Coleco Vision Cart Info

All carts start at 8000h with a header that tells the BIOS what to do.

```
8000 - 8001:   If bytes are AAh and 55h, the CV will show a title screen
               and game name, etc.

               If bytes are 55h and AAh, the CV will jump directly to the
               start of code vector.

8002 - 8003:   Vector; ???  --\
                               \
8004 - 8005:   Vector; ???      \
                                 >--- These always point to 0000 or RAM
(7xxx)
8006 - 8007:   Vector; ???      /
                               /
8008 - 8009:   Vector; ???  --/

800A - 800B:   Vector; Start of code

800C - 800E:   Jmp to: RST 08h

800F - 8011:   Jmp to: RST 10h

8012 - 8014:   Jmp to: RST 18h

8015 - 8017:   Jmp to: RST 20h

8018 - 801A:   Jmp to: RST 28h

801B - 801D:   Jmp to: RST 30h

801E - 8020:   Jmp to: RST 38h

8021 - 8023:   JMP to: NMI (Vertical Blanking Interrupt from video chip)

8024 - nnnn:   Title screen data:
```

Data for the title screen is composed of 4 lines in the format:

```
+--------------+
| COLECOVISION |
|              |
|    LINE 2    |
|    LINE 3    |
```

```
|(c)xxxx COLECO|
+--------------+
```

 Typical Screen

The 'ColecoVision' line cannot be changed, as well as the '(C)xxxx Coleco'
part of the bottom line. Only the xxxx part can be changed.

The data is stored as one string with the '/' character (2Fh) used as a delimiter. It signals the end of a line, and isn't printed.

The lines are stored out of order like so:

"LINE 3/LINE 2/xxxx" There isn't an end-of-line delimiter, because the last line is always 4 characters (it's meant for a year like 1983)

So, if we want to see the following:

```
+--------------+
| COLECOVISION |
|              |
|   MY GAME!   |
|    BY: ME    |
|(c)1995 COLECO|
+--------------+
```

We would use the string:

"BY: ME!/MY GAME!/1995"

Remember, we cannot change the "(c)xxxx COLECO" part, only the xxxx in the
middle of the line.

The lines are self-centering on the screen.

Altho the BIOS ROM has both upper-case and lower-case characters in the character set, only upper-case is supported.

All printable characters are based on the ASCII character set:

(all values in hex)

00-1C: Blank

```
1D   :     (c)    (Copyright symbol)
1E-1F:     tm     (TradeMark symbol, uses 2 chars side-by-side)
20-2E: (respectively)  space ! " # $ % & ' ( ) * + , - .
2F   :     Delimiter- used to end a line, not printable
30-39:     0-9
3A-40: (respectively)  : ; < = > ? @
41-5A:     A-Z
5B-5F: (respectively)  [ \ ] ^ _
```

The chars # 60-8F are the 4-char blocks that make up the 'COLECOVISION' name at the top, arranged like so:

```
         6062 6466 686A 6C6E 7072 7476 787A 7C7E 8082 8486 888A 8C8E
         6163 6567 696B 6D6F 7173 7577 797B 7D7F 8183 8587 898B 8D8F


          C    O    L    E    C    O    V    I    S    I    O    N

         (purple)  (orange)   (pink)   (yellow)   (green)   (blue)
```

What's intresting, is when these are in the title lines, they show up in their respective colours! All other printable chars are white.

Chars 90-FF are all blank

Controls:

There are 4 ports governing the operation of the controls. They are:

80- When written to, enables the keypads and right buttons on both controls.

C0- When written to, enables the joysticks and left buttons on both controls.

These 2 ports toggle a flip-flop, so the data going out the ports is irrelevant.

FC- Reading this port gives the status of controller #1. (farthest from front)

FF- Reading this one gives the status of controller #2. (closest to front)

All switch closures are represented by a '0' Open switches are '1'

Only 5 bits of ports FC and FF are used:

'80' mode (port 80 written to)

bit #6= status of right button

The keypad returns a 4-bit binary word for a button pressed:

	bit #			
btn:	0	1	2	3
0	0	1	0	1
1	1	0	1	1
2	1	1	1	0
3	0	0	1	1
4	0	1	0	0
5	1	1	0	0
6	0	1	1	1
7	1	0	1	0
8	1	0	0	0
9	1	1	0	1
*	1	0	0	1
#	0	1	1	0

Re-arranged, in order:

btn:	0	1	2	3	hex#	
inv.	0	0	0	0	0	
8	1	0	0	0	1	
4	0	1	0	0	2	
5	1	1	0	0	3	
inv.	0	0	1	0	4	
7	1	0	1	0	5	
#	0	1	1	0	6	
2	1	1	1	0	7	
inv.	0	0	0	1	8	
*	1	0	0	1	9	
0	0	1	0	1	A	
9	1	1	0	1	B	
3	0	0	1	1	C	
1	1	0	1	1	D	
6	0	1	1	1	E	
inv.	1	1	1	1	F	(No buttons down)

The controllers have 28 diodes in them that generate the above table.
Coleco did this so the programmer wouldn't have to scan the keys. Also,

there wouldn't have been enough pins on the end of the cable to accomodate
the array. (There are only 7 pins on the connector. That's just enough to scan a 3*4 matrix, but that doesn't include the stick or other 2 buttons.)

'C0' mode (port C0 written to)

This mode allows you to read the stick and left button:

Bit 6=Left button
Bit 0=Left
Bit 1=Down
Bit 2=Right
Bit 3=Up

CV's memory/IO map

The CV uses 2 74138 3-8 line decoders to generate the memory and IO maps.

As you would expect, memory is broken up into 8 8K blocks. However, the IO
map is broken up into 4 write and 4 read ports.

Memory:

(ABC lines of decoder go to A5, A6, and A7 respectively /E1 -> /M_request
/E2 -> Reset; E3 -> +V)

0000-1FFF = BIOS ROM
2000-3FFF = Expansion port
4000-5FFF = Expansion port
6000-7FFF = SRAM (1K)
8000-9FFF = Cart
A000-BFFF = Cart
C000-DFFF = Cart
E000-FFFF = Cart

IO:

(ABC lines of decoder go to /WR, A5, and A6 respectively /E1 -> /IO_request
/E2 -> Reset; E3 -> A7)

80-9F (W) = Controls _ Set to keypad mode
80-9F (R) = Not Connected

```
A0-BF (W) = Video \___ A0 also decoded by video chip
A0-BF (R) = Video /

C0-DF (W) = Controls _ Set to joystick mode
C0-DF (R) = Not Connected

E0-FF (W) = Sound
E0-FF (R) = Controls _ A1 also decoded by chips  (A1=0 ctrl 1; A1=1 ctrl
2)
```

CV's sound chip

The sound chip in the CV is a SN76489AN manufactured by Texas Instruments.
It has 4 sound channels- 3 tone and 1 noise.

The volume of each channel can be controlled seperately in 16 steps from full volume to silence.

A byte written into the sound chip determines which register is used, along
with the frequency/ attenuation information.

The frequency of each channel is represented by 10 bits. 10 bits won't fit into 1 byte, so the data is written in as 2 bytes.

Here's the control word:

```
+--+--+--+--+--+--+--+--+
|1 |R2|R1|R0|D3|D2|D1|D0|
+--+--+--+--+--+--+--+--+
```

1: This denotes that this is a control word
R2-R0 the register number:

```
000 Tone 1 Frequency
001 Tone 1 Volume
010 Tone 2 Frequency
011 Tone 2 Volume
100 Tone 3 Frequency
101 Tone 3 Volume
110 Noise Control
111 Noise Volume
```

D3-D0 is the data

Here's the second frequency register:

```
+--+--+--+--+--+--+--+--+
|0 |xx|D9|D8|D7|D6|D5|D4|
```

```
+--+--+--+--+--+--+--+--+
```

0: This denotes that we are sending the 2nd part of the frequency

D9-D4 is 6 more bits of frequency

To write a 10-bit word for frequenct into the sound chip you must first send the control word, then the second frequency register. Note that the second frequency register doesn't have a register number. When you write to it, it uses which ever register you used in the control word.

So, if we want to output 11 0011 1010b to tone channel 1:

First, we write the control word:

```
LD A,1000 1010b
OUT (F0h),A
```

Then, the second half of the frequency:

```
LD A,0011 0011b
OUT (F0h),A
```

To tell the frequency of the wave generated, use this formula:

```
   3579545
f= -------
     32n
```

Where f= frequency out,
and n= your 10-bit binary number in

To control the Volume:

```
+--+--+--+--+--+--+--+--+
|1 |R2|R1|R0|V3|V2|V1|V0|
+--+--+--+--+--+--+--+--+
```

R2-R0 tell the register

V3-V0 tell the volume:

0000=Full volume
.
.
.
1111=Silence

The noise source is quite intresting. It has several modes of operation.

Here's a control word:

```
+--+--+--+--+--+--+--+--+
|1 |1 |1 |0 |xx|FB|M1|M0|
+--+--+--+--+--+--+--+--+
```

FB= Feedback:

0= 'Periodic' noise
1= 'white' noise

The white noise sounds, well, like white noise.
The periodic noise is intresting. Depending on the frequency, it can sound very tonal and smooth.

M1-M0= mode bits:

```
00= Fosc/512  Very 'hissy'; like grease frying
01= Fosc/1024 Slightly lower
10= Fosc/2048 More of a high rumble
11= output of tone generator #3
```

You can use the output of gen. #3 for intresting effects. If you sweep the frequency of gen. #3, it'll cause a cool sweeping effect of the noise.
The usual way of using this mode is to attenuate gen. #3, and use the output of the noise source only.

The attenuator for noise works in the same way as it does for the other channels.

Video

The CV uses a TMS9918a chip made by Texas Instruments.