WPC Match Value

Document History

May 29, 2012, Initial write-up showing how IJ_L7 code derives the match value used at end of game match sequence.

Document Scope

This document is just a quick write-up showing a couple functions. This shows logic used by the game code to determine whether to have a match, and how such match value is determined. This shows code from IJ_L7 and it is likely similar to other WPC games. As always, if you need help finding code in your game feel free to shoot me an email or make a post in the pinhacks.com forum. I'll be happy to help.

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.

If you modify the characteristics of the match value determination, and you run such modified code in a public setting, it is important that you understand the ramifications of giving away too many free games or artificially reducing the free game awards. It is the opinion of this author that it is a bad idea to operate a hacked or modified ROM in a public setting.

<to be named function>()

This discussion will start at the following function which I've yet to fully document.

```
;-------------------------------------;-------------------------------------
48A5: 34 02       PSHS  A
48A7: 1C FE       ANDCC #$00FE
48A9: BD 86 74    JSR   $8674
48AC: 80 5B       SUBA  #$5B
48AE: 25 1A       BCS   $48CA
48B0: BD AC 18    JSR   $AC18
48B3: 7F 04 6C    CLR   $046C
48B6: BD 83 52    JSR   $8352           ; LoadAWithTableIndexByteParameter()
48B9: 93                                ; Match Feature %
48BA: 81 00       CMPA  #$00            ;
48BC: 27 0C       BEQ   $48CA           ; Match feature disabled? Skip to the end
48BE: 8D 0C       BSR   $48CC           ; GenerateRandomMatchValue()
48C0: BD 85 A6    JSR   $85A6           ; RunFunctionIndexByteParameterFromTable74052F()
48C3: 0A                                ; ShowMatchSequenceAnimation()
48C4: BD 49 D1    JSR   $49D1           ; WaitForMatchSequenceToFinish()
48C7: BD 49 3F    JSR   $493F           ; AwardMatchSequenceResult()
48CA: 35 82       PULS  A,PC            ; (PUL? PC=RTS)
;-------------------------------------;-------------------------------------
```

The above function is part of code that gets called at the end of game prior to the match sequence. You can see that I've somewhat figured out that the second half of the function fetches the user-adjustment for "Match Feature %" and , if configured to anything other than "Off" will proceed with the match value determination and then play the match sequence animation.

This document, at this time, focuses on the "GenerateRandomMatchValue()" portion of the above commands.

GenerateRandomMatchValue()

This function does the main work for the match value determination. It determines if a match should be awarded, does some work to generate a match value that either matches or not matches players scores, and puts the match value into RAM so that the subsequent calls to display the match animation will have the desired effect (either not match, or a match along with credit accumulation and knocker pulses, etc).

This function was annotated with a mix of C-like pseudo-code and some line-by-line conversion into human-readable code.

```
;--------------------------------------;-------------------------------------
;
; GenerateRandomMatchValue()
;
; A has match percentage adjustment
;
; Function checks various conditions, decides if we must have a match or non-match
; then finds appropriate BCD value to suit (either a match or non-match) and puts
; it into ram at $046B for use during the imminent match sequence.
;
48CC: 34 06       PSHS  B,A             ;
48CE: 1F 89       TFR   A,B             ; B =  match percentage, adjustment value
                                        ;
48D0: BD 89 48    JSR   $8948           ; CallBankedFunction_Param_WPCAddr()
48D3: 4E 4A 3D                          ; -->CreditsAwardedThisGameDetermine(), C-bit clear if 1 or more credits awarded this game.
                                        ;
48D6: 24 0C       BCC   $48E4           ; if (CreditsAwardedThisGame == 0)
                                        ; {
                                        ;    // The following function will clear C-bit if game thinks a match
                                        ;    // should be awarded due to long attract-mode time (prior to start of
                                        ;    // this game) and the matches awarded thus far have been less than or
                                        ;    // equal to the configured match award percentage.
48D8: BD 49 A5    JSR   $49A5           ;    if (CheckMatchPrecedenceGameIdleAndLowMatchPercentageHistory())
48DB: 24 20       BCC   $48FD           ;    {
                                        ;       goto MATCH_POSITIVE
                                        ;    }
48DD: BD B2 3A    JSR   $B23A           ; }
48E0: 81 01       CMPA  #$01            ; else if (GetPreviouslyPlayedNumberOfPlayers() == 1) // players this game == 1
48E2: 26 03       BNE   $48E7           ; {
48E4: 54          LSRB                  ;    // For multiplayer games the MatchPercentageAdjustmentValue is artifically lowered here
48E5: C9 00       ADCB  #$00            ;    MatchPercentageAdjustmentValue = (MatchPercentageAdjustmentValue+1)/2
                                        ; }
48E7: BD B2 3A    JSR   $B23A           ; GetPreviouslyPlayedNumberOfPlayers() // 1 - 4 players, returned in A register
```

```
48EA: 3D          MUL                 ; D = (PreviouslyPlayedNumberOfPlayers * MatchPercentageAdjustmentValue)
                                      ; //
                                      ; // Here, D has the number of players this game, multiplied by the match
                                      ; // percentage adjustment value.  For multi-player games, the match percentage
                                      ; // adjustment value was artifically lowered prior to this.  Some example values
                                      ; //
                                      ; // Match % Adjustment,   Players,   D-Register
                                      ; // ------------------------------------------
                                      ; //        7                1         0x0007
                                      ; //        8                1         0x0008
                                      ; //        8                2         0x0008
                                      ; //        8                3         0x000C
                                      ; //        8                4         0x0010
                                      ; //
48EB: 86 0A       LDA   #$0A          ; Highest random number is 10, Match digit 0-9.
48ED: BD A7 D7    JSR   $A7D7         ; Get16BitPseudoRandomValueintoA()
48F0: 48          ASLA                ;
48F1: 48          ASLA                ;
48F2: 48          ASLA                ; Shift random number to the left nibble, A now has BCD match value.
48F3: 48          ASLA                ;
48F4: 1E 98       EXG   B,A           ; B has random match value BCD, A has MUL result from above.
                                      ;
                                      ; //
                                      ; // The following function will check the A register (MUL result from above)
                                      ; // against a random number between 0 and 100.  It will return C-bit set if
                                      ; // the random number is greater or equal to the MUL result (from above).
                                      ; // It will return C-bit clear if the random number is less than the MUL result.
                                      ;
48F6: BD A7 8F    JSR   $A78F         ; CheckRandomNumberPercentage()
                                      ;
48F9: 1F 98       TFR   B,A           ; A has random match value BCD, B has MUL result from above.
                                      ;
48FB: 25 08       BCS   $4905         ; if (C-bit set)
                                      ; {
                                      ;    goto MATCH_NEGATIVE
                                      ; }
                                      ;
                                      ; If C-bit clear then the above logic gets here.
                                      ; If C-bit set then the above logic will skip over to MATCH_NEGATIVE.
                                      ;
                                      ; MATCH_POSITIVE:
                                      ; while (1)
                                      ; {
48FD: 8D 19       BSR   $4918         ;    GetMatchHitCount() // See if BCD value in A matches any players scores
48FF: 24 0C       BCC   $490D         ;    if (A-register is a match to one-or-more players scores)
```

```
4901: 8D 0F       BSR   $4912           ;    {
                                        ;       goto MATCH_DONE
4903: 20 F8       BRA   $48FD           ;    }
                                        ; } AdvanceMatchBcdValueA()
                                        ;
                                        ;
                                        ; MATCH_NEGATIVE:
                                        ; while (1)
                                        ; {
4905: 8D 11       BSR   $4918           ;    GetMatchHitCount() // See if BCD value in a does not match any players scores
4907: 25 04       BCS   $490D           ;    if (A-register is a non-match to any player's scores)
4909: 8D 07       BSR   $4912           ;    {
                                        ;       goto MATCH_DONE
490B: 20 F8       BRA   $4905           ;    }
                                        ; } AdvanceMatchBcdValueA()
                                        ;
                                        ; MATCH_DONE:
490D: B7 04 6B    STA   $046B           ; Put match value into $046B
4910: 35 86       PULS  A,B,PC          ;
                                        ;
;---------------------------------------;-----------------------------------
```

The above function is mostly self-documenting. You can see how it first decides whether a match should be awarded based on various conditions such as:

- Whether any player won a replay during the previous game
- Result of CheckMatchPrecedenceGameIdleAndLowMatchPercentageHistory(), function described later
- Number of players during this game
- User adjustment value for Match Award %

Once the above function decides whether match should be awarded, it starts a random BCD value in A and advances it up to 10 times until it finds either a match or a non-match, depending on whether the previous logic decided to award a match.

AdvanceMatchBcdValueA()

This is just a little helper function called from the previous function. For completeness it's listed here as a separate function in this document.

```
;-------------------------------------;-------------------------------------
;
; AdvanceMatchBcdValueA()
;
4912: 8B 10       ADDA  #$10            ; A += 0x10 // Advance to next match value
4914: 19          DAA                   ;
4915: 84 F0       ANDA  #$F0            ; A &= 0xF0 // ensure only high nibble has anything
4917: 39          RTS                   ;
;-------------------------------------;-------------------------------------
```

The function above simply advances the BCD match value in A. It's used while cycling the match value up to 10 times as the previous function tries to find a value that represents a match or represents a non-match.

GetMatchHitCount()

This is another helper function used by GenerateRandomMatchValue().

It checks if the current match value in the A register is an actual match against any of the player's scores from the previously played game.

```
;-------------------------------------;-------------------------------------
;
; GetMatchHitCount()
;
; Called with A holding actual match value
; Checks if match value matches any of the players' scores.
; C-bit set if no match.
; C-bit clear if match!
;
; Returns B with number of matches
;
4918: 34 06       PSHS  B,A             ;
491A: 6F 61       CLR   $0001,S         ; Clear B on the stack
491C: 1F 89       TFR   A,B             ; B gets A, actual match value
491E: BD B2 3A    JSR   $B23A           ; GetPreviouslyPlayedNumberOfPlayers()
                                        ;
4921: 8D 0D       BSR   $4930           ;--\
4923: 25 02       BCS   $4927           ;  | C-bit set, then no match
4925: 6C 61       INC   $0001,S         ;  | Increment B
4927: 4A          DECA                  ;  | Decrement # of players
4928: 26 F7       BNE   $4921           ;  | Keep looping for each player
                                        ;--/
492A: E6 61       LDB   $0001,S         ;
492C: C1 01       CMPB  #$01            ;
492E: 35 86       PULS  A,B,PC          ; (PUL? PC=RTS)
;-------------------------------------;-------------------------------------
```

As indicated, it returns with C-bit indicating whether the BCD value in A is an actual match against any of the player's scores, and also the number of matches is returned in the B register.

CheckIfPlayerAGotAMatch()

This is a helper function called by GetMatchHitCount(), above.

```
;-------------------------------------;-------------------------------------
                                      ;
;
; CheckIfPlayerAGotAMatch()
;
; A has player number to check (loop starts at highest and works down to 1)
; B has actual match value
;
4930: 34 44       PSHS  U,B           ;
4932: BD BB E7    JSR   $BBE7         ; GetPlayerScoreIndexAintoU()
4935: E6 44       LDB   $0004,U       ; B gets 5th byte of score
4937: C4 F0       ANDB  #$F0          ; Get high nibble of last score byte
4939: E0 E4       SUBB  ,S            ; Subtract match value from score
493B: CB FF       ADDB  #$FF          ;
493D: 35 C4       PULS  B,U,PC        ; (PUL? PC=RTS)
                                      ;
;-------------------------------------;-------------------------------------
```

This function simply checks if a single player's score matches the match value. It returns C-bit clear if it's a match. C-bit set when no match.

AwardMatchSequenceResult()

This function is located immediately after the previously shown function in ROM so I figured I'd show it here in this document. It isn't so much about determining the match value, rather it is called after the match number is shown on the DMD during the match sequence animation. This is shown here to give you an idea how the code works in general but not so much about match number determination.

```
;-------------------------------------;-----------------------------------
;
; AwardMatchSequenceResult()
;
; called just after match sequence animation has been shown
;
493F: 34 16       PSHS  X,B,A           ;
4941: 1A 01       ORCC  #$0001          ;
4943: B6 04 6C    LDA   $046C           ; A gets byte from $046C
4946: 26 38       BNE   $4980           ; if $0f6C is non-zero skip to the end
4948: 7C 04 6C    INC   $046C           ; $046C++
494B: B6 04 6B    LDA   $046B           ; A gets match value from $046B
494E: 8D C8       BSR   $4918           ;
4950: 25 2E       BCS   $4980           ; C-bit set? no match, skip to the end
4952: 1F 98       TFR   B,A             ; A gets B
4954: BD 86 AE    JSR   $86AE           ; LookupGameAdjustmentParameter1andCheckIfEqualsParameter2()  C-bit set when not-equal
4957: 90 00                             ; 0x90, Match Award, credit or ticket
4959: 24 12       BCC   $496D           ; If C-bit clear, adjustment matches 0x00 (Credit), skip down
                                        ;
                                        ; --------------------
                                        ; Match Award == Ticket
                                        ; --------------------
495B: F6 04 6B    LDB   $046B           ; B gets match value from $046B
495E: BD B2 3A    JSR   $B23A           ;
4961: BD 49 30    JSR   $4930           ;
4964: 25 02       BCS   $4968           ;
4966: 8D 1A       BSR   $4982           ;
4968: 4A          DECA                  ;
4969: 26 F6       BNE   $4961           ;
496B: 20 11       BRA   $497E           ;
                                        ;
                                        ; --------------------
                                        ; Match Award == Credit
                                        ; --------------------
496D: C6 03       LDB   #$03            ;
496F: BD 89 48    JSR   $8948           ; CallBankedFunction_Param_WPCAddr()
```

```
4972: 4E 50 3D                          ;
4975: 8E 80 18    LDX   #$8018          ;
4978: BD 89 48    JSR   $8948           ; CallBankedFunction_Param_WPCAddr()
497B: 52 89 39                          ; AddValueAToLinkedListTableIndexX()
497E: 1C FE       ANDCC #$00FE          ;
4980: 35 96       PULS  A,B,X,PC        ; (PUL? PC=RTS)
                                        ;
;---------------------------------------;-------------------------------------
```

You can see I don't have the above function fully commented however what I have so far was enough to let me figure out how the solenoid code works. This served as a gateway for learning various other things about the code.

CheckMatchPrecedenceGameIdleAndLowMatchPercentageHistory()

This is the condition-check function used by GenerateRandomMatchValue() which can declare a match to occur based on various conditions outside of simply checking the Match Award % value.

When this function returns with C-bit clear, it signals GenerateRandomMatchValue() to ensure that a match *does* occur. When this function returns with C-bit set it signals GenerateRandomMatchValue() to continue its normal match value determination.

```
;--------------------------------------;--------------------------------------
;
; CheckMatchPrecedenceGameIdleAndLowMatchPercentageHistory()
;
; Checks AttractModeIdleMinutes which is indicates if game attract mode was
; active for awhile prior to the start of this game.
;
; Returns C-bit set if AttractModeIdleMinutes was 0x00, else
; Returns C-bit set if (MatchFeatureAdjustment is <= CurrentMatchAwardedPercentage)
; Returns C-bit set if (MatchFeatureAdjustment is > CurrentMatchAwardedPercentage) and a random number is >= 80
;
; Returns C-bit clear when AttractModeIdleMinutes != 0x00 <and>
;                     when MatchFeatureAdjustment is > CurrentMatchAwardedPercentage <and>
;                     when a random number is < 80
;
49A5: 34 66       PSHS  U,Y,B,A          ; if (AttractModeIdleMinutes == 0x00)
49A7: B6 17 A8    LDA   $17A8            ; {
                                         ;    C-bit set
49AA: 27 21       BEQ   $49CD            ;    return
                                         ; }
                                         ;
                                         ; // We're here if this game had been started after
                                         ; // the attract mode had been running for 9 or more
                                         ; // minutes (assuming it is units of minutes).
                                         ;
49AC: 10 8E 80 18 LDY   #$8018           ; Y = 0x8018 // Table Index: BookkeepingTable, Standard Audits, Match Awards
49B0: CE 80 14    LDU   #$8014           ; U = 0x8014 // Table Index: BookkeepingTable, Standard Audits, Total Plays
49B3: 86 64       LDA   #$64             ; A = 0x64 // Total Percentage
49B5: C6 01       LDB   #$01             ; B = 0x01 // Percentage Step
                                         ;
49B7: BD 89 48    JSR   $8948            ; CallBankedFunction_Param_WPCAddr()
49BA: 51 3F 3A                           ; GetBookkeepingPercentageValueAsciiString8WideRightJustified()
                                         ;
```

```
                                    ; // The previous function fetched the current percentage value of
                                    ; // Match Awards / Total Plays, put percentage result as an ASCII string
                                    ; // at $0326 and hex byte value at $0356
                                    ;
49BD: BD 83 52    JSR   $8352       ; LoadAWithTableIndexByteParameter() // Gets match % adjustment into A
49C0: 93                            ; -->Adjustments, Standard Adjustments, Match Feature
                                    ;
                                    ;
49C1: B1 03 56    CMPA  $0356       ; if (MatchFeatureAdustment > CurrentMatchAwardedPercentage)
49C4: 23 07       BLS   $49CD       ; {
49C6: 86 50       LDA   #$50        ;    A = 0x50 (80 decimal)
49C8: BD A7 8F    JSR   $A78F       ;    CheckRandomNumberPercentage()
49CB: 20 02       BRA   $49CF       ;    // Returns C-bit set if a random number is >= 0x50 (80 decimal)
                                    ;    // Returns C-bit clear if a random number is < 0x50 (80 decimal)
                                    ; }
49CD: 1A 01       ORCC  #$0001      ;
49CF: 35 E6       PULS  A,B,Y,U,PC  ; return C-bit set
                                    ;
;-------------------------------------;-----------------------------------
```

You can see how the function first checks AttractModeIdleMinutes for non-zero value before doing anything else. This value is non-zero if the current game was started after a long time of attract mode activity. The comments are not accurate by saying 9 minutes, actually there is a periodic timer that increments during attract mode. When this timer reaches exactly value 0x09, then AttractModeIdleMinutes is set to a non-zero value. In my examination in PinMame, it seems this is a free-running timer which will hit value 0x09 only after this free-running timer wraps all the way around past 0xFF, so it may be longer than 9 minutes before AttractModeIdleMinutes gets set to non-zero value.

You can then see how the game Bookkeeping values are used to determine the actual match percentage awarded, and when the real match award percentage is less-than or equal to the user configured Match Award % value, then the match percentage is actually bumped to an 80% chance of award! One thing this reveals is that you might be able to increase the odds of getting a match by starting up a game after attract mode has ran for enough time to ensure that AttractModeIdleMinutes has been made non-zero. This will take some more experimentation to say for sure, and to determine exactly how much time must elapse before AttractModeIdleMinutes becomes non-zero on a real pin.

Using the information in this document

This document was produced for educational purposes. You can see how IJ_L7 (and likely other WPC games) determine the match value and perhaps get some ideas on best way to increase your odds of getting a match depending on game idle time, number of players, and whether you've been awarded any free games during the course of the previous game.

If you like to tinker with code, you might find it enjoyable to modify the match code to your liking. Such activities are encouraged only for educational and entertainment benefit of yourself and your friends. I wouldn't recommend operating a hacked rom in a public setting.

-garrett lee, mrglee@yahoo.com