WPC Transistors

Document History

February 27, 2012, initial draft. Basics on WPC transistor driver code including both solenoids and flashers. Brief mention of Fliptronic II solenoids, IRQ Interrupt routine, and main loop housekeeping function.

Document Scope

This document is an informal write-up on what I've discovered about WPC transistor code. The audience of this document is other s/w developers who want some details about WPC transistor code, including some details on how transistors get energized and how the solenoid and flasher code are done differently.

This document does not cover the solenoid and flasher test mode code. Such code will be described in the future when the WPC menu system is detailed in a separate document.

The examples shown here are from IJ_L7. The specific memory addresses are going to be different on other pins, so you need to search the ROM for other pins with matching non-address bytes (ie opcodes and data struct patterns) to determine the equivalent code and addresses used in your game.

The ordering of the information in this document is a mix of the following two concepts:

- Present simple data items first, and then start showing more complex functions as the document progresses.
- Present code in the same order as it resides in the IJ_L7 ROM.

Disclaimer

This information is for educational and entertainment purposes only. Some of the interpretation about code may be incorrect so take some of what is presented with a grain of salt. In fact, the code comments shown in this document have been made over the course of several years. Some parts of the code were commented poorly as little was known about the code being annotated. I try to clean this up as I go.

Exercise caution when modifying ROM images as they could have real physical effects which may be undesirable, especially if you modify code that causes hardware components to operate outside of their specifications. Since this document describes transistor on/off code, you should be particularly aware of the delicate nature of the code and carefully consider the ramifications of making any code changes in the areas depicted.

<u>TransistorTable[]</u>

Let's start with the most generic piece of information regarding the WPC transistors. Below is the TransistorTable[], it contains a 5-byte table entry for every transistor that the s/w will be able to control. This includes transistors for:

- Solenoids
- Flashers
- Motors/Features
- Extra switch columns
- Extra lamp columns

There is no particular requirement that the WPC software always explicitly extract values from the TransistorTable[], however for the most part, this table *is* referenced. For some things other than solenoids and flashers, it's not surprising to find code that directly uses values instead of pulling them out of the TransistorTable[]. For example code that uses a transistor to drive a 9th switch column would just directly read/write to the h/w register associated with the 9th switch column instead of pulling the h/w register value out of the TransistorTable[].

Each TransistorTable[] entry is 5-bytes in length, representing:

- 16-bit pointer to address in WPC ram which caches the transistor state (single bit per transistor, 8 transistors per byte)
- 16-bit pointer to the h/w register that WPC code needs to update to actually update the physical state of the transistor.
- 8-bit mask representing the transistor (stored as all 1's except for single 0-bit representing the transistor).

Each transistor is represented as a single bit within an 8-bit field. The 8-bit field represents 8 different transistors so it's important to read-modify-write so that only the desired transistor bit is modified. A 1-bit represents an "on" transistor and a 0-bit represents an "off" transistor. The cached byte in ram is usually updated with the new bit-state for a transistor and then its value is written thru to the h/w which updates the transistor state.

The TransistorTable[] stores the 8-bit mask for the transistor as a 0-bit. This means that in order to turn ON the transistor, the 8-bit mask needs to be inversed and then ORed into the cache byte and into the h/w register. To turn OFF the transistor the 8-bit mask simply needs to be ANDed with the cache byte and then pushed thru to the h/w register.

Below is the TransistorTable[] from IJ_L7, bank 0x22, for reference. Later parts of this document will reveal how you should be able to find the TransistorTable[] in your game.

```
;---------------------------------------;----------------------------------------------------------
; TransistorTable[]
;
; Table entry for reach transistor.
; Each table entry contains:
;  16-bit address in RAM where the transistor's cached status is stored.
;  16-bit address of the h/w register for this transistor.
;  8-bit AND mask representing this transistor, 0-bit indicates the transistor.
;
7C2B: 00 2D                            ; There are 0x002D (45) entries in the table
7C2D: 05                               ; Each table entry is 5 bytes in length
                                       ;
                                       ;
                                       ; TransistorTable00[], NULL
                                       ; ------------------------------------------------
7C2E: 00 7B                            ; NULL Cache byte, 0x7B
7C30: 00 7B                            ; NULL Cache byte, 0x7B
7C32: FF                               ; 0xFF, NULL
                                       ;
                                       ;
                                       ; TransistorTable01[], Q82/Q81/Q71
                                       ; ------------------------------------------------
7C33: 00 7C                            ; Cache Byte:     0x7C
7C35: 3F E1                            ; H/w Register: 0x3FE1, WPC_TRANSISTOR1
7C37: FE                               ; Mask:           0xFE --> 11111110
                                       ;
                                       ;
                                       ; TransistorTable02[], Q80/Q79/Q74
                                       ; ------------------------------------------------
7C38: 00 7C                            ; Cache Byte:     0x7C
7C3A: 3F E1                            ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C3C: FD                               ; Mask:           0xFD --> 11111101
                                       ;
                                       ;
                                       ; TransistorTable03[], Q78/Q77/Q72
                                       ; ------------------------------------------------
7C3D: 00 7C                            ; Cache Byte:     0x7C
7C3F: 3F E1                            ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C41: FB                               ; Mask:           0xFB --> 11111011
                                       ;
```

```
                                        ;
                                        ; TransistorTable04[], Q76/Q75/Q73
                                        ; ------------------------------------------------
7C42: 00 7C                             ; Cache Byte:     0x7C
7C44: 3F E1                             ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C46: F7                                ; Mask:           0xF7 --> 11110111
                                        ;
                                        ;
                                        ; TransistorTable05[], Q54/Q63/Q61
                                        ; ------------------------------------------------
7C47: 00 7C                             ; Cache Byte:     0x7C
7C49: 3F E1                             ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C4B: EF                                ; Mask:           0xEF --> 11101111
                                        ;
                                        ;
                                        ; TransistorTable06[], Q56/Q65/Q60
                                        ; ------------------------------------------------
7C4C: 00 7C                             ; Cache Byte:     0x7C
7C4E: 3F E1                             ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C50: DF                                ; Mask:           0xDF --> 11011111
                                        ;
                                        ;
                                        ; TransistorTable07[], Q68/Q57/Q52
                                        ; ------------------------------------------------
7C51: 00 7C                             ; Cache Byte:     0x7C
7C53: 3F E1                             ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C55: BF                                ; Mask:           0xBF --> 10111111
                                        ;
                                        ;
                                        ; TransistorTable08[], Q70/Q69/Q59
                                        ; ------------------------------------------------
7C56: 00 7C                             ; Cache Byte:     0x7C
7C58: 3F E1                             ; H/W Register: 0x3FE1, WPC_TRANSISTOR1
7C5A: 7F                                ; Mask:           0xBF --> 01111111
                                        ;
                                        ;
                                        ;
                                        ; TransistorTable09[], Q58/Q57
                                        ; ------------------------------------------------
7C5B: 00 7E                             ; Cache Byte:     0x7E
7C5D: 3F E3                             ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C5F: FE                                ; Mask:           0xFE --> 11111110
                                        ;
                                        ;
                                        ; TransistorTable0A[], Q56/Q55
```

```
                                            ;- ---------------------------------------------
7C60: 00 7E                                 ; Cache Byte:     0x7E
7C62: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C64: FD                                    ; Mask:           0xFD --> 11111101
                                            ;
                                            ;
                                            ; TransistorTable0B[], Q54/Q53
                                            ; ---------------------------------------------
7C65: 00 7E                                 ; Cache Byte:     0x7E
7C67: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C69: FB                                    ; Mask:           0xFB --> 11111011
                                            ;
                                            ;
                                            ; TransistorTable0C[], Q52/Q51
                                            ; ---------------------------------------------
7C6A: 00 7E                                 ; Cache Byte:     0x7E
7C6C: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C6E: F7                                    ; Mask:           0xF7 --> 11110111
                                            ;
                                            ;
                                            ; TransistorTable0D[], Q50/Q49
                                            ; ---------------------------------------------
7C6F: 00 7E                                 ; Cache Byte:     0x7E
7C71: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C73: EF                                    ; Mask:           0xEF --> 11101111
                                            ;
                                            ;
                                            ; TransistorTable0E[], Q48/Q47
                                            ; ---------------------------------------------
7C74: 00 7E                                 ; Cache Byte:     0x7E
7C76: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C78: DF                                    ; Mask:           0xDF --> 11011111
                                            ;
                                            ;
                                            ; TransistorTable0F[], Q46/Q45
                                            ; ---------------------------------------------
7C79: 00 7E                                 ; Cache Byte:     0x7E
7C7B: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
7C7D: BF                                    ; Mask:           0xBF --> 10111111
                                            ;
                                            ;
                                            ; TransistorTable10[], Q44/Q43
                                            ; ---------------------------------------------
7C7E: 00 7E                                 ; Cache Byte:     0x7E
7C80: 3F E3                                 ; H/W Register: 0x3FE3, WPC_TRANSISTOR2
```

```
7C82: 7F                                    ; Mask:           0x7F --> 01111111
                                            ;
                                            ;
                                            ; TransistorTable11[], Q42/Q41
                                            ; ---------------------------------------------
7C83: 00 7D                                 ; Cache Byte:     0x7D
7C85: 3F E2                                 ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7C87: FE                                    ; Mask:           0xFE --> 11111110
                                            ;
                                            ;
                                            ; TransistorTable12[], Q40/Q39
                                            ; ---------------------------------------------
7C88: 00 7D                                 ; Cache Byte:     0x7D
7C8A: 3F E2                                 ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7C8C: FD                                    ; Mask:           0xFD --> 11111101
                                            ;
                                            ;
                                            ; TransistorTable13[], Q38/Q37
                                            ; ---------------------------------------------
7C8D: 00 7D                                 ; Cache Byte:     0x7D
7C8F: 3F E2                                 ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7C91: FB                                    ; Mask:           0xFB --> 11111011
                                            ;
                                            ;
                                            ; TransistorTable14[], Q36/Q35
                                            ; ---------------------------------------------
7C92: 00 7D                                 ; Cache Byte:     0x7D
7C94: 3F E2                                 ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7C96: F7                                    ; Mask:           0xF7 --> 11110111
                                            ;
                                            ;
                                            ; TransistorTable15[], Q28/Q27
                                            ; ---------------------------------------------
7C97: 00 7D                                 ; Cache Byte:     0x7D
7C99: 3F E2                                 ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7C9B: EF                                    ; Mask:           0xEF --> 11101111
                                            ;
                                            ;
                                            ; TransistorTable16[], Q30/Q29
                                            ; ---------------------------------------------
7C9C: 00 7D                                 ; Cache Byte:     0x7D
7C9E: 3F E2                                 ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7CA0: DF                                    ; Mask:           0xDF --> 11011111
                                            ;
                                            ;
```

```
                                          ; TransistorTable17[], Q34/Q33
                                          ; ---------------------------------------------
7CA1: 00 7D                               ; Cache Byte:     0x7D
7CA3: 3F E2                               ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7CA5: BF                                  ; Mask:           0xBF --> 10111111
                                          ;
                                          ;
                                          ; TransistorTable18[], Q32/Q31
                                          ; ---------------------------------------------
7Ca6: 00 7D                               ; Cache Byte:     0x7D
7CA8: 3F E2                               ; H/W Register: 0x3FE2, WPC_TRANSISTOR3
7CAA: 7F                                  ; Mask:           0x7F --> 01111111
                                          ;
                                          ;
                                          ; TransistorTable19[], Q26/Q25
                                          ; ---------------------------------------------
7CAB: 00 7B                               ; Cache Byte:     0x7B
7CAD: 3F E0                               ; H/W Register: 0x3FE0, WPC_TRANSISTOR4
7CAF: FE                                  ; Mask:           0xFE --> 11111110
                                          ;
                                          ;
                                          ; TransistorTable1A[], Q24/Q23
                                          ; ---------------------------------------------
7CB0: 00 7B                               ; Cache Byte:     0x7B
7CB2: 3F E0                               ; H/W Register: 0x3FE0, WPC_TRANSISTOR4
7CB4: FD                                  ; Mask:           0xFD --> 11111101
                                          ;
                                          ;
                                          ; TransistorTable1B[], Q22/Q21
                                          ; ---------------------------------------------
7CB5: 00 7B                               ; Cache Byte:     0x7B
7CB7: 3F E0                               ; H/W Register: 0x3FE0, WPC_TRANSISTOR4
7CB9: FB                                  ; Mask:           0xFB --> 11111011
                                          ;
                                          ;
                                          ; TransistorTable1C[], Q20/Q19
                                          ; ---------------------------------------------
7CBA: 00 7B                               ; Cache Byte:     0x7B
7CBC: 3F E0                               ; H/W Register: 0x3FE0 , WPC_TRANSISTOR4
7CBE: F7                                  ; Mask:           0xF7 --> 11110111
                                          ;
                                          ;
                                          ; TransistorTable1D[], Q4/Q12/Q20 (Fliptronic II)
                                          ; ---------------------------------------------
7CBF: 00 A0                               ; Cache Byte:     0xA0
```

```
7CC1: 04 F2                               ; H/W Register: 0x04F2
7CC3: FE                                  ; Mask:           0xFE --> 11111110
                                          ;
                                          ;
                                          ; TransistorTable1E[], Q11/Q19 (Fliptronic II)
                                          ; ------------------------------------------------
7CC4: 00 A0                               ; Cache Byte:     0xA0
7CC6: 04 F2                               ; H/W Register: 0x04F2
7CC8: FD                                  ; Mask:           0xFD --> 11111101
                                          ;
                                          ;
                                          ; TransistorTable1F[], Q3/Q10/Q18 (Fliptronic II)
                                          ; ------------------------------------------------
7CC9: 00 A0                               ; Cache Byte:     0xA0
7CCB: 04 F2                               ; H/W Register: 0x04F2
7CCD: FB                                  ; Mask:           0xFB --> 11111011
                                          ;
                                          ;
                                          ; TransistorTable20[], Q9/Q17 (Fliptronic II)
                                          ; ------------------------------------------------
7CCE: 00 A0                               ; Cache Byte:     0xA0
7CD0: 04 F2                               ; H/W Register: 0x04F2
7CD2: F7                                  ; Mask:           0xF7 --> 11110111
                                          ;
                                          ;
                                          ; TransistorTable21[], Q2/Q8/Q16 (Fliptronic II)
                                          ; ------------------------------------------------
7CD3: 00 A0                               ; Cache Byte:     0xA0
7CD5: 04 F2                               ; H/W Register: 0x04F2
7CD7: EF                                  ; Mask:           0xEF --> 11101111
                                          ;
                                          ;
                                          ; TransistorTable22[], Q7/Q15 (Fliptronic II)
                                          ; ------------------------------------------------
7CD8: 00 A0                               ; Cache Byte:     0xA0
7CDA: 04 F2                               ; H/W Register: 0x04F2
7CDC: DF                                  ; Mask:           0xDF --> 11011111
                                          ;
                                          ;
                                          ; TransistorTable23[], Q1/Q6/Q14 (Fliptronic II)
                                          ; ------------------------------------------------
7CDD: 00 A0                               ; Cache Byte:     0xA0
7CDF: 04 F2                               ; H/W Register: 0x04F2
7CE1: BF                                  ; Mask:           0xBF --> 10111111
                                          ;
```

```
                                       ;
                                       ; TransistorTable24[], Q5/Q13 (Fliptronic II)
                                       ; ---------------------------------------------
7CE2: 00 A0                            ; Cache Byte:     0xA0
7CE4: 04 F2                            ; H/W Register: 0x04F2
7CE6: 7F                               ; Mask:           0x7F --> 01111111
                                       ;
                                       ;
                                       ; TransistorTable25[], Q16/Q8 (8-Driver Board)
                                       ; ---------------------------------------------
7CE7: 00 ED                            ; Cache Byte:     0xED
7CE9: 3F EB                            ; H/W Register: 0x3FEB
7CEB: FE                               ; Mask:           0xFE --> 11111110
                                       ;
                                       ;
                                       ; TransistorTable26[], Q15/Q7 (8-Driver Board)
                                       ; ---------------------------------------------
7CEC: 00 ED                            ; Cache Byte:     0xED
7CEE: 3F EB                            ; H/W Register: 0x3FEB
7CF0: FD                               ; Mask:           0xFD --> 11111101
                                       ;
                                       ;
                                       ; TransistorTable27[], Q14/Q6 (8-Driver Board)
                                       ; ---------------------------------------------
7CF1: 00 ED                            ; Cache Byte:     0xED
7CF3: 3F EB                            ; H/W Register: 0x3FEB
7CF5: FB                               ; Mask:           0xFB --> 11111011
                                       ;
                                       ;
                                       ; TransistorTable28[], Q13/Q5 (8-Driver Board)
                                       ; ---------------------------------------------
7CF6: 00 ED                            ; Cache Byte:     0xED
7CF8: 3F EB                            ; H/W Register: 0x3FEB
7CFA: F7                               ; Mask:           0xF7 --> 11110111
                                       ;
                                       ;
                                       ; TransistorTable29[], Q9/Q4 (8-Driver Board)
                                       ; ---------------------------------------------
7CFB: 00 ED                            ; Cache Byte:     0xED
7CFD: 3F EB                            ; H/W Register: 0x3FEB
7CFF: EF                               ; Mask:           0xEF --> 11101111
                                       ;
                                       ;
                                       ; TransistorTable2A[], Q10/Q3 (8-Driver Board)
                                       ; ---------------------------------------------
```

```
7CD0: 00 ED                             ; Cache Byte:     0xED
7D02: 3F EB                             ; H/W Register: 0x3FEB
7D04: DF                                ; Mask:           0xDF --> 11011111
                                        ;
                                        ;
                                        ; TransistorTable2B[], Q11/Q2 (8-Driver Board)
                                        ; ----------------------------------------------
7D05: 00 ED                             ; Cache Byte:     0xED
7D07: 3F EB                             ; H/W Register: 0x3FEB
7D09: BF                                ; Mask:           0xBF --> 10111111
                                        ;
                                        ;
                                        ; TransistorTable2C[], Q12/Q1 (8-Driver Board)
                                        ; ----------------------------------------------
7D0A: 00 ED                             ; Cache Byte:     0xED
7D0C: 3F EB                             ; H/W Register: 0x3FEB
7D0E: 7F                                ; Mask:           0x7F --> 01111111
                                        ;
;---------------------------------------;------------------------------------------------------
```

SolenoidTable[]

Next is the SolenoidTable[]. This table contains multiple entries which the game s/w may use to pulse a particular solenoid for a particular period of time. Each SolenoidTable[] entry is 2 bytes in length, consisting of:

- 8-bit TransistorTable[] index to which the solenoid is associated.
- 8-bit Pulse-time byte.

The pulse time (also referred to as "energization time" and other phrases in the code comments) can have special meaning. Details on the pulse-time will be described in detail later in this document, although in the comments below give us a sneak peek into the different ways the pulse-time value can be used for determining how long the s/w will keep the transistor on.

It is also important to note that the same TransistorTable[] index may appear in multiple SolenoidTable[] entries. This allows the table to hold different pulse-times for the same transistor. The game s/w must find it handy to do this for certain situations. TBD but one example might be that the game uses longer pulses during a ball search.

```
;----------------------------------------;----------------------------------------------------------
; SolenoidTable[]
;
; Each table entry is 2 bytes:
;  Byte 1 is the TransistorTable[] index value and also the index number shown on the screen
;           during test mode for the particular solenoid.
;  Byte 2 is timer value for how long to energize solenoid during test mode, with special meanings:
;         0x00 == turn transistor off
;         0xff == turn transistor on
;         0x01 == special long delay, using SolenoidSpecialDelay[0] table entry
;         0x02 == special long delay, using SolenoidSpecialDelay[1] table entry
;         0x03 == special long delay, using SolenoidSpecialDelay[2] table entry
;         0x04 == special long delay, using SolenoidSpecialDelay[3] table entry
;         0x05 == special long delay, using SolenoidSpecialDelay[4] table entry
;         0x06 == special long delay, using SolenoidSpecialDelay[5] table entry
;         0x07-0xF0 == actual time value used used for the pulse time
;         0xFx in high nibble, then actual time is low-nibble * 30 (0x1e)
;
41B0: 00 35                             ; Table entries
41B2: 02                                ; Bytes per entry
                                        ;
```

```
41B3: 00 00                          ; SolenoidTableEntry00, NULL
41B5: 01 20                          ; SolenoidTableEntry01, Ball Popper, 20
41B6: 01 40                          ; SolenoidTableEntry02, Ball Popper, 40
41B9: 04 40                          ; SolenoidTableEntry03, Ball Release, 40
41BB: 08 20                          ; SolenoidTableEntry04, Left Eject, 20
41BD: 08 40                          ; SolenoidTableEntry05, Left Eject, 40
41BF: 23 60                          ; SolenoidTableEntry06, Top Lockup Power, 60
41C1: 23 80                          ; SolenoidTableEntry07, Top Lockup Power, 80
41C3: 24 FF                          ; SolenoidTableEntry08, Top Lockup Hold, <on>
41C5: 24 00                          ; SolenoidTableEntry09, Top Lockup Hold, <off>
41C7: 0A 20                          ; SolenoidTableEntry0A, Right Jet Bumper, 20
41C9: 0A 40                          ; SolenoidTableEntry0B, Right Jet Bumper, 40
41CB: 09 20                          ; SolenoidTableEntry0C, Left Jet Bumper, 20
41CD: 09 40                          ; SolenoidTableEntry0D, Left Jet Bumper, 40
41CF: 0B 20                          ; SolenoidTableEntry0E, Center Jet Bumper, 20
41D1: 0B 40                          ; SolenoidTableEntry0F, Center Jet Bumper, 40
41D3: 0D 20                          ; SolenoidTableEntry10, Right Slingshot, 20
41D5: 0D 40                          ; SolenoidTableEntry11, Right Slingshot, 40
41D7: 0C 20                          ; SolenoidTableEntry12, Left Slingshot, 20
41D9: 0C 40                          ; SolenoidTableEntry13, Left Slingshot, 40
41DB: 07 40                          ; SolenoidTableEntry14, Solenoid_Knocker, 40
41DD: 02 40                          ; SolenoidTableEntry15, Ball Launch, 40
41DF: 02 60                          ; SolenoidTableEntry16, Ball Launch, 60
41E1: 06 FF                          ; SolenoidTableEntry17, Idol Release, <on>
41E3: 06 00                          ; SolenoidTableEntry18, Idol Release, <off>
41E5: 06 FA                          ; SolenoidTableEntry19, Idol Release, 300
41E7: 00 F0                          ; SolenoidTableEntry1A, NULL
41E9: 05 40                          ; SolenoidTableEntry1B, Center Drop Bank, 40
41EB: 05 60                          ; SolenoidTableEntry1C, Center Drop Bank, 60
41ED: 0E FF                          ; SolenoidTableEntry1D, Left Control Gate, <on>
41EF: 0E 00                          ; SolenoidTableEntry1E, Left Control Gate, <off>
41F1: 0F FF                          ; SolenoidTableEntry1F, Right Control Gate, <on>
41F3: 0F 00                          ; SolenoidTableEntry20, Right Control Gate, <off>
41F5: 03 20                          ; SolenoidTableEntry21, Totem Drop Up, 20
41F7: 03 40                          ; SolenoidTableEntry22, Totem Drop Up, 40
41F9: 10 20                          ; SolenoidTableEntry23, Totem Drop Down, 20
41FB: 10 40                          ; SolenoidTableEntry24, Totem Drop Down, 40
41FD: 1C FF                          ; SolenoidTableEntry25, Subway Release, <on>
41FF: 1C 00                          ; SolenoidTableEntry26, Subway Release, <off>
4201: 1C E0                          ; SolenoidTableEntry27, Subway Release, E0
4203: 1C FE                          ; SolenoidTableEntry28, Subway Release, 420
4205: 22 FF                          ; SolenoidTableEntry29, Diverter Hold, <on>
4207: 22 00                          ; SolenoidTableEntry2A, Diverter Hold, <off>
4209: 21 40                          ; SolenoidTableEntry2B, Diverter Power, 40
420B: 17 FF                          ; SolenoidTableEntry2C, MPF Motor Right, <on>
```

```
420D: 17 00                                 ; SolenoidTableEntry2D, MPF Motor Right, <off>
420F: 16 FF                                 ; SolenoidTableEntry2E, MPF Motor Left, <on>
4211: 16 00                                 ; SolenoidTableEntry2F, MPF Motor Left, <off>
4213: 06 02                                 ; SolenoidTableEntry30, Idol Release, 2
4215: 08 01                                 ; SolenoidTableEntry31, Left Eject, 1
4217: 23 02                                 ; SolenoidTableEntry32, Top Lockup Power, 2
4219: 23 FA                                 ; SolenoidTableEntry33, Top Lockup Power, 300
421B: 1C 03                                 ; SolenoidTableEntry34, Subway Release, 3
                                            ;
;-------------------------------------------;------------------------------------------------------------
```

Note the few entries which have pulse-time values of 0x01, 0x02, or 0x03 (above). As the comments at the start of the function indicate, these special pulse-time values are used to trigger a special, longer pulse utilizing a SolenoidSpecialDelay[] data table. Details on this will be provided later in this document.

FlasherTable[]

Here we have the FlasherTable[] (which in IJ_L7 is located immediately after the SolenoidTable[] in the ROM).

This table has similar characteristics as the SolenoidTable[], namely, it consists of multiple 2-byte entries, each entry representing:

- 8-bit Flasher Index.
- 8-bit Pulse-time byte.

As with the SolenoidTable[], the FlasherTable[] contains multiple entries with the same flasher, but with different pulse-times.

Unlike the SolenoidTable[], the FlasherTable[] doesn't have the various special meanings for pulse-time byte. The pulse-time for all FlasherTable[] entries represent real transistor on-time periods. The code doesn't support putting in special pulse-time values as in the SolenoidTable[] (if you tried, you'd just have a pulse-time equal to whatever value you put into the 8-bit field, and possibly a lot of burnt out flasher bulbs).

Also, unlike the SolenoidTable[], the first byte of each table entry does NOT represent the corresponding index into the TransistorTable[]. Rather, the first byte is what I called, the "Flasher Index" which needs to be cross referenced into another table, FlasherIndexToTransistorIndexTable[] which will be shown next.

```
;-----------------------------------------;----------------------------------------------------------
; FlasherTable[]
;
; Each table entry is 2 bytes:
;  Byte 1 represents a flasher index
;  Byte 2 is timer value for how long to energize transistor
;
421D: 00 2B                               ; Table entries
421F: 02                                  ; Bytes per entry
                                          ;
4220: 00 01                               ; FlasherLookupEntry00, NULL
4222: 08 10                               ; FlasherLookupEntry01, 0x08, "RIGHT RAMP", 10
4224: 08 20                               ; FlasherLookupEntry02, 0x08, "RIGHT RAMP", 20
4226: 08 40                               ; FlasherLookupEntry02, 0x08, "RIGHT RAMP", 40
4228: 04 10                               ; FlasherLookupEntry04, 0x04, "JACKPOT", 10
422A: 04 20                               ; FlasherLookupEntry05, 0x04, "JACKPOT", 20
422C: 04 40                               ; FlasherLookupEntry06, 0x04, "JACKPOT", 40
422E: 03 10                               ; FlasherLookupEntry07, 0x03, "SUPER JACKPOT", 10
```

```
4230: 03 20                               ; FlasherLookupEntry08, 0x03, "SUPER JACKPOT", 20
4232: 03 40                               ; FlasherLookupEntry09, 0x03, "SUPER JACKPOT", 40
4234: 0D 10                               ; FlasherLookupEntry0A, 0x0D, "TOTEM MULTIBALL", 10
4236: 0D 20                               ; FlasherLookupEntry0B, 0x0D, "TOTEM MULTIBALL", 20
4238: 0D 40                               ; FlasherLookupEntry0C, 0x0D, "TOTEM MULTIBALL", 40
423A: 0E 10                               ; FlasherLookupEntry0D, 0x0E, "JACKPOT MULTPLER.", 10
423C: 0E 20                               ; FlasherLookupEntry0E, 0x0E, "JACKPOT MULTPLER.", 20
423E: 0E 40                               ; FlasherLookupEntry0F, 0x0E, "JACKPOT MULTPLER.", 40
4240: 09 10                               ; FlasherLookupEntry10, 0x09, "LEFT RAMP", 10
4242: 09 20                               ; FlasherLookupEntry11, 0x09, "LEFT RAMP", 20
4244: 09 40                               ; FlasherLookupEntry12, 0x09, "LEFT RAMP", 40
4246: 0A 10                               ; FlasherLookupEntry13, 0x0A, "LEFT SIDE", 10
4248: 0A 20                               ; FlasherLookupEntry14, 0x0A, "LEFT SIDE", 20
424A: 0A 40                               ; FlasherLookupEntry15, 0x0A, "LEFT SIDE", 40
424C: 0B 10                               ; FlasherLookupEntry16, 0x0B, "RIGHT SIDE", 10
424E: 0B 20                               ; FlasherLookupEntry17, 0x0B, "RIGHT SIDE", 20
4250: 0B 40                               ; FlasherLookupEntry18, 0x0B, "RIGHT SIDE", 40
4252: 05 10                               ; FlasherLookupEntry19, 0x05, "PATH OF ADVENTURE", 10
4254: 05 20                               ; FlasherLookupEntry1A, 0x05, "PATH OF ADVENTURE", 20
4256: 05 40                               ; FlasherLookupEntry1B, 0x05, "PATH OF ADVENTURE", 40
4258: 07 10                               ; FlasherLookupEntry1C, 0x07, "DOGFIGHT HURRYUP", 10
425A: 07 20                               ; FlasherLookupEntry1D, 0x07, "DOGFIGHT HURRYUP", 20
425C: 07 40                               ; FlasherLookupEntry1E, 0x07, "DOGFIGHT HURRYUP", 40
425E: 02 10                               ; FlasherLookupEntry1F, 0x02, "LIGHT JACKPOT", 10
4260: 02 20                               ; FlasherLookupEntry20, 0x02, "LIGHT JACKPOT", 20
4262: 02 40                               ; FlasherLookupEntry21, 0x02, "LIGHT JACKPOT", 40
4264: 01 10                               ; FlasherLookupEntry22, 0x01, "ETERNAL LIFE", 10
4266: 01 20                               ; FlasherLookupEntry23, 0x01, "ETERNAL LIFE", 20
4268: 01 40                               ; FlasherLookupEntry24, 0x01, "ETERNAL LIFE", 40
426A: 06 10                               ; FlasherLookupEntry25, 0x06, "PLANE/SUPER BALL", 10
426C: 06 20                               ; FlasherLookupEntry26, 0x06, "PLANE/SUPER BALL", 20
426E: 06 40                               ; FlasherLookupEntry27, 0x06, "PLANE/SUPER BALL", 40
4270: 0C 10                               ; FlasherLookupEntry28, 0x0C, "SPECIAL", 10
4272: 0C 20                               ; FlasherLookupEntry29, 0x0C, "SPECIAL", 20
4274: 0C 40                               ; FlasherLookupEntry2A, 0x0C, "SPECIAL", 40
                                          ;
;-----------------------------------------;-------------------------------------------------------
```

<u>FlasherIndexToTransistorIndexTable[]</u>

As mentioned, the FlasherTable[] contains a “Flasher Index” which must be cross-referenced against the FlasherIndexToTransistorIndexTable[] in order to determine the corresponding index into the TransistorTable[] for the particular flasher. This means that when dealing with flasher code, it is important to know whether you’re dealing with:

- Index into the FlasherTable[], or
- Index into the FlasherIndexToTransistorIndexTable[], or
- Index into the TransistorTable[]

Below is the contents of the FlasherIndexToTransistorIndexTable[]. It consists of multiple 2-byte entries, each entry corresponding to one of the “Flasher Index” values in the FlasherTable[] (above). Each entry consisting of:

- 8-bit TransistorTable[] index to which the flasher is associated.
- 8-bit byte, value 0x40

The second byte of each entry has unknown purpose. All entries have 0x40 which is consistent with a pulse-time byte as shown in the FlasherTable[] however I haven’t encountered code that actually utilizes the second byte of a FlasherIndexToTransistorIndexTable[] entry.

```
;-----------------------------------------;----------------------------------------------------------
; FlasherIndexToTransistorIndexTable[]
;
; Each table entry is 2 bytes:
;  Byte 1 is the TransistorTable[] index value and also the index number shown on the screen
;           during test mode for the particular flasher.
;  Byte 2 is 0x40 which might be a pulse-time, although whether it’s ever used is tbd.
;
; This table is used for converting the "flasher index" number from the FlasherTable[]
; into indexes that are used in the TransistorTable[]
;
6D9E: 00 0F                               ; Number of table entries 0x000F
6DA0: 02                                  ; Entry length
                                          ;
6DA1: 00 40                               ; Data00 for: "NULL"
6DA3: 11 40                               ; Data01 for: "ETERNAL LIFE"
6DA5: 12 40                               ; Data02 for: "LIGHT JACKPOT"
6DA7: 13 40                               ; Data03 for: "SUPER JACKPOT"
6DA9: 14 40                               ; Data04 for: "JACKPOT"
```

```
6DAB: 15 40                                  ; Data05 for: "PATH OF ADVENTURE"
6DAD: 18 40                                  ; Data06 for: "PLANE/SUPER BALL"
6DAF: 19 40                                  ; Data07 for: "DOGFIGHT HURRYUP"
6DB1: 1A 40                                  ; Data08 for: "RIGHT RAMP"
6DB3: 1B 40                                  ; Data09 for: "LEFT RAMP"
6DB5: 25 40                                  ; Data0A for: "LEFT SIDE"
6DB7: 26 40                                  ; Data0B for: "RIGHT SIDE"
6DB9: 27 40                                  ; Data0C for: "SPECIAL"
6DBB: 28 40                                  ; Data0D for: "TOTEM MULTIBALL"
6DBD: 29 40                                  ; Data0E for: "JACKPOT MULTPLER."
                                             ;
;--------------------------------------------;--------------------------------------
```

EnergizeSolenoidEnqueueOnlyParameterByte()

Now we will start looking at functions that WPC s/w can call when it wants to pulse a transistor.

This function takes a single parameter byte (single byte immediately following the JSR opcode that got into this function) and uses it at an index into the SolenoidTable[]. The specified SolenoidTable[] entry is enqueued and the function returns. This function reveals that the solenoid transistor code has a concept of "enqueuing" solenoid pulses, presumably, in a FIFO buffer of some sort (details will be revealed later that this is, in fact, the case).

This function only ensures the solenoid pulse is put into a queue, but not necessarily actually pulsed by the time this function returns. This is likely intended for lower priority solenoid pulses when the function caller doesn't care whether the solenoid is actually energized at any particular moment, although it will likely be energized *very* soon.

```
;---------------------------------------;-----------------------------------------
; EnergizeSolenoidEnqueueOnlyParameterByte()
;
83D8: 34 12       PSHS  X,A             ;
83DA: AE 63       LDX   $0003,S         ;
83DC: A6 80       LDA   ,X+             ;
83DE: AF 63       STX   $0003,S         ;
83E0: BD A9 11    JSR   $A911           ; EnergizeSolenoidAEnqueueOnly()
83E3: 35 92       PULS  A,X,PC          ; Done, RTS
;---------------------------------------;-----------------------------------------
```

Later in this document we'll reveal details about EnergizeSolenoidAEnqueueOnly().

In IJ_L7 I see that EnergizeSolenoidEnqueueOnlyParameterByte() is used to energize the left and right control gates (at the top of the playfield). This is not to say that *every* call to pulse these gates use this function, rather, I found occurrences of this function which happen to specify the left and right control gates.

PulseFlasherParameterByte()

As implied by its name, this function will cause the specified flasher to pulse. This function takes in the FlasherTable[] index as a parameter byte (byte immediately following the JSR opcode that got us here):

```
;---------------------------------------;------------------------------------------
; PulseFlasherParameterByte()
;
83E5: 34 12       PSHS  X,A              ;
83E7: AE 63       LDX   $0003,S          ;
83E9: A6 80       LDA   ,X+              ;
83EB: AF 63       STX   $0003,S          ;
83ED: BD AA 68    JSR   $AA68            ; FlasherPulseIndexA()
83F0: 35 92       PULS  A,X,PC           ; Done, RTS
;---------------------------------------;------------------------------------------
```

We will see details about FlasherPulseIndexA() later. We will find that it will make a “best-effort” attempt to get the flasher pulsed, however it may not necessarily happen if it was very recently pulsed, or it may mean that we prematurely turn off some other flasher (if the other flasher is in the middle of a pulse) in the process of pulsing this flasher. Keep in mind this all happens in the blink of an eye!

<u>EnergizeSolenoidWaitForDeEnergizedParameterByte()</u>

This function takes a single parameter byte (single byte immediately following the JSR opcode that got into this function) and uses it as an index into the SolenoidTable[]. The specified SolenoidTable[] entry is put through the solenoid logic to ensure it gets engaged (details later) and this function will only return after the entire solenoid pulse-time has elapsed and after the solenoid has de-engaged.

This function is probably intended for very time-sensitive code that requires particular timing of the solenoid engagement amongst other things such as DMD animation, sound-effect, or with relation to other solenoid or flasher pulses.

```
;---------------------------------------;-------------------------------------------
; EnergizeSolenoidWaitForDeEnergizedParameterByte()
;
83F2: 34 02       PSHS  A                ;
83F4: 34 10       PSHS  X                ;
83F6: AE 63       LDX   $0003,S          ;
83F8: A6 80       LDA   ,X+              ;
83FA: AF 63       STX   $0003,S          ;
83FC: 35 10       PULS  X                ;
83FE: BD A8 C9    JSR   $A8C9            ; EnergizeSolenoidAWaitForDeEnergized()
8401: 35 82       PULS  A,PC             ; Done, RTS
;---------------------------------------;-------------------------------------------
```

Later in this document we’ll reveal details about EnergizeSolenoidAWaitForDeEnergized().

EnergizeSolenoidWaitForEnergizedParameterByte()

This function takes a single parameter byte (single byte immediately following the JSR opcode that got into this function) and uses it at an index into the SolenoidTable[]. The specified SolenoidTable[] entry is put through the solenoid logic to ensure it gets engaged (details later) and then returns.

This function is probably intended for time-sensitive code that requires particular timing of the solenoid engagement to sync with other things like DMD animation or a sound effect.

```
;-----------------------------------------;---------------------------------------------
; EnergizeSolenoidWaitForEnergizedParameterByte()
;
8403: 34 02       PSHS  A                ;
8405: 34 10       PSHS  X                ;
8407: AE 63       LDX   $0003,S          ;
8409: A6 80       LDA   ,X+              ;
840B: AF 63       STX   $0003,S          ;
840D: 35 10       PULS  X                ;
840F: BD A8 D0    JSR   $A8D0            ; EnergizeSolenoidAWaitForEnergized()
8412: 35 82       PULS  A,PC             ; Done, RTS
;-----------------------------------------;---------------------------------------------
```

Later in this document we’ll reveal details about EnergizeSolenoidAWaitForEnergized().

EnergizeSolenoidAWaitForDeEnergized()

This function was called by one of the functions shown above. This function uses the SolenoidTable[] index in the A register, and puts it through the solenoid code (details later) which will ensure the solenoid gets pulsed and won't return until after the specified solenoid has fully been turned on and then off.

```
;---------------------------------------;-------------------------------------------
; EnergizeSolenoidAWaitForDeEnergized()
;
;   A has solenoid index number (used to lookup entry in SolenoidTable[])
;
A8C9: 34 04       PSHS  B                 ; Save B
A8CB: 5F          CLRB                    ; B gets 0x00 == wait until solenoid has been turned on and off
A8CC: 8D 18       BSR   $A8E6             ; EnergizeSolenoidAWaitForProcessingB()
A8CE: 35 84       PULS  B,PC              ; Done, RTS
;---------------------------------------;-------------------------------------------
```

Obviously, the heavy lifting is done by EnergizeSolenoidAWaitForProcessingB(), and it will take the 0x00 in the B register as an indication that it needs to wait until the solenoid has been turned on and then off before returning.

<u>EnergizeSolenoidAWaitForEnergized()</u>

This function was called by one of the functions shown above. This function uses the SolenoidTable[] index in the A register, and puts it through the solenoid code (details later) which will ensure the solenoid gets turned on, and then returns.

```
;---------------------------------------;-------------------------------------------
; EnergizeSolenoidAWaitForEnergized()
;
; A has SolenoidTable[] index to energize.
;
A8D0: 34 16       PSHS  X,B,A           ; Save regs
A8D2: BE 03 8A    LDX   $038A           ; X gets $038A, solenoid circular buffer, head pointer
A8D5: BC 03 8C    CPX   $038C           ; Compare with $038C, solenoid circular buffer, tail pointer
A8D8: 26 06       BNE   $A8E0           ; If they differ, skip down to energize and wait, no enqueue
A8DA: 4F          CLRA                  ; A gets 0x00
A8DB: BD A9 11    JSR   $A911           ; Circular buffer is empty, call EnergizeSolenoidAEnqueueOnly()
A8DE: A6 E4       LDA   ,S              ; A gets A off the stack
A8E0: C6 01       LDB   #$01            ; B gets 0x01 == wait until solenoid has started being energized
A8E2: 8D 02       BSR   $A8E6           ; EnergizeSolenoidAWaitForProcessingB()
A8E4: 35 96       PULS  A,B,X,PC        ; Done, RTS
;---------------------------------------;-------------------------------------------
```

The comments in the function (above) reveal that solenoid pulses can be queued into a circular buffer. This statement then reveals that the system will only allow a single solenoid to be turned on at a time, since they are serviced by a queue. The function also shows that if the queue is empty, it will call a function to EnergizeSolenoidAEnqueueOnly() prior to calling EnergizeSolenoidAWaitForProcessingB(). The call to EnergizeSolenoidAEnqueueOnly() is done presumably to speed things up however tracing through the code it seems like it's not really necessary since the EnergizeSolenoidAWaitForProcessingB will do the same thing (one of the first things it does, see below), but maybe I'm not looking at this correctly.

Again, the heavy lifting is done by EnergizeSolenoidAWaitForProcessingB(), and it will take the 0x01 in the B register as an indication that it only needs to wait until the solenoid has been turned on before returning.

EnergizeSolenoidAWaitForProcessingB()

At last we get to a function that does some significant work. This function was called from the previous two functions with the following register parameters:

- The A register contains the SolenoidTable[] index of the solenoid to process.
- The B register has 0x00 when we want to wait for the specified solenoid to turn on then off.
- The B register has 0x01 when we only want to wait for the specified solenoid to turn on.

The important thing to understand is that this is a *wait* function. It should not be called by code that doesn't care about any sort of waiting.

```
;---------------------------------------;-------------------------------------------
; EnergizeSolenoidAWaitForProcessingB()
;
;  Called to energize a solenoid and wait.
;   A has SolenoidTable[] index
;   B has value == 0x00 --> wait until solenoid has been energized then deenergized
;   B has value != 0x00 --> wait until solenoid has started being energized
;
A8E6: 34 06       PSHS  B,A             ; Save registers
A8E8: 8D 2D       BSR   $A917           ; EnqueueSolenoidPulse()
                                        ;
A8EA: BD 83 99    JSR   $8399           ; --\--\--\ Sleep()
A8ED: 01                                ;   |  |  |
A8EE: 1F 89       TFR   A,B             ;   |  |  | B gets Head Entries Added (minus 1, not accounting for entry we just added)
A8F0: F0 03 8F    SUBB  $038F           ;   |  |  | B -= Tail Entries Processed
A8F3: 2B 0D       BMI   $A902           ;   |  |  | If B < 0, goto $A902
A8F5: 26 F3       BNE   $A8EA           ;   |  |--/ if B > 0, keep looping
                                        ;   |  |
A8F7: E6 61       LDB   $0001,S         ;   |  | B gets original B value off stack from calling function parameters
A8F9: 27 EF       BEQ   $A8EA           ;   |--/ If it was 0x00, keep looping until we know the enqueued signal has
                                        ;   |      been handled by the tail pointer.
                                        ;   |
A8FB: F6 03 90    LDB   $0390           ;   | B gets $0390, tail entry solenoid energize in process
A8FE: 26 EA       BNE   $A8EA           ; --/ If tail entry solenoid energize in process, keep looping
                                        ;
A900: 20 0D       BRA   $A90F           ; return
                                        ;
                                        ;------------------------------------------------------------------
                                        ; original head entries added < tail entries processed
                                        ; This means we know our enqueued head pointer has been hit by
                                        ; the tail pointer processing.
                                        ;-----------------------------------------------------------------
A902: C1 FF       CMPB  #$FF            ; Is count -1?
A904: 26 09       BNE   $A90F           ; if not, return, something is really messed up
A906: E6 61       LDB   $0001,S         ; B gets original B value off stack from calling function parameters
A908: 26 05       BNE   $A90F           ; If it was not 0x00, return
A90A: F6 03 90    LDB   $0390           ; B gets $0390, tail entry solenoid energize in process
A90D: 26 DB       BNE   $A8EA           ; if tail entry solenoid energize in process, go back to $A8EA and try again
                                        ;-----------------------------------------------------------------
                                        ;-----------------------------------------------------------------
                                        ;
A90F: 35 86       PULS  A,B,PC          ; Done, RTS
;---------------------------------------;-------------------------------------------
```

This function (above) first adds the specified solenoid to the queue, then loops until the circular buffer statistics reveal that the solenoid has been processed. Then depending on the value of the B register, it will either return as soon as it's been determined that the solenoid has been turned on, or it will continue to wait and only return after the solenoid has been turned back off.

Remember your game will probably use different ram values for the circular-buffer and for the head/tail statistics, but this is a good function to find and then set breakpoints and watch the on-screen solenoid indicators in pinmame, in order to understand what's going on.

EnergizeSolenoidAEnqueueOnly()

This is a simple function. It takes in the SolenoidTable[] index in the A register, and calls a function to push the SolenoidTable[] entry into the queue. That's all it does, there is no guarantee that the solenoid is actually on when this function is finished, only that a request to turn it on has been put into the queue (circular buffer referenced in the previous function).

```
;--------------------------------------;------------------------------------------
; EnergizeSolenoidAEnqueueOnly()
;
;  A has index number into the SolenoidTable[] for the solenoid/time values
;
A911: 34 02       PSHS  A                ; Save A
A913: 8D 02       BSR   $A917            ; EnqueueSolenoidPulse()
A915: 35 82       PULS  A,PC             ; Done, RTS
;--------------------------------------;------------------------------------------
```

Details on EnqueueSolenoidPulse() are next.

EnqueueSolenoidPulse()

This function, as its name implies, takes in a SolenoidTable[] index and puts it into a queue for purposes of turning on a solenoid. The function, below, is heavily commented with C-like pseudo-code, to help make it easier to understand what it's doing. The comments in the function header also show the various solenoid queue addresses that are used in IJ_L7 (which will likely be different in other games).

```
;----------------------------------------;-----------------------------------------
; EnqueueSolenoidPulse()
;
;  $038A:$038B  head pointer
;  $038C:$038D  tail pointer
;  $038E        head entries added
;  $038F        tail entries processed
;  $0390        tail entry solenoid energize in progress
;  $0391-$03AC  28-byte circular table
;
A917: 34 70       PSHS  U,Y,X            ; Save registers
A919: BE 03 8A    LDX   $038A            ; X gets bytes from $038A:$038B, head pointer
                                         ;
A91C: 8C 03 91    CMPX  #$0391           ; if ($038A:$038B < 0x0391), if head pointer is too low, error
A91F: 25 05       BCS   $A926            ;    goto ErrorHandler82BC(12)
A921: 8C 03 AD    CMPX  #$03AD           ; if ($038A:$038B >= 0x03AD), if head pointer is too high, error
A924: 25 04       BCS   $A92A            ;    goto ErrorHandler82BC(12)
A926: BD 82 BC    JSR   $82BC            ; ErrorHandler82BC(12)
A929: 12                                 ; -------------------------------------------------------
                                         ; -------------------------------------------------------
                                         ; Here when ($038A:$038B >= 0x0391) && ($038A:$038B < 0x03AD)
                                         ; -------------------------------------------------------
A92A: 8D 28       BSR   $A954            ; AdvanceCircularSolenoidQueueIndexX()
A92C: BC 03 8C    CPX   $038C            ; if (X == $038C:$038D)  // if X (incremented head pointer) is at the tail pointer
A92F: 26 11       BNE   $A942            ; {
A931: BD 82 E4    JSR   $82E4            ;    ThrowGenError(0x0F) // we had a circular buffer overrun, record/report it
A934: 0F                                 ; }
                                         ; // So the circular buffer is full, we reported it with above call, now sit around and
                                         ; // wait for an entry to free up…
                                         ; while (x == $038C:$038D)  // while X (incremented head pointer) is at the tail pointer
                                         ; {
A935: BD A9 65    JSR   $A965            ;    ProcessNextSolenoidTableEntry(), $81-$8B gets data specific to the
                                         ;                                    next solenoid in the circular queue.
A938: BE 03 8A    LDX   $038A            ;    X gets $038A:$038B     // get head pointer value again
A93B: 8D 17       BSR   $A954            ;    AdvanceCircularSolenoidQueueIndexX()
A93D: BC 03 8C    CPX   $038C            ;    Keep looping as long as the buffer is filled
A940: 27 F3       BEQ   $A935            ; }
A942: FE 03 8A    LDU   $038A            ; U gets $038A:$038B  // U gets current head pointer
A945: A7 C4       STA   ,U               ; Save A (solenoid index) at current U pointer
A947: BF 03 8A    STX   $038A            ; Save new X head value at $038A:$038B
A94A: B6 03 8E    LDA   $038E            ; A gets $038E (circular buffer entry count)
A94D: 7C 03 8E    INC   $038E            ; Increment Head Entries Added count
A950: 8D 13       BSR   $A965            ; ProcessNextSolenoidTableEntry(), $81-$8B gets data specific to the
                                         ;                                  next solenoid in the circular queue.
A952: 35 F0       PULS  X,Y,U,PC         ; Done, RTS
;-----------------------------------------;--------------------------------------------
```

The code (above) is actually pretty simple. Checks circular buffer and adds the SolenoidTable[] index into the buffer. If the circular buffer is full (meaning we have 28 solenoids in the queue! Not likely to happen), then the function will throw an error (details of error throwing outside scope of this document) and then wait in a loop until an entry becomes available in the queue. Pretty simple and effective.

AdvanceCircularSolenoidQueueIndexX()

This function was called from the above function for purposes of advancing the solenoid queue pointer. Since the solenoid queue is in the form of a circular buffer, this function is simply needed for the purpose of handling the wrap-around moment when the pointer gets incremented and then needs to end up being the address of the very first address in the buffer. It's easier for code to call this function than to have the code inline (since it's used in more than just one place).

```
;---------------------------------------;-----------------------------------------
; AdvanceCircularSolenoidQueueIndexX()
;
; X has ram pointer value from circular solenoid queue ($038A:$038B)
; which has been validated.  This just advances X to the next address
; in the circular buffer.
;
A954: 30 01       LEAX  $0001,X         ; X++
A956: 8C 03 AD    CMPX  #$03AD          ; compare X with end of circular buffer 0x03AD address
A959: 25 09       BCS   $A964           ; if X is less than 0x03AD, nothing to do, return
A95B: 27 04       BEQ   $A961           ; if X is at end of buffer, set X to 0x0391 and return
                                        ; if X is > 0x03AD, unexpected error,
A95D: BD 82 BC    JSR   $82BC           ; ErrorHandler82BC(12)
A960: 12                                ;
A961: 8E 03 91    LDX   #$0391          ; X gets 0x0391, start of circular buffer
A964: 39          RTS                   ; return
;---------------------------------------;-----------------------------------------
```

Again, the above function includes an error reporting function, ErrorHandler82BC(), the details of which, are outside the scope of this document.

<u>ProcessNextSolenoidTableEntry()</u>

This is an important function. It handles the solenoid queue, advancing to the next SolenoidTable[] index stored in the queue, if necessary. This is what ensures the serialization of solenoid queue pulses, and is further evidence to the fact that the WPC system only allows a single solenoid to be energized at any given moment during game play (not including flippers). Of course this leads one to question things like motors and feature elements which are energized during the course of game play. It would seem that such things happen outside of the solenoid queue all together (something to explore and to watch for next time you play your pin).

```
;--------------------------------------;-------------------------------------------
; ProcessNextSolenoidTableEntry()
;
; Process a table entry from the circular solenoid queue.  Populates memory at
;  $81-$8B with data specific to the next solenoid in the circular queue.
;  This area is then used by the ISR to determine what physical outputs to assert.
;
;  $038A:$038B  solenoid circular buffer, head pointer
;  $038C:$038D  solenoid circular buffer, tail pointer
;
A965: 34 16       PSHS  X,B,A           ; save registers
A967: 8D 42       BSR   $A9AB           ; CheckIfAnySolenoidPresentlyEnergized()
A969: 25 3E       BCS   $A9A9           ; If C-bit set, return, a solenoid is presently energized, so try later
                                        ;
                                        ; // We are here when we know there is no physical transistor energized, however
                                        ; // the $0390 might still be flagged which is what we set prior to telling the
                                        ; // ISR about the transistor we want energized....
                                        ;
A96B: B6 03 90    LDA   $0390           ; if ($0390 != 0x00)  // tail entry solenoid energize in process
A96E: 27 11       BEQ   $A981           ; {
                                        ;    // It appears this little block will schedule an "off-time" that must elapse
                                        ;    // prior to the energization of next solenoid in the queue.  We know the previously
                                        ;    // energized solenoid is now off, but this will put in a time of 0x50 and flag the
                                        ;    // logic so that this 0x50 must time back to 0x00 before we'll move on and energize
                                        ;    // the next solenoid in the queue.
                                        ;    //
A970: CC 00 03    LDD   #$0003          ;    D = 0x0003, length of TransistorTable[] header
A973: F3 81 BD    ADDD  $81BD           ;    D += $81BD, at $81BD is 7C2B22 so D+= 0x7C2B, D is at start of TransistorTable[] data
A976: DD 82       STD   $82             ;    $82 gets D, address of start of TransistorTable[] table data entry[0]
A978: 86 50       LDA   #$50            ;    A = 0x50
A97A: 97 81       STA   $81             ;    $81 gets A, energize time for solenoid? maybe prep ICs for solenoid call?
A97C: 7F 03 90    CLR   $0390           ;    $0390 = 0x00 // tail entry solenoid energize not in process
A97F: 20 28       BRA   $A9A9           ;    return;
                                        ; }
                                        ;
                                        ; // We are here when there is no physical solenoid energized <and> the schedule
                                        ; // inter-solenoid "off-time" has elapsed (above).  So now we should safely be
                                        ; // able to turn on the next solenoid in the queue.
                                        ;
A981: BE 03 8C    LDX   $038C           ; X gets $038C:$038D, tail pointer
A984: BC 03 8A    CPX   $038A           ; if (tail == head)
A987: 27 20       BEQ   $A9A9           ;    return // nothing in the circular buffer, no solenoids to energize, return
                                        ;
A989: 8C 03 91    CMPX  #$0391          ; if ($038A:$038B < 0x0391)
A98C: 25 05       BCS   $A993           ;    goto ErrorHandler82BC(12)
A98E: 8C 03 AD    CMPX  #$03AD          ; if ($038A:$038B >= 0x03AD)
A991: 25 04       BCS   $A997           ;    goto ErrorHandler82BC(12)
A993: BD 82 BC    JSR   $82BC           ; ErrorHandler82BC(12)
A996: 12                                ; -------------------------------------------------------
                                        ; -------------------------------------------------------
                                        ; Here when ($038A:$038B >= 0x0391) && ($038A:$038B < 0x03AD)
                                        ; -------------------------------------------------------
A997: A6 84       LDA   ,X              ; A gets byte from tail pointer
A999: 8D B9       BSR   $A954           ; AdvanceCircularSolenoidQueueIndexX()
A99B: BF 03 8C    STX   $038C           ; Update tail pointer with new queue index X
A99E: 7C 03 8F    INC   $038F           ; Increment tail entries counter
A9A1: 7C 03 90    INC   $0390           ; tail entry solenoid energize in process
A9A4: BD AA 36    JSR   $AA36           ; GetSolenoidIndexATableEntryValueIntoD(), SolenoidTable[] 16-bit value into D
A9A7: 8D 0F       BSR   $A9B8           ; WriteSolenoidTableEntryDataToRam81_8B()
A9A9: 35 96       PULS  A,B,X,PC        ; Done, RTS
;---------------------------------------;-------------------------------------------
```

The comments in the function (above) describe everything the function does but here's a summary:

- If any solenoid is presently energized, do nothing and return.
- If no solenoids are energized but we haven't done the inter-solenoid off-time, then start up the off-time and return.
- If no solenoids are energized, and the inter-solenoid off time has elapsed, then:
    - If the solenoid queue is empty, return, nothing else to do. Else,
    - Put the details about the next solenoid queue entry into memory so that the ISR will handle the transistor on/off

Details about ISR will be described later. All that you need to know at this point is that on a regular basis, the timer interrupt will kick in and check things including timer-based transistor turn-on and turn-off events. Outside of the ISR is a "main loop" that constantly performs various housekeeping tasks, including making periodic calls to this function to kick off the next solenoid (details on this are also later in this document).

<u>CheckIfAnySolenoidPresentlyEnergized()</u>

This is a utility function used to determine if a solenoid is presently turned on (or off during the inter-solenoid delay period). This function reveals some of the details about the memory used by the solenoid queue logic and the ISR to manage the solenoid on/off behavior.

```
;---------------------------------------;---------------------------------------------
; CheckIfAnySolenoidPresentlyEnergized()
;
; Returns C-bit set when a solenoid is presently on (or during the inter-solenoid off time).
; Returns C-bit clr when the solenoid logic is idle and its safe to energize another solenoid.
;
; Checks $84, flag byte used between the queue logic and the ISR.
; Checks $81, time byte used between the queue logic and the ISR.
;
A9AB: 1A 01       ORCC  #$0001           ; if (($84 == 0x00) && ($81 == 0x00))
A9AD: 0D 84       TST   $84              ; {
A9AF: 26 06       BNE   $A9B7            ;    return C-bit cleared, okay to process linked list entry
A9B1: 0D 81       TST   $81              ; }
A9B3: 26 02       BNE   $A9B7            ; else
A9B5: 1C FE       ANDCC #$00FE           ; {
A9B7: 39          RTS                    ;    return C-bit set, do not process linked list entry
                                         ; }
;---------------------------------------;---------------------------------------------
```

The above function reveals that the $84 and $81 bytes are important when the solenoid queue logic needs to flag things to the ISR. We'll soon see these, along with a few other bytes in the same general area of RAM are used for negotiation of solenoid on and off between normal code and the ISR.

You may also notice my C pseudo-code tends to use curly-brackets, even to enclose single-lines of "code" which isn't necessary in the C programming language. I tend to do this just to make it clear that's being communicated and to give a consistent appearance.

WriteSolenoidTableEntryDataToRam81_8B()

This function takes in a TransistorTable[] index, along with the pulse-time value (as fetched from the SolenoidTable[] entry) and sets up the RAM in a way that the ISR will take notice, and turn on or off the transistor accordingly. Additionally the ISR takes care of the pulse-time value decrements so that it eventually reaches 0x00 at which point the transistor will get turned off.

```
;----------------------------------------;-------------------------------------------
; WriteSolenoidTableEntryDataToRam81_8B()
;
; Called after retrieving 16-bit solenoid index table entry value into D
;  A == TransistorTable[] index number
;  B == energize time:
;       0xFF == Just turn on
;       0x00 == turn off
;       0x01 == Load special delay time from table[0]
;       0x02 == Load special delay time from table[1]
;       0x03 == Load special delay time from table[2]
;       0x04 == Load special delay time from table[3]
;       0x05 == Load special delay time from table[4]
;       0x06 == Load special delay time from table[5]
;   all else == actual energize time to put into $81
;
; If turning on the transistor for time:  0x06 < Time < 0xFF
;  $81 gets Time
;  $82:$83 gets address of the transistor's TransistorTable[] 5-byte entry
;  $84 gets 0x02
;
; If turning on the transistor for time:  0x00 < Time < 0x07
;  $81 gets 0x00
;  $82:$83 gets address of the transistors TransistorTable[] 5-byte entry
;  $84 gets 0x01, in the ISR this decrements to 0xFF which triggers processing of this data.
;  $8A:$8B gets 2 bytes of SolenoidSpecialDelay[] table entry data
;  $88:$89 gets address of the next SolenoidSpecialDelay[] entry
;
; If just turning, and leaving the transistor ON:
;  $81 gets 0x00
;  $82:$83 gets address of the transistors TransistorTable[] 5-byte entry
;  $84 gets 0x02
;
; If just turning, and leaving the transistor OFF:
;  $81 gets 0x01
;  $82:$83 gets address of the transistors TransistorTable[] 5-byte entry
;
```

```
;
A9B8: 34 16       PSHS  X,B,A            ; Save registers
A9BA: 1C FE       ANDCC #$00FE           ; Clear C-bit
                                         ;
A9BC: AD 9F 81 2D JSR   [$812D]          ; $812D has $FECE,FF ; JumpTableEntry64, Null_Rts_FECE_FF() // debug hook for solenoid pulses
A9C0: 25 37       BCS   $A9F9            ; if C-bit set, return
                                         ;
A9C2: C1 FF       CMPB  #$FF             ; if 2nd byte of solenoid table entry is 0xFF
A9C4: 26 03       BNE   $A9C9            ; {
A9C6: 5F          CLRB                   ;    B = 0x00, just turn on the solenoid
A9C7: 20 0E       BRA   $A9D7            ; }
A9C9: C1 00       CMPB  #$00             ; else if 2nd byte of solenoid table entry is 0x00
A9CB: 26 0A       BNE   $A9D7            ; {
A9CD: 8D 5D       BSR   $AA2C            ;    LoadDwithTransistorTableAddressIndexA(), D has addr of 5-byte table entry $7c2b,22
A9CF: DD 82       STD   $82              ;    Write SolenoidTableEntry address to $82:$83
A9D1: 86 01       LDA   #$01             ;    A = 0x01, in the ISR this decrements -2 to 0xFF which triggers processing of this data.
A9D3: 97 81       STA   $81              ;    Write 0x01 to $81
A9D5: 20 22       BRA   $A9F9            ;    return
                                         ; }
                                         ;
A9D7: 34 04       PSHS  B                ; Save B, solenoid energize time
A9D9: 8D 51       BSR   $AA2C            ; LoadDwithTransistorTableAddressIndexA(), D has addr of 5-byte table entry $7c2b,22
A9DB: DD 82       STD   $82              ; Write TransistorTable[] Entry address to $82:$83
A9DD: 35 04       PULS  B                ; Restore B
A9DF: 86 02       LDA   #$02             ; A = 0x02
A9E1: 34 01       PSHS  CC               ; Save CC reg
                                         ;
A9E3: 5D          TSTB                   ; if (solenoid energize time != 0x00)
A9E4: 27 0B       BEQ   $A9F1            ; {
A9E6: C1 06       CMPB  #$06             ;    if (2nd byte of solenoid table entry <= 0x06)
A9E8: 22 07       BHI   $A9F1            ;    {
A9EA: 8D 0F       BSR   $A9FB            ;       LoadSolenoidSpecialDelayTimeB()
A9EC: 25 09       BCS   $A9F7            ;       if C-bit set, error, return
A9EE: 5F          CLRB                   ;       B = 0x00
A9EF: 86 01       LDA   #$01             ;       A = 0x01
                                         ;    }
                                         ; }
A9F1: 1A 10       ORCC  #$0010           ; Disable IRQ interrupts
A9F3: D7 81       STB   $81              ; $81 gets B, Solenoid pulse time, ISR decrements this towards zero
A9F5: 97 84       STA   $84              ; $84 gets A, Code used by isr to indicate type of pulse, see above
A9F7: 35 01       PULS  CC               ; Enable interrupts
                                         ;
A9F9: 35 96       PULS  A,B,X,PC         ; Done, RTS
                                         ;
;----------------------------------------;-----------------------------------------
```

LoadSolenoidSpecialDelayTimeB()

As mentioned in a few places above, when the SolenoidTable[] pulse-time value is 01, 02, 03, 04, 05, or 06, then the a special extra long pulse is used via other mechanism. This function is part of that special mechanism. This function gets called with the pulse-time code (01 – 06) value in the B register. We will see that this will be used as in index into another table to load the actual solenoid pulse time, and ram will be set up so that the ISR will know to traverse this table, adding additional time to the pulse, until a 0x0000 values is reached.

```
;--------------------------------------;-------------------------------------------
; LoadSolenoidSpecialDelayTimeB()
;
; Called when processing solenoid table entry, and 2nd byte was 1, 2, 3, 4, 5 or 6
; Used to perform special solenoid delays > than 0xFE (maximum single-byte value that could cause a timed pulse0
; Set C-bit if error of some sort.
;
; Loads extended pulse time for the solenoid from $EAA2 or $EA7A (depending on $75), and puts it into RAM:
;  $8A:$8B gets 2 bytes of metadata entry index B
;  $88:$89 gets address of the next metadata entry
;
A9FB: 34 16       PSHS  X,B,A           ; Save registers
A9FD: 0D 75       TST   $75             ; if ($75 != 0x00) // not sure when this might happen, usually $75 is 0x00
A9FF: 27 05       BEQ   $AA06           ; {
AA01: BE 82 34    LDX   $8234           ;    X = $8234, at $8234 is $EAA2,FF ; TablePointer23 so X gets $EAA2,
                                        ;                                 SolenoidSpecialDelay_when75NotNull[]
AA04: 20 03       BRA   $AA09           ; }
                                        ; else
                                        ; {
AA06: BE 82 31    LDX   $8231           ;    X = $8231, at $8231 is $EA7A,FF ; TablePointer22 so X gets $EA7A
                                        ;                                 SolenoidSpecialDelay_when75Null[]
                                        ; }
AA09: 5A          DECB                  ; B--, make the 1-6 index 0-based.
AA0A: 58          ASLB                  ; B *= 2, 2 bytes per SolenoidSpecialDelay_x[] table entries
AA0B: AE 85       LDX   B,X             ; X gets pointer to SolenoidSpecialDelay_x[] table entry
AA0D: A6 84       LDA   ,X              ; A gets 1st byte of the SolenoidSpecialDelay_x[] entry
AA0F: 97 8A       STA   $8A             ; $8A gets the 1st byte of the SolenoidSpecialDelay_x[] entry
AA11: A6 01       LDA   $0001,X         ; A gets 2nd byte of the SolenoidSpecialDelay_x[] entry
AA13: 27 0A       BEQ   $AA1F           ; If 2nd byte of SolenoidSpecialDelay_x[] entry 0x00, return C-bit set, error pulse time 0x00
AA15: 97 8B       STA   $8B             ; Store 2nd byte of SolenoidSpecialDelay_x[] at $8B
AA17: 30 02       LEAX  $0002,X         ; X += 2, go to next SolenoidSpecialDelay_x[] table entry
AA19: 9F 88       STX   $88             ; $88:$89 gets next the pointer to next SolenoidSpecialDelay_x[] entry
AA1B: 1C FE       ANDCC #$00FE          ; Clear C-bit, all good
AA1D: 20 02       BRA   $AA21           ; return
AA1F: 1A 01       ORCC  #$0001          ; Set C-bit
```

```
AA21: 35 96       PULS  A,B,X,PC         ; return
;-------------------------------------;----------------------------------------
```

The function (above) will load a pulse time from a “SolenoidSpecialDelay” table, and also save the pointer to the next SolenoidSpecialDelay_x[] table entry. We will see later how the ISR will keep the solenoid energized for the period of time defined by the SolenoidSpecialDelay_x[] entry 2nd byte, and continue to keep the solenoid energized for the period of time for every subsequent SolenoidSpecialDelay_x[] entry’s second byte until a value of 0x0000 is encountered.

At this time it’s not clear why there are two different SolenoidSpecialDelay_x[] tables. As shown in the above function, the value of ram $75 will determine which SolenoidSpecialDelay_x[] table to load. While tracing this code in PinMame, I find the value of $75 is always 0x00 so the same table is always used. It’s possible this is a debug thing that the s/w developer originally put in to experiment with different extended time periods. A thorough scan of the code to determine when $75 gets changed might shed some light on this.

As you can see, the function looks up the address of the SolenoidSpecialDelay_x[] table by dereferencing a pointer. If you find this function in your ROM, you can use it to determine the address of your SolenoidSpecialDelay_x[] tables by dereferencing the pointer when you find it in this function in your ROM.

It is worth reminding the reader that other games will likely use different values for all ram addresses used in this, and all functions. It is likely all WPC games share these same functions but use different addresses so you will need to search your ROM for byte patterns which do not contain ram address bytes when searching for these functions in your game.

Next we will show the actual SolenoidSpecialDelay_x[] tables…

SolenoidSpecialDelay_when75Null[]

This is the SolenoidSpecialDelay_x[] table that's used when $75 is 0x00 (which in IJ_L7 trace, this appears to always be the case). There is a table entry for each possible special delay code that the code supports. As described above, the code supports 6 special delays. In IJ_L7 we see in its SolenoidTable[] only the values 01, 02, and 03 are used. This means the entries for 04, 05, and 06 are not used, and we see below how these unused values, instead of defining a unique special delay, just point to the first delay table entry.

```
;--------------------------------------;-------------------------------------
; SolenoidSpecialDelay_when75Null[]
;
;  This table is used to produce extra long energize times for general purpose solenoids.
;  The first byte of each table entry is just which, of every 8 ISR calls the transistor
;  will be turned on.  ISR has a continuous counter that goes 0..7 and when it matches
;  first byte of each table entry's 2-byte entry, the solenoid will be refreshed to the
;  on state.
;
;  The second byte of each table entry is a timer value that will count down to zero before
;  the next 2-byte entry is loaded and aged.  After all of the 2-byte table entries are
;  loaded and timed out, then the transistor will finally get turned off.
;
;  The 0x0000 entry marks the end of the table entry and will cause the ISR logic to
;  finally turn off the transistor, when it gets reached.
;
EA7A: EA 86                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x01
EA7C: EA 92                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x02
EA7E: EA 9A                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x03
EA80: EA 86                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x04
EA82: EA 86                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x05
EA84: EA 86                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x06
                                       ;
                                       ;------------------------------------
EA86: 06 3C                            ; entry 0x01, 0x04, 0x05, 0x06, Used for: Top Eject
EA88: 05 3C                            ;
EA8A: 04 3C                            ;
EA8C: 02 3C                            ;
EA8E: 00 3C                            ;
EA90: 00 00                            ; 0x0000 Marker for end of table entry
                                       ;
                                       ;
                                       ;------------------------------------
EA92: 00 60                            ; entry 0x02, Used for: Top Lockup Power
EA94: 03 D0                            ;
```

```
EA96: 03 A0                          ;
EA98: 00 00                          ; 0x0000 Marker for end of table entry
                                     ;-----------------------------------
                                     ;
                                     ;-----------------------------------
EA9A: 00 40                          ; entry 0x03, Used for: Subway Release
EA9C: 02 FA                          ;
EA9E: 02 20                          ;
EAA0: 00 00                          ; 0x0000 Marker for end of table entry
                                     ;-----------------------------------
                                     ;
;------------------------------------;------------------------------------
```

The table (above) first contains a just a pointer for each table entry. The pointer then points to bytes immediately following the main table. The table, and all of its entries are shown above.

The comments in the function header (above) also describe how the first byte of each table entry is used to define which ISR pass will be used to process the particular table entry. The ISR contains a constantly wrapping 0..7 number which different parts of the code can hook into for purposes of staggering different events. In this case, the table can pick one of those 0..7 passes when to process each table entry. We will show details on the ISR handling of the transistor data later in this document.

SolenoidSpecialDelay_when75NotNull[]

This is the alternate SolenoidSpecialDelay_x[] table which would be used when $75 is not 0x00. Since it appears $75 is always 0x00 then it would seem this is only used for debug purposes. TBD if there is any certain normal circumstance when $75 would be made non-zero and then this table used for long solenoid pulses. Comparing this table to the previous table reveals that this table has longer pulse times for the SolenoidTable[] entries with values 0x01 and 0x02 for their pulse-time bytes.

```
;--------------------------------------;------------------------------------
; SolenoidSpecialDelay_when75NotNull[]
;
; When processing a solenoid table entry, and 2nd byte is > 0x00 and <= 0x06
; and when ($75 != 0x00) then this table is indexed with 2nd byte index -1.
;
;  $88:$89 gets metadata bytes[2..3]
;  $8A gets metadata byte[0]
;  $8B gets metadata byte[1]
;
;  This table is used to produce extra long energize times for general purpose solenoids.
;  The first byte of each table entry is just which, of every 8 ISR calls the transistor
;  will be turned on.  ISR has a continuous counter that goes 0..7 and when it matches
;  first byte of each table entry's 2-byte entry, the solenoid will be refreshed to the
;  on state.
;
;  The second byte of each tabel entry is a timer value that will count down to zero before
;  the next 2-byte entry is loaded and aged.  After all of the 2-byte table entries are
;  loaded and timed out, then the transistor will finally get turned off.
;
;  The 0x0000 entry marks the end of the table entry and will cause the ISR logic to
;  finally turn off the transistor, when it gets reached.
;
EAA2: EA AE                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x01
EAA4: EA BC                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x02
EAA6: EA C4                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x03
EAA8: EA AE                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x04
EAAA: EA AE                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x05
EAAC: EA AE                            ; Referenced when 2nd byte of SolenoidTable[] entry is 0x06
                                       ;
                                       ;------------------------------------
EAAE: 07 3C                            ; entry 0x01, 0x04, 0x05, 0x06, Used for: Top Eject
EAB0: 06 3C                            ;
EAB2: 05 3C                            ;
EAB4: 03 1E                            ;
```

```
EAB6: 02 1E                          ;
EAB8: 00 3C                          ;
EABA: 00 00                          ; 0x0000 Marker for end of table entry
                                     ;-----------------------------------
                                     ;
                                     ;-----------------------------------
EABC: 00 60                          ; entry 0x02, Used for: Top Lockup Power
EABE: 05 D0                          ;
EAC0: 05 A0                          ;
EAC2: 00 00                          ; 0x0000 Marker for end of table entry
                                     ;-----------------------------------
                                     ;
                                     ;-----------------------------------
EAC4: 00 40                          ; entry 0x03, Used for: Subway Release
EAC6: 02 FA                          ;
EAC8: 02 20                          ;
EACA: 00 00                          ; 0x0000 Marker for end of table entry
                                     ;-----------------------------------
                                     ;
;------------------------------------;-----------------------------------------------------
```

<u>SolenoidEnergizeIndexDIfIdle()</u>

This is a utility function that is used by the menu system test mode for purposes of pulsing a solenoid without using the solenoid queue (circular buffer). It would probably be bad to call this function outside of the test mode since it might conflict with code that utilizes the solenoid queue.

```
;---------------------------------------;-------------------------------------------
; SolenoidEnergizeIndexDIfIdle()
;
; A has TransistorTable[] index
; B has energize time If 0xFF, just stick on, 0x00 turn off.
;
AA23: 8D 86       BSR   $A9AB           ; CheckIfAnySolenoidPresentlyEnergized()
AA25: 25 04       BCS   $AA2B           ; A solenoid is energized, skip to the end
AA27: 8D 8F       BSR   $A9B8           ; WriteSolenoidTableEntryDataToRam81_8B()
AA29: 1C FE       ANDCC #$00FE          ; Clear C-bit
AA2B: 39          RTS                   ; return
;---------------------------------------;-------------------------------------------
```

As you can see, this function simply checks if the ISR is presently busy with a transistor, if not, then the specified transistor gets energized. The A register has the TransistorTable[] index for the transistor to energize and B has the energize time. Since this function ends up calling WriteSolenoidTableEntryDataToRam81_8B(), which handles the special extra long pulse times, it should be okay for any of the pulse-time values to be specified when calling this function, including real delay times, and special delay values 01-06, as well as values 0xFF to leave the solenoid on, and 0x00 to turn solenoid off.

LoadDwithTransistorTableAddressIndexA()

This is a utility function called by functions above used to retrieve a pointer to the TransistorTable[] using index in the A register.

```
;---------------------------------------;-------------------------------------------
; LoadDwithTransistorTableAddressIndexA()
;
; A has the TransistorTable[] index.
;
; Returns in D the starting address for the corresponding transistor data table entry
; in the TransistorTable[].   Each table entry is 5 bytes in length.
;  First 2 bytes are the address of the transistor's cache byte in ram
;  Next 2 bytes are the address of the h/w register corresponding to the transistor
;  Last byte is the bitmask for the transistor, 0-bit corresponding to the transistor.
;
AA2C: C6 05       LDB   #$05             ; B gets 0x05, number of bytes per table entry at $7c2b,22
AA2E: 3D          MUL                    ; D solenoid index MUL 5, offset into the $7c2b,22 data table.
AA2F: C3 00 03    ADDD  #$0003           ; D += 0x0003, account for the data table header to jump past at $7c2b,22
AA32: F3 81 BD    ADDD  $81BD            ; D += at $81BD is 7C2B22 so D+= 0x7C2B, at last D points to TransistorTable[] entry.
AA35: 39          RTS                    ; return
;---------------------------------------;-------------------------------------------
```

It is notable that the TransistorTable[] is located in banked ROM, so callers of this function shouldn’t assume they can just read data from the returned pointer in D. They must first set the bank to that of the TransistorTable[] before trying to access the returned pointer.

<u>GetSolenoidIndexATableEntryValueIntoD()</u>

This is a utility function called from one of the solenoid handling functions above. This function will take a SolenoidTable[] index in the A register and return the corresponding SolenoidTable[] entry value in D (A:B) register.

```
;---------------------------------------;-----------------------------------------
; GetSolenoidIndexATableEntryValueIntoD()
;
; Called while processing a SolenoidTable[] entry.
; A has the 1-byte SolenoidTable[] index
; Returns in D, the 2-byte solenoid table entry:
;  High Byte (A) has the TransistorTable[] index corresponding to the solenoid.
;  Low Byte (B) has energize time (00==off, FF==0n, 01-06==special long delay, 07-FE==actual delay)
;
AA36: 34 30       PSHS  Y,X             ; Save registers
AA38: 8E 81 C0    LDX   #$81C0          ; X gets 0x81C0, $81C0 has $41B0,30  ; JumpTableEntry95, SolenoidTable[]
AA3B: BD AC F3    JSR   $ACF3           ; GetPageAndLoadTableEntryPointerIndexAIntoXandY()
AA3E: BD 90 52    JSR   $9052           ; LoadTableEntry()
AA41: 1F 20       TFR   Y,D             ; D gets 16-bit solenoid table entry value
AA43: 35 B0       PULS  X,Y,PC          ; Done, RTS
;---------------------------------------;-----------------------------------------
```

Note that this function can allow you to determine the address of the SolenoidTable[] within your own ROM. If you can find this function in your ROM then you need to look at the value loaded into the X register. That value is a pointer to the SolenoidTable[] in banked ROM.

GetFlasherIndexATableEntryValueIntoD()

This function takes a FlasherTable[] index in the A register and returns, in the D register, its corresponding TransistorTable[] index, and the pulse-time corresponding to the particular FlasherTable[] entry.

As noted earlier in this document, unlike the SolenoidTable[], the FlasherTable[] index numbers are not used directly as TransistorTable[] indexes, but the FlasherTable[] index must be passed through a cross-reference table to convert the Flasher Index into a Transistor Index.

```
;-----------------------------------------;-------------------------------------------
; GetFlasherIndexATableEntryValueIntoD()
;
; Called for processing of a flasher table entry.
; A has the 1-byte table index (flasher index)
; The FlasherTable[] has a 2-byte entry for every flasher:
;  Byte 1: Flasher Number
;  Byte 2: Pulse-time
;
; The Flasher number (byte 1) is an arbitrary number which, unlike solenoid table data, is NOT the same
; as the transistor index in the transistor table, so another look is done.  The flasher number from the
; FlasherTable[] is then used to look up the transistor number in the FlasherIndexToTransistorIndexTable[].
;
; The FlasherIndexToTransistorIndexTable[] has a 2-byte entry for every flasher number where the first byte is the
; actual transistor number for the flasher and the 2nd byte appears to always be 0x40. The second byte isn't
; really used here (tbd if the 2nd byte is used anywhere).
;
; This function returns in D the following:
;  High Byte (A) has the TransistorTable[] index that corresponds to the specified flasher
;  Low Byte (B) has the energize time for the flasher, from the FlasherTable[].
;
; This function returns 0x00xx in Y where xx is the flasher number from the FlasherTable[]
;
AA45: 34 10       PSHS  X                 ; Save registers
AA47: 8E 81 CB    LDX   #$81CB            ; X gets 0x81CB, $81CB has $421D,30  ; TablePointer00, FlasherTable[]
AA4A: BD AC F3    JSR   $ACF3             ; GetPageAndLoadTableEntryPointerIndexAIntoXandY()
AA4D: BD 90 52    JSR   $9052             ; LoadTableEntry()
AA50: 34 20       PSHS  Y                 ; Save the 2-byte table entry data onto stack
AA52: E6 E4       LDB   ,S                ; B gets first byte from 2-byte table entry, flasher number
AA54: 4F          CLRA                    ; A gets 0x00
AA55: 1F 02       TFR   D,Y               ; Y gets 0x00xx where xx is the flasher number
AA57: A6 E4       LDA   ,S                ; A gets flasher number off stack
AA59: 8E 81 C6    LDX   #$81C6            ; X gets 0x81C6, $81C6 has $6D9E,2F  ; JumpTableEntry97, FlasherIndexToTransistorIndexTable[]
AA5C: BD AC F3    JSR   $ACF3             ; GetPageAndLoadTableEntryPointerIndexAIntoXandY()
```

```
AA5F: BD 90 2B    JSR   $902B           ; DereferenceXintoA_PageB()
AA62: E6 61       LDB   $0001,S         ; B gets the original energize time for the flasher, from FlasherTable[]
AA64: 32 62       LEAS  $0002,S         ; Fixup stack
AA66: 35 90       PULS  X,PC            ; Done, RTS
;---------------------------------------;-------------------------------------------
```

In a nutshell, the above function will return the following information for a given FlasherTable[] index:

- The flasher number, used to look up other characteristics about the flasher in other tables
- The TransistorTable[] index, used to look up details about the transistor associated with the flasher
- The flasher pulse time

FlasherPulseIndexA()

This is a simple utility function that can be used to pulse the specified flasher. The A register has the FlasherTable[] index for the desired flasher pulse. This function is called by PulseFlasherParameterByte(), earlier in this document.

```
;----------------------------------------;--------------------------------------------
; FlasherPulseIndexA()
;
; A has the flasher index (which can be used to look up flasher info in the
; FlasherTable[] and the FlasherIndexToTransistorIndexTable[].
;
AA68: 34 26       PSHS  Y,B,A           ;
AA6A: BD AA 45    JSR   $AA45           ; GetFlasherIndexATableEntryValueIntoD()
AA6D: 8D 02       BSR   $AA71           ; StartFlasherPulseTransistorATimeB()
AA6F: 35 A6       PULS  A,B,Y,PC        ; Done, RTS
;----------------------------------------;--------------------------------------------
```

The StartFlasherPulseTransistorATimeB() function will be described next. It does most of the heavy lifting involved with the flasher pulse.

StartFlasherPulseTransistorATimeB()

This function performs a lot work related to the pulsing of a flasher. It is important to keep in mind that the flashers might be the most delicate component of all of the transistor-driven devices. A solenoid can take a bit of abuse before it fails, but a flasher can fail quite quickly if not properly handled. It is important that the period of time that the flasher is illuminated must be very carefully controlled to avoid premature failure of the flash bulbs.

```
;---------------------------------------;-------------------------------------------
; StartFlasherPulseTransistorATimeB()
;
; Called with D having: Transistor Index (in A) and energize time (in B)
; Y has 0x00xx where xx is the flasher number from the FlasherTable[]
;
AA71: 34 76       PSHS  U,Y,X,B,A        ;
AA73: 32 7D       LEAS  $FFFD,S          ; Make room for 3 bytes on the stack
AA75: D6 78       LDB   $78              ; $78 appears to be 0x80 duing attract, test, and game modes.
AA77: 27 19       BEQ   $AA92            ; if ($78 != 0x00)
                                         ; {
AA79: 1F 20       TFR   Y,D              ;    // D gets flasher number (from FlasherTable[])
AA7B: 86 02       LDA   #$02             ;    D *= 2;
AA7D: 3D          MUL                    ;
AA7E: 10 BE 81 C9 LDY   $81C9            ;    Y gets *81C9 which has 0x051C, &FlasherNextEnergizeTimeTable[]
AA82: 31 AB       LEAY  D,Y              ;    Y = 0x051C + D  (gets address of 2-byte entry table starting at 0x051C,
                                         ;                     FlasherNextEnergizeTimeTable[])
AA84: 10 AF 61    STY   $0001,S          ;    Store the 2-byte flasher data entry address onto last 2 of the 3 temp bytes on stack
AA87: EC A4       LDD   ,Y               ;    D gets the 2 bytes of FlasherNextEnergizeTimeTable[] entry for this flasher
AA89: 27 0B       BEQ   $AA96            ;    If loaded 0x0000 then goto $AA96
                                         ;    //
                                         ;    // $55:$56 (CurrentTimerValue) 2-byte timer value (increments each timer interrupt)
                                         ;    // ---------------------------------------------------------------------------
                                         ;    // Logic below will:
                                         ;    // a. If this transistor was last set to be turn on-able <NOW> then okay to re-energize
                                         ;    // b. If this transistor was last set to be turn on-able time in the past, then okay
                                         ;    //    to re-energize it now.
                                         ;    // c. Else, transistor next turn on-able time is some time in the future
                                         ;    //    then it's not allowed to be turned on again now, set C-bit and return (error).
                                         ;    //
AA8B: 10 93 55    CMPD  $55              ;    else if == $55:$56 then goto $AA96
AA8E: 27 06       BEQ   $AA96            ;    else if 2-bytes are < $55:$56 then goto $AA96 expired in the past,
AA90: 2B 04       BMI   $AA96            ;    else set C-bit (error) and return
                                         ; }
                                         ; else
```

```
AA92: 1A 01       ORCC  #$0001          ; {
AA94: 20 62       BRA   $AAF8           ;    set C-bit (error) and return
                                        ; }
                                        ;
AA96: D6 11       LDB   $11             ; Save current bank onto temp byte on stack
AA98: E7 E4       STB   ,S              ;
AA9A: F6 81 BF    LDB   $81BF           ; B gets 0x22 from $81BF, Bank value of TransistorTable[]
AA9D: BD 8F FB    JSR   $8FFB           ; SetPageBANK1()
                                        ;
                                        ; Two FlasherEnergizeArray[] entries:
                                        ;------------------------------------
                                        ; $0384-$0386, FlasherEnergizeArray[] entry
                                        ;                 Byte1=EnergizeTime,
                                        ;                 Bytes2:3, Address of TransistorTable[] entry
                                        ; $0387-$0386, FlasherEnergizeArray[] entry
                                        ;                 Byte1=EnergizeTime,
                                        ;                 Bytes2:3, Address of TransistorTable[] entry
                                        ;
AAA0: 9E 7F       LDX   $7F             ; X gets flasher FlasherEnergizeArray[] entry address.  This address of next free entry.
AAA2: 8C 03 8A    CMPX  #$038A          ; if (next free flasher address is >= 0x038A) // if FlasherEnergizeArray[] is full
AAA5: 25 02       BCS   $AAA9           ; {
                                        ;    // turn off the one with shortest energize time and free up an entry
AAA7: 8D 53       BSR   $AAFC           ;    FlasherEnergizeArrayFreeOneEntry()
                                        ; }
                                        ;
AAA9: EC 63       LDD   $0003,S         ; Reload D from function start, TransistorIndex:EnergizeTime
AAAB: 5D          TSTB                  ; if (flasher energize time == 0x00)
AAAC: 26 04       BNE   $AAB2           ; {
AAAE: BD 82 BC    JSR   $82BC           ;    ErrorHandler82BC(14)
AAB1: 14                                ; }
AAB2: BD AA 2C    JSR   $AA2C           ; LoadDwithTransistorTableAddressIndexA()
AAB5: 1F 02       TFR   D,Y             ; Y gets address of TransistorTable[] entry for this flasher
AAB7: A6 64       LDA   $0004,S         ; A gets transistor energize time off the stack
AAB9: AD 9F 81 42 JSR   [$8142]         ; At $8142 is $FEF8,FF, so call function at $FEF8, Null_Rts_FEF8_FF()
AABD: 34 01       PSHS  CC              ;
AABF: 1A 10       ORCC  #$0010          ; Disable IRQ interrupts
AAC1: 9E 7F       LDX   $7F             ; X gets pointer to next free flasher FlasherEnergizeArray[] entry,
                                        ;  code above has validated it is a valid,free entry.
AAC3: A7 84       STA   ,X              ; Push the energize-time value into the FlasherEnergizeArray[] entry
AAC5: 10 AF 01    STY   $0001,X         ; Push the TransistorTable[] address into the FlasherEnergizeArray[] entry
AAC8: 30 03       LEAX  $0003,X         ; X += 3, next flasher FlasherEnergizeArray[] entry
AACA: 9F 7F       STX   $7F             ; Update $7F:$80 with address of next entry
AACC: A6 24       LDA   $0004,Y         ; Fetch the transistor bitmask for this transistor from its TransistorTable[] entry
AACE: 43          COMA                  ; Flip bits so now the 1-bit represents the transistor
AACF: AA B4       ORA   [,Y]            ;
```

```
AAD1: A7 B4       STA   [,Y]            ; Update cache byte with 1-bit set for this transistor
AAD3: A7 B8 02    STA   [$02,Y]         ; Write-thru to h/w with 1-bit set for thsi transistor (turns on the flasher)
                                        ;
AAD6: 35 01       PULS  CC              ; Enable IRQ interrupts
AAD8: E6 E4       LDB   ,S              ; Restore original bank
AADA: BD 8F FB    JSR   $8FFB           ; SetPageBANK1()
AADD: E6 64       LDB   $0004,S         ; B gets the transistor energize-time off stack
AADF: 54          LSRB                  ; Divide the transistor energize-time by 2
AAE0: 34 04       PSHS  B               ; Save it on the stack
                                        ;
                                        ;-------------------------------------------------------------------------------------
                                        ; The following will convert the energize-time value associated with the flasher in
                                        ; the FlasherTable[] into a minimum time period (ISR ticks value) which must
                                        ; elapse before the flasher is allowed to be energized again, to increase bulb life.
                                        ;-------------------------------------------------------------------------------------
AAE2: 54          LSRB                  ; Divide the transistor energize-time by 2 again (now its been divided by 4 in total)
AAE3: EB E0       ADDB  ,S+             ; B = (EnergizeTime/4) + (EnergizeTime/2), so it has 75% of original energize-time value
AAE5: 54          LSRB                  ; B /= 2
AAE6: 54          LSRB                  ; B /= 2
AAE7: 54          LSRB                  ; B /= 2, so not B has 75% of original energize time value, divided by 8
AAE8: C9 00       ADCB  #$00            ;
AAEA: 5C          INCB                  ; B++
AAEB: 4F          CLRA                  ; Now D has 0x00xx where xx is 75% of original energize time value, divided by 8, plus 1
AAEC: D3 55       ADDD  $55             ; Now add in $55:$56 (CurrentTimerValue) 2-byte timer value (increments each timer interrupt)
                                        ;
                                        ; // catch a rollover situation, if time calculated to be ticks 0x0000, bump it up by 1.
AAEE: 26 01       BNE   $AAF1           ; If (D == 0x0000)
                                        ; {
AAF0: 5C          INCB                  ;    B++; // increment it to 0x0001
                                        ; }
AAF1: 10 AE 61    LDY   $0001,S         ; Y gets the value pushed on earlier,
                                        ;   FlasherNextEnergizeTimeTable[] entry address for this flasher.
                                        ;
AAF4: ED A4       STD   ,Y              ; The $051C table entry for this flasher gets the flasher expiry ticks value
                                        ;
AAF6: 1C FE       ANDCC #$00FE          ; Clear C-bit
AAF8: 32 63       LEAS  $0003,S         ; Fixup stack
AAFA: 35 F6       PULS  A,B,X,Y,U,PC    ; Done. RTS.
                                        ;
;---------------------------------------;------------------------------------------
```

The StartFlasherPulseTransistorATimeB() function, above, contains a lot of logic and reveals a lot of details about the way the flasher pulse is performed. A brief summary of the steps performed is as follows:

- If the flasher to pulse has recently been pulsed and its inter-pulse time indicates that we shouldn't pulse it again, return.
- If the FlasherEnergizeArray[] is full then turn off the one with least amount of on-time remaining.
- Put the desired flasher pulse information into the FlasherEnergizeArray[]
- Turn on the specified flasher.
- Set the inter-pulse time for the flasher so its next flash won't happen after a minimum period has elapsed.

The FlasherNextEnergizeTimeTable[] is what's used to store the inter-pulse time for each flasher. The table contains a 2-byte entry for each flasher. When the 2-byte entry is 0x0000 then it's perfectly safe to energize the flasher. Otherwise, the 2-byte entry is a timer number which is used to determine if enough time has elapsed since the previous pulse of the same flasher. In IJ_L7, as shown in the code above, there is a pointer to the FlasherNextEnergizeTimeTable[] at $81C9. At $81C9 is $051C which is the starting address of the FlasherNextEnergizeTimeTable[] in ram. The timer value is compared against a free-running number that gets incremented at each pass through the ISR (timer interrupt function). When you can find this function in your ROM you will need to find your game's pointer and de-reference it to determine the starting address of your FlasherNextEnergizeTimeTable[]. Later we'll show a cleanup function that sets all stale entries back to 0x0000 so they won't cause any trouble after the timer value rolls over back to 0x0000.

The FlasherEnergizeArray[] contains multiple entries allowing multiple flashers to be energized at the same time. In IJ_L7 there are two entries in this array, thus allowing a maximum of two flashers to be illuminated at the same time. Each entry in this array is 3 bytes long containing:

- Byte 1, time remaining for this flasher to be on. This starts at the flasher's pulse time, then decrements during calls to the ISR, and when it reaches value 0x00, the flasher is turned off in the ISR (details will be shown later in this document).
- Bytes 2 & 3 contain a pointer to the flasher's corresponding entry in the TransistorTable[], thus giving the ISR access to this flasher's cache byte, h/w register and bitmask, which will allow the ISR to turn off this flasher when the time in Byte 1 has expired.

You can also see how this function, above, loads up the TransistorTable[] entry and turns on the flasher. This means the flasher is turned ON outside of the ISR, but the ISR is what is responsible for turning OFF the flasher, with the exception of the logic in this function that prematurely turns off a flasher to accommodate this new flasher pulse request. Details on FlasherEnergizeArrayFreeOneEntry() are shown next.

FlasherEnergizeArrayFreeOneEntry()

This is the function that called during StartFlasherPulseTransistorATimeB(), above, which will find the FlasherEnergizeArray[] entry with the shortest on-time remaining, and then turn off that flasher and mark that FlasherEnergizeArray[] entry as free so that a new flasher pulse can be stored there. This function should only be called when the FlasherEnergizeArray[] has been found to be full, although it does contain a check to ensure this is the case prior to turning off the flasher with shortest on-time.

```
;-----------------------------------------;---------------------------------------------
; FlasherEnergizeArrayFreeOneEntry()
;
; Called from previous function when $7F:$80 pointer indicates that
; FlasherEnergizeArray[] is full.
;
; Finds an entry in the FlasherEnergizeArray[] with shortest energize-time remaining,
; and turns it off, thus freeing up an entry in the FlasherEnergizeArray[].
;
AAFC: 34 76       PSHS  U,Y,X,B,A        ;
AAFE: C6 FF       LDB   #$FF             ; Starting value 0xFF, then scan the FlasherEnergizeArray[]
                                         ;  for entry with smallest energize time
                                         ;
AB00: 10 8E 03 84 LDY   #$0384           ; Y gets first address of FlasherEnergizeArray[] entries
                                         ;
                                         ; Two FlasherEnergizeArray[] entries:
                                         ;--------------------------------------
                                         ; $0384-$0386, FlasherEnergizeArray[] entry
                                         ;                Byte1=EnergizeTime,
                                         ;                Bytes2:3, Address of TransistorTable[] entry
                                         ; $0387-$0386, FlasherEnergizeArray[] entry
                                         ;                Byte1=EnergizeTime,
                                         ;                Bytes2:3, Address of TransistorTable[] entry
                                         ;
                                         ;CHECK_FLASHER_ARRAY_ENTRY_Y
AB04: E1 A4       CMPB  ,Y               ; if (FlasherArrayEntry.Byte1 is <= B) // trying to find entry with shortest energize time
AB06: 25 06       BCS   $AB0E            ; {
AB08: EE 21       LDU   $0001,Y          ;    U gets next 2 bytes of the 3-byte FlasherEnergizeArray[] entry,
                                         ;      address of TransistorTable[] entry
AB0A: E6 A4       LDB   ,Y               ;    B gets first byte of the 3-byte FlasherEnergizeArray[] entry,   flasher index
AB0C: 30 A4       LEAX  ,Y               ;    X gets Y, pointer to this 3-byte FlasherEnergizeArray[] entry
                                         ; }
                                         ;
AB0E: 31 23       LEAY  $0003,Y          ; Y += 3, advance to next FlasherEnergizeArray[] entry
```

```
AB10: 10 8C 03 8A CMPY  #$038A           ; if (FlasherEnergizeArray[] Pointer < 0x038A) // haven't gone through all of the entires yet
                                         ; {
AB14: 25 EE       BCS   $AB04            ;    goto CHECK_FLASHER_ARRAY_ENTRY_Y
                                         ; }
                                         ;
                                         ; At this piont, X has pointer to one of the FlasherEnergizeArray[] entries,
                                         ; whichever has the shortest energize time.
                                         ;  B has the shortest energize time associated with that entry.
                                         ;  U has TransistorTable[] associated with that entry.
AB16: 34 01       PSHS  CC               ;
AB18: 1A 10       ORCC  #$0010           ; Disable IRQ interrupt
AB1A: 10 9E 7F    LDY   $7F              ; Y gets next-free-array-entry pointer
                                         ;
                                         ; // if there's no more free entries in FlasherEnergizeArray[]...
AB1D: 10 8C 03 8A CMPY  #$038A           ; if (next-free-array-entry >= 0x038A)
AB21: 25 17       BCS   $AB3A            ; {
AB23: A6 D4       LDA   [,U]             ;    A gets cache byte associated with transistor having shortest energize time
AB25: A4 44       ANDA  $0004,U          ;    Clear the transistor's bit
AB27: A7 D4       STA   [,U]             ;    Update cache byte with transistor tuned off
AB29: A7 D8 02    STA   [$02,U]          ;    Write-thru to h/w register with transistor turned off
AB2C: DE 7F       LDU   $7F              ;    U gets next-free-array-entry pointer
AB2E: 33 5D       LEAU  $FFFD,U          ;    U -= 3
AB30: A6 C4       LDA   ,U               ;    A gets energize-time from last valid FlasherEnergizeArray[] entry
AB32: A7 84       STA   ,X               ;    Move the last valid FlasherEnergizeArray[] entry into the entry we just turned off,
                                         ;     energize time
AB34: EC 41       LDD   $0001,U          ;    Move the last valid FlasherEnergizeArray[] entry into the entry we just turned off,
                                         ;     TransistorTable[] pointer
AB36: ED 01       STD   $0001,X          ;
AB38: DF 7F       STU   $7F              ;    next-free-array-entry gets address of this entry we just freed up (last entry $038A)
                                         ; }
AB3A: 35 01       PULS  CC               ; Enable ISQ interrupt
AB3C: 35 F6       PULS  A,B,X,Y,U,PC     ; (PUL? PC=RTS)
;----------------------------------------;------------------------------------------
```

Notice that the functions above will disable the timer interrupt prior to manipulating the FlasherEnergizeArray[] contents. This is important to ensure that the entire data is modified at once, without worrying about the ISR cutting in half way through the operation and changing the data which could result in unpredictable behavior (pretty basic stuff to an embedded s/w developer but good to point out anyway).

FlasherNextEnergizeTimeTableCleanup()

This is the cleanup function mentioned earlier. It removes any stale entries from the FlasherNextEnergizeTimeTable[]. This will prevent problems especially after the ISR timer value has rolled over. See my note in the function header where there still could be some timing issues but it may a non-issue or handled by some other code (the WPC code is pretty solid and robust).

```
;----------------------------------------;-------------------------------------------
; FlasherNextEnergizeTimeTableCleanup()
;
; Runs through the FlasherNextEnergizeTimeTable[] and cleans it up.  Any entries which
; have time value < current time ticks, write with value 0x0000.  This helps avoid
; having problems with stale ticks in there after ticks has rolled over.
;
; TBD, code still may have issues when flasher is energized while time ticks is very
; close to 0xFFFF, it seems like the FlasherNextEnergizeTimeTable[] entry might get
; loaded with time just over 0x0000, however if this cleanup function comes in before
; time ticks actually rolled over, then this table entry would appear as a very old,
; stale entry and get wiped out.  It's possible this condition is handled and hasn't
; been noticed, so just something to think about.
;
AB3E: 34 76       PSHS  U,Y,X,B,A        ;
AB40: 8E 81 C6    LDX   #$81C6           ; X gets 0x81C6, $81C6 has $6D9E,2F  ; JumpTableEntry97, FlasherIndexToTransistorIndexTable[]
AB43: BD AC D0    JSR   $ACD0            ; GetTableEntryLimitIntoY()
AB46: BE 81 C9    LDX   $81C9            ; X gets *81C9 which has 0x051C, &FlasherNextEnergizeTimeTable[]
AB49: DC 55       LDD   $55              ; Load $55:$56 (CurrentTimerValue) 2-byte timer value (increments each timer interrupt)
AB4B: CE 00 00    LDU   #$0000           ; U = 0x0000, reset value for any table entries
                                         ;
                                         ; while (1)
                                         ; {
AB4E: 10 8C 00 00 CMPY  #$0000           ;   if (flasherCount == 0x0000)
AB52: 27 0D       BEQ   $AB61            ;      done, goto RTS
AB54: 10 A3 84    CMPD  ,X               ;   Compare current timer ticks with FlasherNextEnergizeTimeTable[] entry value
AB57: 2B 02       BMI   $AB5B            ;   if (FlasherNextEnergizeTimeTable[] value is <= NOW)
                                         ;   {
AB59: EF 84       STU   ,X               ;      Push 0x0000 into FlasherNextEnergizeTimeTable[] entry for this flasher
AB5B: 31 3F       LEAY  $FFFF,Y          ;      flasherCount--
AB5D: 30 02       LEAX  $0002,X          ;      FlasherNextEnergizeTimeTable[] pointer ++
                                         ;   }
AB5F: 20 ED       BRA   $AB4E            ; }
                                         ;
AB61: 35 F6       PULS  A,B,X,Y,U,PC     ; Done. RTS.
;----------------------------------------;-------------------------------------------------------
```

InitializeTransistors()

Now that we know all sorts of details about the transistors, solenoids and flashers, we can show one of the first functions that gets called when the WPC software boots. The InitializeTransistors() function initializes all of the data structs, pointers, etc that have been shown above. Additionall all of the transistors in the TransistorTable[] are turned off.

```
;---------------------------------------;---------------------------------------------------------
; InitializeTransistors()
;
; Called during initialization routine during power-up.
;
; Walks the TransistorTable[] and forces all transistors to OFF state.
; Sets all transistor cache bits to OFF.
; Inits all transistor ram bytes, used by ISR to OFF.
; Inits the circular transistor queue buffer to all OFF.
; Inits the circular transistor head/tail buffer pointers to start of the buffer.
;
A852: 34 36       PSHS  Y,X,B,A          ; Save registers
A854: 32 7F       LEAS  $FFFF,S          ; Make a spare byte available on the stack for temp usage here
A856: D6 11       LDB   $11              ; Load current ROM page into B
A858: E7 E4       STB   ,S               ; Save the original ROM page into our spare stack byte we just made
A85A: 1C FE       ANDCC #$00FE           ; Clear the carry bit (it gets set if an error occurs...)
                                         ;
A85C: BD 86 74    JSR   $8674            ; CallPagedFunctionPreserve_U_X_B_A_CC()  This calls function pointed to at $8091
A85F: 80 91                              ; -->JumpTableEntry30, Null_Rts_7F58_3D(), Debug/development hook, just an RTS now.
A861: 25 52       BCS   $A8B5            ; If some sort of error occurred in the above, skip way down to $A8B5
                                         ;
A863: 8E 03 84    LDX   #$0384           ; X gets 0x0384
A866: 9F 7F       STX   $7F              ; Store 0x0384 into $7F:$80, start of FlasherEnergizeArray[], 0x0384 means queue is empty
A868: 4F          CLRA                   ; A gets 0x00
A869: B7 03 90    STA   $0390            ; Store 0x00 into $0390, clears: tail entry solenoid energize in progress
A86C: 97 84       STA   $84              ; Store 0x00 into $84, clears solenoid-energize data used by ISR
A86E: 97 81       STA   $81              ; Store 0x00 into $81, clears solenoid-energize data used by ISR
A870: 97 86       STA   $86              ; Store 0x00 into $86, clears solenoid-energize data used by ISR
A872: BE 81 BD    LDX   $81BD            ; X gets value from $81BD, at $81BD is 7C2B22 so X gets 0x7C2B, &TransistorTable[]
A875: F6 81 BF    LDB   $81BF            ; B gets 0x22 from $81BF, Bank value of TransistorTable[]
A878: BD AC DA    JSR   $ACDA            ; CallLoadTableEntry()  The 2 bytes at 7C2B,22 are loaded into Y.
                                         ;  This loads 0x002D from 7C2B,22 into Y, # of TransistorTable[] entries, 45
                                         ;
A87B: 31 3F       LEAY  $FFFF,Y          ; Decrement Y so it now contains 1 less than table maximum. (0-based/1-based)
A87D: 86 01       LDA   #$01             ; Set table index to 0x01
A87F: BD AC F6    JSR   $ACF6            ; LoadTableEntryPointerIndexA()
                                         ;  A has table index we want to look up
                                         ;  X has table address
                                         ;  B has the ROM page of the table
                                         ; When function is done, X points to the desired entry into the table
                                         ;
A882: BD 8F FB    JSR   $8FFB            ; SetPageBANK1()  B has 0x22, the bank of the table we're looking at.
                                         ;
                                         ;---------------------------------------------------------------------------------
                                         ; Loop below turns off all transistors defined in the TransistorTable[]
                                         ;---------------------------------------------------------------------------------
A885: 1C FE       ANDCC #$00FE           ; --\ Clear the C bit.
A887: AD 9F 81 39 JSR   [$8139]          ;   | NullFEF7_FF()  This just does an RTS, debug hook.
A88B: 25 09       BCS   $A896            ;   | If an error occured (?) skip down to $A896  Since the function is just
                                         ;   |  an RTS, there is no error so we NEVER skip over the following...
                                         ;   |
A88D: E6 94       LDB   [,X]             ;   |
A88F: E4 04       ANDB  $0004,X          ;   | Clear this bit from cache ram
                                         ;   |
A891: E7 94       STB   [,X]             ;   | Store 0-bit for this transistor in cache
A893: E7 98 02    STB   [$02,X]          ;   | Store 0-bit for this transistor in h/w, turns off transistor
                                         ;   |
A896: 4C          INCA                   ;   | Bump A to the next index
A897: 30 05       LEAX  $0005,X          ;   | Advance X directly to the next table entry.  Here we are hard-coded with
                                         ;   |  knowledge that each table entry is 5 bytes in length.
                                         ;   |
A899: BD 9C DA    JSR   $9CDA            ;   | RefreshWatchdogTimeout()
A89C: 31 3F       LEAY  $FFFF,Y          ;   | Decrement Y... Y is storing the number of table entries.
A89E: 26 E5       BNE   $A885            ; --/ If not zero, keep looping until all table entires has been handled.
                                         ;
A8A0: B6 03 8E    LDA   $038E            ; Get $038E, head entries added
A8A3: B7 03 8F    STA   $038F            ; Store it to $038F, tail entries processed
A8A6: 8E 03 91    LDX   #$0391           ; X gets 0x0391
A8A9: BF 03 8A    STX   $038A            ; Store 0x0391 into $038A, head pointer at start of circular buffer
A8AC: BF 03 8C    STX   $038C            ; Store 0x0391 into $038C, tail pointer at start of circular buffer
A8AF: BD AB 63    JSR   $AB63            ; Init037B_0383_7ARAMwithNOT1F()
A8B2: BD AC 16    JSR   $AC16            ; InitRAM7A_DisableIRQ_ClearAxRAM()
                                         ;
                                         ; We will jump here if an error occurred in one of the above function(s)
                                         ; Normally, we're here after running through all of the above code without
                                         ; any problems.
                                         ;
A8B5: E6 E4       LDB   ,S               ; Get our original Bank off the stack (saved at the start of this function)
A8B7: BD 8F FB    JSR   $8FFB            ; SetPageBANK1()   Set bank back to original value (0x3D in our trace)
A8BA: 32 61       LEAS  $0001,S          ; Get stack pointer back to normal
A8BC: 35 B6       PULS  A,B,X,Y,PC       ; Done, RTS
;----------------------------------------;-------------------------------------------------------
```

IRQ_Routine()

This section will only discuss the ISR as it applies to transistor/solenoid/flasher code. Only relevant portions of code related to this topic will be shown. This document uses the terms "IRQ" and "ISR" and "timer interrupt" interchangeably. They all refer to the same thing.

The WPC CPU board uses the Motorola 68B09 microprocessor. Any datasheet on this part will describe that the IRQ interrupt vector is at $FFF8:FFF9. This is where the microprocessor will jump when the IRQ condition occurs (periodic timer).

You can look at your ROM's non-banked area and see where your IRQ_Routine() is located, for IJ_L7 it is as follows:

```
FFF8: DC 2E                ; Interrupt Vector: IRQ, Interrupt Request, external hardware triggers this.
```

The IJ_L7 will jump to $DC2E in non-banked region whenever the IRQ_Routine() needs to be called.

Within the IRQ_Routine() is a lot of code that handles timers, multi-threaded operation of WPC, switches, lamps, general-illumination, flashers, transistors and a variety of other things outside the scope of this document.

Below is a chunk of the IRQ_Routine() which turns on transistors:

```
DC5F: AD 9F 81 3C JSR   [$813C]          ; Set8DriverPCBTransistorOutputs()
                                         ; This jumps to subroutine at [$813C], at $813C is EA CC FF so we go to $EACC
DC63: 96 7B       LDA   $7B              ; A gets $7B
DC65: B7 3F E0    STA   $3FE0            ; Store to WPC_TRANSISTOR4, Solenoid 25-28
DC68: 96 7C       LDA   $7C              ; A gets $7C
DC6A: B7 3F E1    STA   $3FE1            ; Store to WPC_TRANSISTOR1, Solenoids 1-8
DC6D: 96 7D       LDA   $7D              ; A gets $7D
DC6F: B7 3F E2    STA   $3FE2            ; Store to WPC_TRANSISTOR3, Solenoid 17-24
DC72: 96 7E       LDA   $7E              ; A gets $7E
DC74: B7 3F E3    STA   $3FE3            ; Store to WPC_TRANSISTOR2, Solenoid  9-16
```

The code chunk above gets called every pass through the ISR. All it does is read each transistor cache byte (same byte pointed to by the TransistorTable[] entries) and push its value into the transistor's h/w register (again, same as defined in the TransistorTable[]). Also note that this includes a call for the IJ_L7 8-driver PCB, so that its transistors get set as well. Below is that function:

```
;--------------------------------------;----------------------------------------------------
; Set8DriverPCBTransistorOutputs()
;
; This little routine is called during timer interrupt just after the CPU LED is toggled
;
EACC: 96 ED       LDA   $ED              ; A gets $ED, Cache byte for 8-driver transistors
EACE: B7 3F EB    STA   $3FEB            ; Write the cache thru to h/w register for 8-driver transistor outputs
EAD1: 39          RTS                    ; Done, RTS
;--------------------------------------;----------------------------------------------------
```

If your game doesn't use the 8-driver PCB then you probably won't have the above function. Or perhaps the function is in place and you can optionally add an 8-driver pcb to your game to light up new and exciting features. This is something that would need further research.

Later, during the ISR, the following code is executed which handles a lot of the nitty gritty details of transistor logic:

```
                                         ; ---------------------------------------------------------
                                         ;
                                         ;
E17D: 96 78       LDA   $78              ; Let's check $78 for non-zero, appears to be 0x80 during attract, test, and game modes.
E17F: 26 03       BNE   $E184            ; If non-zero do handler a couple lines down...
E181: 7E E2 6D    JMP   $E26D            ; ...otherwise $78 was 0x00, jump to $E26D
                                         ; // A gets transistor energize flag byte, 0x01==Special Timed energize, 0x2==Timed energize
E184: 96 84       LDA   $84              ; // Further down, $84 gets -= 2 which is what causes 0x01 to go to 0xff which
                                         ; // triggers this code...
E186: 2A 3D       BPL   $E1C5            ; If ($84 & 0x80)
                                         ; {
E188: 0A 8B       DEC   $8B              ;    if (--$8B == 0x00), decrement 2nd special-delay byte, check if reached 0x00
E18A: 26 21       BNE   $E1AD            ;    {
E18C: 9E 88       LDX   $88              ;       X gets pointer to NEXT special delay time bytes
E18E: EC 81       LDD   ,X++             ;       D gets NEXT special delay time bytes
E190: 9F 88       STX   $88              ;       Update $88:$89 with address of NEXT special delay time bytes
E192: DD 8A       STD   $8A              ;       $8A:$8B gets the NEXT special delay time bytes from special delay table
                                         ;
                                         ;       // if reached a 0x0000 2-byte entry value
E194: 26 17       BNE   $E1AD            ;       if (reached end of special delay time table entry)
                                         ;       {
E196: 97 84       STA   $84              ;          $84 gets 0x00, transistor energize flag byte
E198: 97 81       STA   $81              ;          $81 gets 0x00, transistor energize time byte
E19A: B6 81 BF    LDA   $81BF            ;          A gets 0x22 from $81BF, Bank value of TransistorTable[]
E19D: B7 3F FC    STA   $3FFC            ;          Set bank to that of TransistorTable[]
E1A0: DE 82       LDU   $82              ;          U gets $82:$83 Address of the TransistorTable[] entry to energize
E1A2: A6 D4       LDA   [,U]             ;          A gets value of the transistor's cache byte in ram
E1A4: A4 44       ANDA  $0004,U         ;          Clear the bit in this transistor's cache byte
E1A6: A7 D4       STA   [,U]            ;          Put it back with transistor's bit cleared
E1A8: A7 D8 02    STA   [$02,U]         ;          Lastly, push updated cache byte thru to h/w register (turns off transistor)
                                        ;       }
                                        ;       else
                                        ;       {
                                        ;          goto REFRESH_THIS_SPECIAL_DELAY_TRANSISTOR_ON
                                        ;       }
E1AB: 20 18       BRA   $E1C5           ;    }
                                        ;    else
                                        ;    {
                                        ;    REFRESH_THIS_SPECIAL_DELAY_TRANSISTOR_ON:
E1AD: 96 76       LDA   $76             ;       //  ($8A is 1st byte of special delay)
E1AF: 91 8A       CMPA  $8A             ;       if ($76 == $8A) ($76 cycles from 0-7 each time through ISR)
E1B1: 26 12       BNE   $E1C5           ;       {
                                        ;          // This is when special solenoid time logic finally get the solenoid turned on
                                        ;          // when the $76 ISR 0-7 byte matches the 1st byte of the 2-byte special solenoid
                                        ;          // timer table entry.  The 2nd byte of the special table entry is a timer byte.
E1B3: F6 81 BF    LDB   $81BF           ;          B gets 0x22 from $81BF, Bank value of TransistorTable[]
E1B6: F7 3F FC    STB   $3FFC           ;          Set bank to that of TransistorTable[]
E1B9: DE 82       LDU   $82             ;          U gets $82:$83 Address of the TransistorTable[] entry to energize
E1BB: E6 44       LDB   $0004,U         ;          B gets value of the transistor's bitmask
E1BD: 53          COMB                  ;          Flip bits so it's all 0s except for the 1-bit for the particular transistor
E1BE: EA D4       ORB   [,U]            ;          OR in the existing transistor set in this cache byte
E1C0: E7 D4       STB   [,U]            ;          Update the cache byte with this transistor's bit set
E1C2: E7 D8 02    STB   [$02,U]         ;          Lastly, push updated cache byte thru to h/w register (turns on transistor)
                                        ;       }
                                        ;    }
                                        ; }
                                        ;
E1C5: 96 76       LDA   $76             ; if ($76 != 0x00)  ($76 cycles from 0-7 each time through ISR)
E1C7: 27 3E       BEQ   $E207           ; {
E1C9: 81 02       CMPA  #$02            ;    if ($76 == 0x02)
E1CB: 27 69       BEQ   $E236           ;       goto SINGLE_TRANSISTOR_ENERGIZE
E1CD: 81 01       CMPA  #$01            ;    if ($76 != 0x01)
E1CF: 10 26 00 81 LBNE  $E254           ;       goto POST_GENERAL_TRANSISITORS_CHECK
                                        ;
                                        ; ---------------------------------------------------------------------------
                                        ; // Only do the following loop every 8 passes through the ISR (when $76 == 0x01)
                                        ; ---------------------------------------------------------------------------
                                        ; ---------------------------------------------------------------------------
                                        ; // Flasher FlasherEnergizeArray[] check, start
                                        ; ---------------------------------------------------------------------------
E1D3: 8E 03 84    LDX   #$0384          ;    X gets start of buffer 0x0384
                                        ;
```

```
                                           ; $0384 - $0389, FlasherEnergizeArray[]
                                           ; ----------------------------
                                           ; $0384-$0386, FlasherEnergizeArray[] entry
                                           ; $0387-$0389, FlasherEnergizeArray[] entry
                                           ;
                                           ;
                                           ; $7F:$80 FlasherEnergizeArray[] pointer, next free entry, $0384, $0387, or $038A (array full)
                                           ;         Points to the next free entry, points to $0384 when no flashers energized.
                                           ;                                        points to $038A when FlasherEnergizeArray[] is full
                                           ;
                                           ; Each FlasherEnergizeArray[] entry:
                                           ;                  Byte[0] == on-time
                                           ;                  Bytes[1..2] == pointer into TransistorTable[] for this transistor.
                                           ;
                                           ; Flasher entries are in the FlasherEnergizeArray[] in no particular order. All entries
                                           ; are scaned and if their on-time byte reaches zero then that entry's transistor is turned
                                           ; off and the table is consolidated by moving the last table entry into the one that just
                                           ; expired.
                                           ;
                                           ;
                                           ;    // for each active flasher FlasherEnergizeArray[] entry (entries at address < $7F:$80)
                                           ;    while (1)
                                           ;    {
                                           ;       // if "last free FlasherEnergizeArray[] entry" pointer matches start of buffer,
                                           ;       //  then nothing to do
E1D6: 9C 7F       CPX   $7F                ;       if (X <= $7F:$80)
                                           ;       {
E1D8: 24 7A       BCC   $E254              ;          goto POST_GENERAL_TRANSISITORS_CHECK
                                           ;       }
E1DA: A6 84       LDA   ,X                 ;       A gets 1st byte of the X pointer 3-byte FlasherEnergizeArray[] ($0384+) entry
E1DC: 90 77       SUBA  $77                ;       Subtract $77 from the 1st byte of the $7F:$80 3-byte FlasherEnergizeArray[] entry
E1DE: A7 84       STA   ,X                 ;       if (($0384 -= $77) <= 0x00), if 1st byte of 3-byte buffer entry finally reached 0x00
for this entry
E1E0: 22 21       BHI   $E203              ;       {
E1E2: B6 81 BF    LDA   $81BF              ;          A gets 0x22 from $81BF, Bank value of TransistorTable[]
E1E5: B7 3F FC    STA   $3FFC              ;          Set bank to that of TransistorTable[]
E1E8: EE 01       LDU   $0001,X            ;          U gets pointer value from 2nd and 3rd bytes of FlasherEnergizeArray[] entry,
                                           ;           points to TransistorTable[]
E1EA: A6 D4       LDA   [,U]               ;          A gets value of transistor's cache byte in ram
E1EC: A4 44       ANDA  $0004,U            ;          AND A with transistor's mask, clears its bit
E1EE: A7 D4       STA   [,U]               ;          Update the cache byte with transistor's bit cleared
E1F0: A7 D8 02    STA   [$02,U]            ;          Lastly, push updated cache byte thru to h/w register (turns off transistor)
                                           ;          //-------------------------------------------------------------------------
                                           ;          // Now just write the last valid table entry over the current table entry.  This
                                           ;          // just overwrites the entry which we just aged (and turned off transistor), thus
```

```
                                          ;         // shrinking the FlasherEnergizeArray[] by 1 entry
                                          ;         //---------------------------------------------------------------------------
E1F3: DE 7F       LDU   $7F               ;         U gets $7F:80 pointer, "last free FlasherEnergizeArray[] entry"
E1F5: 33 5D       LEAU  $FFFD,U           ;         U -= 3, point to previous bytes in $0384+ buffer
E1F7: A6 C4       LDA   ,U                ;         A gets 1st byte of previous 3-byte entry
E1F9: A7 84       STA   ,X                ;         Put it at current X pointer (Some address in buffer starting at $0384)
E1FB: EC 41       LDD   $0001,U           ;         D gets 2nd & 3rd bytes of previous 3-byte entry
E1FD: ED 01       STD   $0001,X           ;         Put it at X pointer
E1FF: DF 7F       STU   $7F               ;         Put address of previus 3-byte entry at $7F:$80
E201: 20 D3       BRA   $E1D6             ;      }
E203: 30 03       LEAX  $0003,X           ;      X += 3, next 3-byte entry in buffer (that started at $0384)
E205: 20 CF       BRA   $E1D6             ;   }
                                          ; }
                                          ; ------------------------------------------------------------------------------
                                          ; // FlasherEnergizeArray[] check, done
                                          ; ------------------------------------------------------------------------------
                                          ;
E207: 96 84       LDA   $84               ; if ($84 != 0x00)
E209: 27 08       BEQ   $E213             ; {
E20B: 2A 23       BPL   $E230             ;   if ($84 & 0x80)
                                          ;   {
E20D: 96 8A       LDA   $8A               ;      if ($8A != 0x00) // $8A is 1st byte of 2-byte special solenoid table entry bytes
                                          ;      {
E20F: 26 0E       BNE   $E21F             ;         goto PROCESS_SPECIAL_SOLENOID_TIMER_BYTES_82_83
                                          ;      }
                                          ;   }
                                          ;   goto  POST_SPECIAL_SOLENOID_TIMERS_CHECK // skip over special solenoid table stuff
E211: 20 1D       BRA   $E230             ; }
                                          ;
E213: 96 81       LDA   $81               ; if (current transistor energize time != 0x00)
E215: 27 19       BEQ   $E230             ; {
E217: 90 77       SUBA  $77               ;  if ((time -= $77) <= 0x00)
E219: 97 81       STA   $81               ;  {
E21B: 22 13       BHI   $E230             ;
E21D: 0F 81       CLR   $81               ;     $81 gets 0x00 // in case it decremented to below zero, force to 0x00
                                          ;
                                          ;   PROCESS_SPECIAL_SOLENOID_TIMER_BYTES_82_83:
E21F: B6 81 BF    LDA   $81BF             ;     A gets 0x22 from $81BF, Bank value of TransistorTable[]
E222: B7 3F FC    STA   $3FFC             ;     Set bank to that of TransistorTable[]
E225: DE 82       LDU   $82               ;     U gets $82:$83 Address of the TransistorTable[] entry to energize
E227: A6 D4       LDA   [,U]              ;     A gets value of the transistor's cache byte in ram
E229: A4 44       ANDA  $0004,U           ;     Clear the bit in this transistor's cache byte
E22B: A7 D4       STA   [,U]              ;     Put it back with transistor's bit cleared
E22D: A7 D8 02    STA   [$02,U]           ;     Lastly, push updated cache byte thru to h/w register (turns off transistor)
                                          ;   }
```

```
                                       ; }
                                       ;
                                       ;
                                       ;
                                       ; POST_SPECIAL_SOLENOID_TIMERS_CHECK:
                                       ;
E230: AD 9F 81 2A JSR   [$812A]        ; Call [$812A], at $812A is $E61E,FF, so this calls $E61E, CheckPopBumpersSlingshotsOff()
E234: 20 1E       BRA   $E254          ; goto POST_GENERAL_TRANSISITORS_CHECK
                                       ;
                                       ;
                                       ;
                                       ; SINGLE_TRANSISTOR_ENERGIZE
                                       ;
E236: 96 84       LDA   $84            ; if ($84 > 0x00)
E238: 27 1A       BEQ   $E254          ; {
E23A: 2B 18       BMI   $E254          ;    // Here is where $84 decrements by 2.  For special timings, it was loaded with 0x01
E23C: 80 02       SUBA  #$02           ;    // which now decrements to 0xFF which triggers the code above when 0x80 bit is set.
E23E: 97 84       STA   $84            ;    if (($84 -= 0x02) == 0x00)
E240: 26 12       BNE   $E254          ;    {
E242: B6 81 BF    LDA   $81BF          ;       A gets 0x22 from $81BF, Bank value of TransistorTable[]
E245: B7 3F FC    STA   $3FFC          ;       Set bank to that of TransistorTable[]
E248: DE 82       LDU   $82            ;       U gets the TransistorTable[] address out of $82
E24A: A6 44       LDA   $0004,U        ;       A gets this transistor's bitmask
E24C: 43          COMA                 ;       Flip the bitmask bits so it has single 1-bit for the transistor
E24D: AA D4       ORA   [,U]           ;       Include any existing bits set in this transistor's cache byte
E24F: A7 D4       STA   [,U]           ;       Set this transistors cache byte, set bit for this transistor
E251: A7 D8 02    STA   [$02,U]        ;       Lastly, push updated cache byte thru to h/w register (turns on transistor)
                                       ;    }
                                       ; }
                                       ;
                                       ;
                                       ;
                                       ; POST_GENERAL_TRANSISITORS_CHECK:
                                       ;
E254: 96 11       LDA   $11            ; A gets current bank
E256: B7 3F FC    STA   $3FFC          ; Make sure we're still in the current bank
                                       ;
```

If you trace through the code, especially after tracing into the other functions described earlier in this document, you should be able to see how a particular transistor will get turned on or off. Due to the wealth of comments added into the code above, no additional commentary seems necessary here.

Immediately after the code above, in the ISR, is the following code chunk. You can see it is dealing with the Fliptronic II board which is somewhat off topic here since this document is mainly about the transistors on the WPC power driver board:

```
E259: 96 0C       LDA   $0C              ; if ($0C & 0x01)
E25B: 85 01       BITA  #$01             ; {
E25D: 26 12       BNE   $E271            ;    goto POST_FLIPTRONIC_SWITCHES_READ
                                         ; }
E25F: 85 02       BITA  #$02             ; if ($0C & 0x02)
                                         ; {
E261: 26 0A       BNE   $E26D            ;    goto FLIPTRONIC_SWITCHES_READ
                                         ; }
E263: 96 86       LDA   $86              ; if ($86 == 0x00) // if WPC Test Mode (menu system) isn't in progress
                                         ; {
E265: 27 0A       BEQ   $E271            ;    goto POST_FLIPTRONIC_SWITCHES_READ
                                         ; }
                                         ;
E267: AD 9F 81 27 JSR   [$8127]          ; Call [$8127], at $8127 is $E580,FF, so this calls $E580, CheckPopBumpersSlingshotsOn()
E26B: 20 04       BRA   $E271            ; goto POST_FLIPTRONIC_SWITCHES_READ
                                         ;
                                         ;---------------------------------------------------------------------
                                         ;---------------------------------------------------------------------
                                         ;
                                         ;FLIPTRONIC_SWITCHES_READ:
E26D: AD 9F 80 DF JSR   [$80DF]          ; Call [$80DF], at $80DF is E711,FF so this calls $E711
                                         ; ProcessFliptronicIISwitches()
                                         ;
                                         ;POST_FLIPTRONIC_SWITCHES_READ:
E271: 96 A0       LDA   $A0              ; A gets $A0
E273: 43          COMA                   ; Flip the bits
E274: B7 3F D4    STA   $3FD4            ; Store value to FliptronicII Solenoids
                                         ;
```

The logic above shows that, depending on the value of $0C, the Fliptronic II switches get read. The $0C byte changes each time through the ISR. It advances from value 0x00 through 0x0F and repeats. With this information you can figure out how frequently each of the different functions, above, get called. The important thing to know is that all of the code gets called on a regular basis, just not every single time through the ISR.

The code chunk above consists of the following components, each of which will be described in more detail later:

- Call to CheckPopBumpersSlingshotsOn(), checks pop-bumper switches and sets cache bytes with appropriate bits and also updates the h/w registers directly (for immediate solenoid energization which is obviously desirable for pop-bumpers and slingshots).
    - Note the complementary call to CheckPopBumpersSlingshotsOff() was buried in a previous ISR Code chunk at $E230, both of these functions will be shown in detail later next.

- Call to ProcessFliptronicIISwitches(), updates $A0 byte with a 1-bit set for each Fliptronic II board transistor to energize. Since IJ_L7 only has 2 flippers, it's expected that its call to ProcessFliptronicIISwitches() will only manipulate 4 of the 8 bits in $A0, those 4 that correspond to the 2 flippers (each flipper coil is associated with 2 transistors, the "flip" and the "hold").

- Updating of the Fliptronic II Solenoids with inverse of $A0 byte. Apparently the Fliptronic II uses 0-bit to turn on each transistor. The WPC code will store 1-bits in $A0 for transistors to energize (so it looks consistent with everything else) and so here is the one place where it gets inversed just prior to pushing the value to the h/w register at $3FD4. These values ($A0 and $3FD4) match some of the entries in the TransistorTable[].

CheckPopBumpersSlingshotsOn()

This function is called periodically during the IRQ_Routine() above. This function will check the switch for each pop-bumper and slingshot and decide if it should energize its solenoid. If so, it updates the solenoid's cache byte <and> immediately updates its h/w register as well.

It is important to note that these solenoids are subject to an inter-pulse delay time which must elapse before the solenoid is allowed to pulse again. This is more evident in the next function, CheckPopBumpersSlingshotsOff(), however the comments in this function do refer to this concept with the slingshot handling. The code comments below should help describe how this function operates.

```
;---------------------------------------;------------------------------------------------------
; CheckPopBumpersSlingshotsOn()
;
; Called from ISR
;
                                         ; ---------------------------------------------------
                                         ;
E580: D6 22       LDB   $22              ; B = ($22 & $14) // 3rd switch column <and> non-shorted switch row mask
E582: D4 14       ANDB  $14              ;
E584: 96 C9       LDA   $C9              ; if ($C9 == 0x00) // if transistor off: left jet bumper
E586: 26 15       BNE   $E59D            ; {
E588: C5 10       BITB  #$10             ;    if (B & 0x10) // if switch closed: left-jet
E58A: 27 11       BEQ   $E59D            ;    {
E58C: 96 7E       LDA   $7E              ;       A gets Transistor Cache Byte 0x7E value
E58E: 8A 01       ORA   #$01             ;       Set 0x01 transistor (left jet bumper)
E590: 97 7E       STA   $7E              ;       Update cache byte with transistor-on bit
E592: B7 3F E3    STA   $3FE3            ;       Update h/w register with transistor-on bit
E595: 86 03       LDA   #$03             ;       $C9 = 0x03
E597: 97 C9       STA   $C9              ;       $CA = 0x16 // set transistor on-time 0x0316
E599: 86 16       LDA   #$16             ;    }
E59B: 97 CA       STA   $CA              ; }
                                         ;
                                         ; ---------------------------------------------------
                                         ;
E59D: 96 E2       LDA   $E2              ; if (!($E2 & 0x01)) <special check only for bottom jet bumper> ?
E59F: 85 01       BITA  #$01             ; {
E5A1: 26 19       BNE   $E5BC            ;    if ($CB == 0x00) // if transistor off: bottom jet bumper
E5A3: 96 CB       LDA   $CB              ;    {
E5A5: 26 15       BNE   $E5BC            ;       if (B & 0x40) // if switch closed: bottom-jet
E5A7: C5 40       BITB  #$40             ;       {
E5A9: 27 11       BEQ   $E5BC            ;          A gets Transistor Cache Byte 0x7E value
E5AB: 96 7E       LDA   $7E              ;          Set 0x04 transistor (lower jet bumper)
E5AD: 8A 04       ORA   #$04             ;          Update cache byte with transistor-on bit
E5AF: 97 7E       STA   $7E              ;          Update h/w register with transistor-on bit
E5B1: B7 3F E3    STA   $3FE3            ;          $CB = 0x03
E5B4: 86 03       LDA   #$03             ;          $CC = 0x16 // set transistor on-time 0x0316
E5B6: 97 CB       STA   $CB              ;       }
E5B8: 86 16       LDA   #$16             ;    }
E5BA: 97 CC       STA   $CC              ; }
                                         ;
                                         ; ---------------------------------------------------
                                         ;
E5BC: 96 CD       LDA   $CD              ; if ($CD == 0x00) // if transistor off: right jet bumper
E5BE: 26 15       BNE   $E5D5            ; {
E5C0: C5 20       BITB  #$20             ;    if (B & 0x20) // if switch closed: right-jet
E5C2: 27 11       BEQ   $E5D5            ;    {
E5C4: 96 7E       LDA   $7E              ;       A gets Transistor Cache Byte 0x7E value
E5C6: 8A 02       ORA   #$02             ;       Set 0x02 transistor (right jet bumper)
E5C8: 97 7E       STA   $7E              ;       Update cache byte with transistor-on bit
E5CA: B7 3F E3    STA   $3FE3            ;       Update h/w register with transistor-on bit
E5CD: 86 03       LDA   #$03             ;       $CD = 0x03
E5CF: 97 CD       STA   $CD              ;       $CE = 0x16 // set transistor on-time 0x0316
E5D1: 86 16       LDA   #$16             ;    }
E5D3: 97 CE       STA   $CE              ; }
                                         ;
                                         ; ---------------------------------------------------
                                         ;
E5D5: D6 23       LDB   $23              ; B = ($23 & $14) // 4th switch column <and> non-shorted switch row mask
E5D7: D4 14       ANDB  $14              ; if (CF == 0x00) // if transistor off: right slingshot
E5D9: 96 CF       LDA   $CF              ; {
E5DB: 26 1B       BNE   $E5F8            ;    if (B & 0x80) // if switch closed: right slingshot
E5DD: C5 80       BITB  #$80             ;    {
E5DF: 27 17       BEQ   $E5F8            ;       A gets Transistor Cache Byte 0x7E value
E5E1: 96 7E       LDA   $7E              ;       Set 0x10 transistor (right slingshot)
E5E3: 8A 10       ORA   #$10             ;       Update cache byte with transistor-on bit
E5E5: 97 7E       STA   $7E              ;       Update h/w register with transistor-on bit
E5E7: B7 3F E3    STA   $3FE3            ;       $CF = 0x40 // set transistor on-time 0x40, right slingshot
E5EA: 86 40       LDA   #$40             ;       if ($D0 < 0) // If other slingshot is in the inter-pulse delay, refresh it to -24.
E5EC: 97 CF       STA   $CF              ;       {
E5EE: B6 00 D0    LDA   $00D0            ;          $D0 = 0xE8 // left slingshot needs to increase this to 0x00 before it kicks again.
E5F1: 2A 05       BPL   $E5F8            ;       }
E5F3: 86 E8       LDA   #$E8             ;    }
E5F5: B7 00 D0    STA   $00D0            ; }
                                         ;
                                         ; ---------------------------------------------------
                                         ;
E5F8: D6 22       LDB   $22              ; B = ($22 & $14) // 3rd switch column <and> non-shorted switch row mask
E5FA: D4 14       ANDB  $14              ; if ($EA == 0x00) // <special disable just for left-slingshot> ?
```

```
E5FC: 0D EA       TST   $EA             ; {
E5FE: 26 1D       BNE   $E61D           ;    if ($D0 == 0x00) // if transistor off: left slingshot
E600: 96 D0       LDA   $D0             ;    {
E602: 26 19       BNE   $E61D           ;       if (B & 0x04) // if switch closed: left slingshot
E604: C5 04       BITB  #$04            ;       {
E606: 27 15       BEQ   $E61D           ;          A gets Transistor Cache Byte 0x7E value
E608: 96 7E       LDA   $7E             ;          Set 0x08 transistor (left slingshot)
E60A: 8A 08       ORA   #$08            ;          Update cache byte with transistor-on bit
E60C: 97 7E       STA   $7E             ;          Update h/w register with transistor-on bit
E60E: B7 3F E3    STA   $3FE3           ;          $D0 = 0x40 // set transistor on-time 0x40, left slingshot
E611: 86 40       LDA   #$40            ;          if ($CF < 0) // If other slingshot is in the inter-pulse delay, refresh it to -24.
E613: 97 D0       STA   $D0             ;          {
E615: 96 CF       LDA   $CF             ;             $CF = 0xE8 // right slingshot needs to increase this to 0x00 kicking again.
E617: 2A 04       BPL   $E61D           ;          }
E619: 86 E8       LDA   #$E8            ;       }
E61B: 97 CF       STA   $CF             ;    }
                                        ; }
                                        ;
E61D: 39          RTS                   ; Done.
                                        ;
----------------------------------------;--------------------------------------------------
```

CheckPopBumpersSlingshotsOff()

As mentioned above, this is the complementary function to CheckPopBumpersSlingshotsOn() and it gets called periodically in the IRQ_Routine().

Also, as mentioned, this function will ensure that an inter-pulse delay period is observed prior to allowing the same solenoid to be re-energized. This probably is needed to help reduce wear and tear on the rapidly firing pop-bumper and slingshot solenoids.

```
----------------------------------------;---------------------------------------------------
; CheckPopBumpersSlingshotsOff()
;
; Called from ISR
;
E61E: 96 C9       LDA   $C9             ; if ($C9 != 0x00) // if transistor minimum on-time: left jet bumper
E620: 27 34       BEQ   $E656           ; {
E622: 2B 24       BMI   $E648           ;    if ($C9 >= 0x00) // if transistor minimum on-time >= 0
E624: D6 22       LDB   $22             ;    {
E626: C5 10       BITB  #$10            ;       if (!($22 & 0x10)) // if switch open: left-jet
E628: 26 10       BNE   $E63A           ;       {
E62A: 4A          DECA                  ;          if (--$C9 == 0x00) // if transistor minimum on-time decremented <= 0
E62B: 97 C9       STA   $C9             ;          {
E62D: 26 27       BNE   $E656           ;             A gets Transistor Cache Byte 0x7E value
E62F: 96 7E       LDA   $7E             ;             Clear 0x01 transistor (left jet bumper)
E631: 84 FE       ANDA  #$FE            ;             Update cache byte with transistor-off bit
E633: 97 7E       STA   $7E             ;             Update h/w register with transistor-off bit
E635: B7 3F E3    STA   $3FE3           ;          }
E638: 20 1C       BRA   $E656           ;          goto <check next transistor>
                                        ;       }
                                        ;       // switch is still closed, enforce maximum on-time
E63A: 86 03       LDA   #$03            ;       $C9 = 0x03 // reset minimum on-time to 0x03
E63C: 97 C9       STA   $C9             ;       if (--$CA == 0x00) // if maximum on-time finally reached 0x00, turn off transistor
E63E: 0A CA       DEC   $CA             ;       {
E640: 26 14       BNE   $E656           ;          $C9 = 0xF0 // inter-pulse delay time -16
E642: 86 F0       LDA   #$F0            ;          goto <this transistor off>
E644: 97 C9       STA   $C9             ;       }
E646: 20 E7       BRA   $E62F           ;    }
                                        ;    // Transistor is in its inter-pulse period (forced off)
E648: D6 22       LDB   $22             ;    if ($22 & 0x10) // if switch closed: left-jet
E64A: C5 10       BITB  #$10            ;    {
E64C: 27 06       BEQ   $E654           ;       $C9 = 0xF0 // refresh inter-pulse delay to -16
E64E: 86 F0       LDA   #$F0            ;    }
E650: 97 C9       STA   $C9             ;    else // else switch is open, allow inter-pulse timeout to increment
                                        ;    {
E652: 20 02       BRA   $E656           ;       $C9++ // increment inter-pulse time towards 0
```

```
                                               ;    }
E654: 0C C9       INC   $C9                    ; }
                                               ;
                                               ; ----------------------------------------------------
                                               ;
E656: 96 CB       LDA   $CB                    ; if ($CB != 0x00) // if transistor minimum on-time: lower jet bumper
E658: 27 34       BEQ   $E68E                  ; {
E65A: 2B 24       BMI   $E680                  ;    if ($CB >= 0x00) // if transistor minimum on-time >= 0
E65C: D6 22       LDB   $22                    ;    {
E65E: C5 40       BITB  #$40                   ;       if (!($22 & 0x40)) // if switch open: bottom-jet
E660: 26 10       BNE   $E672                  ;       {
E662: 4A          DECA                         ;          if (--$CB == 0x00) // if transistor minimum on-time decremented <= 0
E663: 97 CB       STA   $CB                    ;          {
E665: 26 27       BNE   $E68E                  ;             A gets Transistor Cache Byte 0x7E value
E667: 96 7E       LDA   $7E                    ;             Clear 0x04 transistor (lower jet bumper)
E669: 84 FB       ANDA  #$FB                   ;             Update cache byte with transistor-off bit
E66B: 97 7E       STA   $7E                    ;             Update h/w register with transistor-off bit
E66D: B7 3F E3    STA   $3FE3                  ;          }
E670: 20 1C       BRA   $E68E                  ;          goto <check next transistor>
                                               ;       }
                                               ;       // switch is still closed, enforce maximum on-time
E672: 86 03       LDA   #$03                   ;       $CB = 0x03 // reset minimum on-time to 0x03
E674: 97 CB       STA   $CB                    ;       if (--$CC == 0x00) // if maximum on-time finally reached 0x00, turn off transistor
E676: 0A CC       DEC   $CC                    ;       {
E678: 26 14       BNE   $E68E                  ;          $CB = 0xF0 // inter-pulse delay time -16
E67A: 86 F0       LDA   #$F0                   ;          goto <this transistor off>
E67C: 97 CB       STA   $CB                    ;       }
E67E: 20 E7       BRA   $E667                  ;    }
                                               ;    // Transistor is in its inter-pulse period (forced off)
E680: D6 22       LDB   $22                    ;    if ($22 & 0x40) // if switch closed: bottom-jet
E682: C5 40       BITB  #$40                   ;    {
E684: 27 06       BEQ   $E68C                  ;       $CB = 0xF0 // refresh inter-pulse delay to -16
E686: 86 F0       LDA   #$F0                   ;    }
E688: 97 CB       STA   $CB                    ;    else // else switch is open, allow inter-pulse timeout to increment
                                               ;    {
E68A: 20 02       BRA   $E68E                  ;       $CB++ // increment inter-pulse time towards 0
                                               ;    }
E68C: 0C CB       INC   $CB                    ; }
                                               ;
                                               ; ----------------------------------------------------
                                               ;
E68E: 96 CD       LDA   $CD                    ; if ($CD != 0x00) // if transistor minimum on-time: right jet bumper
E690: 27 34       BEQ   $E6C6                  ; {
E692: 2B 24       BMI   $E6B8                  ;    if ($CD >= 0x00) // if transistor minimum on-time >= 0
E694: D6 22       LDB   $22                    ;    {
```

```
E696: C5 20       BITB  #$20            ;       if (!($22 & 0x20)) // if switch open: right-jet
E698: 26 10       BNE   $E6AA           ;       {
E69A: 4A          DECA                  ;          if (--$CD == 0x00) // if transistor minimum on-time decremented <= 0
E69B: 97 CD       STA   $CD             ;          {
E69D: 26 27       BNE   $E6C6           ;             A gets Transistor Cache Byte 0x7E value
E69F: 96 7E       LDA   $7E             ;             Clear 0x20 transistor (right jet bumper)
E6A1: 84 FD       ANDA  #$FD            ;             Update cache byte with transistor-off bit
E6A3: 97 7E       STA   $7E             ;             Update h/w register with transistor-off bit
E6A5: B7 3F E3    STA   $3FE3           ;          }
E6A8: 20 1C       BRA   $E6C6           ;          goto <check next transistor>
                                        ;       }
                                        ;       // switch is still closed, enforce maximum on-time
E6AA: 86 03       LDA   #$03            ;       $CD = 0x03 // reset minimum on-time to 0x03
E6AC: 97 CD       STA   $CD             ;       if (--$CE == 0x00) // if maximum on-time finally reached 0x00, turn off transistor
E6AE: 0A CE       DEC   $CE             ;       {
E6B0: 26 14       BNE   $E6C6           ;          $CD = 0xF0 // inter-pulse delay time -16
E6B2: 86 F0       LDA   #$F0            ;          goto <this transistor off>
E6B4: 97 CD       STA   $CD             ;       }
E6B6: 20 E7       BRA   $E69F           ;    }
                                        ;    // Transistor is in its inter-pulse period (forced off)
E6B8: D6 22       LDB   $22             ;    if ($22 & 0x20) // if switch closed: right-jet
E6BA: C5 20       BITB  #$20            ;    {
E6BC: 27 06       BEQ   $E6C4           ;       $CD = 0xF0 // refresh inter-pulse delay time to -16
E6BE: 86 F0       LDA   #$F0            ;    }
E6C0: 97 CD       STA   $CD             ;    else // else switch is open, allow inter-pulse time to increment
                                        ;    {
E6C2: 20 02       BRA   $E6C6           ;       $CD++ // increment inter-pulse time towards 0
                                        ;    }
E6C4: 0C CD       INC   $CD             ; }
                                        ;
                                        ; ------------------------------------------------------
                                        ;
E6C6: 96 CF       LDA   $CF             ; if ($CF != 0x00) // if transistor on-time: right slingshot
E6C8: 27 1F       BEQ   $E6E9           ; {
E6CA: 2B 15       BMI   $E6E1           ;    if ($CF >= 0x00) // if transistor on-time >= 0
E6CC: 90 77       SUBA  $77             ;    {
E6CE: 97 CF       STA   $CF             ;       if (($CF -= $77) <= 0x00) // if transistor time decremented <= 0
E6D0: 22 17       BHI   $E6E9           ;       {
E6D2: 96 7E       LDA   $7E             ;          A gets Transistor Cache Byte 0x7E value
E6D4: 84 EF       ANDA  #$EF            ;          Clear 0x10 transistor (right slingshot)
E6D6: 97 7E       STA   $7E             ;          Update cache byte with transistor-off bit
E6D8: B7 3F E3    STA   $3FE3           ;          Update h/w register with transistor-off bit
E6DB: 86 E8       LDA   #$E8            ;          $CF = 0xE8 // set transistor off-time -24
E6DD: 97 CF       STA   $CF             ;       }
E6DF: 20 08       BRA   $E6E9           ;       goto <check next transistor>
```

```
                                        ;    }
                                        ;    // Transistor is in its inter-pulse period (forced off)
E6E1: D6 23       LDB   $23             ;    if (!($23 & 0x80)) // if switch open: right slingshot
E6E3: C5 80       BITB  #$80            ;    {
E6E5: 26 F4       BNE   $E6DB           ;       $CF++; // inter-pulse time, increment to 0x00 before it can turn on again
                                        ;    }
E6E7: 0C CF       INC   $CF             ; }
                                        ;
                                        ; ---------------------------------------------------
                                        ;
E6E9: 96 D0       LDA   $D0             ; if ($D0 != 0x00) // if transistor on-time: left slingshot
E6EB: 27 1F       BEQ   $E70C           ; {
E6ED: 2B 15       BMI   $E704           ;    if ($D0 >= 0x00) // if transistor on-time >= 0
E6EF: 90 77       SUBA  $77             ;    {
E6F1: 97 D0       STA   $D0             ;       if (($D0 -= $77) <= 0x00) // if transistor time decremented <= 0
E6F3: 22 17       BHI   $E70C           ;       {
E6F5: 96 7E       LDA   $7E             ;          A gets Transistor Cache Byte 0x7E value
E6F7: 84 F7       ANDA  #$F7            ;          Clear 0x08 transistor (left slingshot)
E6F9: 97 7E       STA   $7E             ;          Update cache byte with transistor-off bit
E6FB: B7 3F E3    STA   $3FE3           ;          Update h/w register with transistor-off bit
E6FE: 86 E8       LDA   #$E8            ;          $D0 = 0xE8 // set transistor off-time -24
E700: 97 D0       STA   $D0             ;       }
E702: 20 08       BRA   $E70C           ;       goto <end of function>
                                        ;    }
                                        ;    // Transistor is in its inter-pulse period (forced off)
E704: D6 22       LDB   $22             ;    if (!($22 & 0x04)) // if switch open: left slingshot
E706: C5 04       BITB  #$04            ;    {
E708: 26 F4       BNE   $E6FE           ;       $D0++; // inter-pulse time, increment to 0x00 before it can turn on again
                                        ;    }
E70A: 0C D0       INC   $D0             ; }
                                        ;
E70C: 39          RTS                   ; Done.
                                        ;
;---------------------------------------;-------------------------------------------------------
```

ProcessFliptronicIISwitches()

This function was shown earlier as being called prior to pushing the (inverse of the) $A0 byte into h/w register $3FD4 in order to set the state of the 8 transistors on the Fliptronic II board.

As you can see this function hasn't yet been fully annotated. This function is getting outside the scope of this document, so the important thing to understand is that this function will handle Fliptronic II board switches (and presumably, the flipper button switches) and update the $A0 byte with the desired state of the flipper-related solenoids in $A0.

Also shown below, for reference, are the 4 bytes that immediately proceed this function. These bytes are used during the initialization of Fliptronic II related ram, used to indicate the fact that the lower left and right flippers are to be used in this game (and not the upper flippers). At a future date it is expected that all of the Fliptronic II code will be mapped out and this, and other, code will be described in a separate document.

```
;----------------------------------------;----------------------------------------------------------
                                         ;
E70D: 08 00                              ; FlipperDataMask_LowerLeft[], value loaded into $B5 in bank $3D
E70F: 02 00                              ; FlipperDataMask_LowerRight[], value loaded into $B5 in bank $3D
                                         ;
;----------------------------------------;----------------------------------------------------------
; ProcessFliptronicIISwitches()
;
                                         ;
E711: F6 3F D4    LDB   $3FD4            ; Read Fliptronic II switches into B
E714: 53          COMB                   ; Reverse the logic on the inputs
E715: D7 B1       STB   $B1              ; Store this value to $B1
E717: 96 B1       LDA   $B1              ; Get a copy of the value to A
                                         ;
                                         ; EOS Inputs on J906 use mask 0xAA
                                         ; Cabinet buttons on J906 use mask 0x55
                                         ;
E719: 84 55       ANDA  #$55             ; Clear EOS switch indicators in A
                                         ;
E71B: C5 02       BITB  #$02             ; Test the 0x02 bit of the B register holding of inputs
E71D: 27 02       BEQ   $E721            ; If 0x02 bit is NOT set skip to $E721, J906-3, Lower-Right flipper EOS
E71F: 9A B6       ORA   $B6              ; ..otherwise 0x02 bit *IS* set, Set the EOS flag, for Lower-Right flipper,
                                         ;   if $B6 was initialized with it.  On IJ_L7 $B6 has 0x02.
                                         ;
E721: C5 20       BITB  #$20             ; Test the 0x20 bit of the B register holding of inputs
E723: 27 02       BEQ   $E727            ; If 0x20 bit is NOT set skip to $E727, J906-4, Upper-right flipper EOS
```

```
E725: 9A B8       ORA   $B8             ; ..otherwise 0x20 bit *IS* set, Set the EOS flag, for Upper-right flipper EOS,
                                        ;   if $B8 was initialized with it.  On IJ_L7 $B8 has 0x00, so no bit gets set.
                                        ;
E727: C5 08       BITB  #$08            ; Test the 0x08 bit of the B register holding of inputs
E729: 27 02       BEQ   $E72D           ; If 0x08 bit is NOT set skip to $E72D, J906-5, Lower-Left flipper EOS
E72B: 9A B5       ORA   $B5             ; ..otherwise 0x08 bit *IS* set, Set the EOS flag, for Lower-Left flipper EOS,
                                        ;   if $B5 was initialized with it.  On IJ_L7 $B5 has 0x08
                                        ;
E72D: C5 80       BITB  #$80            ; Test the 0x80 bit of the B register holding of inputs
E72F: 27 02       BEQ   $E733           ; If 0x80 bit is NOT set skip to $E733, J906-1, Upper-Left flipper EOS
E731: 9A B7       ORA   $B7             ; ..otherwise 0x80 bit *IS* set, Set the EOS flag, for Upper-Left flipper EOS,
                                        ;   if $B7 was initialized with it.  On IJ_L7 B7 has 0x00, so no bit gets set.
                                        ;
E733: D6 B2       LDB   $B2             ; Load up $B2
E735: 27 0D       BEQ   $E744           ; If $B2 is 0x00, skip to $E744, usually 0x00 but gets set to nonzero in code in bank 0x30
E737: 97 A2       STA   $A2             ;
E739: 96 B2       LDA   $B2             ;
E73B: 43          COMA                  ;
E73C: 94 A2       ANDA  $A2             ;
E73E: D4 B3       ANDB  $B3             ;
E740: D7 A2       STB   $A2             ;
E742: 9A A2       ORA   $A2             ;
                                        ;
E744: D6 A1       LDB   $A1             ; B gets byte from $A1
E746: D7 A2       STB   $A2             ; Copy it to $A2
E748: 97 A1       STA   $A1             ; Store new value of switch inputs A into $A1
E74A: 53          COMB                  ; Flip bits in byte from $A1
E74B: D4 A1       ANDB  $A1             ; AND with $A1 clear out bits
E74D: D7 A4       STB   $A4             ; Store result into $A4
                                        ;
E74F: 96 A1       LDA   $A1             ; A gets $A1
E751: 43          COMA                  ; Flip its bits
E752: 94 A2       ANDA  $A2             ; and clear out bits from $A1 within $A2
E754: 97 A3       STA   $A3             ; Store result in $A3
                                        ;
                                        ;--------------------------------------------------------------------------
                                        ;
E756: 96 B1       LDA   $B1             ; A gets byte from $B1, The raw Fliptronic II switch reads
E758: 98 AD       EORA  $AD             ; XOR with $AD
E75A: 1F 89       TFR   A,B             ; Transfer to B
E75C: 94 AF       ANDA  $AF             ; AND with $AF
E75E: 9A AE       ORA   $AE             ; OR with $AE
E760: 97 AE       STA   $AE             ; Store A to $AE
E762: D7 AF       STB   $AF             ; Store B to $AF
                                        ;
```

```
E764: D6 87       LDB   $87              ; B gets $87
E766: 26 03       BNE   $E76B            ; If not 0x00, then process things
E768: 7E E8 47    JMP   $E847            ; ..otherwise $87 is 0x00, skip to the end
                                         ;
                                         ; The following code is extended processing on the Fliptronic II
                                         ; switches and possibly set the solenoid output byte $A0 which
                                         ; gets written after this function RTS back into the interrupt
                                         ; routine.
                                         ;
E76B: D6 A0       LDB   $A0              ;
E76D: 96 A1       LDA   $A1              ;
E76F: 85 08       BITA  #$08             ;
E771: 27 62       BEQ   $E7D5            ;
E773: 96 A5       LDA   $A5              ;
E775: 2B 26       BMI   $E79D            ;
E777: 26 38       BNE   $E7B1            ;
E779: 96 A1       LDA   $A1              ;
E77B: 84 04       ANDA  #$04             ;
E77D: 27 06       BEQ   $E785            ;
E77F: 96 B4       LDA   $B4              ;
E781: 8A 04       ORA   #$04             ;
E783: 20 04       BRA   $E789            ;
E785: 96 B4       LDA   $B4              ;
E787: 84 F3       ANDA  #$F3             ;
E789: 97 B4       STA   $B4              ;
E78B: 86 0A       LDA   #$0A             ;
E78D: 97 A5       STA   $A5              ;
E78F: CA 0C       ORB   #$0C             ;
E791: 20 46       BRA   $E7D9            ;
E793: 86 80       LDA   #$80             ;
E795: 97 A5       STA   $A5              ;
E797: C4 FB       ANDB  #$FB             ;
E799: CA 08       ORB   #$08             ;
E79B: 20 3C       BRA   $E7D9            ;
E79D: 96 A9       LDA   $A9              ;
E79F: 27 04       BEQ   $E7A5            ;
E7A1: 0A A9       DEC   $A9              ;
E7A3: 20 34       BRA   $E7D9            ;
E7A5: 96 A3       LDA   $A3              ;
E7A7: 85 04       BITA  #$04             ;
E7A9: 27 2E       BEQ   $E7D9            ;
E7AB: 86 3E       LDA   #$3E             ;
E7AD: 97 A9       STA   $A9              ;
E7AF: 20 C8       BRA   $E779            ;
E7B1: 0A A5       DEC   $A5              ;
```

```
E7B3: 27 DE        BEQ   $E793            ;
E7B5: 96 B4        LDA   $B4              ;
E7B7: 85 04        BITA  #$04             ;
E7B9: 26 1E        BNE   $E7D9            ;
E7BB: 85 08        BITA  #$08             ;
E7BD: 27 08        BEQ   $E7C7            ;
E7BF: 96 A5        LDA   $A5              ;
E7C1: 81 05        CMPA  #$05             ;
E7C3: 23 CE        BLS   $E793            ;
E7C5: 20 12        BRA   $E7D9            ;
E7C7: 96 A4        LDA   $A4              ;
E7C9: 85 04        BITA  #$04             ;
E7CB: 27 0C        BEQ   $E7D9            ;
E7CD: 96 B4        LDA   $B4              ;
E7CF: 8A 08        ORA   #$08             ;
E7D1: 97 B4        STA   $B4              ;
E7D3: 20 04        BRA   $E7D9            ;
                                          ;
E7D5: C4 F3        ANDB  #$F3             ;
E7D7: 0F A5        CLR   $A5              ;
E7D9: 96 A1        LDA   $A1              ;
E7DB: 85 02        BITA  #$02             ;
E7DD: 27 62        BEQ   $E841            ;
E7DF: 96 A6        LDA   $A6              ;
E7E1: 2B 26        BMI   $E809            ;
E7E3: 26 38        BNE   $E81D            ;
E7E5: 96 A1        LDA   $A1              ;
E7E7: 84 01        ANDA  #$01             ;
E7E9: 27 06        BEQ   $E7F1            ;
E7EB: 96 B4        LDA   $B4              ;
E7ED: 8A 01        ORA   #$01             ;
E7EF: 20 04        BRA   $E7F5            ;
E7F1: 96 B4        LDA   $B4              ;
E7F3: 84 FC        ANDA  #$FC             ;
E7F5: 97 B4        STA   $B4              ;
E7F7: 86 0A        LDA   #$0A             ;
E7F9: 97 A6        STA   $A6              ;
E7FB: CA 03        ORB   #$03             ;
E7FD: 20 46        BRA   $E845            ;
E7FF: 86 80        LDA   #$80             ;
E801: 97 A6        STA   $A6              ;
E803: C4 FE        ANDB  #$FE             ;
E805: CA 02        ORB   #$02             ;
E807: 20 3C        BRA   $E845            ;
E809: 96 AA        LDA   $AA              ;
```

```
E80B: 27 04       BEQ   $E811            ;
E80D: 0A AA       DEC   $AA              ;
E80F: 20 34       BRA   $E845            ;
E811: 96 A3       LDA   $A3              ;
E813: 85 01       BITA  #$01             ;
E815: 27 2E       BEQ   $E845            ;
E817: 86 3E       LDA   #$3E             ;
E819: 97 AA       STA   $AA              ;
E81B: 20 C8       BRA   $E7E5            ;
E81D: 0A A6       DEC   $A6              ;
E81F: 27 DE       BEQ   $E7FF            ;
E821: 96 B4       LDA   $B4              ;
E823: 85 01       BITA  #$01             ;
E825: 26 1E       BNE   $E845            ;
E827: 85 02       BITA  #$02             ;
E829: 27 08       BEQ   $E833            ;
E82B: 96 A6       LDA   $A6              ;
E82D: 81 05       CMPA  #$05             ;
E82F: 23 CE       BLS   $E7FF            ;
E831: 20 12       BRA   $E845            ;
E833: 96 A4       LDA   $A4              ;
E835: 85 01       BITA  #$01             ;
E837: 27 0C       BEQ   $E845            ;
E839: 96 B4       LDA   $B4              ;
E83B: 8A 02       ORA   #$02             ;
E83D: 97 B4       STA   $B4              ;
E83F: 20 04       BRA   $E845            ;
E841: C4 FC       ANDB  #$FC             ;
E843: 0F A6       CLR   $A6              ;
E845: D7 A0       STB   $A0              ;
E847: 39          RTS                    ;
                                         ;
;----------------------------------------;------------------------------------
```

Main Loop & Housekeeping

Lastly I want to show how the game will ensure the solenoid queue gets serviced regularly. This happens in a piece of code I have described simply as the "main loop", however since I don't yet have this function traced out, it's not necessarily an accurate description of this function. Regardless, this function does get called early and loops indefinitely. I have only traced this enough, at this time, to identify a function that will perform housekeeping related to solenoids and flashers.

```
;---------------------------------------;---------------------------------------------------------
; Main caller to begin attract mode
;
; This is pretty much the main-loop of the entire game.
;
96E5: 8E 03 11    LDX   #$0311           ;  x=0x0311
96E8: 9F 59       STX   $59              ;
96EA: BF 03 0F    STX   $030F            ;
96ED: 96 56       LDA   $56              ;
96EF: 91 57       CMPA  $57              ;
96F1: 27 FA       BEQ   $96ED            ;
96F3: 97 57       STA   $57              ;
96F5: 96 56       LDA   $56              ;
96F7: 91 58       CMPA  $58              ;
96F9: 27 1A       BEQ   $9715            ;
96FB: 97 58       STA   $58              ;
96FD: 8E 03 11    LDX   #$0311           ;
9700: 9F 59       STX   $59              ;
9702: BD 9D 3F    JSR   $9D3F            ; Housekeeping, including read all switch inputs!
9705: 86 01       LDA   #$01             ;
9707: 97 5D       STA   $5D              ;
9709: 8E 03 06    LDX   #$0306           ;
970C: 9F 59       STX   $59              ;
970E: DE 59       LDU   $59              ;
9710: BD 97 3C    JSR   $973C            ;
9713: 0F 5D       CLR   $5D              ;
9715: FE 03 0F    LDU   $030F            ;
9718: BD 97 3C    JSR   $973C            ;
971B: 20 C8       BRA   $96E5            ;
                                         ;
;---------------------------------------;---------------------------------------------------------
```

Obviously, the function that we're interested in (above) is called at $9702. I have listed as a housekeeping function, which we'll see next:

```
;--------------------------------------;------------------------------------------
; Housekeeping, including read all switch inputs!
;
9D3F: B6 03 1E    LDA   $031E
9D42: 27 1C       BEQ   $9D60
9D44: F6 3D 61    LDB   $3D61
9D47: C5 02       BITB  #$02
9D49: 27 15       BEQ   $9D60
9D4B: BD 89 48    JSR   $8948           ; CallPagedFunctionPreserve_U_A_B_CC()
9D4E: 63 87 39                          ; -->
9D51: 24 0D       BCC   $9D60
9D53: B6 3D 60    LDA   $3D60
9D56: 84 7F       ANDA  #$7F
9D58: 81 20       CMPA  #$20
9D5A: 26 04       BNE   $9D60
9D5C: BD 82 E4    JSR   $82E4           ; ThrowGenError(0x01)
9D5F: 01          Illegal Opcode
9D60: A6 9F 81 72 LDA   [$8172]
9D64: 27 3B       BEQ   $9DA1
9D66: B6 17 A4    LDA   $17A4
9D69: 26 36       BNE   $9DA1
9D6B: BD 83 7C    JSR   $837C
9D6E: 07 25       ASR   $25
9D70: 30 BD 83 7C LEAX  [$837C,Y]
9D74: 06 25       ROR   $25
9D76: 2A A6       BPL   $9D1E
9D78: 9F 81       STX   $81
9D7A: 96 BD       LDA   $BD
9D7C: C1 C7       CMPB  #$C7
9D7E: BD 83 7C    JSR   $837C
9D81: 08 24       ASL   $24
9D83: 0A BD       DEC   $BD
9D85: 83 7C 05    SUBD  #$7C05
9D88: 24 04       BCC   $9D8E
9D8A: 0F 6B       CLR   $6B
9D8C: 20 F0       BRA   $9D7E
9D8E: A6 9F 81 8D LDA   [$818D]         ;
9D92: BD C1 C7    JSR   $C1C7           ; PlaySoundIndexRegisterA()
9D95: BD 83 7C    JSR   $837C
9D98: 08 24       ASL   $24
9D9A: FA BD 83    ORB   $BD83
9D9D: 7C 05 24    INC   $0524
9DA0: F4 B6 1D    ANDB  $B61D
9DA3: 27 7C       BEQ   $9E21
9DA5: 1D          SEX
```

```
9DA6: 27 B1       BEQ   $9D59
9DA8: 1D          SEX
9DA9: 27 27       BEQ   $9DD2
9DAB: 07 BD       ASR   $BD
9DAD: 82 E4       SBCA  #$E4
9DAF: 5C          INCB
9DB0: BD 91 6B    JSR   $916B
9DB3: 0F 6B       CLR   $6B
9DB5: DC 55       LDD   $55              ; Load $55:$56 (CurrentTimerValue) 2-byte timer value (increments each timer interrupt)
9DB7: DD 69       STD   $69
9DB9: 10 93 67    CMPD  $67
9DBC: 2B 17       BMI   $9DD5
9DBE: C3 02 58    ADDD  #$0258
9DC1: DD 67       STD   $67
9DC3: BD AB 3E    JSR   $AB3E            ; FlasherNextEnergizeTimeTableCleanup() // clean up stale next-flasher-on-time-allowed values
9DC6: BD 89 48    JSR   $8948            ; CallPagedFunctionPreserve_U_A_B_CC()
9DC9: 41 07 39                           ; -->
9DCC: BD 89 48    JSR   $8948            ; CallPagedFunctionPreserve_U_A_B_CC()
9DCF: 4D 29 38                           ; -->
9DD2: BD A7 A1    JSR   $A7A1
9DD5: BD A9 65    JSR   $A965            ; ProcessNextSolenoidTableEntry() // fire off the next solenoid, if appropriate to do so
9DD8: BD 8F E0    JSR   $8FE0            ; Test02D8AndWriteStringUsingFont_7SSx5_FullTableB()
9DDB: BD A0 7B    JSR   $A07B
9DDE: BD 93 73    JSR   $9373            ; Read switch matrix and coin switches (read all switches!)
9DE1: BD DA 5D    JSR   $DA5D
9DE4: 7E 9D E7    JMP   $9DE7
9DE7: BD 9C DA    JSR   $9CDA
9DEA: BD BE AD    JSR   $BEAD
9DED: BD D1 A2    JSR   $D1A2
9DF0: AD 9F 81 1E JSR   [$811E]
9DF4: 39          RTS
;---------------------------------------;------------------------------------------
```

This housekeeping function isn't close to being fully annotated (and some of the opcode alignment needs fixed up), but the important parts are:

- $9DC3, FlasherNextEnergizeTimeTableCleanup(), remove stale entries from FlasherNextEnergizeTimeTable[]
- $9DD5, ProcessNextSolenoidTableEntry(), check for next solenoid to pulse from the circular solenoid queue.

<u>Using the information in this document</u>

You need to remember to be very careful when modifying code that is shown in this document. This is especially true for code related to pulsing the flashers. As you know, the flasher bulbs can fail rather quickly if they are left on for too long.

If you can identify these functions in your game, then you can determine the parts of the code related to a particular feature. This can aid as a sanity check when you're tracing through the code. If you have a certain area of the code all mapped out, including the function calls described here, then you may even be able to change the code to your liking, for example changing the Fishtales topper knocker behavior, or changing the pulse time to see if the game behaves better, such as extending the pulse time of a catapult launcher to see if it makes the game behave better for you. This is all intended for fun and experimental projects. I understand the original s/w was very well written and doesn't necessarily require any such modification.

If you have spare transistors on your game you can use the information in this document to add functionality to the unused transistors (for example, to control a playfield add-on). Note, to add a new entry to a table, you will need to find an unused region in the ROM (usually needs to be in the same ROM bank) where you would need to copy the entire table with your new entries at the bottom. After moving the entire table, you would need to find the pointer to the original table and redirect it to your new table. If you need help with this, just send me an email and I can give some pointers (pun not intended).

If you *really* want to experiment, you can even tie into the 4 unused outputs on WPC power driver board at the 74LS374 at U2. Then you can either modify its h/w register directly, or neatly update the TransistorTable[], and SolenoidTable[] or FlasherTable[]. If you do that, you will need to connect an external transistor circuit to the 74LS374 at U2, as shown in the CV Knocker project which added external transistor to the WPC-95 board (on such board, they simply routed these 4 outputs of the 74LS374 at U2 down to a little connector on the bottom edge of the board (with buffer circuitry in series).

-garrett lee, mrglee@yahoo.com