

DIY Calculator: File Formats

Introduction

The purpose of this paper is to document the formats of the various files that are generated and used by the DIY Calculator.

An Example Program

As a basis for these discussions, we shall use the linear-feedback shift-register (LFSR) pseudo-random number generator program example that is provided on the DIY Calculator's website (also provided on the website is a full description as to how this program provides its magic). The source code for this program is as follows:

```
##### Generate pseudo-random numbers between 0 and 7

##### Declare constants
MAINDISP: .EQU    $F031    # Output port for main display
KEYPAD:   .EQU    $F011    # Input port for keypad

CLRCODE:  .EQU    $10      # Code to clear the main display
QCODE:   .EQU    $3F      # ASCII code for "?" (Query)
CCODE:   .EQU    $3A      # ASCII code for ":" (Colon)
ENTERKEY: .EQU    $13      # Code associated with "Enter" key

xx000xxx: .EQU    %00000000 # Mask value (XOR with 0s)
xx111xxx: .EQU    %11111111 # Mask value (XOR with 1s)

                .ORG    $4000    # Set program origin

##### Clear main display and show a "?"
INIT:          LDA     CLRCODE    # Load accumulator with clear code
                STA     [MAINDISP] # Copy to main display
                LDA     QCODE     # Load accumulator with "?"
                STA     [MAINDISP] # Copy to main display

##### Get a seed value, store it, and display it
GETSEED:      LDA     [KEYPAD]    # Load accumulator from the keypad
                CMPA    $0F       # Compare accumulator to code $0F
                JC      [GETSEED] # Jump back if C flag is set

DISPSEED:     STA     [LFSR]     # Otherwise copy to LFSR
                LDA     CLRCODE    # Load accumulator with clear code
                STA     [MAINDISP] # Copy to main display
                LDA     [LFSR]     # Load accumulator with seed value
                STA     [MAINDISP] # Copy to main display
                LDA     CCODE     # Load accumulator with ":"
                STA     [MAINDISP] # Copy to main display

##### Generate a random number and display it
##### Wait for "Enter" key
```

```

WAITENT:  LDA    [KEYPAD]    # Load accumulator from the keypad
          CMPA   ENTERKEY    # Compare to code for "Enter" key
          JNZ    [WAITENT]   # Jump if not the right code

##### Generate the new random value and display it
GENRAND:  LDA    [LFSR]     # Load accumulator with LFSR
          SHR                    # Shift right 1 bit

TESTOOR1: JC     [ITS1]     # Jump to XOR with 1

ITS0:     AND    %01111111   # Clear MS bit to 0
          STA    [LFSR]     # Store value in LFSR
          XOR    xx000xxx    # XOR bits [5,4,3] with 0
          AND    %00111000   # Mask out bits [7,6,2,1,0]
          STA    [TEMP]     # Store it
          JMP    [MERGE]    # Jump to "Merge" bits together

ITS1:     OR     %10000000   # Set MS bit to 1
          STA    [LFSR]     # Store value in LFSR
          XOR    xx111xxx    # XOR bits [5,4,3] with 1
          AND    %00111000   # Mask out bits [7,6,2,1,0]
          STA    [TEMP]     # Store it

MERGE:    LDA    [LFSR]     # Load accumulator with LFSR
          AND    %11000111   # Mask out bits [5,4,3]
          OR     [TEMP]     # Bring in the XORed bits
          STA    [LFSR]     # Store in LFSR
          AND    %00000111   # Mask out LS 3 bits
          STA    [MAINDISP]  # Copy to main display
          JMP    [WAITENT]   # Jump back and do it again

LFSR:     .BYTE                                # The LFSR itself
TEMP:     .BYTE                                # A temp storage location

          .END                                # This is the end of the program

```

The *.lst (List) File

When the assembler is run, one of the files it generates is called the "List" file, which is used to debug programs. If our original assembly source code program were called *lfsr.asm*, then the corresponding list file would be called *lfsr.lst*. The list file associated with our example program would be as follows:

```
# FILE TYPE: DIY Calculator List (*.lst) file
# GENERATED: DIY Calculator Assembler V2.0
# DATE-TIME: Jul 24 15:29:35 2005
# SOURCEWAS: C:\DIY Calculator\Work\lfsr.asm

INSTBYTES: 98
DATABYTES: 2

LINE  ADDR  DATA      LABEL  OPCODE OPERAND
-----
00001 ##### Generate pseudo-random numbers between 0 and 7
00002
00003 ##### Declare constants
00004                MAINDISP: .EQU  $F031
00005                KEYPAD:   .EQU  $F011
00006
00007                CLRCODE:  .EQU  $10
00008                QCODE:   .EQU  $3F
00009                CCODE:   .EQU  $3A
00010                ENTERKEY: .EQU  $13
00011
00012                XX000XXX: .EQU  %00000000
00013                XX111XXX: .EQU  %11111111
00014
00015                .ORG      $4000
00016
00017 ##### Clear main display and show a "?"
00018 4000 90 10      INIT:   LDA    CLRCODE
00019 4002 99 F0 31          STA    [MAINDISP]
00020 4005 90 3F          LDA    QCODE
00021 4007 99 F0 31          STA    [MAINDISP]
00022
00023 ##### Get a seed value, store it, and display it
00024 400A 91 F0 11  GETSEED: LDA    [KEYPAD]
00025 400D 60 0F          CMPA   $0F
00026 400F E1 40 0A          JC     [GETSEED]
00027
00028 4012 99 40 62  DISPSEED: STA    [LFSR]
00029 4015 90 10          LDA    CLRCODE
00030 4017 99 F0 31          STA    [MAINDISP]
00031 401A 91 40 62          LDA    [LFSR]
00032 401D 99 F0 31          STA    [MAINDISP]
00033 4020 90 3A          LDA    CCODE
00034 4022 99 F0 31          STA    [MAINDISP]
00035
```

```

00036 ##### Generate a random number and display it
00037 ##### Wait for "Enter" key
00038 4025 91 F0 11 WAITENT: LDA [KEYPAD]
00039 4028 60 13 CMPA ENTERKEY
00040 402A D6 40 25 JNZ [WAITENT]
00041
00042 ##### Generate the new random value and display it
00043 402D 91 40 62 GENRAND: LDA [LFSR]
00044 4030 71 SHR
00045
00046 4031 E1 40 43 TESTOOR1: JC [ITS1]
00047
00048 4034 30 7F ITS0: AND %01111111
00049 4036 99 40 62 STA [LFSR]
00050 4039 40 00 XOR XX000XXX
00051 403B 30 38 AND %00111000
00052 403D 99 40 63 STA [TEMP]
00053 4040 C1 40 4F JMP [MERGE]
00054
00055 4043 38 80 ITS1: OR %10000000
00056 4045 99 40 62 STA [LFSR]
00057 4048 40 FF XOR XX111XXX
00058 404A 30 38 AND %00111000
00059 404C 99 40 63 STA [TEMP]
00060
00061 404F 91 40 62 MERGE: LDA [LFSR]
00062 4052 30 C7 AND %11000111
00063 4054 39 40 63 OR [TEMP]
00064 4057 99 40 62 STA [LFSR]
00065 405A 30 07 AND %00000111
00066 405C 99 F0 31 STA [MAINDISP]
00067 405F C1 40 25 JMP [WAITENT]
00068
00069 4062 00 LFSR: .BYTE
00070 4063 00 TEMP: .BYTE
00071
00072 .END

```

CONSTANT LABELS CROSS-REFERENCE

NAME	VALUE	LINE NUMBERS WHERE USED (* INDICATES DECLARATION)
CCODE	0000003A	00009* 00033
CLRCODE	00000010	00007* 00018 00029
ENTERKEY	00000013	00010* 00039
KEYPAD	0000F011	00005* 00024 00038
MAINDISP	0000F031	00004* 00019 00021 00030 00032 00034 00066
QCODE	0000003F	00008* 00020
XX000XXX	00000000	00012* 00050
XX111XXX	000000FF	00013* 00057

ADDRESS LABELS CROSS-REFERENCE

NAME	VALUE	LINE NUMBERS WHERE USED (* INDICATES DECLARATION)
DISPSEED4012	00028*
GENRAND402D	00043*
GETSEED400A	00024* 00026
INIT4000	00018*
ITS04034	00048*
ITS14043	00046 00055*
LFSR4062	00028 00031 00043 00049 00056 00061 00064 00069*
MERGE404F	00053 00061*
TEMP4063	00052 00059 00063 00070*
TESTOOR14031	00046*
WAITENT4025	00038* 00040 00067

Following a block of comments, we see two keywords `INSTBYTES` and `DATABYTES` whose associated values (specified as decimal numbers) detail the number of bytes used for instructions and data values, respectively. Note that, in this context, the term “instructions” includes both opcodes and their associated operands. Also, the term “data” refers to memory locations reserved using `.BYTE`, `.2BYTE`, and `.4BYTE` directives.

With regard to the body of the list file, during the process of reading the source code, the assembler assigns a unique number to each line of code, and these line numbers are displayed in decimal in the left hand column.

The original source code is displayed to the right of the list file. The machine code relating to each line of source code appears between the line number and the original source code. The first value to the right of the line number is the hexadecimal address of the instruction. Following the address we may see one, two, or three hexadecimal values, where the first value is always the opcode of the instruction (except in the case of `.BYTE` directives, along with their `.2BYTE` and `.4BYTE` cousins). On line 00067, for example, the `JMP` (“unconditional jump”) instruction is used to jump to label `WAITENT`. The `JMP` instruction occurs at address `$405F`; the opcode for the `JMP` instruction is `$C1`; and the two data bytes following the opcode are `$40` and `$25` forming address `$4025`, which is the address associated with the `WAITENT` label.

Following the body of the program is the *Constant Labels* cross-reference table, which details the constant labels associated with any `.EQU` directives. Reading this cross-reference table from left-to-right shows that we have a constant label called `CCODE` with a value of `$3A`. This entry also shows that `CCODE` was declared on line 00009 and used on line 00033 (* characters are used to indicate the line in which a label is declared). Similarly, we have a constant label called `MAINDISP` with a value of `$F031` that was declared on line 00004 and used on lines 00019, 00021, 00030, 00032, 00034, and 00066.

Finally, the *Address Labels* cross-reference table appears at the bottom of the list file. This table shows that the label `DISPSEED` is associated with address `$4012`; it was declared on line 00028 but is not used (referenced) anywhere else. By comparison, the label `LFSR` is associated with address `$4062`; it was declared on line 00069, and it is used (referenced) on lines 00028, 00031, 00043, 00049, 00056, 00061, and 00064.

The *.ram (RAM) File

When the assembler is run, one of the files it generates is called the “RAM” file. This is the machine code version of the program that will be loaded into the DIY Calculator’s memory by means of a **Memory > Load RAM** command. If our original assembly source code program were called *lfsr.asm*, then the corresponding RAM file would be called *lfsr.ram*. The list file associated with our example program would be as follows:

```
# FILE TYPE: DIY Calculator List (*.ram) file
# GENERATED: DIY Calculator Assembler V2.0
# DATE-TIME: Jul 24 15:29:35 2005
# SOURCEWAS: C:\DIY Calculator\Work\lfsr.asm

STARTADDR: $4000
FINALADDR: $4063

$4000 90 10 99 F0 31 90 3F 99 F0 31 91 F0 11 60 0F E1
$4010 40 0A 99 40 62 90 10 99 F0 31 91 40 62 99 F0 31
$4020 90 3A 99 F0 31 91 F0 11 60 13 D6 40 25 91 40 62
$4030 71 E1 40 43 30 7F 99 40 62 40 00 30 38 99 40 63
$4040 C1 40 4F 38 80 99 40 62 40 FF 30 38 99 40 63 91
$4050 40 62 30 C7 39 40 63 99 40 62 30 07 99 F0 31 C1
$4060 40 25 00 00 -- -- -- -- -- -- -- -- -- -- --
```

Following a block of comments, we see two keywords `STARTADDR` and `FINALADDR` whose associated values (specified as hexadecimal numbers) reflect the start and end addresses of the program, respectively.

With regard to the body of the RAM file, each line commences with a 2-byte hexadecimal address followed by sixteen 1-byte hexadecimal data values. In this context, the data values may contain opcodes, operands, or raw data.

If we look at the “List” file discussed in the previous section, we see that the last entry in the program was the `.BYTE` directive associated with the `TEMP` label at address \$4063. Similarly, if we look at the last line of the RAM file above, we see that the address associated with the first entry in this line is \$4060. This first entry is \$40. The next entry, \$25, would occur at address \$4061; the next, \$00, would occur at address \$4062; and the final entry, \$00 (the value assigned to a `.BYTE` directive by default), would occur at address \$4063. Any remaining (unused) locations on this line are indicated by “--” characters.

The clock-cycles.txt File

As discussed in *The Official DIY Calculator Data Book* provided on the CD-ROM accompanying our book *How Computers Do Math*, each of the DIY Calculator's instructions takes a certain number of clock cycles to execute.

This information, which is captured in the *clock-cycles.txt* file, may be used for profiling – and related – applications (see the *More Tools* page on the DIY Calculator website for additional discussions on profiling and code coverage applications).

You may wish to change the clock cycle values provided in this file to reflect alternative architectural implementations of the DIY Calculator's CPU. The format of the *clock-cycles.txt* file is as follows:

```
# FILE TYPE: DIY Calculator Clock Cycle (*.txt) file
# GENERATED: By Hand
# DATE-TIME: Jul 24 15:29:35 2005
# SOURCEWAS: The Official DIY Calculator Data Book
```

```
ADD    IMM    $10  5  0
ADD    ABS    $11 10  0
ADD    ABSX   $12 10  0
ADDC   IMM    $18  5  0
ADDC   ABS    $19 10  0
ADDC   ABSX   $1A 10  0
AND    IMM    $30  5  0
AND    ABS    $31 10  0
AND    ABSX   $32 10  0
BLDIV  IMM    $F0  7  0
BLDIV  ABS    $F1 11  0
BLDSP  IMM    $50  7  0
BLDSP  ABS    $51 11  0
BLDX   IMM    $A0  7  0
BLDX   ABS    $A1 11  0
BSTSP  ABS    $59 13  0
BSTX   ABS    $A9 13  0
CLRIM  IMP    $09  3  0
CMPA   IMM    $60  5  0
CMPA   ABS    $61 10  0
CMPA   ABSX   $62 10  0
DADD   IMM    $48  5  0
DADD   ABS    $49 10  0
DADD   ABSX   $4A 10  0
DADDC  IMM    $68  5  0
DADDC  ABS    $69 10  0
DADDC  ABSX   $6A 10  0
DECA   IMP    $81  3  0
DECX   IMP    $83  3  0
DSUB   IMM    $88  5  0
DSUB   ABS    $89 10  0
DSUB   ABSX   $8A 10  0
DSUBC  IMM    $B8  5  0
```

DSUBC	ABS	\$B9	10	0
DSUBC	ABSX	\$BA	10	0
HALT	IMP	\$01	3	0
INCA	IMP	\$80	3	0
INCX	IMP	\$82	3	0
JC	ABS	\$E1	7	4
JMP	ABS	\$C1	7	0
JMP	ABSX	\$C2	8	0
JMP	IND	\$C3	12	0
JMP	XIND	\$C4	12	0
JMP	INDX	\$C5	13	0
JN	ABS	\$D9	7	4
JNC	ABS	\$E6	7	4
JNN	ABS	\$DE	7	4
JNO	ABS	\$EE	7	4
JNZ	ABS	\$D6	7	4
JO	ABS	\$E9	7	4
JSR	ABS	\$C9	13	0
JSR	ABSX	\$CA	14	0
JSR	IND	\$CB	18	0
JSR	XIND	\$CC	18	0
JSR	INDX	\$CD	19	0
JZ	ABS	\$D1	7	4
LDA	IMM	\$90	4	0
LDA	ABS	\$91	9	0
LDA	ABSX	\$92	9	0
LDA	IND	\$93	14	0
LDA	XIND	\$94	14	0
LDA	INDX	\$95	14	0
NOP	IMP	\$00	3	0
OR	IMM	\$38	5	0
OR	ABS	\$39	10	0
OR	ABSX	\$3A	10	0
POPA	IMP	\$B0	5	0
POPSR	IMP	\$B1	5	0
PUSHA	IMP	\$B2	6	0
PUSHSR	IMP	\$B3	6	0
ROLC	IMP	\$78	3	0
RIRC	IMP	\$79	3	0
RTI	IMP	\$C7	10	0
RTS	IMP	\$CF	8	0
SETIM	IMP	\$08	3	0
SHL	IMP	\$70	3	0
SHR	IMP	\$71	3	0
STA	ABS	\$99	10	0
STA	ABSX	\$9A	10	0
STA	IND	\$9B	15	0
STA	XIND	\$9C	15	0
STA	INDX	\$9D	15	0
SUB	IMM	\$20	5	0
SUB	ABS	\$21	10	0
SUB	ABSX	\$22	10	0

SUBC	IMM	\$28	5	0
SUBC	ABS	\$29	10	0
SUBC	ABSX	\$2A	10	0
XOR	IMM	\$40	5	0
XOR	ABS	\$41	10	0
XOR	ABSX	\$42	10	0

Following a block of comment lines, we have the main body of the file. Each entry consists of an instruction mnemonic, its addressing mode, and three data values.

With regards to the addressing modes:

IMM	=	Immediate
IMP	=	Implied
ABS	=	Absolute
ABSX	=	Indexed
IND	=	Indirect
XIND	=	Pre-indexed indirect
INDX	=	Indirect post-indexed

With regards to the three data values associated with each mnemonic/addressing-mode combo; for example: \$71 3 0. The first (hex) value is the opcode; the next (decimal) value is the number of clocks (in the case of a standard instruction) or the number of clocks for a test that *passes* (in the case of a conditional jump instruction); and the last (decimal) value is the number of clocks for a test that *fails* (in the case of a conditional jump instruction).

The *.rad (Raw Assembly Dump) File

When the assembler is run, one of the files it generates is called the “Raw Assembly Dump” file. This file will be loaded into the simulator and used as the basis to gather profiling information that can subsequently be used by profiling and code coverage applications. If our original assembly source code program were called *lfsr.asm*, then the corresponding RAD file would be called *lfsr.rad*. The RAD file associated with our example program would be as follows:

```
# FILE TYPE: DIY Calculator Raw Assembly Dump (*.rad) file
# GENERATED: DIY Calculator Assembler V2.0
# DATE-TIME: Jul 24 15:29:35 2005
# SOURCEWAS: C:\DIY Calculator\Work\lfsr.asm
```

```
STARTLINE: 00001
FINALLINE: 00072
```

```
00001 B ----- ---
00002 B ----- ---
00003 B ----- ---
00004 P ----- ---
00005 P ----- ---
00006 B ----- ---
00007 P ----- ---
00008 P ----- ---
00009 P ----- ---
00010 P ----- ---
00011 B ----- ---
00012 P ----- ---
00013 P ----- ---
00014 B ----- ---
00015 P ----- ---
00016 B ----- ---
00017 B ----- ---
00018 I $4000 $90
00019 I $4002 $99
00020 I $4005 $90
00021 I $4007 $99
00022 B ----- ---
00023 B ----- ---
00024 I $400A $91
00025 I $400D $60
00026 J $400F $E1
00027 B ----- ---
00028 I $4012 $99
00029 I $4015 $90
00030 I $4017 $99
00031 I $401A $91
00032 I $401D $99
00033 I $4020 $90
00034 I $4022 $99
00035 B ----- ---
00036 B ----- ---
```

```

00037 B ----- ---
00038 I $4025 $91
00039 I $4028 $60
00040 J $402A $D6
00041 B ----- ---
00042 B ----- ---
00043 I $402D $91
00044 I $4030 $71
00045 B ----- ---
00046 J $4031 $E1
00047 B ----- ---
00048 I $4034 $30
00049 I $4036 $99
00050 I $4039 $40
00051 I $403B $30
00052 I $403D $99
00053 I $4040 $C1
00054 B ----- ---
00055 I $4043 $38
00056 I $4045 $99
00057 I $4048 $40
00058 I $404A $30
00059 I $404C $99
00060 B ----- ---
00061 I $404F $91
00062 I $4052 $30
00063 I $4054 $39
00064 I $4057 $99
00065 I $405A $30
00066 I $405C $99
00067 I $405F $C1
00068 B ----- ---
00069 D $4062 $00
00070 D $4063 $00
00071 B ----- ---
00072 P ----- ---

```

Following a block of comments, we see two keywords `STARTLINE` and `FINALLINE` whose associated values (specified as decimal numbers with leading zeros) reflect the first and last line numbers associated with the source code program, respectively.

With regard to the body of the RAM file, each line commences with the line number (specified as decimal numbers with leading zeros) of a source code line. This is followed by a letter, whose meaning is as follows:

- B = Blank line or comment
- P = Pseudo-command (directive) such as `.EQU`, `.ORG`, and `.END`
- I = Standard instruction just as `LDA`, `STA`, `AND`, and `OR` (also unconditional jumps)
- J = Conditional jump instructions such as `JZ` ("jump if zero")
- D = Data value associated with a `.BYTE`, `.2BYTE`, or `.4BYTE` directive.

In the case of instructions (both standard “I” and conditional “J” instructions), these letters are followed by the 2-byte address of the opcode associated with the instruction (in hexadecimal) and the 1-byte opcode itself (in hexadecimal).

In the case of data values declared using `.BYTE`, `.2BYTE`, or `.4BYTE` directives, the letter ‘D’ is followed by the 2-byte address associated with the directive (in hexadecimal) and the data value itself (in hexadecimal).

Note that if there are multiple values associated with a `.BYTE`, `.2BYTE`, or `.4BYTE` directive, then these will be presented on separate lines without line numbers. For example, consider the following source code statements:

```
FRED: .BYTE $23, $24, $25
BERT: .2BYTE $1234
JOHN: .4BYTE
```

If we assume that the label `FRED` occurred on line 100 of a source code program and that it was associated with address `$4090`, then the corresponding RAD file entries would appear as follows (excluding the comments shown to the right):

```
00100 D $4090 $23           {Values associated with FRED}
      $4091 $24
      $4092 $25
00101 D $4093 $12         {Values associated with BERT}
      $4094 $34
00102 D $4095 $00         {Values associated with JOHN}
      $4096 $00
      $4097 $00
      $4098 $00
```

In the case of the `.4BYTE` directive, we didn’t assign any values in the assembly source file, so the assembler will assign these locations default values of `$00`.

The *.pad (Processed Assembly Dump) File

When the DIY Calculator is run, it reads in the Raw Assembly Dump (RAD) file discussed in the previous section. If you use the profiling data gathering facilities in the DIY Calculator, you can output a Processed Assembly Dump (PAD) file.

If our original assembly source code program were called *lfsr.asm*, then – by default – the corresponding PAD file would be called *lfsr.pad*. However, you may wish to perform different profiling runs, in which case you may end up with a suite of PAD files called something like:

```
lfsr-test-1.pad  
lfsr-test-2.pad  
lfsr-test-3.pad  
:  
etc.
```

Assuming we decided to generate a PAD file associated with our example program, but that we hadn't actually performed a simulation, it would appear as follows:

```
# FILE TYPE: DIY Calculator Processed Assembly Dump (*.pad) file  
# GENERATED: DIY Calculator  
# DATE-TIME: Jul 24 16:29:35 2005  
# SOURCEWAS: C:\DIY Calculator\Work\lfsr.asm and lfsr.rad
```

```
STARTLINE: 00001  
FINALLINE: 00072
```

```
00001 B ----- --- -----  
00002 B ----- --- -----  
00003 B ----- --- -----  
00004 P ----- --- -----  
00005 P ----- --- -----  
00006 B ----- --- -----  
00007 P ----- --- -----  
00008 P ----- --- -----  
00009 P ----- --- -----  
00010 P ----- --- -----  
00011 B ----- --- -----  
00012 P ----- --- -----  
00013 P ----- --- -----  
00014 B ----- --- -----  
00015 P ----- --- -----  
00016 B ----- --- -----  
00017 B ----- --- -----  
00018 I $4000 $90 $00000000 -----  
00019 I $4002 $99 $00000000 -----  
00020 I $4005 $90 $00000000 -----  
00021 I $4007 $99 $00000000 -----  
00022 B ----- --- -----  
00023 B ----- --- -----  
00024 I $400A $91 $00000000 -----  
00025 I $400D $60 $00000000 -----
```

00026	J	\$400F	\$E1	\$00000000	\$00000000
00027	B	-----	---	-----	-----
00028	I	\$4012	\$99	\$00000000	-----
00029	I	\$4015	\$90	\$00000000	-----
00030	I	\$4017	\$99	\$00000000	-----
00031	I	\$401A	\$91	\$00000000	-----
00032	I	\$401D	\$99	\$00000000	-----
00033	I	\$4020	\$90	\$00000000	-----
00034	I	\$4022	\$99	\$00000000	-----
00035	B	-----	---	-----	-----
00036	B	-----	---	-----	-----
00037	B	-----	---	-----	-----
00038	I	\$4025	\$91	\$00000000	-----
00039	I	\$4028	\$60	\$00000000	-----
00040	J	\$402A	\$D6	\$00000000	\$00000000
00041	B	-----	---	-----	-----
00042	B	-----	---	-----	-----
00043	I	\$402D	\$91	\$00000000	-----
00044	I	\$4030	\$71	\$00000000	-----
00045	B	-----	---	-----	-----
00046	J	\$4031	\$E1	\$00000000	\$00000000
00047	B	-----	---	-----	-----
00048	I	\$4034	\$30	\$00000000	-----
00049	I	\$4036	\$99	\$00000000	-----
00050	I	\$4039	\$40	\$00000000	-----
00051	I	\$403B	\$30	\$00000000	-----
00052	I	\$403D	\$99	\$00000000	-----
00053	I	\$4040	\$C1	\$00000000	-----
00054	B	-----	---	-----	-----
00055	I	\$4043	\$38	\$00000000	-----
00056	I	\$4045	\$99	\$00000000	-----
00057	I	\$4048	\$40	\$00000000	-----
00058	I	\$404A	\$30	\$00000000	-----
00059	I	\$404C	\$99	\$00000000	-----
00060	B	-----	---	-----	-----
00061	I	\$404F	\$91	\$00000000	-----
00062	I	\$4052	\$30	\$00000000	-----
00063	I	\$4054	\$39	\$00000000	-----
00064	I	\$4057	\$99	\$00000000	-----
00065	I	\$405A	\$30	\$00000000	-----
00066	I	\$405C	\$99	\$00000000	-----
00067	I	\$405F	\$C1	\$00000000	-----
00068	B	-----	---	-----	-----
00069	D	\$4062	\$00	\$00000000	\$00000000
00070	D	\$4063	\$00	\$00000000	\$00000000
00071	B	-----	---	-----	-----
00072	P	-----	---	-----	-----
99999	S	-----	---	\$00000000	\$00000000
99999	X	\$F000	---	\$00000000	-----

99999	X	\$F001	---	\$00000000	-----
99999	X	\$F002	---	\$00000000	-----
99999	X	\$F003	---	\$00000000	-----
99999	X	\$F004	---	\$00000000	-----
99999	X	\$F005	---	\$00000000	-----
99999	X	\$F006	---	\$00000000	-----
99999	X	\$F007	---	\$00000000	-----
99999	X	\$F008	---	\$00000000	-----
99999	X	\$F009	---	\$00000000	-----
99999	X	\$F00A	---	\$00000000	-----
99999	X	\$F00B	---	\$00000000	-----
99999	X	\$F00C	---	\$00000000	-----
99999	X	\$F00D	---	\$00000000	-----
99999	X	\$F00E	---	\$00000000	-----
99999	X	\$F00F	---	\$00000000	-----
99999	X	\$F010	---	\$00000000	-----
99999	X	\$F011	---	\$00000000	-----
99999	X	\$F012	---	\$00000000	-----
99999	X	\$F013	---	\$00000000	-----
99999	X	\$F014	---	\$00000000	-----
99999	X	\$F015	---	\$00000000	-----
99999	X	\$F016	---	\$00000000	-----
99999	X	\$F017	---	\$00000000	-----
99999	X	\$F018	---	\$00000000	-----
99999	X	\$F019	---	\$00000000	-----
99999	X	\$F01A	---	\$00000000	-----
99999	X	\$F01B	---	\$00000000	-----
99999	X	\$F01C	---	\$00000000	-----
99999	X	\$F01D	---	\$00000000	-----
99999	X	\$F01E	---	\$00000000	-----
99999	X	\$F01F	---	\$00000000	-----
99999	X	\$F020	---	-----	\$00000000
99999	X	\$F021	---	-----	\$00000000
99999	X	\$F022	---	-----	\$00000000
99999	X	\$F023	---	-----	\$00000000
99999	X	\$F024	---	-----	\$00000000
99999	X	\$F025	---	-----	\$00000000
99999	X	\$F026	---	-----	\$00000000
99999	X	\$F027	---	-----	\$00000000
99999	X	\$F028	---	-----	\$00000000
99999	X	\$F029	---	-----	\$00000000
99999	X	\$F02A	---	-----	\$00000000
99999	X	\$F02B	---	-----	\$00000000
99999	X	\$F02C	---	-----	\$00000000
99999	X	\$F02D	---	-----	\$00000000
99999	X	\$F02E	---	-----	\$00000000
99999	X	\$F02F	---	-----	\$00000000
99999	X	\$F030	---	-----	\$00000000
99999	X	\$F031	---	-----	\$00000000
99999	X	\$F032	---	-----	\$00000000
99999	X	\$F033	---	-----	\$00000000
99999	X	\$F034	---	-----	\$00000000

```

99999 X $F035 --- ----- $00000000
99999 X $F036 --- ----- $00000000
99999 X $F037 --- ----- $00000000
99999 X $F038 --- ----- $00000000
99999 X $F039 --- ----- $00000000
99999 X $F03A --- ----- $00000000
99999 X $F03B --- ----- $00000000
99999 X $F03C --- ----- $00000000
99999 X $F03D --- ----- $00000000
99999 X $F03E --- ----- $00000000
99999 X $F03F --- ----- $00000000

```

As we see, this is very similar to our original RAD file. The differences are as follows:

- o In the case of blank/comment lines (“B”) and pseudo-commands (“P”), we have two new additions on the end as follows:

```
RAD File: 00071 B ----- ---
```

```
PAD File: 00071 B ----- --- -----
```

These are really only here for aesthetic reasons and serve no functional purpose.

- o In the case of standard instruction lines (“I”) and pseudo-commands (“P”), we have two new additions on the end (shown in bold) as follows:

```
RAD File: 00018 I $4000 $90
```

```
PAD File: 00018 I $4000 $90 $00000000 -----
```

The first (left-hand) new field shows the number of times this opcode is executed (zero in this case because we haven’t actually run the simulator); the second is for aesthetic reasons and serves no functional purpose.

- o In the case of conditional jump instruction lines (“J”), we have two new additions on the end (shown in bold) as follows:

```
RAD File: 00026 J $400F $E1
```

```
PAD File: 00026 J $400F $E1 $00000000 $00000000
```

The first (left-hand) new field shows the number of times this jump instruction opcode is executed and passes (zero in this case because we haven’t actually run the simulator); for example, if this was a JC (“jump if carry”) instruction and the carry flag happens to be 1, this test will pass and the first of the new fields will be incremented.

Similarly, the second (right-hand) new field shows the number of times this jump instruction opcode is executed and fails (zero in this case because we haven’t actually run the simulator); for example, if this was a JC (“jump if carry”) instruction and the carry flag happens to be 0, this test will fail and the second of the new fields will be incremented.

- In the case of data lines (“D”), we have two new additions on the end (shown in bold) as follows:

RAD File: 00069 D \$4062 \$00

PAD File: 00069 D \$4062 \$00 **\$00000000 \$00000000**

The first new field shows the number of times this data byte is read from (zero in this case because we haven’t actually run the simulator). By comparison, the second new field shows the number of times this data byte is written to.

- We’ve added a line associated with the DIY Calculator’s stack. This is indicated by line number 99999, an “S” character, and our two new fields (shown in bold):

PAD File: 99999 S ----- --- **\$00000000 \$00000000**

The first (left-hand) field shows the number of times data is popped off the stack using the `POPA` and/or `POPSR` instructions. By comparison, the second (right-hand) new field shows the number of times data is pushed onto the stack using the `PUSHA` and/or `PUSHSR` instructions.

- We’ve also added lines associated with the DIY Calculator’s 32 input ports and 32 output ports at the end of the file. These are indicated by line numbers of 99999 and “X” characters.

Addresses \$F000 through \$F01F are occupied by the 32 input ports (of which the calculator actually uses only one port at address \$F011), while addresses \$F020 through \$F03F are occupied by the 32 output ports (of which the calculator actually employs only two ports at addresses \$F031 and \$F032); for example:

PAD File: 99999 X \$F000 --- **\$00000000** -----

PAD File: 99999 X \$F020 --- ----- **\$00000000**

In the case of an input port, its corresponding \$00000000 field will show the number of times the CPU has read from this particular port. Similarly, in the case of an output port, its corresponding \$00000000 field will show the number of times the CPU has written to this particular port.