E5: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
                                        ;    C-clear if on last ball
78E8: 39          RTS                   ;
                                        ;
----------------------------------------;-------------------------------------------------
```

The L8.4 function checks if adjustment A.1 03 is set to “NO EX BALL” and, if so, returns right away with C-bit set to signal the calling function that consolidation ball is not to be allowed so no extra ball will be lit if player is doing poorly at last ball. If the A.1 03 is not set to “NO EX BALL” then the original logic from L-8 will resume where the $B1D1 function is called to check if player is on the last ball, returning C-bit clear if on last ball (signaling the calling function that consolidation ball is allowed).

With this and all of the preceding changes in place, the L8.4, when A.1 03 is set to “NO EX BALL” will not award an extra ball or lit extra ball to the player at any point during game play.

Skill-Shot 5-Bank Failure Investigation, L8.4

As part of L8.4, an investigation was made into how the game handles the failure scenario where none of the 5-bank targets are hit during skill shot. No code changes were done in L8.4 as a result of this investigation however the results are briefly documented here for posterity.

What was noticed is that when none of the 5-bank targets are hit during skill shot, the 5-bank targets continue to cycle even as other playfield switches are being hit. The database award could even be collected while the 5-bank targets continue to cycle due to the missed 5-bank at skill shot.

Eventually, the 5-bank targets stop cycling, yet the conditions, at first, appear to be random under what condition the targets stop cycling. At first it seemed as if this could be an unintentional and random way in which the game behaves since it seemed reasonable that the first non-5-bank target should immediately stop the 5-bank cycling targets.

What was discovered is that the current game behavior is intentional and specifically designed to behave this way. What is actually happening is that the game simply tracks the playfield switches that are hit after the ball is launched and as soon as 3 unique playfield switches have been hit, the code necessarily ends the 5-bank skill shot target cycling. It happens to be that some shots such as left-loop and left ramp/inlane shots cause 3 unique switches to close (which ends the 5-bank cycling) while other shots such as drop-target and then left lock, only consist of 2 unique switches which is why the 5-bank targets continue to cycle even as the database is being awarded. As soon as a 3rd playfield switch is hit, the 5-bank cycling will explicitly stop.

These findings reveal the game is behaving in the way intended and, as such, no code changes for L8.4 are done to change this behavior.

Attract Mode Improvements, L8.4

For L8.4 some improvements to the attract mode are added, with the following goals:

- Fix issue where start-button w/zero credits makes sound even if attract sounds are disabled.
- Add support for start-button sound when zero credits, allow sound to play up to 3 times.
- Add support for gun-trigger sound during attract mode.
- Add support for Tournament Mode display of previous game scores when gun-trigger is pulled.

The desire is to have a little bit of added user-interaction during attract mode to add a little fun for those who try to interact with the game during attract mode. The Indiana Jones game is appropriate game to follow since it has a gun trigger similar to Terminator 2. For L8.4 the changes in attract mode can be selectable by selecting L8.4 from the "Attract Mode" adjustment. *The bug-fix for the start-button sound takes effect regardless of the "Attract Mode" adjustment since it represents a bug-fix and not an added feature for L8.4. It may be debatable if this is truly a bug however during L8.4 development the decision was that "Attract Sounds" being set to "Off" should mean that no sounds are emitted unless a game is in play.*

The table below shows some of the differences in behavior in existing L-8 code as compared to IJ L-7:

Activity	**T2 L-8/L8.3**	**Indiana Jones L-7**
Start Button Press during Attract Mode when game is reporting zero credits		
Number of times allowed	Single sound	Single sound
When can initially play sounds	At game-over attract mode start	At every attract mode start
When can play sounds again after silent period	When main attract mode loop cycles twice, when "I am the future" plays	After 4m41s has passed since previous sound
Gun Trigger Press during Attract Mode		
Number of times allowed	No gun trigger sound	Three sounds
When can initially play sounds	n/a	At every attract mode start
Reinitialized, can play sounds again	n/a	After 2m48s has passed since previous 3 sounds had played
Flipper Button Left/Right during Attract Mode		
Number of times allowed	No flipper button sounds	Three sounds
When can initially play sounds	n/a	At every attract mode start
When can play sounds again	n/a	After 4m41s has passed since previous 3 sounds had played

Since Terminator 2 uses different flipper circuitry than Indiana Jones, it is not possible for T2 to detect flipper button presses without actually allowing the flippers to be powered. The T2 flipper end-of-stroke switch is what actually feeds into the CPU board as the flipper button left or right switch closures.

Because of this, the idea for L8.4 is to give a little more fun to the attract mode by combining some of what was done in Indiana Jones with T-2, resulting in the following goals for L8.4 when the game adjustment for "Attract Mode" is set to "L8.4":

Activity	**T2 L-8/L8.3**	**T2 L8.4**

Start Button Press during Attract Mode when game is reporting zero credits		
Number of times allowed	Single sound	Three sounds
When can initially play sounds	At game-over attract mode start	At every attract mode start
When can play sounds again	When main attract mode loop cycles twice, when "I am the future" plays	After 4m41s has passed since previous 3 sounds had played
Gun Trigger Press during Attract Mode		
Number of times allowed	No gun trigger sound	Three sounds
When can initially play sounds	n/a	At every attract mode start
When can play sounds again	n/a	After 2m48s has passed since previous 3 sounds had played

In addition to these changes for L8.4, there is added improvement that will be pulled in from IJ L-7. The IJ code, when flipper button is pressed in attract mode while "Tournament Mode" is enabled, will show on the display the scores from the most recently played game. After a short while, the attract mode will resume from the top of the current loop (main loop or game-over loop). For L8.4, since the flipper buttons cannot be used in this way, the gun-trigger can be used in this way to allow previously played scores to be shown when the gun trigger is pulled during attract mode when Tournament Mode is enabled.

After a careful survey of IJ and T2 code, the following is a list of the needed work for L8.4:

- Add "L8.4" as a possible selection for the "Attract Mode" setting.
- Work and code changes affecting both start-button and gun-trigger:
    - Identify bytes in RAM that are suitable for L8.4 start-button and trigger switch logic.
    - Identify 2 function ID values that can be used for start/trigger reset timer functions.
    - Put in code to initialize the RAM counter bytes at every attract mode start.
- Code changes specific to start-button handling
    - Put in code to relocate the existing start-button sound to different bank with more room.
    - Enhance the start-button sound code to check attract-mode for L8.4
    - Have the start-button handler, in L8.4 mode, play up to 3 sounds & reset timer code.
    - Fix bug for all cases so start-button only makes sound when Attract Sounds are enabled.
- Code changes specific to gun-trigger handling
    - Gun-trigger switch table metadata so its callback is used even at game-over.
    - Gun-trigger callback function checks for attract mode.
    - Gun-trigger callback function during attract mode, check attract-mode for L8.4.
    - Gun-trigger callback function L8.4 mode, play up to 3 sounds and reset timer code.
    - Gun-trigger callback function in Tournament mode, shows previously played scores.

Attract Mode Improvement: Adding "L8.4" Attract Mode Setting

The new L8.3 adjustment "Attract Mode" is updated in L8.4 to support a new setting for "L8.4" attract mode. This requires a change to the adjustment metadata to increase the maximum value by 1. Out of completeness, the factory setting will also be changed to the new "L8.4" value. This change in factory setting only takes effect when the game resets its settings back to factory default.

The L8.4 updated row of adjustment metadata is highlighted below, allowing maximum value 03. This row of table data starts at $7117,3D, ROM offset 0x77117.

```
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
7117: 00 03 00 00 00 03 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
```

As shown in the metadata table entry, the handler for showing this adjustment in the menu system is at $657C,3D. This function was previously described in the L8.3 code changes and now must undergo some changes to accommodate the new menu selection. The first change is to simply move the strings that it cites to a region later in the ROM image so that the function has room for a new check for L8.4 setting. The change of addresses is highlighted below, starting address of old strings was at $65AB,3D, and they are moved to a region of unused bytes starting at $66E0,3D, ROM offset 0x766E0.

```
--------------------------------;--------------------------------------------------
                                ;
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"
66E0: 4D 45 4E 55 20 45 52 52 4F 52 00  ; "MENU ERROR"
66EB: 4C 38 2E 31 00            ; "L8.1"
66F0: 4C 38 2E 32 00            ; "L8.2"
66F5: 4C 38 2E 33 00            ; "L8.3"
66FA: 4C 38 2E 34 00            ; "L8.4"
                                ;
--------------------------------;--------------------------------------------------
```

With the moved strings, the updated menu handler code is shown below with modifications from its L8.3 version highlighted. Shown below is the portion of the entire menu handling code that requires modification. Refer to the new attract mode setting described in the L8.3 document for a more thorough description of the "Attract Mode" menu handler function. The portion shown below starts at $658A,3D, ROM offset 0x7658A.

```
--------------------------------;--------------------------------------------------
                                ;
                                ;
658A: 8E 65 AB      LDX   #$65AB ; Load default/error string
658A: 8E 66 E0      LDX   #$66E0 ; 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 ;
6593: 8E 66 EB      LDX   #$66EB ;
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 ;~~
659D: 8E 66 F0     LDX   #$66F0 ;
65A0: 39           RTS          ;
65A1: 11 83 00 02  CMPU  #$0002 ; Check if at "L8.3"
~~65A5: 26 03        BNE   $65AA  ;~~
65A5: 26 04        BNE   $65AB  ;
~~65A7: 8E 65 C0     LDX   #$65C0 ;~~
65A7: 8E 66 F5     LDX   #$66F5 ;
65AA: 39           RTS          ;
65AB: 11 83 00 03  CMPU  #$0003 ; Check if at "L8.4"
65AF: 26 03        BNE   $65B4  ;
65B1: 8E 66 FA     LDX   #$66FA ;
65B4: 39           RTS          ;
                                ;
--------------------------------;-------------------------------------------------
```

With the above changes in place, code may now read the “Attract Mode” setting value to determine next course of action. A survey of the L8.3 code changes regarding attract mode reveals that the L8.3 code changes that performs certain attract mode sequences based on “Attract Mode” setting do so in such a way that any checks for “L8.3” will automatically include “L8.4” due to the way in which the code checks for values greater or less-than instead of checking or absolute values of the adjustment. This means there are no corrections needed to the existing L8.3 code related to the “Attract Mode” setting.

Attract Mode Improvement: Overall Changes for Start-Button and Gun-Trigger

This section covers work items applicable to both the start-button and the gun-trigger enhancements to the L8.4 attract mode. This covers (a) identifying bytes in RAM to use for tracking number of start-button and gun-trigger switches, (b) identifying 2 function ID values to use for start-button and gun-trigger timers, and (c) adding code to ensure the RAM bytes are reset at the appropriate times so attract mode will allow them to count up to 3 switch closures before starting its respective quiet timer.

Attract Mode Improvement: Overall Changes: Identifying available RAM Bytes

For L8.4 attract mode changes, two bytes of RAM are needed to keep track of the number of start-button and gun-trigger switch closures. By counting the number of closures, then code can know when 3 closures have occurred and can then stop playing sounds for a while (based on a timer that gets set). Additionally, two more bytes of RAM are needed to store the previously played sound effect identifier to prevent the same sound from playing two times in a row.

An additional 1 byte of RAM is needed to store whether the L8.4 attract mode enhancement is handling the start-button or the gun-trigger. This ended up being needed in order to allow the single attract-mode interrupt function to be used for either start-button or gun-trigger. This allows the attract-mode to be interrupted by either the normal start-button “Insert coin” message or the L8.4 gun-trigger display of previous game scores (when tournament mode is enabled).

With L8.4 having recently added code in Video Mode (for Tournament Mode fixups) some relatively “safe” RAM bytes are already apparent. There are a handful of RAM bytes used by video mode for purpose of tracking the video mode progress. A scan of the ROM of other locations where such RAM bytes reveals they are not used elsewhere for any other purpose. In fact, after the video mode, the

bytes are left in RAM, unchanged, until the next video mode starts up. They remain in their most-recently used state even into the game-over and attract mode.

The table below captures identified Video Mode RAM bytes that may safely be used for this purpose:

RAM Byte	Video Mode Usage	Number of references in ROM	L8.4 Attract Mode
$0643	Tallied hits on player	5 references, bank $2B, all used in Video Mode	Stores number of start-button sounds played so far.
$0644	Number of terminator kills	7 references, bank $2B, all used in Video Mode	Stores number of gun-trigger sounds played so far.
$0645	Current of hits on player	3 references, bank $2B, all used in Video Mode	Stores last played start-button sound index.
$0646	Max terminator kills	3 references, bank $2B, all used in Video Mode	Stores last played gun-trigger sound index.
$0647	Kills until Hunter ship appears	3 references, bank $2B, all used in video Mode	Indicates if start-button or gun-trigger is being handled.

Reference count is only counting where the RAM byte is referenced in an opcode. Other occurrences of the 2-byte RAM byte addresses in ROM are not in relation to accessing these RAM bytes.

The L8.4 can neatly use these RAM bytes during attract mode for purpose of counting start-button and gun-trigger switch closures and during game play these RAM bytes can simply be used in the way currently are used without any code changes to accommodate their extra use in L8.4 attract mode.

As a side note, it was noticed that the RAM bytes that IJ L-7 uses to count attract mode flipper-button and gun-trigger switch closures are also used during regular game-play (such as in mode for crossing the rope bridge). This method of using video mode RAM bytes in L8.4 is consistent with IJ L-7.

Rows 3 and 4 in the table above identify RAM bytes that L8.4 can used to track the previously played sound so that the same sound doesn't play two times in a row. This is also consistent with what IJ L-7 does for flipper-button sounds during attract mode.

Attract Mode Improvement: Overall Changes: Identifying 2 Function ID values

As suggested, the L8.4 code will mimic what was observed in Indiana Jones L-7 code whereby the attract mode will count the button hits and when their hit limit is reached, then a timer is set during which subsequent button closures will not cause sound to play.

As described in L8.3 documentation and in earlier parts of L8.4 documentation, the WPC code will assign a scheduled function with a 16-bit identifier value. Typically this is important during game play so that code knows if a particular feature is active by checking if the given 16-bit ID value is currently in the scheduler. Because of this it is important to choose the ID value carefully so as not to cause inadvertent game logic when existing code checks for running ID values. During attract mode, the likelihood of trouble is less of a concern since there is no game play taking place however the ID values still need to be considered carefully to avoid any unintended behaviors.

It should be noted that in IJ L-7 upon game-play startup, the active running timers are cancelled as part of game startup procedures. A sweeping cancellation of active running functions takes place. The sweeping cancellation includes any of the 3 possible attract mode timer functions that might be active at the moment of game start. For reference, the table below summarizes the function ID values used in IJ L-7 for attract mode timers.

IJ L-7 Attract Mode Feature	Used in ROM Bank	Function ID	Timer Period
Start-Button Sound	Bank $30	Function ID: 00 04	0x4650
Flipper-Button Sounds	Bank $36	Function ID: 00 F0	0x4650
Gun Trigger Sounds	Bank $36	Function ID: 00 E9	0x2A30

For those curious about the cancellation of startup timers at game start, below is the IJ L-7 function that starts the cancellation of all scheduled functions. This is located in IJ L-7 at $4B4A,38, ROM offset 0x60B4A.

```
---------------------------------------;-------------------------------------------
                                       ; IJ L-7
                                       ; GameStartupCancelCallbacks()
                                       ;
4B4A: 34 26       PSHS  Y,B,A          ;
4B4C: 10 8E BF FF LDY   #$BFFF         ;
4B50: CC 40 00    LDD   #$4000         ;
4B53: BD 9A A8    JSR   $9AA8          ; CancelAllCallbacksIdYMaskD
4B56: 35 A6       PULS  A,B,Y,PC       ;
                                       ;
---------------------------------------;-------------------------------------------
```

The function gets called through a call chain of other initialization code that takes place when a game is starting up.

A survey of T2 L-8 code reveals it has the same sweeping cancellation of timer functions. This means no special code is needed other than starting up the timer functions in a similar manner as IJ code. The corresponding function in T2 L-8 code is located at $4B7F,38, ROM offset 0x60B7F as shown below:

```
---------------------------------------;-------------------------------------------------
                                       ; T2 L-8
                                       ; GameStartupCancelCallbacks()
                                       ;
4B7F: 34 26       PSHS  Y,B,A          ;
4B81: 10 8E BF FF LDY   #$BFFF         ;
4B85: CC 40 00    LDD   #$4000         ;
4B88: BD 9A FD    JSR   $9AFD          ; CancelAllCallbacksIdYMaskD()
4B8B: 35 A6       PULS  A,B,Y,PC       ;
                                       ;
---------------------------------------;-------------------------------------------------
```

As mentioned earlier in this document, the ID/Mask are used in the following way. All scheduled callback functions are examined, and for each existing scheduled function ID:

IF <existing ID> AND <yyyy> EQUALS <xxxx> AND <yyyy>, then function is cancelled.

Readers are encouraged to experiment with emulator and set breakpoints to see how this code behaves, being called at game startup, and to see how it ultimately cancels functions. After this function has completed, the timer functions are no longer running.

For T2 L8.4 a survey of ROM code of the usual byte patterns where function IDs are specified is done to see if the IJ L-7 function IDs can be carried into L8.4. Initial checking for IDs 00E9 and 00F0 indicate they are already used in T2 L-8. Other nearby ID values were checked until two were found to be unused by existing code, 00F2 and 00F3. Check of all places where IDs are cited shows they are unused in T2 L-8 and should be able to be used in L8.4 safely. Results are shown below.

Function Name	L-8 Function Usage Signature	Description	Search Result
ScheduleFunctionStart()	BD 8B 77 xx xx yy yy yy	Schedules function ID xx xx to start at WPC Addr yy yy yy	No occurrence of: BD 8B 77 00 F2 or BD 8B 77 00 F3
SearchLinkedListForId()	BD 86 90 xx xx	Searches for scheduled function ID xx xx	No occurrence of: BD 86 90 00 F2 or BD 86 90 00 F3
CancelScheduled CallbackFunction()	BD 86 9E xx xx	Cancels scheduled function ID xx xx	No occurrence of: BD 86 9E 00 F2 or BD 86 9E 00 F3
UpdateCurrentRunningSchedule FunctionIDParameterBytes()	BD 86 AC xx xx	Sets currently running function ID to xx xx	No occurrence of: BD 86 AC 00 F2 or BD 86 AC 00 F3
TBD()	BD 86 BA xx xx	TBD, where xx xx is function ID	No occurrence of: BD 86 BA 00 F2 or BD 86 BA 00 F3
CancelScheduledCallbackID ParameterBytes()	BD 86 D0 xx xx	Cancels scheduled ID xx xx	No occurrence of: BD 86 D0 00 F2 or BD 86 D0 00 F3
CancelAllCallbacksIdMask ParameterBytes()	BD 8A 9A xx xx yy yy	Cancels scheduled functions matching ID pattern of xx xx bitwise-and yy yy	No occurrence of: BD 8A 9A 00 Fx,
SearchLinkedListAndMask ParameterBytes()	BD 8A AA xx xx yy yy	Searches for schedule functions matching ID pattern of xx xx bitwise-and yy yy	No occurrence of: BD 8A AA 00 Fx
AddLinkedListEntry()	BD 8B 3D xx xx yy yy yy	Adds function yy yy yy to linked list as ID xx xx	No occurrence of: BD 8B 3D 00 F2 or BD 8B 3D 00 F3
ScheduleFunctionStart()	BD 8B 77 xx xx yy yy yy	Schedules function yy yy yy ID xx xx	No occurrence of: BD 8B 77 00 F2 or BD 8B 77 00 F3
TBD()	BD 8B 9D xx xx yy yy yy	TBD, where xx xx is ID and yy yy yy is addr.	No occurrence of: BD 8B 9D 00 F2 or BD 8B 9D 00 F3
ScheduleFunctionCallback()	BD 8B C3 xx xx yy yy yy	Schedules function yy yy yy ID xx xx	No occurrence of: BD 8B C3 00 F2 or BD 8B C3 00 F3
TBD()	BD 8B F7 xx xx yy yy yy	Schedules function yy yy yy ID xx xx	No occurrence of: BD 8B F7 00 F2 or BD 8B F7 00 F3

With the above information 5 RAM bytes and 2 function ID values for L8.4 attract mode are now identified and can be used in code shown next.

Attract Mode Improvement: Overall Changes: Initialize the RAM bytes at attract mode start

The final set of L8.4 attract mode work that is applicable to both start-button and gun-trigger involves inserting new code that ensures the RAM bytes that count number of switch closures are cleared to 0x00 so that L8.4 attract mode can then increase these counters to track number of start-button presses and gun-trigger pulls.

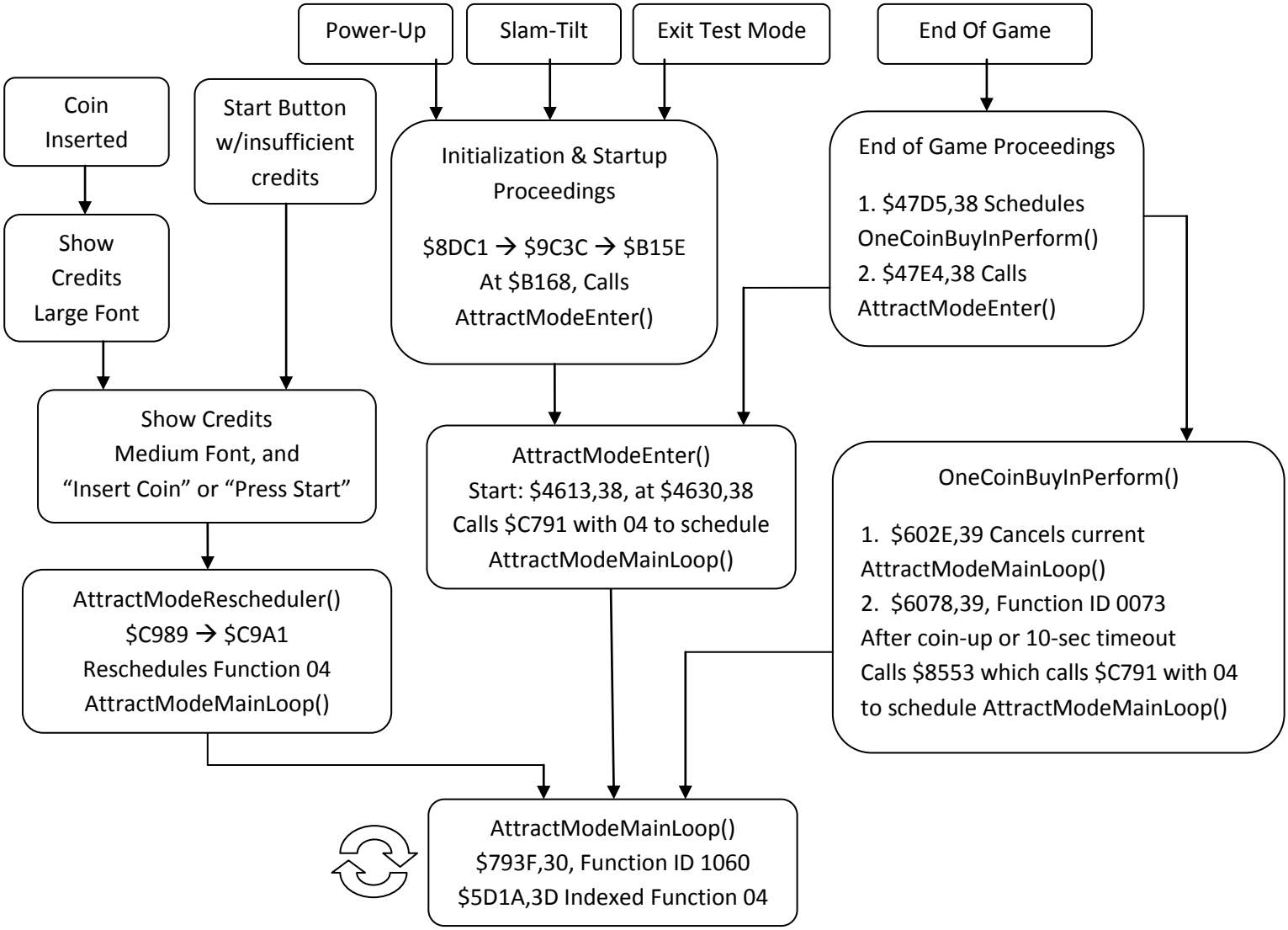
Initially the L8.4 was going to match IJ L-7 and reset the counters at attract mode start, however it was discovered this method would not allow the start-button count to accumulate. This is because the start-button (when zero credits) will show "Insert Coins" message for a short period and then restart attract mode. The restart of attract mode then (in IJ L-7) resets the switch counters. *Astute readers will see that on IJ L-7 the start-button (when zero credits) results in the counters being reset so they can play two flipper-button sounds and get two gun-trigger clicks and then use the start-button to reset the counts, then get two more flipper-button and gun-trigger sounds, repeat, without incurring the timeout period where no sounds are allowed from flipper-button and gun-trigger.*

For reference, the IJ L-7 code that resets the two counters (for start-button and gun-trigger) is shown below. In IJ L-7 this is done at $6D4D,36, as shown below:

```
6D4D: BD 83 99                        ; Sleep(0x01)
6D50: 01                              ; Parameter 0x01
6D51: 7F 0A AB    CLR   $0AAB         ; $0AAB gets 0x00, clears attract-mode flipper-button
                                      ;  sounds counter at attract mode start
6D54: 7F 0A A4    CLR   $0AA4         ; $0AA4 gets 0x00, clears attract-mode gun-trigger
                                      ;  sounds counter at attract mode start
6D57: BD 67 6A    JSR   $676A         ; InitDMDAndCycleC7andClockItIn()
6D5A: 7E 6D 5D    JMP   $6D5D         ; <nop>
```

For T2 L8.4 the clearing of the RAM must be done *prior* to the invocation of the attract-mode so that the start-button and its subsequent restart of attract-mode does not cause the counters to get reset.

In order to ensure the L8.4 code update for clearing the start-button and gun-trigger RAM values is done correctly, the L-8 code was analyzed for full understanding in all ways in which the attract mode gets started. The flowchart, below, depicts the various paths that can start attract mode.



As shown in the attract mode logic diagram above, the AttractModeMainLoop() can be started from various sources. The one-coin buy-in option introduces an interesting behavior where the game-over code schedules the OneCoinBuyInPerform() function and then calls AttractModeEnter() which schedules the AttractModeMainLoop() however the OneCoinBuyInPerform() function then proceeds to cancel the AttractModeMainLoop() function before presenting the player with the option to buy a credit for a coin. Once the player has inserted a coin or the 10 second timeout expires, then the OneCoinBuyInPerform() will then schedule the AttractModeMainLoop() itself. This is awkward design in that the attract mode is started then cancelled and restarted. To get one-coin buy-in, it must be enabled in Pricing Adjustments along with the pricing type specifically set to 2/$1.00 (and free-play is not enabled). When the adjustments are set this way, the game offers the player the opportunity to get 1 credit for a single coin

at the end of a game. The one credit for one coin occurs only during the 10 second buy-in period and may only be done for as many players that were in the game that had just ended.

As previously stated, the start-button handler results in the restart of the attract mode loop and, therefore, the L8.4 cannot reset the button-press counts at the start of AttractModeMainLoop() which is what IJ L-7 does for flipper-button and gun-trigger sound counts. For L8.4 the memory clearing needs to be done prior to the AttractModeMainLoop() only in the cases where the attract mode is genuinely starting and not in the case of the restart of attract mode due to start-button during zero credits.

An analysis of the L-8 code and referring to the logic chart above, the AttractModeEnter() function is the place where L8.4 can safely reset the RAM counters for start-button and gun-trigger pulls. This function is common to all places where attract mode starts where subsequent start-button and gun-trigger sounds can, from then on, be counted. As shown in the logic chart, subsequent restarts of attract mode due to start-button (with zero credits) will not cause the counts to reset since AttractModeEnter() is not involved at such attract mode restarts. In the case of one-coin buy-in, the game-over code still calls AttractModeEnter() even though the subsequent one-coin buy-in will cancel the AttractModeMainLoop() that AttractModeEnter() had tried to launch. In this case the AttractModeEnter() is still an appropriate place to clear the RAM for start-button and gun-trigger switch counts when one-coin buy-in is in use.

Shown below is the partially annotated AttractModeEnter() function. This is at $4613,38, ROM offset 0x60613.

```
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; AttractModeEnter()
                                       ;
4613: 34 16       PSHS  X,B,A          ;
4615: 86 01       LDA   #$01           ;
4617: 97 87       STA   $87            ; $87 is non-zero when in attract mode or test mode
4619: BD 88 F5    JSR   $88F5          ; CallBankedFunction_Param_WPCAddr()
461C: 53 5F 39                         ;
461F: BD A8 4A    JSR   $A84A          ;
4622: BD AB 98    JSR   $AB98          ; FlippersRelayDisable()
4625: BD 83 DB    JSR   $83DB          ;
4628: 1F                               ;
4629: B6 17 93    LDA   $1793          ; test mode indicator (non-zero in test mode)
462C: 26 20       BNE   $464E          ; If $1793 was non-zero then skip to the end
                                       ; This is where attract mode is launched at power-up and
                                       ; when leaving menu mode
462E: 86 04       LDA   #$04           ; Index 0x04 into $5D1A,3D tbl == AttractMode() $793F,30
4630: BD C7 91    JSR   $C791          ; This function saves the 0x04 into $0487 which is how
                                       ; main code knows the function ID when re-scheduling
                                       ; attract mode after "insert coin" w/zero credits
4633: BD 4B 6D    JSR   $4B6D          ;
4636: BD A2 52    JSR   $A252          ;
4639: 8E 4B 8D    LDX   #$4B8D         ;
463C: C6 38       LDB   #$38           ;
463E: BD C7 22    JSR   $C722          ;
4641: BD 86 21    JSR   $8621          ; CallFunctionPointerParameterBytes()
4644: 80 2B                            ; $802B has $5EDD,3B
4646: BD 4D 3A    JSR   $4D3A          ;
4649: BD 86 21    JSR   $8621          ; CallFunctionPointerParameterBytes()
464C: 80 D0                            ; $80D0 has $6585,3B
```

```
464E: 35 96       PULS  A,B,X,PC         ;
                                         ;
-----------------------------------------;-------------------------------------------------
```

The function, above, is modified so it will jump to a new L8.4 function located at the end of the bank $38 which performs the instructions that were replaced with the jump instruction and then clear the RAM for L8.4 attract mode. The updated portion of the code is as follows:

```
4629: B6 17 93    LDA   $1793           ; test mode indicator (non-zero in test mode)
462C: 26 20       BNE   $464E           ; If $1793 was non-zero then skip to the end
                                        ; This is where attract mode is launched at power-up and
                                        ; when leaving menu mode
462E: 86 04       LDA   #$04            ; Index 0x04 into $5D1A,3D tbl == AttractMode() $793F,30
4630: BD C7 91    JSR   $C791           ; This function saves the 0x04 into $0487 which is how
                                        ; main code knows the function ID when re-scheduling
                                        ; attract mode after "insert coin" w/zero credits
462E: BD 7F EC    JSR   $7FEC           ; Call new L8.4 attract-mode start hook
4631: 20 00       BRA   $4633           ; Dummy instructions to fill in the bytes
4633: BD 4B 6D    JSR   $4B6D           ;
4636: BD A2 52    JSR   $A252           ;
```

At the end of bank $38 is the new L8.4 function, as shown below. This is located at $7FEC,38, ROM offset 0x63FEC.

```
-----------------------------------------;-------------------------------------------------
                                         ;
                                         ;
7FEC: 7F 06 43    CLR   $0643           ; For L8.4, Clear RAM byte for start-button closures
7FEF: 7F 06 44    CLR   $0644           ; For L8.4, Clear RAM byte for gun-trigger closures
7FF2: 86 FF       LDA   #$FF            ;
7FF4: B7 06 45    STA   $0645           ; Push 0xFF into last played sound index, start-button
7FF7: B7 06 46    STA   $0646           ; Push 0xFF into last played sound index, gun-trigger
                                        ;
7FFA: 86 04       LDA   #$04            ; Index 0x04 into $5D1A,3D tbl == AttractMode() $793F,30
7FFC: BD C7 91    JSR   $C791           ; This function saves the 0x04 into $0487 which is how
                                        ; main code knows the function ID when re-scheduling
                                        ; attract mode after "insert coin" w/zero credits
7FFF: 39          RTS                   ;
                                         ;
-----------------------------------------;-------------------------------------------------
```

The new L8.4 memory clearing ensures the start-button and gun-trigger counts are reset to zero and also the saved index from the previously played sound is set to 0xFF which is not a valid sound index so the first played sound is not inadvertently determined to have been played most recently.

Attract Mode Improvement: Start-Button Handling

In this section the new L8.4 changes for attract-mode start-button handling are described. This refers to the following items:

- Put in code to relocate the existing start-button sound to different bank with more room.
- Enhance the start-button sound code to check attract-mode for L8.4
- Have the start-button sound code, in L8.4 mode, play up to 3 sounds & reset timer code.
- Fix bug for all cases so start-button only makes sound when Attract Sounds are enabled.

First, the start-button handling code where sound is played needs to be relocated to where there is more room in ROM space to implement the L8.4 function when Attract Mode is set to L8.4. To understand how this gets relocated, the entire start-button switch handler is shown below, partially annotated with relevant section highlighted. This code is at $4540,38, ROM offset 0x60540.

```
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; SwitchMatrixHdlr_StartButton()
                                        ;
4540: BD 86 D0    JSR   $86D0           ; CancelScheduledCallbackIDParameterBytes()
4543: 78 7F                             ;
4545: BD 4D 4C    JSR   $4D4C           ;
4548: 7D 03 83    TST   $0383           ; In menu-system indicator
454B: 10 26 00 9B LBNE  $45EA           ; If $0383 is is non-zero then we're done, do nothing
                                        ;
454F: BD 48 D1    JSR   $48D1           ;
4552: 10 24 00 94 LBCC  $45EA           ; C-clear? do nothing
                                        ;
4556: BD 83 19    JSR   $8319           ; GetSwitchClosedParameterByte()  // C-clr = sw closed
4559: 11                                ; 0x11 == Start Button
455A: 10 24 00 8C LBCC  $45EA           ; C-clear? switch is closed, do nothing
                                        ;
455E: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
4561: 5C 0A 39                          ; Checks for 1 or more credits, C-clr = 1 or more cred.
4564: 24 12       BCC   $4578           ; C-clr means theres 1 or more credits, C-set means zero
                                        ;
                                        ; Code only when no credits:
                                        ; {
4566: BD 86 21    JSR   $8621           ;    CallFunctionPointerParameterBytes()
4569: 80 52                             ;    $8052 == $5ECB,3B, debug RTS
456B: BD 86 90    JSR   $8690           ;    SearchLinkedListForId()  // c-bit clear = ID found
456E: 00 0C                             ;
4570: 24 78       BCC   $45EA           ;
4572: BD 85 53    JSR   $8553           ;    Calls function from table at $5D1A,3D
4575: 17                                ;    0x17 --> $7DDD,30, 0-credits sound handler
4576: 20 72       BRA   $45EA           ; }
                                        ;
4578: 96 87       LDA   $87             ; $87 = 0x00 when Game in progress
457A: 26 5E       BNE   $45DA           ; Game not in progress? Go to $45DA
                                        ;
457C: BD B1 D1    JSR   $B1D1           ; GameOnLastBallCheckCBitClearCurrentBallInA()
457F: 81 01       CMPA  #$01            ; Ball 1?
4581: 27 2E       BEQ   $45B1           ;
4583: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;    C-bit set when not-equal
4586: A1 00                             ; StandardAdjustment033, Game Re-start,  Adj=0xA1
4588: 24 60       BCC   $45EA           ;
458A: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
                                        ;    C-bit set when not-equal
458D: A1 02                             ; StandardAdjustment033, Game Re-start,  Adj=0xA1
458F: 24 13       BCC   $45A4           ;
                                        ;
4591: 86 1E       LDA   #$1E            ; This many sleeps will be 0.46875 seconds
4593: BD 83 46    JSR   $8346           ;--\ Sleep()
4596: 01                                ;  | 0.015625 seconds
                                        ;  |
4597: BD 83 19    JSR   $8319           ;  | GetSwitchClosedParameterByte() // C-clr = sw closed
459A: 0B                                ;  | 0x0B == Start Button
459B: 25 4D       BCS   $45EA           ;  | Switch is no longer pushed, do nothing
459D: 4A          DECA                  ;  |
```

```
459E: 26 F3       BNE   $4593           ;--/
                                        ; Switch was held down for entire 0.46875 seconds,
                                        ; proceed to add player
                                        ;
45A0: 96 87       LDA   $87             ; $87 has 0x00 when Game in progress
45A2: 26 46       BNE   $45EA           ; Game no longer in progress? done
                                        ;
45A4: 86 02       LDA   #$02            ;
45A6: BD 4A 8D    JSR   $4A8D           ; EndOfGameBookkeepingUpdates()
45A9: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
45AC: 62 84 39                          ;
45AF: 20 37       BRA   $45E8           ;
                                        ;
45B1: BD B1 9F    JSR   $B19F           ; GetCurrentGameNumberOfPlayersIntoA()
45B4: 81 04       CMPA  #$04            ;
45B6: 24 32       BCC   $45EA           ;
45B8: BD 88 F5    JSR   $88F5           ;
45BB: 5C 1F 39                          ;
45BE: 1C FE       ANDCC #$00FE          ;
45C0: BD 86 21    JSR   $8621           ;
45C3: 80 1C       SUBA  #$1C            ;
45C5: 25 23       BCS   $45EA           ;
45C7: 4C          INCA                  ;
45C8: BD B1 9B    JSR   $B19B           ;
45CB: BD 88 F5    JSR   $88F5           ;
45CE: 62 56 39                          ;
45D1: 8D 2C       BSR   $45FF           ;
45D3: BD CA E2    JSR   $CAE2           ;
45D6: 8D 30       BSR   $4608           ;
45D8: 20 10       BRA   $45EA           ;
                                        ;
45DA: BD BC 5F    JSR   $BC5F           ;
45DD: 25 0B       BCS   $45EA           ;
45DF: 1C FE       ANDCC #$00FE          ; C-clear
45E1: BD 86 21    JSR   $8621           ; CallFunctionPointerParameterBytes()
45E4: 80 55                             ; $8055 == $61A9,3B, debug RTS
45E6: 25 02       BCS   $45EA           ;
45E8: 8D 03       BSR   $45ED           ;
45EA: 7E 99 A2    JMP   $99A2           ;
                                        ;
----------------------------------------;------------------------------------------------
```

For L8.4, it was found that the callback function Index 0x17 needs to be updated so it can be called from start-button or from gun-trigger alike. This is needed as a safe method of allowing the gun trigger to get the attract-mode to interrupt with the display of previously played game scores (when Tournament mode is enabled). In order to support this, the portion of start-button handler shown above is modified as shown below. This code begins at $4566,38, ROM offset 0x60566.

```
                                        ; Code only when no credits:
                                        ; {
4566: BD 86 21    JSR   $8621           ;    CallFunctionPointerParameterBytes()
4569: 80 52                             ;    $8052 == $5ECB,3B, debug RTS
456B: BD 86 90    JSR   $8690           ;    SearchLinkedListForId()  // c-bit clear = ID found
456E: 00 0C                             ;
4570: 24 78       BCC   $45EA           ;
~~4572: BD 85 53    JSR   $8553~~       ;    ~~Calls function from table at $5D1A,3D~~
~~4575: 17~~                            ;    ~~0x17 --> $7DDD,30, 0-credits sound handler~~
4572: BD 7F 05    JSR   $7F05           ;    Calls L8.4 Function to play sound and call $8553
4575: 12          NOP                   ;     with 0x17 --> $7DDD,30, 0-credits sound handler
```

```
4576: 20 72       BRA   $45EA           ; }
```

This will cause the start-button handler, when there are zero credits, to first call the new L8.4 function that is shown below. This is located at $7F05,38, ROM offset 0x63F05. This function will play a L8.4 sound if appropriate to do so, and then call $7FE4,38 to proceed. Initial L8.4 design had the code at $4572,38 call directly to $7FE4,38. The $7F05,38 was inserted in later L8.4 revisions to allow the L8.4 start-button sound to play during the period of time in which the “Insert Coin” message is being displayed. This is why this code flows through $7F05,38 before getting to $7FE4,38.

```
---------------------------------------;-----------------------------------------------------
                                       ;
                                       ; The L8.4 attract mode allows up to 3 start-button
                                       ; sounds which get played here prior to scheduling the
                                       ; function that shows the "Insert Coin" message.
                                       ; This way, all 3 sounds can be triggered during the
                                       ; "Insert Coin" message, if button is repeatedly pressed.
                                       ;
7F05: BD 86 5B    JSR   $865B          ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
7F08: 10 00                            ; 0x10 == Attract Sounds, C-set when not equal to 0x00
7F0A: 24 0A       BCC   $7F16          ; If no attract sounds, branch to show CreditsInsertCOin
                                       ;
7F0C: BD 82 FF    JSR   $82FF          ; LoadAWithTableIndexByteParameter()
7F0F: 17                               ; 0x17, $1BE7:$1BE8 FeatureAdjustment023, Attract Mode
7F10: 81 03       CMPA  #$03           ; Adjustment Value = 0, L8.1     C-bit set
                                       ;                    1, L8.2     C-bit set
                                       ;                    2, L8.3     C-bit set
                                       ;                    3, L8.4     C-bit clear
7F12: 25 02       BCS   $7F16          ; If c-set then not L8.4
7F14: 8D 03       BSR   $7F19          ; Call the L8.4 start-button handler
7F16: 7E 7F E4    JMP   $7FE4          ; Go schedule the remainder of the start-button work
                                       ;
---------------------------------------;-----------------------------------------------------
                                       ;
                                       ;
                                       ; Perform the L8.4 start-button sound code
                                       ;
7F19: 34 12       PSHS  X,A            ;
7F1B: BD 86 90    JSR   $8690          ; SearchLinkedListForId() // c-clear means ID is found
7F1E: 00 F2                            ; Parameter 0x00F2
7F20: 24 36       BCC   $7F58          ; If timer is running, no sounds, skip to the end
                                       ;
7F22: 8E 7F 65    LDX   #$7F65         ; X=0x7F65, start of sound numbers for start-button
                                       ;
7F25: 86 0A       LDA   #$0A           ;--\ A gets 0x0A
7F27: BD A7 5B    JSR   $A75B          ;  | Get16BitPseudoRandomValueintoA()
7F2A: B1 06 45    CMPA  $0645          ;  | Is it same random 8-bit as last time?
7F2D: 27 F6       BEQ   $7F25          ;  | If so, keep trying for different random number
7F2F: 81 05       CMPA  #$05           ;  | Check if we have FUA sound index
7F31: 26 09       BNE   $7F3C          ;  | If not at FUA then use this sound
7F33: BD 86 5B    JSR   $865B          ;  | LookupGameAdjustmentParam1andCheckIfEqualsParam2()
7F36: 16 00                            ;  | FeatureAdjustment022, Profanity, Adj=0x16
7F38: 25 02       BCS   $7F3C          ;  | If C-set then it Profanity is ON, okay to play FUA
7F3A: 20 E9       BRA   $7F25          ;--/ FUA was picked but Profanity is OFF, try another
                                       ;
                                       ;
7F3C: B7 06 45    STA   $0645          ; Save random number A into $0645
                                       ;
7F3F: A6 86       LDA   A,X            ; A = StartButtonZeroCreditsSounds[A]
                                       ;
```

```
7F41: BD BD FB    JSR   $BDFB           ; DoSoundTableIndexA(), plays random sound effect
7F44: 25 12       BCS   $7F58           ; if Error then return without counting it
                                        ;
7F46: 7C 06 43    INC   $0643           ; $0643++
7F49: B6 06 43    LDA   $0643           ; A = $0643
7F4C: 81 03       CMPA  #$03            ; if (A >= 3)  // if played 3 or more sound effects
7F4E: 25 08       BCS   $7F58           ; {
7F50: BD 8B 77    JSR   $8B77           ;    ScheduleFunctionStart()
7F53: 00 F2                             ;    ID = 00F2
7F55: 7F 5A 38                          ;    StartButtonZeroCreditsSilenceTimer()
                                        ; }
7F58: 35 92       PULS  A,X,PC          ;
                                        ;
----------------------------------------;-------------------------------------------------
                                        ;
                                        ; StartButtonZeroCreditsSilenceTimer()
                                        ; Function ID == 00F2
                                        ;
7F5A: BD 86 79    JSR   $8679           ; SleepLong()
7F5D: 46 50                             ; 4m41.25 seconds
7F5F: 7F 06 43    CLR   $0643           ; Reset the start-button sound count back to zero
7F62: 7E 99 A2    JMP   $99A2           ; ThreadedFunctionDone()
                                        ;
----------------------------------------;-------------------------------------------------------
                                        ;
                                        ; StartButtonZeroCreditsSounds[]
                                        ;
7F65: A3                                ; "No way, Jose"
7F66: 83                                ; "Let's go"
7F67: 8B                                ; "Way to go"
7F68: 8D                                ; "Hasta la vista, baby"
7F69: 8E                                ; "Chill out"
7F6A: 92                                ; FUA
7F6B: 93                                ; "Woopdadidoo"
7F6C: 97                                ; "Time to go"
7F6D: 7E                                ; "Get out"
7F6E: 9B                                ; "Out of the way"
                                        ;
----------------------------------------;-------------------------------------------------------
```

The code, above, will allow the start-button to play a sound. After the 3rd such sound, a timer is set before another round of 3 sounds can be triggered. The duration of the timer is set to match that of IJ when flipper buttons experience a quiet period after triggering 3 different sound effects.

The code, above, then jumps to $7FE4,38 function, depicted below.

```
----------------------------------------;-------------------------------------------------------
                                        ;
7FE4: 7F 06 47    CLR   $0647           ; For L8.4, Clear $0647 = handling start-button
7FE7: BD 85 53    JSR   $8553           ; Calls function from table at $5D1A,3D
7FEA: 17                                ; 0x17 --> $7DDD,30, start-button zero-credits function
7FEB: 39          RTS                   ;
                                        ;
----------------------------------------;-------------------------------------------------------
```

The new function, above, for L8.4 will simply clear the previously identified RAM byte used in attract mode to inform the $8553 index function 0x17 that the start-button is being handled. Later, it will be

shown how this same RAM byte is made non-zero to allow the 0x17 function to know it is to handle a request from the gun-trigger being pulled during attract mode and during Tournament Mode.

The original code, and now the new code calls $8553 with parameter byte 0x17. This causes the code to call a lookup function and run function at index 0x17. The lookup table that $8553 is located starting at $5D1A,3D, ROM offset 0x75D1A, with entry 17 containing the following data:

```
5D90: 18 1C                             ; Index17
5D92: 7D DD 30                          ; Start-button zero-credits sound player
```

The first two bytes are metadata used by the $8553 function caller (details of how they are used have not been fully analyzed or documented). The last three bytes are the address of the function to be invoked. As shown, the function is $7DDD,30 which is at ROM offset 0x43DDD. The content of this function as it is in L-8 is shown below for reference.

```
----------------------------------------;------------------------------------------------
                                        ;
7DDD: BD 84 AD        JSR   $84AD       ; GetMemoryFlag() // C-bit clear when flag set
7DE0: DC                                ;  0xDC flag for main inner attract block played twice
7DE1: 25 08           BCS   $7DEB       ;
7DE3: BD 85 46        JSR   $8546       ; DoSoundTableParameterByte()
7DE6: 51                                ; 0x51 == Twangz
7DE7: BD 84 8F        JSR   $848F       ; ClearMemoryFlag()
7DEA: DC                                ;   0xDC flag for main inner attract block played twice
7DEB: BD 7B 74        JSR   $7B74       ; AttractMode_CreditsInsertCoin()
7DEE: 7E C9 52        JMP   $C952       ;
                                        ;
----------------------------------------;------------------------------------------------
```

For L8.4 the function at $7DDD,30 will be removed (bytes set to 0xFF) and the table data at index 0x17 will point to new function instead of pointing to $7DDD,30. Other L8.4 code has been added near the end of bank $3A, and this new code for start-button is placed there also. Note the entire L-8 ROM only contains a single call to $8553 with index 0x17 (the call shown above) so this change can safely be made. The $5D1A,3D table entry for 0x17 is changed to the following:

```
5D90: 18 1C                             ; Index17
~~5D92: 7D DD 30                          ; Start-button zero-credits sound player~~
5D92: 7F 48 3A                          ; Start-button zero-credits sound player, L8.4
```

The new start-button sound handler is located at $7F48,3A, ROM offset 0x6BF48 and is as follows.

```
----------------------------------------;------------------------------------------------
                                        ;
                                        ; L8.4 Start-Button Sound Handler & Gun-Trigger helper
                                        ;
7F48: 7D 06 47    TST   $0647           ; 00 == Start-button, else gun-trigger
7F4B: 26 29       BNE   $7F76           ; If non-zero do gun-trigger tournament mode scores
                                        ;
7F4D: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
7F50: 10 00                             ; 0x10 == Attract Sounds, C-set when not equal to 0x00
7F52: 24 0A       BCC   $7F5E           ; If no attract sounds, branch to show CreditsInsertCoin
                                        ;
7F54: BD 82 FF    JSR   $82FF           ; LoadAWithTableIndexByteParameter()
7F57: 17                                ; 0x17, $1BE7:$1BE8 FeatureAdjustment023, Attract Mode
```

```
7F58: 81 03       CMPA  #$03            ; Adjustment Value = 0, L8.1     C-bit set
                                        ;                    1, L8.2     C-bit set
                                        ;                    2, L8.3     C-bit set
                                        ;                    3, L8.4     C-bit clear
7F5A: 24 02       BCC   $7F5E           ; C-clr is L8.4, skip the pre L8.4 start-button sound
7F5C: 8D 09       BSR   $7F67           ; Call the pre L8.4 start-button handler
                                        ;
7F5E: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
7F61: 7B 74 30                          ; AttractMode_CreditsInsertCoin()
7F64: 7E C9 52    JMP   $C952           ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; Perform the original L-8 start-button sound code
                                        ;
7F67: BD 84 AD    JSR   $84AD           ; GetMemoryFlag() // C-bit clear when flag set
7F6A: DC                                ;  0xDC indicates main inner attract block played 2x
7F6B: 25 08       BCS   $7F79           ;
7F6D: BD 85 46    JSR   $8546           ; DoSoundTableParameterByte()
7F70: 51                                ; 0x51 == Twangz
7F71: BD 84 8F    JSR   $848F           ; ClearMemoryFlag()
7F74: DC                                ;   0xDC indicates main inner attract block played 2x
7F75: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------
                                        ;
                                        ; AttractMode_LastGameScores_Tournament()
                                        ;
                                        ; When this function is called for gun-trigger, logic
                                        ; has already determined that game is in tournament mode,
                                        ; the only work to do here is to show the last game
                                        ; scores and sleep for same period as IJ L-7 before
                                        ; being done.  This will then restart attract-mode.
                                        ;
7F76: BD FB AE    JSR   $FBAE           ; ClearDisplayMemory()
7F79: BD 88 F5    JSR   $88F5           ; CallBankedFunction_Param_WPCAddr()
7F7C: 6A F4 3B                          ; Put scores into pixel memory
7F7F: BD E2 74    JSR   $E274           ; Instant-on display
7F82: BD 86 79    JSR   $8679           ; SleepLong(0x04B0)
7F85: 04 B0                             ; 18.75 seconds
7F87: 7E C9 52    JMP   $C952           ;
                                        ;
---------------------------------------;-------------------------------------------------
```

Readers are encouraged to trace through the code above to see how the attract mode behaves for the start button when it has been pressed while there are zero credits. This code also has a preview on how the gun-trigger will end up showing last-game scores when tournament mode is set. Details on gun-trigger are next. Attentive readers may notice the original L-8 sound gets played in the scheduled 0x17 function immediately prior to showing “Insert Coin” message while the L8.4 plays up to 3 sounds in the start-button handler prior to scheduling the 0x17 function. This change is needed in L8.4 to ensure the start-button presses can play sounds even while the “Insert Coin” message is playing. Since the 0x17 function must run to completion, it is not allowed to be re-entered until it has finished its “Insert Coin” message and, as such, to get start-button to play sound during the display of “Insert Coin”, this design in L8.4 is needed.

Attract Mode Improvement: Gun-Trigger Handling

In this section the new L8.4 changes for attract-mode gun-trigger handling are described. This refers to the following items:

- Gun-trigger switch table metadata so its callback is used even at game-over.
- Gun-trigger callback function checks for attract mode.
- Gun-trigger callback function during attract mode, check attract-mode for L8.4.
- Gun-trigger callback function L8.4 mode, play up to 3 sounds and reset timer code.
- Gun-trigger callback function in Tournament mode, shows previously played scores.

First, the switch-table data needs updated so that the gun-trigger handler code gets called when the gun-trigger switch is closed during attract mode. As described in the L8.3 document, each switch is represented in the switch data table with an 11-byte structure containing various characteristics about how the switch is to be handled. Some of the data bytes contain flags that define how or when the switch-closure is to be processed. Specifically the 8th and 9th bytes of each switch table entry contains bit-mapped flags. The meaning of some of these flags have been determined and listed below. Shown below is a brief description of the full 11-bytes of data that exists for each playfield switch.

```
; ----------------------------------------------------------------
; 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  Callback Address in ROM where code is ran upon switch closure
; 06        ; This value is reset value for this switch's byte in idle-switch array at $174E.
; 07
; 08        ; This is a flag byte with the following bit-flag definitions for this switch
;           ; 11111111
;           ; ||||||||
;           ; |||||||\-- 0x01 bit.
;           ; ||||||\--- 0x02 bit.
;           ; |||||\---- 0x04 bit.
;           ; ||||\----- 0x08 bit.
;           ; |||\------ 0x10 bit.
;           ; ||\------- 0x20 bit.  When set, switch is NOT included 60-ball switch error report
;           ; |\-------- 0x40 bit.
;           ; \--------- 0x80 bit.
;           ;
;           ;
;
; 09        ; This is another flag byte
;           ; 11111111
;           ; ||||||||
;           ; |||||||\-- 0x01 bit.
;           ; ||||||\--- 0x02 bit.
;           ; |||||\---- 0x04 bit.  Set = switch is ignored if closed during bonus addup
;           ; ||||\----- 0x08 bit.
;           ; |||\------ 0x10 bit.  Set = switch processed if closed in attract-mode/test-mode
;           ; ||\------- 0x20 bit.  Set = switch processed if closed during TILT period
;           ; |\-------- 0x40 bit.  Set = switch is valid, its callback may be invoked
;           ; \--------- 0x80 bit.  Set = switch processed if closed during bonus addup
;           ;
;           ; The 0x04 bit is checked prior to the 0x80 bit.  Further research is needed.
```

```
;           ;
; 0A
; 0B
;
;-------------------------------------------------------------------------------------------
```

Some of the behaviors/descriptions are subject to further investigation especially on byte 09 bits 0x04 and 0x80 both being related to switch closure during bonus award. In all cases, “switch processed” means that the callback function at bytes 03 04 05 gets called.

For L8.4 it was immediately obvious that a change is needed in this gun-trigger switch table entry so that the gun-trigger callback function gets called when the switch gets closed during attract mode. A comparison is in order between IJ L-7 and T2 L-8, as shown below.

```
;-------------------------------------------------------------------------------------------
; Gun Trigger flag bytes IJ-L7: 00 02 79 D6 36 03 00 00 70 00 04
;                                                       ^
;                                                       |
; 0x40 bit == switch is valid --------------------------+
; 0x20 bit == switch allowed in tilt-mode --------------+
; 0x10 bit == switch allowed in attract-mode/test-mode -+
;
; Gun Trigger flag bytes T2-L8: 00 02 5D 0A 31 3C 00 00 40 00 04
;                                                       ^
;                                                       |
; 0x40 bit == switch is valid --------------------------+
;
;-------------------------------------------------------------------------------------------
```

Comparing IJ and T2, it is evident the only differences are as follows:

- Byte 06, T2 uses value 0x3C (60) where IJ uses 0x03. This is the number to balls in play that can play without the switch closing before game flags the switch as bad in the test report. The T2 software allows 60 balls to play before declaring the gun-trigger as bad (if it has not been closed) while IJ only allows 3 balls in play to go by without gun-trigger switch closure before declaring the gun-trigger switch as bad in the test report.
- Byte 09, T2 only has 0x40 bit set which means switch is valid. By default this means its callback function will only be called during game play and not during TILT or bonus award period. The IJ, on the other hand has 3 bits set, 0x04, 0x02 and 0x01 which means the IJ gun-trigger callback function can be called during game, during TILT and during attract mode or test mode.

Clearly, the change needed for T2 L8.4 gun-trigger switch table data is that its byte 09 needs to have the 0x10 bit set so that its callback function is invoked when the switch is closed during attract mode. The changed data is depicted below. This is the gun-trigger switch table entry is at $4A68,3D, ROM offset 0x74A68.

```
4A68: 00 02                         ; SwitchTableEntry1C, 34, Grip Trigger
4A6A: 5D 0A 31                      ; SwitchMatrixHdlr_GunTrigger()
4A6D: 3C 00                         ;
4A6F: 00 40                         ; 40=Valid
4A6F: 00 50                         ; 40=Valid, 10=AttractMode (L8.4)
```

```
4A71: 00 04                         ;
```

With the 0x10 bit added to the table entry's 9th byte, the gun-trigger callback $5D0A,31 will be invoked when the gun-trigger is closed during attract mode. This necessarily means the gun-trigger handler code needs to ensure it invokes the normal L-8 code if called during game-play or plays a L8.4 sound if called during attract-mode. In order to do this, the IJ L-7 code is analyzed to determine how it handles gun-trigger switch closure so something similar can be done for T2 L8.4.

In addition to the change to the 9th byte, the 3rd, 4th and 5th bytes need to be updated so that the gun-trigger switch closure calls a new L8.4 function. The new function will decide if the game is in game-mode or not and determine if original L-8 gun-trigger code at $5D0A,31 should be called or new L8.4 code called instead. This new function is located at some free space at $7F73,31 which means the gun-trigger switch table entry is further updated as follows:

```
4A68: 00 02                         ; SwitchTableEntry1C, 34, Grip Trigger
4A6A: 5D 0A 31                      ; SwitchMatrixHdlr_GunTrigger()
4A6A: 7F 73 31                      ; SwitchMatrixHdlr_GunTriggerL84()
4A6D: 3C 00                         ;
4A6F: 00 40                         ; 40=Valid
4A6F: 00 50                         ; 40=Valid, 10=AttractMode (L8.4)
4A71: 00 04                         ;
```

The new function at $7F73,31, ROM offset 0x47F73, is as follows:

```
-------------------------------------;-------------------------------------------------------
                                    ;
                                    ; L8.4 Gun-Trigger Code Inserted at $7F73,31
                                    ;
7F73: BD F7 59    JSR   $F759       ; CheckGameMode() // z-bit set if game in progress
7F76: 26 03       BNE   $7F7B       ; If z-bit clear, go do L8.4 attract mode
7F78: 7E 5D 0A    JMP   $5D0A       ; Go to normal L-8 gun-trigger handling during game mode
7F7B: BD 89 2F    JMP   $892F       ; CallBankedFunction_Param_WPCAddr_NoReturn()
7F7E: 7F 6F 38                      ; GunTriggerL84Handler()
                                    ;
-------------------------------------;-------------------------------------------------------
```

This new function calls a common L-8 function to check if game-mode is active and, if so, jumps to the $5D0A,31 function which is the normal L-8 handler for gun-trigger. If game-mode is not active, then new L8.4 function is called. Since there is not enough free ROM space in bank $31, the L8.4 handler is in bank $38 so the $892F function is used to effectively jump to the code in the other bank. There is no need for such code to return back here since it will end its work with the same jump to $99A2 as what the original L-8 switch handler does when it's finished.

The L8.4 gun-trigger function at $7F6F,38 ROM offset 0x63F6F is shown below.

```
-------------------------------------;-------------------------------------------------------
                                    ;
                                    ; GunTriggerL84Handler()
                                    ;
                                    ; Called here after confirming not in game mode.
                                    ; Could be in attract mode or test mode, so first need
                                    ; to check $1793.
                                    ;
```

```
7F6F: 7D 17 93    TST   $1793           ; $1793 is 0 during attract mode, non-zero in test mode
7F72: 26 04       BNE   $7F78           ; If non-zero branch to the end, no work to do here
                                        ;
7F74: 8D 05       BSR   $7F7B           ; Branch to routine to deal with L8.4 gun-trigger sounds
7F76: 8D 58       BSR   $7FD0           ; Branch to routine to deal with Tournament Mode Scores
                                        ;
7F78: 7E 99 A2    JMP   $99A2           ; All done
                                        ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; Do L8.4 gun-trigger attract-mode sounds
                                        ;
7F7B: BD 82 FF    JSR   $82FF           ; LoadAWithTableIndexByteParameter()
7F7E: 17                                ; FeatureAdjustment023, Attract Mode, Adj 0x17
7F7F: 81 03       CMPA  #$03            ; Adjustment Value = 0, L8.1     C-bit set
                                        ;                    1, L8.2     C-bit set
                                        ;                    2, L8.3     C-bit set
                                        ;                    3, L8.4     C-bit clear
7F81: 25 37       BCS   $7FBA           ; If c-set then not L8.4, no gun-trigger sound, goto end
7F83: BD 86 5B    JSR   $865B           ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
7F86: 10 00                             ; 0x10 == Attract Sounds, C-bit set when not 0x00
7F88: 24 30       BCC   $7FBA           ; If no attract sounds, no gun-trigger sound, goto end
                                        ;
7F8A: BD 86 90    JSR   $8690           ; SearchLinkedListForId() // c-clear means ID is found
7F8D: 00 F3                             ; Parameter 0x00F3
7F8F: 24 29       BCC   $7FBA           ; If timer is running, no sounds, goto end
                                        ;
7F91: 8E 7F C6    LDX   #$7FC6          ; X=0x7FC6, start of list of sound #s for gun-trigger
                                        ;
7F94: 86 0A       LDA   #$0A            ;--\ A gets 0x0A
7F96: BD A7 5B    JSR   $A75B           ;  | Get16BitPseudoRandomValueintoA()
7F99: B1 06 46    CMPA  $0646           ;  | Is it same random 8-bit as last time?
7F9C: 27 F6       BEQ   $7F94           ;--/ If so, keep trying for different random number
                                        ;
7F9E: B7 06 46    STA   $0646           ; Save random number A into $0646
                                        ;
7FA1: A6 86       LDA   A,X             ; A = GunTriggerSounds[A]
                                        ;
7FA3: BD BD FB    JSR   $BDFB           ; DoSoundTableIndexA(), plays random sound effect
7FA6: 25 12       BCS   $7FBA           ; if Error then return without counting it
                                        ;
7FA8: 7C 06 44    INC   $0644           ; $0644++
7FAB: B6 06 44    LDA   $0644           ; A = $0644
7FAE: 81 03       CMPA  #$03            ; if (A >= 3)  // if played 3 or more sound effects
7FB0: 25 08       BCS   $7FBA           ; {
7FB2: BD 8B 77    JSR   $8B77           ;    ScheduleFunctionStart()
7FB5: 00 F3                             ;    ID = 00F2
7FB7: 7F BB 38                          ;    GunTriggerSilenceTimer()
                                        ; }
7FBA: 39          RTS                   ;
                                        ;
---------------------------------------;-------------------------------------------------------
                                        ;
                                        ; GunTriggerSilenceTimer()
                                        ; Function ID == 00F2
                                        ;
7FBB: BD 86 79    JSR   $8679           ; SleepLong()
7FBE: 2A 30                             ; 2m48.75s
7FC0: 7F 06 44    CLR   $0644           ; Reset the gun-trigger sound count back to zero
7FC3: 7E 99 A2    JMP   $99A2           ; ThreadedFunctionDone()
                                        ;
```

```
---------------------------------------;-------------------------------------------------------
                                       ;
                                       ; GunTriggerSounds[]
                                       ;
7FC6: A1                               ; "You're targeted for termination"
7FC7: AE                               ; Gunshot
7FC8: 65                               ; "Take your best shot"
7FC9: 69                               ; "Great shot"
7FCA: 7C                               ; "Load the cannon"
7FCB: 7D                               ; "Shoot again"
7FCC: 90                               ; "Get down"
7FCD: 99                               ; "Don't move"
7FCE: 9E                               ; "Reloaded"
7FCF: 9F                               ; "Destroy everything"
                                       ;
---------------------------------------;-------------------------------------------------------
                                       ;
                                       ; Check if Tournament mode, if set, show scores
                                       ;
7FD0: BD 86 5B    JSR   $865B          ; LookupGameAdjustmentParam1andCheckIfEqualsParam2()
7FD3: 9A 01                            ; StandardAdjustment026, Tournament Play, Adj=0x9A
7FD5: 25 02       BCS   $7FD9          ; C-set = tournament mode is off, skip to end, no work
7FD7: 8D 01       BSR   $7FDA          ; ScheduleGunTriggerTournamentModeScoresDisplay()
7FD9: 39          RTS                  ;
                                       ;
---------------------------------------;-------------------------------------------------
                                       ;
                                       ; ScheduleGunTriggerTournamentModeScoresDisplay()
                                       ;
7FDA: 86 01       LDA   #$01           ;
7FDC: B7 06 47    STA   $0647          ; Push 0x01 into $0647 to signal 0x17 fn to show scores
7FDF: BD 85 53    JSR   $8553          ; Calls function from table at $5D1A,3D
7FE2: 17                               ; 0x17 --> $7F48,3A, L8.4 start-button/gun-trigger hdlr
7FE3: 39          RTS                  ;
                                       ;
---------------------------------------;-------------------------------------------------------
```

Readers are encouraged to trace through the L8.4 gun-trigger sound handler, above, starting at $7F6F,38. When tournament mode is set, the code sets $0647 to non-zero and then calls the indexed function 0x17 same as start-button. As shown in earlier code-change for L8.4 the start-button indexed function 0x17 is modified to check $0647 and determine whether to handle it as gun-trigger (and show previously played game scores) or as a start-button and play sound then show the "Insert Coin" message. The way in which indexed function 0x17 is called necessarily means the attract mode is going to be interrupted and then restarted. The interruption will either be the "Insert Coin" message or the last played scores display.

Note that all timers used are done to match that of IJ L-7. This includes the timeout period where start-button won't make noise (equal to IJ flipper button sound timeout period) and the gun-trigger timeout period. The duration in which tournament mode previously played scores is shown also matches IJ when flipper button is pressed in attract mode when tournament mode is enabled.

Appendix

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

Indexed Display Effect Functions

During development of L8.4 and especially during the attract-mode updates, the L-8 function $8553 was given a closer examination for the possible need of having to use it for gun-trigger display of previously played game scores (when tournament mode is enabled) interrupting and restarting the attract mode in the process. It was noticed that the start-button handler in L-8 necessarily needs to invoke the function to play sound effect and show "Insert coin" by using $8553 in order for the attract mode to be allowed to be interrupted and restarted. Details into precisely how the $8553 function call of indexed function 0x17 causes attract mode to be interrupted and restarted have not been fully studied.

What was done in L8.4 was survey the ROM for how $8553 is called with various index values in order to trigger different display effects. This was done in hopes of finding an unused index or possibly consider increasing the table at $5D1A,3D to accommodate the gun-trigger's need to interrupt the display with previously played game scores.

Ultimately it was determined to be too uncertain and too much risk in attempting to use an unused index or to attempt to increase the size of $5D1A,3D table, especially given the nature of the code change being implemented (a simple and fun way to enhance the L8.4 attract mode). The final solution for L8.4 was to simply use the existing start-button index function 0x17 for both start-button *and* gun-trigger. Prior to calling the $8553 with index 0x17, the L8.4 code ensures a RAM byte is set to 0x00 or 0x01. The function handler for index 0x17 is updated in L8.4 to then check this RAM byte and proceed with attract-mode interruption of either the "Insert coins" message or the display of previously played game scores. Refer to text earlier in this document for technical details of this code change. It should be noted the RAM byte that is used to flag index function 0x17 was vetted and determined to be safe to use as it is only otherwise used during video mode during game play. Using it in this way during attract mode is safe and poses no risk.

While surveying the $8553 index functions, a list of the various indexed functions was made and recorded here for posterity. The $8553 function calls another function, $C791, to call up the indexed function. Both of these function usages was examined and documented in the table below.

$8553/ $C791 Index	**$8553 Called From, or $C791 Called from**		**$5D1A,3D Entry Data**		**Game Usage/Feature**
	ROM Offset	WPC Address	Flag Bytes	Vector	
0x00			00 24	$C952	
0x01	0x607C9	$47C9,38	01 01	$6AC1,3B	Player score, steady
0x02	0x7B22A	$B22A	01 01	$68F5,3B	Player score, blinking
0x03	0x45703	$5703,31	01 01	$69EC,3B	Player score, scrolling down/up
	0x7B342	$B342	01 01	$69EC,3B	Player score, scrolling down/up
0x04	0x66078	$6078,39	01 01	$793F,30	Attract mode
	0x60630	$4630,38	01 01	$793F,30	Attract mode
0x05	0x7C750	$C750	01 01	$6899,3B	Testing… / Test report

0x06	0x607DD	$47DD,38	F3 3A	$7E19,30	Game Over / Player Scores
	0x66074	$6074,39	F3 3A	$7E19,30	Game Over / Player Scores
0x07	0x6083B	$483B,38	F3 3A	$505E,24	Player X is a top marksman
0x08	0x60851	$4851,38	F3 3C	$4DC3,24	HSTD Initials Entry
0x09	0x608B1	$48B1,38	F3 3C	$50B9,24	HSTD Initials Received / Award
0x0A	0x608F5	$48F5,38	F3 3A	$44B4,24	Match Sequence animation
0x0B	0x7CFE8	$CFE8	08 64	$51C6,24	Flipper button status report
	0x7D218	$D218	08 64	$51C6,24	Flipper button status report
0x0C	0x65EA0	$5EA0,39	E4 2C	$7DF1,30	Shows credits in large font
0x0D			80 60	$42FF,24	Tilt warning
0x0E			F0 40	$6AB8,3B	
0x0F	0x65949	$5949,39	01 01	$433F,24	TILT
0x10	0x6076C	$476C,38	EE 02	$7553,33	Bonus addup sequence
0x11	0x60CFB	$4CFB,38	40 20	$735F,35	Shoot Again animation sequence
0x12	0x60D31	$4D31,38	40 20	$681B,3B	Replay At score shown
0x13	0x7BD04	$BD04	10 1A	$6928,3B	Press “Enter” for test report
0x14	0x74515	$4515,3D	00 26	$7A1F,33	Match/Replay knocker
0x15	0x6F64B	$764B,3B	90 2A	$47E2,24	Pinball Missing Please Wait
0x16	0x74848	$4848,3D	F1 6C	$6CDF,3B	Volume Level adjustment
0x17	0x60572	$4572,38	18 1C	$7DDD,30	Start-button zero-credits sound
0x18	0x680A3	$40A3,3A	10 6C	$68C7,3B	Open coin-door to use buttons
0x19	0x6602E	$602E,39	F3 3C	$67DE,3B	(?) attract mode restarts
0x1A	0x66042	$6042,39	01 01	$67E7,3B	1-coin buy-in countdown
0x1B	0x4585D	$585D,31	B0 62	$7264,33	Ramp MILLION award
	0x45DCC	$5DCC,31	B0 62	$7264,33	Ramp MILLION award
	0x4607F	$607F,31	B0 62	$7264,33	Ramp MILLION award
0x1C	0x45580	$5580,31	E8 22	$72B9,33	Database Selector
0x1D			DD 22	$7907,33	Extra Ball Animation
0x1E	0x44F02	$4F02,31	DD 22	$783F,33	Jackpot Animation
0x1F	0x44F47	$4F47,31	DD 22	$4B44,24	Super Jackpot Animation
0x20			EA 22	$42E8,2B	Video mode
0x21	0x456C4	$56C4,31	01 01	$721C,33	Hurry Up
0x22	0x4636A	$636A,31	DD 62	$7241,33	Shows score/points in huge font
0x23	0x4572A	$572A,31	A0 62	$732A,35	Extra ball is lit
0x24	0x44D38	$4D38,31	B0 62	$7C63,33	Hunter Ship Explosion, direct hit
0x25	0x456D9	$56D9,31	01 01	$715A,33	Payback time
0x26	0x46145	$6145,31	DD 62	$712C,33	Payback Time
0x27	0x46CBC	$6CBC,31	DD 62	$7505,33	Multiball start animation
0x28	0x456EA	$56EA,31	01 01	$6FC6,33	Load the Gun countdown
0x29	0x470E4	$70E4,31	DD 62	$4718,24	Get the super jackpot
0x2A	0x45BAC	$5BAC,31	B4 62	$7203,35	Bonus-X multiplier
0x2B	0x451EA	$51EA,31	71 62	$6EED,33	Jackpot Grows
	0x45A0D	$5A0D,31	71 62	$6EED,33	Jackpot Grows
0x2C	0x444E5	$44E5,31	B3 42	$7910,35	Autofire machine gun

	0x4463B	$463B,31	B3 42	$7910,35	Autofire machine gun
	0x455DE	$55DE,31	B3 42	$7910,35	Autofire machine gun
0x2D	0x44956	$4956,31	B2 62	$7DD5,35	Kickback animation
0x2E	0x44A71	$4A71,31	B0 42	$73BD,35	Chase Loop motorcycle
0x2F			71 62	$7A91,35	Blocky screen fill bottom up
0x30	0x457D4	$57D4,31	B0 62	$7C30,35	Blocky Bonus Held message
0x31	0x45EB7	$5EB7,31	B6 62	$4088,24	Security Level doors opening
0x32	0x45D66	$5D66,31	DD 62	$7321,35	Hurry Up is Lit
0x33	0x46B8C	$6B8C,31	DD 62	$41CD,24	Pull Trigger / Hunter Ship target
0x34	0x6E263	$6263,3B	B0 62	$438D,24	Los Angeles July 11, 2029
0x35	0x464BE	$64BE,31	DD 62	$43BD,24	Let's Go
0x36	0x464CE	$64CE,31	DD 62	$43D3,24	GO
0x37	0x464DE	$64DE,31	DD 62	$43E9,24	RUN
0x38	0x445F5	$45F5,31	04 62	$43FF,24	Pull Trigger to shoot ball
	0x445FE	$45FE,31	04 62	$43FF,24	Pull Trigger to shoot ball
0x39	0x451B3	$51B3,31	DE 62	$46A1,24	Ball Locked, Jackpot multiplied
	0x459E5	$59E5,31	DE 62	$46A1,24	Ball Locked, Jackpot multiplied
	0x47032	$7032,31	DE 62	$46A1,24	Ball Locked, Jackpot multiplied
0x3A	0x457AF	$57AF,31	B0 62	$414D,24	Return Lanes are lit
0x3B	0x45E30	$5E30,31	B6 62	$7311,35	Get the CPU
	0x460E3	$60E3,31	B6 62	$7311,35	Get the CPU
0x3C	0x45AA8	$5AA8,31	B2 62	$790B,35	Kickback Lit
	0x45ACB	$5ACB,31	B2 62	$790B,35	Kickback Lit
0x3D	0x461C7	$61C7,31	DD 62	$7264,33	Million jumbled characters
0x3E	0x44C8F	$4C8F,31	DD 62	$72DE,35	# Targets Remaining
0x3F	0x47379	$7379,31	F5 28	$7390,31	(?) attract mode restarts
0x40	0x75F89	$5F89,3D	F4 3A	$60C6,3D	Gun Test
0x41	0x46017	$6017,31	EA 62	$72C0,35	Shoot for video mode
0x42	0x456FD	$56FD,31	01 01	$6F82,33	Load the Gun, super jackpot value
0x43	0x461BC	$61BC,31	DD 42	$53C4,24	Payback Time total, # million
0x44	0x44AB4	$4AB4,31	B1 62	$7264,33	Million jumbled characters
0x45			C6 26	$7A20,33	Replay award animation
0x46			C6 26	$7A56,33	Special award animation
0x47	0x45CB6	$5CB6,31	C6 06	$53FE,24	Easter Egg: Hello Xaqery
0x48	0x45CB6	$5CB6,31	C6 06	$5424,24	Easter Egg: Hello Anna
0x49	0x45CB6	$5CB6,31	C6 06	$544A,24	Easter Egg: Hello Doc X
0x4A	0x45CB6	$5CB6,31	C6 06	$5468,24	Easter Egg: Hello World

The table above serves as supporting data for future enhancements or code examinations.

ROM Image Changes

The table, below, identifies every ROM change in L8.4 as compared to the L8.3 ROM image with a brief description of each ROM change. Refer to text throughout this document for technical details of each ROM byte change described below. *Refer to the L8.3 document for all ROM changes between L8.3 and the official L-8 ROM image.*

ROM Offset	WPC Address	Original Bytes	Original Description	New Bytes	New Description
0x113A8	$53A8,24	BD D3 60	Flipper-button status-report code about to report number of bonus-x for extra-ball lit.	7E 57 57	Jump to new L8.4 code at $5757,24 to determine whether or not to exclude bonus-x for extra-ball from the flipper-button status-report.
0x11757	$5757,24	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Start of region of unused ROM bytes in bank $24.	BD 86 5B 83 00 25 03 7E 53 C2 BD D3 60 7E 53 AB	Code that checks "Max E.B. Count" adjustment and decides whether to show the bonux-x for lit extra ball or to exclude the message from flipper-button status-report.
0x2C36E	$436E,2B	A7 25	In Video Mode startup, this is the address portion of a JSR instruction. Originally this jumps to $A725 to load up a random number into A register.	49 E7	The JSR now jumps to new L8.4 code at $49E7,2B which checks for tournament mode and, if set, returns same value for each player in multi-player game. Otherwise returns genuinely random number as per original code.
0x2C387	$4387,2B	A7 25	In Video Mode startup, this is another call to fetch a random number from $A725.	49 EE	The JSR now jumps to new L8.4 code at $49EE,2B to get either a tournament mode value or genuine random number.
0x2C3A4	$43A4,2B	7E 43 A7 25 01 4C	In Video Mode startup, when it has been determined the EB should be shown during the video mode this code loads A with 0x01 as flag value into $064C.	25 04 8A 80 20 00	The L8.4 code repurposes the $064C so it contains a 7-bit counter in low 7-bits. Here code is changed so when EB is to be awarded A gets 0x80 high bit flag to indicate EB will be part of the Video Mode.
0x2C543	$4543,2B	7D 06 4C	During Video Mode this is code that checks for non-zero value in $064C	BD 4A 2C	The L8.4 code calls new routine that will check if EB should be shown. The

			to see if EB should be shown.		routine will check 0x80 bit of $064C and return Z-bit set/cleared accordingly.
0x2C591	$4591,2B	`A7 25`	During Video Mode this is address of a JSR instruction where code fetches random value to determine next robot direction.	`4A 1E`	The JSR now jumps to new L8.4 code at $4A1E,2B to get either a tournament mode value or genuine random number.
0x2C5AA	$45AA,2B	`A7`	During Video Mode this is part of $A725 address of a JSR instruction where code fetches random value to determine next robot timing/speed.	`4A`	The L8.4 code calls new routine at $4A25,2B (which means only the high byte of address is changed in ROM). This gets either a tournament mode value or genuine random number.
0x2C91E	$491E,2B	`FF FF FF FF FF FF FF FF` `FF FF FF FF FF`	Start of region of unused ROM bytes in bank $2B.	`18 A3 18 EB 18 F1 18 F7` `18 FD 19 21 19 2D 19 39` `19 45 19 51 19 5D 19 69` `19 75 19 81 19 8D 19 99` `19 A5 19 B1 00 00 18 EB` `18 F1 18 F7 18 FD 19 27` `19 33 19 3F 19 4B 19 57` `19 63 19 6F 19 7B 19 87` `19 93 19 9F 19 AB 19 B7` `00 00 18 A3 18 EB 18 F1` `18 F7 18 FD 19 21 19 27` `19 39 19 3F 19 51 19 57` `19 69 19 6F 19 81 19 87` `19 99 19 9F 19 B1 19 B7` `00 00 18 EB 18 F1 18 F7` `18 FD 19 2D 19 33 19 45` `19 4B 19 5D 19 63 19 75` `19 7B 19 8D 19 93 19 A5` `19 AB 00 00 10 AE 81 27` `04 AB A4 20 F7 39 8D F4` `8E F9 00 A6 86 39 34 10` `8E 05 B5 BD FB 29 A6 84` `35 90 BD 86 5B 9A 01 24` `07 BD A7 25 32 62 35 30` `39 8D EF 8D E1 8D D7 35` `B0 34 30 8E 49 1E 20 F1` `34 30 8E 49 44 20 EA 8D` `D9 B6 06 4C 85 80 27 05` `4C 8A 80 20 03 4C 84 7F` `B7 06 4C 84 7F 34 02 8D` `B5 8C 49 68 27 04 AB E0` `20 02 A0 E0 8D A0 35 B0` `34 30 8E 49 68 20 D0 34` `30 8E 49 90 20 C9 34 02` `B6 06 4C 85 80 35 82 BD` `84 80 49 BD 84 80 4A BD` `86 5B 83 00 39`	Block of code performs the Video Mode code for getting a set of identical seemingly random numbers for all players in a multi-player game when Tournament Mode is enabled. This includes all jump points from the previous changes described above. If Tournament Mode is not enabled then the regular L-8 logic is used where genuine random numbers are used.
0x41B04	$5B04,30	`73`	When adjustments are showing number of seconds for a particular adjustment, this is the starting 's' in "sek." That is shown when language is set to German.	`53`	For L8.4 the abbreviated seconds in German is corrected to use upper case 'S' for "Sek.".

0x43DDD	$7DDD,30	`BD 84 AD DC 25 08 BD 85`
		`46 51 BD 84 8F DC BD 7B`
		`74 7E C9 52`
			When the start button	`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF FF FF FF`
				`FF FF FF FF`
					The L8.4 code moves this

			is pushed during attract mode and there are not enough credits to start a game, this function is called to possibly play a sound effect and to show the “Insert Coin” message for a brief moment before attract mode restarts.		logic with enhancements to $7F48,3A thus making this $7DDD,30 function no longer used. It is reset to unused 0xFF bytes for completeness and for possible reuse by other code in the future.
0x44676	$4676,31	AF	This is byte used to indicate function id 00AF, Ball-saver countdown timer loop. This is part of code that checks for whether or not to show the Autofire animation as part of the ball stoppage accounting logic.	83	This is byte used to indicate function id 0083, Ball-saver maintenance function. This L8.4 code changes modifies the function ID that the ball stoppage accounting searches for when determining whether or not to show the Autofire animation. Refer to “Multiball Auto-Fire Code Correction” section of this document for technical details.
0x44BCE	$4BCE,31	86 90 00 84 24 07 BD 86 90 00 E1 25 57	This is code used for 5-bank target hits used for determining whether the target hit is for ordinary game play, or if it is an attempt to shoot the hunter ship from the cannon.	88 F5 7A B9 3A 25 5D 27 59 7E 4B DB 12	The updated L8.4 code calls into a new function at $7AB9,3A which performs a more elaborate set of checks for determining how the 5-bank target should be processed. Depending on whether the $7AB9,3A function returns C and Z bits, the new code here proceeds accordingly. Refer to “Hunter Ship 5-Bank Target L8.4 Code Changes” section of this document for technical details.
0x45565	$5565,31	F7 06 12 BD A7 25	During Database Award setup this code saves the winning award index from B into $0612 and then calls $A725 to get a random number into A for remaining of	BD 88 F5 6B C0 34	The updated L8.4 code calls into a new function at $57FA,24 which will check if Tournament Mode is enabled and, if so, ensures all players in a multi-player game get the same award,

			the database award logic.		otherwise original L-8 logic is retained.
0x4574C	$574C,31	8E 05 AD BD FB 29 7E 57 55	During the awarding of extra ball this code is after the $B383 function has been called which checks or skips the accumulation of an extra ball if player is ineligible. This L-8 code proceeds with assumption EB was awarded even if it was not accumulated, and shows the extra ball animation on the display.	25 3B 12 8E 05 AD BD FB 29	The L8.4 code takes advantage of the fact that the $B383 function returns with C-bit set if the extra ball was not actually accumulated. This code checks the C-bit and, if set, skips over the extra ball animation sequence. If c-bit is clear then the L-8 logic proceeds with incrementing of the per-player extra ball award count and display of the extra ball animation.
0x45BA2	$5BA2,31	84 80 49 BD 84 80 4A	During Bonus-x advancement this is part of code that is executed when the extra ball is being lit. This code sets some logic flags to control the bonus-x behavior regarding extra ball lit and then it calls $572E to illuminate the extra ball lamp at the skull shot.	88 F5 4A 35 2B 24 03	The L8.4 code calls a new function at $4A35,2B to set the same logic flags as L-8 and then determine if the extra ball should be lit based on the "Max E.B. Count" adjustment. If the extra ball can be lit then it returns C-bit set. This code checks the return C-bit value and proceeds to either light the extra ball or not.
0x45ED4	$5ED4,31	84 49 30 24 06 8D 47 8D 07 20 04 8D 03 8D 3F 39	During the "Security Pass" award this function is called to advance left/right security awards in L-8.	88 F5 7A F4 3A 39 FF FF FF FF FF FF FF FF FF FF	In L8.4 the "Security Pass" award function now calls a new L8.4 function at $7AF4,3A to handle all of the security lamp advancement. This results in unused bytes in this function which are replaced with 0xFF so they can be repurposed later if needed.
0x46A2C	$6A2C,31	A7 5B 1F 89 3A	This is code used when game is determining the 5-bank lamp(s) that need to be lit and subquently hit by cannon-shot in order to start multiball. The L-8	88 F5 6C 0C 34	In L8.4 this replacement code will call a new function at $6C0C,34 which will ensure that if Tournament Mode is enabled, all players in a multi-player game get the

			code here is getting a random number and advancing the value of X by such random number.		same 5-bank lamp patterns for each multiball. If Tournament Mode is not enabled then the original L-8 logic is used.
0x46E66	$6E66,31	8B 77 00 83 6E 8A 31	When the autofire timer is being set due to skill-shot, this is the code where the timer is about to be set. The number of seconds is in the B register and this code schedules function ID 0083 at $6E8A,31 which will ensure balls are returned to the playfield during the time period in the B register.	88 F5 7C 00 3A 20 00	The L8.4 code has a call to a new function for L8.4 at $7C00,3A which will check the new L8.4 "SS Autofire Time" adjustment and schedule the function ID 0083 at $6E8A,31 with the timer value set accordingly.
0x47099	$7099,31	F7 59 7E 70 9E	This function is called when the ball is transitioning from the ball-popper to the cannon with the hits-remaining being reported on the display. This code is checking if game is in progress (or if such ball movement is happening during attract mode if ball got into the ball-popper). This code has a dummy JMP instruction taking up three bytes of space.	FB 77 BD F7 59	The L8.4 code adjusts the code so that a common L-8 function at $FB77 is first called which waits for any animation in progress to complete before returning. After this the $F759 function is called to check game mode same as L-8 logic. This code change is needed for fixing the "PAPA Lost Super Jackpot" problem. Refer to "PAPA Lost Super Jackpot Code Fixes" section of this document for technical details.
0x47180	$7180,31	84 8F 48	This is a state-check function that is called when the ball is in the ball-popper. This code is a function call to $848F to clear memory flag 0x48 as part of normal state-audit logic.	7F 8F 12	The L8.4 code replaces the $848F function with a call to new L8.4 function at $7F8F,31 to perform logic needed as part of the bug fix for the "PAPA Lost Super Jackpot" problem.
0x47F73	$7F73,31	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Start of region of unused ROM bytes in	BD F7 59 26 03 7E 5D 0A BD 89 2F 7F 6F 38	In L8.4 the gun-trigger switch closure calls this

			bank $31.		new function instead directly calling $5D0A,31 as it does in L-8. This new function checks if game mode or attract mode. If game mode then calls $5D0A,31. If attract mode then it jumps to new L8.4 function at $7F00,38.
0x47F8F	$7F8F,31	FF FF FF FF FF FF FF FF FF FF FF FF	In region of unused ROM bytes in bank $31.	BD 86 90 00 86 24 04 BD 84 8F 48 39	The ROM change at $7180,31, 0x47180 calls this new function as part of the "PAPA Lost Super Jackpot" fix. Logic will only clear the 0x48 memory flag if multiball is not running. Refer to the "The PAPA Lost Super Jackpot Bug" section of this document for technical details.
0x47FAD	$7FAD,31	B9	This is part of L8.3 code where the drop-target "up" function ID 00B9 is referenced by L8.3 code checking if the callback function is currently running.	B1	This updates the function ID for the L8.3 code for drop-target "up" from B9 to B1. Refer to the "Fix the L8.3 Function ID Overlap Issue" section of this document for details.
0x47FE2	$7FE2,31	B9	This is part of L8.3 code where the drop-target "up" function ID 00B9is referenced by L8.3 code where the function is being scheduled.	B1	This updates the function ID for the L8.3 code for drop-target "up" from B9 to B1. Refer to the "Fix the L8.3 Function ID Overlap Issue" section of this document for details.
0x518BA	$58BA,34	59 26 34	When game is about to start sequence for super jackpot attempt, this is where code schedules function that maintains the moving 5-bank lamp. This is address of the scheduled function $5926,34.	7E 95 3A	The L8.4 code replaces the original function call with address $7E95,3A which is the L8.4 super-jackpot lamp movement function.
0x518FE	$58FE,34	A7 5B	When game is about to	6C 25	The L8.4 code changes this

			illuminate one of the 5-bank targets that player must hit for jackpot, this is function call to $A75B where a random number is retrieved for which lamp to illuminate.		to call $6C25,34 which will check if Tournament Mode is set and, if so, ensure all players in a multiplayer game get the same jackpot lamp. If Tournament Mode is not set then a random number is used as original L-8 logic.

0x51926	$5926,34	`86 FF 97 D3 BD 87 15 15`
		`18 86 04 B7 05 F9 BD 83`
		`46 14 BD FD B4 18 18 7A`
		`05 F9 B6 05 F9 4D 26 EE`
		`BD 83 46 14 BD FD A2 18`
		`18 7C 05 F9 B6 05 F9 81`
		`04 25 ED 20 D9`
			This is the super jackpot 5-bank lamp movement function called from code cited above at $58BA,34, 0x518BA.	`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF FF FF FF`
				`FF FF FF FF FF`
					The L8.4 replaces the super jackpot lamp movement function with new code at $7E95,3A. This L-8 function is no longer used so it is removed and set to 0xFF so the ROM space may be repurposed if needed.

0x52B1D	$6B1D,34	`FF FF FF FF FF FF FF FF` `FF FF FF FF FF FF`	Start of region of unused ROM bytes in bank $34.	`18 A3 18 EB 18 F1 18 F7` `18 FD 19 21 19 2D 19 39` `19 45 19 51 19 5D 19 69` `19 75 19 81 19 8D 19 99` `19 A5 19 B1 00 00 18 EB` `18 F1 18 F7 18 FD 19 27` `19 33 19 3F 19 4B 19 57` `19 63 19 6F 19 7B 19 87` `19 93 19 9F 19 AB 19 B7` `00 00 18 EB 18 F1 18 F7` `18 FD 19 21 19 27 19 39` `19 3F 19 51 19 57 19 69` `19 6F 19 81 19 87 19 99` `19 9F 19 B1 19 B7 00 00` `10 AE 81 27 04 AB A4 20` `F7 39 8D F4 8E F9 00 A6` `86 39 1F 12 8D 16 A6 84` `1F 21 8D EE 39 8E 6B 1D` `8D F0 84 0F 4C 39 8E 6B` `43 8D E7 39 8E 05 C5 BD` `FB 29 39 BD 86 5B 9A 01` `24 07 F7 06 12 BD A7 25` `39 34 20 8D D8 BD 86 5B` `83 00 25 0C 81 02 26 02` `86 0D 81 0A 26 02 86 0F` `1F 89 F7 06 12 8D C7 8D` `CB 6C 84 35 A0 34 30 8E` `05 9D BD FB 29 A6 84 AE` `E4 8D 97 84 0F 81 04 23` `04 80 05 20 F8 35 B0 BD` `86 5B 9A 01 24 05 BD A7` `5B 20 09 34 10 8E 6B 1D` `8D D3 35 10 1F 89 3A 39` `BD 86 5B 9A 01 24 05 BD` `A7 5B 20 09 34 10 8E 6B` `67 8D BA 35 10 39`	Block of L8.4 code used for getting consistent behavior for all players in a multi-player game when Tournament Mode is enabled. $6BAA,34 gets database award index. $6BB3,34 gets database selector to blink as winner. $6BC0,34 is entry point for database award handler. $6BF2,34 gets multiball lamp pattern number. $6C0C,24 is entry point for multiball lamp pattern. $6C25,24 is entry point for jackpot lamp selector.
0x572B7	$72B7,35	`86 37 97 B4`	When 3-bank rollovers have been completed enough times for Extra Ball to be lit, this code is storing value 0x37 into RAM byte $B4 as indicator that Extra Ball should be lit.	`7E 7E E8 12`	The L8.4 jumps to new code at $7EE8,35 to determine if the extra ball should be lit or not, depending on the adjustment “Max E.B. Count”.
0x57EE8	$7EE8,35	`FF FF FF FF FF FF FF FF` `FF FF FF FF FF FF FF FF`	Start of region of	`BD 86 5B 83 00 25 03 7E` `C9 52 86 37 97 B4 7E 73`	The L8.4 inserts this new

		FF	unused ROM bytes in bank $35.	2A	function called from the $72B7,35, 0x572B7 change, above. When 3-bank rollovers are about to light the extra ball this function will prevent that from happening if the “Max E.B. Count” adjustment is for no extra balls.
0x57FBF	$7FBF,35	19	This is byte region where L8.x code uses to ensure a value checksum can be calculated.	21	The L8.4 code updates this byte so a valid checksum can be calculated.
0x57FCA	$7FCA,35	33	This is an informational string in the ROM indicating the “T2_l8.3” designation.	34	Updated in L8.4 to indicate “T2_l8.4”.
0x60573	$4573,38	85 53 17	The start-button handler, when pressed during attract mode without enough credits to start a game will use this code to call $8553 to schedule function index 0x17 to possibly play a sound effect and show “Insert Coins” message.	7F 05 12	The L8.4 code has this part of code call new function at $7F05,38 which will play L8.4 attract mode sound if applicable, and then call a function to schedule function 0x17 for pre L8.4 attract-mode sound, if applicable and then show “Insert Coin” message. The 0x12 is a NOP filler byte.
0x6062E	$462E,38	86 04 BD C7 91	When attract mode is being scheduled to start up such as at startup, menu exit or game-over this code is used to set the attract mode index 0x04 into function $C791 as normal part of attract mode scheduling.	BD 7F EC 20 00	The L8.4 code replaces this part of code with a jump to new L8.4 function $7FEC,38 which performs L8.4 attract mode memory setup for start-button and gun-trigger sound effects, and then it calls $C791 with 0x04 to schedule the attract mode.
0x63F05	$7F05,38	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Region of unused ROM bytes at the end of bank $38.	BD 86 5B 10 00 24 0A BD 82 FF 17 81 03 25 02 8D 03 7E 7F E4 34 12 BD 86 90 00 F2 24 36 8E 7F 65 86 0A BD A7 5B B1 06 45 27 F6 81 05 26 09 BD 86 5B 16 00 25 02 20 E9 B7 06 45 A6 86 BD BD FB 25 12 7C 06 43 B6 06 43 81 03 25 08 BD 8B 77 00 F2 7F 5A 38 35 92 BD 86 79 46 50 7F 06 43 7E 99 A2 A3 83 8B 8D 8E 92 93 97 7E 9B	This contains L8.4 code called when start-button is pressed when there is not enough credits to start a game. If Attract Sounds are On and Attract Mode is L8.4 this will play up to 3 sounds and then schedule a

					timeout period. Afterwards, code jumps to $7FE4,38 to continue with start-button handling, doing pre L8.4 attract sound, if needed, and showing “Insert Coin”.
0x63F6F	$7F6F,38	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Region of unused ROM bytes at the end of bank $38.	7D 17 93 26 04 8D 05 8D 58 7E 99 A2 BD 82 FF 17 81 03 25 37 BD 86 5B 10 00 24 30 BD 86 90 00 F3 24 29 8E 7F C6 86 0A BD A7 5B B1 06 46 27 F6 B7 06 46 A6 86 BD BD FB 25 12 7C 06 44 B6 06 44 81 03 25 08 BD 8B 77 00 F3 7F BB 38 39 BD 86 79 2A 30 7F 06 44 7E 99 A2 A1 AE 65 69 7C 7D 90 99 9E 9F BD 86 5B 9A 01 25 02 8D 01 39 86 01 B7 06 47 BD 85 53 17 39 7F 06 47 BD 85 53 17 39 7F 06 43 7F 06 44 86 FF B7 06 45 B7 06 46 86 04 BD C7 91 39	This contains L8.4 functions used for L8.4 attract mode. $7F6F,38 is the gun-trigger L8.4 handler to play sounds and schedule Tournament Mode score display. $7FE4,38 is the start-button handler in L8.4. $7FEC,38 is the attract mode init function for L8.4.
0x681D1	$41D1,3A	8D 74 8D 4E 24 08 8D 5B 25 3B 8D 63 24 F2	The WPC pre-menu messages reach this logic as game information is shown before entering menu.	7E 7A 0B 55 36 20 43 48 45 43 4B 53 55 4D	Logic is replaced with a jump to new L8.4 code at $7A0B,3A where the pre-menu will include the report of the game checksum. The bytes here are replaced with the static string “U6 CHECKSUM” for use by the pre-menu code.
0x6AD96	$6D96,3A	7A 0B	This was L8.3 code that was experimental during sound test after advancing past the database award music, this code would do a forced stop of all sounds to overcome a problem in certain 3rd party sound boards that didn’t otherwise stop playing the database music in the same way original WPC hardware would.	6D F3	In L8.4 the original L-8 code is restored here. There is no longer the experimental L8.3 code to force all sounds to stop when advancing the sound test past the database award music. This restores intended code as the experimental code was not intended to be part of L8.3. Refer to section “Removal of the L8-3 Sound 05 Explicit Stop” of this document for more details.
0x6BA0B	$7A0B,3A	34 02 BD 6D F3 A6 41 81 06 26 03 BD C0 A5 35 82 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	This was the helper function for the L8.3 experimental sound test music stop when advancing past	4D 26 15 BD 42 47 BD 42 23 24 0D BD 42 34 24 03 7E 42 16 BD 42 40 24 EB BD 7A 3B BD 42 23 24 0D BD 42 34 24 03 7E 42 16 BD 42 40 24 D6 7E 41 DF BD 42 4C 8D 03 8D 41 39 34 36 32 7B B6 FF EE 8D 1C A7 E4 E7 61 B6 FF EF	This contains the L8.4 pre-menu display of game checksum value . The entry point to this L8.4 code is at $7A0B,3A with supporing

		FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	database award music. This and unused ROM bytes near the end of bank $3A. The helper function is no longer used and is being replaced with L8.4 code as shown to the right.	8D 13 A7 62 E7 63 6F 64 31 E4 BD 8A DA 81 01 94 02 32 65 35 B6 34 02 8D 0B 1F 89 35 02 8D 01 39 44 44 44 44 84 0F 81 09 23 03 8B 37 39 8B 30 39 34 36 32 74 BD 7A A0 D6 11 31 E4 86 0B BD 91 39 6F 6B BD 8A DA 81 01 94 09 32 6C 35 B6 8E 41 D4 B6 17 4D 85 01 27 03 8E 7A AE 39 55 36 20 42 41 44 20 43 53 55 4D	function located immediately after.
0x6BAB9	$7AB9,3A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region near the end of bank $3A.	34 12 BD 86 90 00 E1 25 09 BD 86 5B 1C 00 27 09 20 23 BD 86 90 00 84 25 20 BD 84 AD 48 24 12 86 04 BD 86 90 00 88 25 09 4A 27 0A BD 83 46 01 20 F0 1C FA 20 04 1C FA 1A 04 35 92	The L8.4 hunter-ship hit determinator. Called from $4BCE,31, 0x44BCE as part of improved L8.4 hunter-ship hit determination. This includes the lookup of new L8.4 adjustment “Cannon 1 Hit” to decide if a secondary target hit can be included to allow multiple target lamps to be extinguished with the same, single, cannon shot.
0x6BAF4	$7AF4,3A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region near the end of bank $3A.	BD 84 49 30 24 18 BD 88 F5 5F 22 31 BD 88 F5 5E E4 31 24 08 8D 1F 27 04 BD 84 2B 30 20 16 BD 88 F5 5E E4 31 BD 88 F5 5F 22 31 24 08 8D 07 27 04 BD 84 2B 3A 39 34 12 8E 05 95 BD FB 29 A6 84 8E 05 99 BD FB 29 A1 84 35 92	The L8.4 adds this function for fixing the “Security Pass” problem where the lamps might be incorrectly lit if security pass award causes Payback Time to start. This is called from the $5ED4,31, 0x45ED4 ROM change shown above.
0x6BB3D	$7B3D,3A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region near the end of bank $3A.	34 10 8E 05 91 BD FB 29 6F 84 BD 87 BE 05 00 BD 84 2B 03 35 90	The L8.4 adds this function to ensure the bonus-x lamps are extinguished when a tilt occurs. This fixes problem where bonus-x lamps are not correctly lit after tilt.
0x6BB52	$7B52,3A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region near the end of bank $3A.	BD 79 E2 25 03 BD 7B 6D 39 34 06 BD 82 FF 95 81 02 22 04 1F 89 58 3A AE 84 35 86 34 36 1F 30 BD 7B C2 BD 7B 5B BD B9 51 35 B6 7B 94 7B 94 7B 9D 7B A7 7B AB 7B AF 7B B4 7B BD 7B B4 7B B8 7B BD 7B B8 4F 52 49 47 49 4E 41 4C 00 4F 52 49 47 49 4E 41 4C 45 00 4F 46 46 00 41 55 53 00 48 4F 52 53 00 73 65 63 00 73 65 63 73 00 53 65 6B 2E 00	This section of L8.4 code contains new code releated to the new L8.4 adjustment “SS Autofire Time”. This has the menu adjustment handler starting at $7B52,3A and the code that the skll shot code calls to

		FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF		C1 00 26 05 8E 7B 7C 20 34 C1 0B 26 05 8E 7B 82 20 2B C1 15 22 EE C1 0A 23 04 C0 0B 20 04 86 2B A7 A0 8E 7B 8E C1 01 26 03 8E 7B 88 C1 09 23 05 86 31 A7 A0 5F CA 30 E7 A0 86 20 A7 A0 39 34 06 BD 82 FF 1D 81 00 27 16 81 0B 27 1A 81 15 22 0E 81 0A 23 06 80 0B 1F 89 20 04 34 02 EB E0 BD 8B 77 00 83 6E 8A 31 35 86	initiate the autofire timer according to the adjustment value, starting at $7C00,3A.
0x6BC2A	$7C2A,3A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region near the end of bank $3A.	BD 79 E2 25 02 8D 01 39 34 36 1F 30 8D 08 BD 7B 5B BD B9 51 35 B6 C1 0C 25 01 5F 86 06 3D 8E 7C 4E 30 8B 39 7B 94 7B 94 7B 9D 7C 96 7C 96 7C 96 7C 9E 7C 9E 7C 9E 7C A6 7C AD 7C B4 7C BA 7C C3 7C CC 7C D4 7C DD 7C E6 7C EE 7C F3 7C FA 7D 04 7D 0B 7D 14 7D 1E 7D 25 7D 2E 7D 38 7D 3F 7D 3F 7D 47 7D 50 7D 50 7D 5A 7D 63 7D 63 4F 52 49 47 2E 2B 31 00 4F 52 49 47 2E 2B 2B 00 4D 45 44 49 55 4D 00 4D 49 54 54 45 4C 00 4D 4F 59 45 4E 00 4D 45 44 49 55 4D 2B 31 00 4D 49 54 54 45 4C 2B 31 00 4D 4F 59 45 4E 2B 31 00 4D 45 44 49 55 4D 2B 2B 00 4D 49 54 54 45 4C 2B 2B 00 4D 4F 59 45 4E 2B 2B 00 48 41 52 44 00 53 43 48 57 45 52 00 44 49 46 46 49 43 49 4C 45 00 48 41 52 44 2B 31 00 53 43 48 57 45 52 2B 31 00 44 49 46 46 2F 4C 45 2B 31 00 48 41 52 44 2B 2B 00 53 43 48 57 45 52 2B 2B 00 44 49 46 46 2F 4C 45 2B 2B 00 45 58 50 45 52 54 00 45 58 50 45 52 54 45 00 45 58 50 45 52 54 2B 31 00 45 58 50 45 52 54 45 2B 31 00 45 58 50 45 52 54 2B 2B 00 45 58 50 45 52 54 45 2B 2B 00 18 A3 18 EB 18 F1 18 F7 18 FD 19 21 19 2D 19 39 19 45 19 51 19 5D 19 69 19 75 19 81 19 8D 19 99 19 A5 19 B1 00 00 18 EB 18 F1 18 F7 18 FD 19 27 19 33 19 3F 19 4B 19 57 19 63 19 6F 19 7B 19 87 19 93 19 9F 19 AB 19 B7 00 00 18 EB 18 F1 18 F7 18 FD 19 21 19 27 19 39 19 3F 19 51 19 57 19 69 19 6F 19 81 19 87 19 99 19 9F 19 B1 19 B7 00 00 4F 10 AE 81 27 04 AB A4 20 F7 8D 59 AB 84 39 8D EF 8E FD 00 A6 86 39 8E 7D 6D 8D F3 84 07 81 05 25 02 80 04 39 8E 7D B7 8D E5 39 8D F8 46 46 46 46 20 E9 8E 7D 93 8D D7 84 01 39 8D E8 84 03 81 03 25 02 80 02 39 86 04 C1 03 25 02 8D C9 81 05 25 02 86 04 8D 01 39 34 04 B7 05 F9 8B 11 C6 18 BD 9E 7F 35 84 8E 05 D1 BD FB 29 39 C1	This section of L8.4 code contains new code related to the L8.4 super-jackpot enhancements. Notable portions are: $7C2A,35 is entry point for the new “Super Jackpot” adjustment handler, showing the various adjustment values. From $7D6D,35 through $7E94,35 is code related to the faux “random” values used in super jackpot to ensure, in fairness, all players in a multi player game get the same super jackpot experience (regardless if Tournament mode or not). $7E95,35 is the entry point to the L8.4 super jackpot lamp movement maintenance during the super jackpot attempt. From $7EC1,35 through $7F47,35 is the code for each of the different super-jackpot lamp movement patterns.

		FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF		01 27 10 C1 04 27 0C C1 07 27 08 C1 0A 27 04 1C FE 20 02 1A 01 39 C1 02 27 10 C1 05 27 0C C1 08 27 08 C1 0B 27 04 1C FE 20 02 1A 01 39 34 10 8D CE 24 08 8D C3 A6 84 27 0C 20 04 8D D9 24 06 BD 83 46 10 20 04 BD 83 46 14 35 90 86 FF 97 D3 BD 83 0C 1E 8D 84 C1 06 24 02 20 1C C1 09 24 05 BD 7E 18 20 03 BD 7E 08 81 04 27 74 81 03 27 6B 81 02 27 35 81 01 27 49 B6 05 F9 27 19 81 04 24 05 BD 7E 10 27 10 8D A6 BD FD B4 18 18 7A 05 F9 B6 05 F9 4D 26 F0 8D 96 BD FD A2 18 18 7C 05 F9 B6 05 F9 81 04 25 EF 20 DD 8D 83 BD FD B4 18 18 B6 05 F9 26 07 86 04 B7 05 F9 20 03 7A 05 F9 20 E8 BD 7E 77 BD FD A2 18 18 B6 05 F9 81 04 25 06 4F B7 05 F9 20 03 7C 05 F9 20 E6 8E 7F 3E 20 03 8E 7F 43 BD 7E 77 BD 87 BE 18 18 B6 05 F9 A6 86 BD 7E 34 20 EE 02 03 04 00 01 03 04 00 01 02	
0x6BF48	$7F48,3A	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region near the end of bank $3A.	7D 06 47 26 29 BD 86 5B 10 00 24 0A BD 82 FF 17 81 03 24 02 8D 09 BD 88 F5 7B 74 30 7E C9 52 BD 84 AD DC 25 08 BD 85 46 51 BD 84 8F DC 39 BD FB AE BD 88 F5 6A F4 3B BD E2 74 BD 86 79 04 B0 7E C9 52	This section of code is the L8.4 new function handler for indexed function 0x17. This is called from attract mode when start-button is pressed (with < 1 credit) or when gun-trigger is pressed when Tournament Mode is enabled. For start-button it optionally plays sound effect when attract mode is less than L8.4 and always shows “Insert Coin” message. For gun-trigger it always shows previously played game scores. The attract mode restarts when this code completes.
0x6E273	$6273,3B	B1 D1	Code during ball-start when playfied state is being set up for the current player/current ball. This is address of the $B1D1 function that is called to determine if the player is doing poorly and is deserving of an “Extra ball is lit”. The $B1D1 returns C-bit	78 DA	The L8.4 replaces the function call to a new L8.4 function at $78DA,3B which will include the “Max E.B. Count” adjustment in the determination of whether player is given “Extra ball is lit”. This new function also calls the original $B1D1 function in making the determination. The

			clear if player is to be given “Extra ball is lit”.		$78DA,3B function returns C-bit clear if player is to be given “Extra ball is lit”.
0x6E2F4	$62F4,3B	8E 05 91 BD FB 29 7E 62 FD 6F 84	Code during ball-start when playfield state is being set up for the current player/current ball. This is the portion of code that clears the current player’s bonus-X value as past of ordinary start of ball.	BD 88 F5 7B 3D 3A 7E 62 FF 12 12	The L8.4 code has this code call a new L8.4 function at $7B3D,3A which will perform the same L-8 logic of clearing the player’s bonus-X value and, new in L8.4 also extinguish the bonus-X lamps. This is part of fix to ensure bonus-X lamps are not incorrectly lit after a tilt.
0x6F8DA	$78DA,3B	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Start of region of unused ROM bytes in bank $3B.	BD 86 5B 83 00 25 04 1A 01 20 03 BD B1 D1 39	This is the new L8.4 function called from $6273,3B, 0x6E273 for the ball-start code to check if player can be given “Extra ball is lit” if they are doing poorly. The new L8.4 logic ensures the player is not given such consideration if the “Max E.B. Count” adjustment is set for no extra balls.
0x73FB7 **0x73FC4** **0x73FD2** **0x73FE0** **0x73FEF** **0x73FFE**	$7FB7,3C $7FC4,3C $7FD2,3C $7FE0,3C $7FEF,3C $7FFE,3C	33	These are 3 occurrences of ‘3’ in fixed version strings for on-display report of L8.3 in various languages and release/prototype designators.	34	The L8.4 simply updates these 6 occurrences to ‘4’ for L8.4.
0x74A6A	$4A6A,3D	5D 0A	In the switch-matrix data table this is the callback address for gun-trigger switch closures. When gun-trigger switch is closed this tells the s/w to call function $5D0A,31.	7F 73	The L8.4 code updates the switch-matrix data table to call new L8.4 function at $7F73,31 so that enhanced logic can be implemented to allow L8.4 attract mode to play sounds and to show previous game scores if Tournament Mode is enabled.
0x74A70	$4A70,3D	40	In the switch-matrix data table this is a flag byte in gun-trigger data. The 0x40 bit indicates	50	For L8.4 code this flag byte adds the 0x10 bit, making the resulting value 0x50. The 0x01 bit indicates the

			the switch is value and its callback address may be called when switch is closed.		gun-trigger's callback function may be called when switch is closed during attract mode.
0x75D92	$5D92,3D	7D DD 30	This is a function callback data table entry data for index 0x17. The 0x17 indexed function is called when start-button is pressed during attract mode when there are not enough credits to start a game. This causes function $7DDD,30 to be called which can optionally play a sound effect and then show the "Insert Coin" message.	7F 48 3A	For L8.4 the 0x17 function address is changed to new L8.4 function $7F48,3A. The new function is designed to be called from L8.4 code for start-button or from gun-trigger. This allows the attract mode to be interrupted by the start-button "Insert Coin" message or from gun-trigger display of previous game scores when Tournament Mode is enabled.
0x7658B	$658B,3D	65 AB 11 83 00 00 26 04 8E 65 B6 39 11 83 00 01 26 04 8E 65 BB 39 11 83 00 02 26 03 8E 65 C0 39 4D 45 4E 55 20 45 52 52 4F 52 00 4C 38 2E 31 00 4C 38 2E 32 00 4C 38 2E 33 00	The L8.3 code for displaying "Attract Mode" adjustment values, this has code and the display strings.	66 E0 11 83 00 00 26 04 8E 66 EB 39 11 83 00 01 26 04 8E 66 F0 39 11 83 00 02 26 04 8E 66 F5 39 11 83 00 03 26 03 8E 66 FA 39 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	For L8.4 the "Attract Mode" adjustment includes "L8.4" and this code is expanded to check for the new adjustment value. Space is not available here for the new "L8.4" string so all adjustment atrings are moved away from this section and the now unused bytes are set to 0xFF.
0x766E0	$66E0,3D	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region in bank $3D.	4D 45 4E 55 20 45 52 52 4F 52 00 4C 38 2E 31 00 4C 38 2E 32 00 4C 38 2E 33 00 4C 38 2E 34 00	For L8.4 the adjustment strings for "Attract Mode" are moved here with the new "L8.4" string added.
0x76701	$6701,3D	1C	At the start of the L8.3 Feature Adjustment English strings table, this byte defines the number of strings.	1F	For L8.4 3 new feature adjustments were added so this value is increased by 3.
0x7673B	$673B,3D	67 43 67 43 67 43	Feature Adjustment English strings table, three placeholder, unused entries.	68 EB 68 F8 69 09	The three new adjustment strings added to the English strings table. These are addresses to the start of the strings in $3D. This refers to $68EB,3D,

					$68F8,3D, and $6909,3D.
0x768EB	$68EB,3D	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region in bank $3D.	43 41 4E 4E 4F 4E 20 31 20 48 49 54 00 53 53 20 41 55 54 4F 46 49 52 45 20 54 49 4D 45 00 53 55 50 45 52 20 4A 41 43 4B 50 4F 54 00	Three new strings for English feature adjustments. $68EB,3D "CANNON 1 HIT", $68F8,3D "SS AUTOFIRE TIME", $6909,3D "SUPER JACKPOT".
0x76A01	$6A01,3D	1C	At the start of the L8.3 Feature Adjustment German strings table, this byte defines the number of strings.	1F	For L8.4 3 new feature adjustments were added so this value is increased by 3.
0x76A3B	$6A3B,3D	67 43 67 43 67 43	Feature Adjustment German strings table, three placeholder, unused entries.	6B 7F 6B 90 69 09	The three new adjustment strings added to the German strings table. These are addresses to the start of the strings in $3D. This refers to $6B7F,3D, $6B90,3D, and $6909,3D. Note the third string points to the existing string defined in the English table as the same string is used for both languages.
0x76B7F	$6B7F,3D	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region in bank $3D.	4B 41 4E 4F 4E 45 20 31 20 54 52 45 46 46 45 52 00 53 53 20 41 55 54 4F 46 49 52 45 20 5A 45 49 54 00	Two new strings for German feature adjustments. $6B7F,3D "KANONE 1 TREFFER", $6B90,3D "SS AUTOFIRE ZEIT".
0x76D01	$6D01,3D	1C	At the start of the L8.3 Feature Adjustment French strings table, this byte defines the number of strings.	1F	For L8.4 3 new feature adjustments were added so this value is increased by 3.
0x76D3B	$6D3B,3D	67 43 67 43 67 43	Feature Adjustment French strings table, three placeholder, unused entries.	6E D4 6E E1 69 09	The three new adjustment strings added to the French strings table. These are addresses to the start of the strings in $3D. This refers to $6ED4,3D, $6EE1,3D, and $6909,3D. Note the third string points to the existing string defined in the English table as the same string is used

					for both languages.
0x76ED4	$6ED4,3D	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused region in bank $3D.	43 41 4E 4F 4E 20 31 20 43 4F 55 50 00 53 53 20 44 55 52 2E 20 41 55 54 4F 46 49 52 45 00	Two new strings for French feature adjustments. $6ED4,3D “CANON 1 COUP”, $6EE1,3D “SS DUR. AUTOFIRE”.
0x77001	$7001,3D	1C	Feature Adjustments metadata table, this byte defines the number of entries.	1F	For L8.4 3 new feature adjustments were added so this value is increased by 3.
0x77118	$7118,3D	02	The “Attract Mode” entry in Feature Adjustments metadata table, this defines the default adjustment value 02 for “L8.3”.	03	In L8.4 the default value for this adjustment changes to 03 for “L8.4”.
0x7711C	$711C,3D	02	The “Attract Mode” entry in Feature Adjustments metadata table, this defines the maximum value the setting can take. 02 for “L8.3”.	03	In L8.4 the maximum value for this adjustment changes to 03 for “L8.4”.
0x77158	$7158,3D	00 00 01 00 8E BC FF	Placeholder data in ROM for unused adjustment A2.28.	01 74 13 3A 72 70 3A	For L8.4 the adjustment A2.28 is enabled for “Cannon 1 Hit”. These byte changes enable the setting using existing string handlers to represent the “ON” and “OFF” option values.
0x77164	$7164,3D	00 00 01 00 8E BC FF	Placeholder data in ROM for unused adjustment A2.29.	15 00 01 00 7B 52 3A	For L8.4 the adjustment A2.29 is enabled for “SS Autofire Time”. These byte changes enable the setting and defines $7B52,3A, new L8.4 code for handling the display of adjustment values.
0x77170	$7170,3D	00 00 01 00 8E BC FF	Placeholder data in ROM for unused adjustment A2.30.	0B 00 01 00 7C 2A 3A	For L8.4 the adjustment A2.30 is enabled for “Super Jackpot”. These byte changes enable the setting and defines $7C2A,3A, new L8.4 code for handling the display of adjustment values.

0x7FFEE	$FFEE	73 08	The L8.3 WPC Checksum value 7308.	9A 08	The L8.4 WPC Checksum value 9A08

Corrections to the L8.3 Document

During the L8.4 development, a few typographical errors in the L8.3 document were observed and recorded here for completeness.

Page	Info
79	ROM offset of moved adj tableis shown as 0x77680 but should be 0x77000
92	Code depicted as `7A12: 81 02 should be depicted as: 81 06`
182	ROM change 0x6AD95 should describe T05 is failing to stop, not T06.

L8.4 Test/Verification

Testing of the L8.4 image was done to ensure all scenarios behave as expected. Each major set of changes provided in each beta L8.4 ROM image were carefully constructed and thoroughly tested.

As each code change was developed, it was ran through with an emulator, single stepping through the new code to ensure flow is as intended. This includes all logic paths the new code can take. Once the code flow is confirmed, then game play in the emulator is done to ensure the new code works as expected.

Beta testing was done on all bug fixes and improvements. Each beta ROM image is provided with a set of instructions and areas to test with focus being on the new elements added. Testing of the release candidate was done to ensure proper and expected behavior prior to releasing the ROM image for general distribution.

Once Again, a big THANK YOU for each and every beta tester. We appreciate the time and effort you have put forth in making L8.4 possible!

Any/All bug reports have been carefully examined and applied toward making the L8.4 toward its final release.

During L8.4 development this technical document was created in parallel so that technical details and code analysis can be double checked as the documentation is being created. This technique serves as a way to preserve quality and consistency of the code changes.

Prior to L8.4 release, the analysis of each and every ROM change was done (and documented, above) to ensure no inadvertent changes were included. In this exercise no inadvertent code changes were found.

Lastly, several passes were made through this document with focus on double-checking the depicted code changes, ensuring correct and intended changes throughout.