CPS3 Sprites

A guide to CPS3 sprites. CPS3 in this context is for the sprites used in the Jojo's Bizarre Adventure and the Street Fighter III series. It is not known if Red Earth uses the same format.

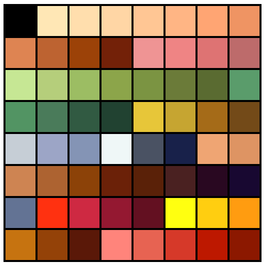
Outline

CPS3 sprites are made up of separate components that form a sprite. We will outline each component from the pallet itself, all the way to how tiles and tile groups forms a sprite.

Pallet

A pallet for CPS3 is nothing out of the ordinary. It is made up of 64 unique colors, with the first index of the pallet used as a background color.

Here is Alex's pallet, which will be used throughout this example



Tile

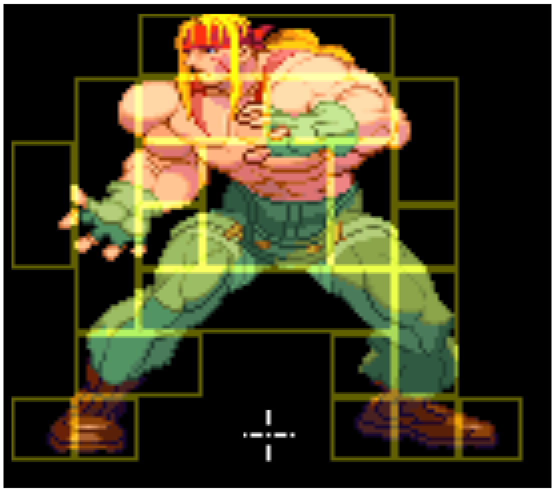
A tile is a 256 byte sequence that forms a 16 by 16 picture involved in a group of tiles. Each byte can be a value between 0 and 63 to refer to a position in the Pallet that is used by the sprite. The tile has no position relative to the sprite, as it is occupied by the group.

The tile is drawn from left to right, top to bottom, relative to its index in the sequence of bytes. This will be referred to as "raw tile data", as it is the de-compressed tile data and refers to indexes in the pallet at face value.

Tile Group

A tile group in this document will refer to ***a group of tiles aligned in a way that form a portion of a sprite***. The position of each sprite group is relative to the game's unit position. The tile group can be 1, 2 or 4 tiles in length and / or height. The group cannot use the amount of 3 as it's width or height.

Below is a sprite showing where it's tile groups are defined to form a sprite. As you can see, there are varying degrees of sizes and are defined for each portion of the sprite:



Sprite Data in the Rom

To know how tiles are decompressed, it is important to know how the sprite information is formed within the game. Sprites are defined in the program rom and are made up of 3 components. We will call these 3 components **Sprite Header, Tile Data Definition**, and **Tile Group Definition**.

A couple of notes before each item is defined:

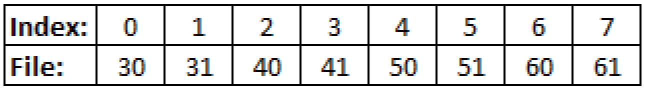
- The Lookup Address or the Tile Data Address must go under a formula to get what is called its “real” address.The formula is:

  Address * 2 - 0x**400000 = Real Address**
  0x14D0000 * 2 - 0x400000 = 0x25A0000

  Address 0 is defined as the User Rom and its length depends on how many memory chips the game uses. Each User Rom chip uses 0x1000000 bytes, split to two roms and is dumped to roms as 30, 31, 40, 41 and so on. If you would like to find the rom file that the tile data is contained in relative to it’s real address, you would first gain the file index from the formula:

  Real Address / 0x800000 = File Index
  0x25A0000 / 0x800000 = 5

  And select the file based on it’s position in the User Rom



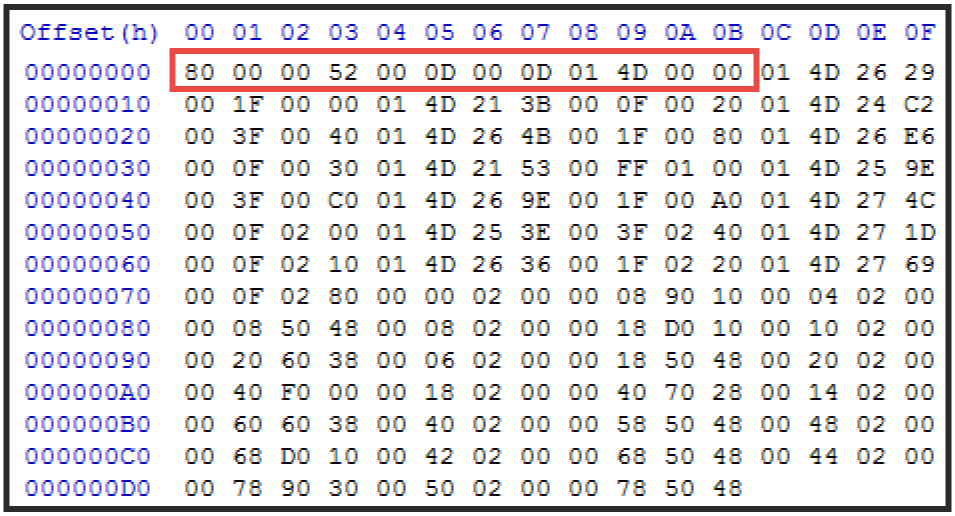
Index:	0	1	2	3	4	5	6	7
File:	30	31	40	41	50	51	60	61

  And use the remainder of the Real Address divided by the file size:

  Real Address - (File Index * 0x800000) = File Offset
  0x25A0000 - (4 * 0x800000) = 0x5A0000

  Address 14D0000 is offset 5A0000 in file 51

Tile Header

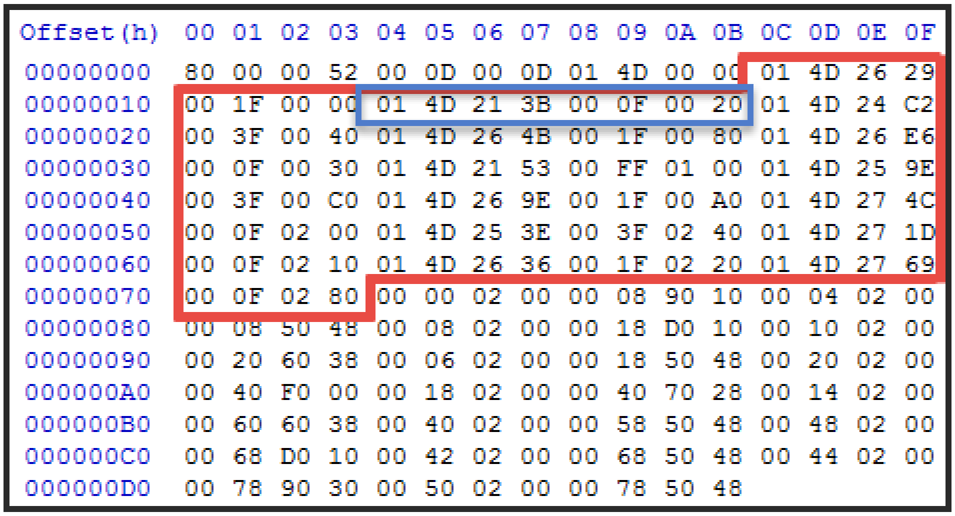


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   80 00 00 52 00 0D 00 0D 01 4D 00 00 01 4D 26 29
00000010   00 1F 00 00 01 4D 21 3B 00 0F 00 20 01 4D 24 C2
00000020   00 3F 00 40 01 4D 26 4B 00 1F 00 80 01 4D 26 E6
00000030   00 0F 00 30 01 4D 21 53 00 FF 01 00 01 4D 25 9E
00000040   00 3F 00 C0 01 4D 26 9E 00 1F 00 A0 01 4D 27 4C
00000050   00 0F 02 00 01 4D 25 3E 00 3F 02 40 01 4D 27 1D
00000060   00 0F 02 10 01 4D 26 36 00 1F 02 20 01 4D 27 69
00000070   00 0F 02 80 00 00 02 00 00 08 90 10 00 04 02 00
00000080   00 08 50 48 00 08 02 00 00 18 D0 10 00 10 02 00
00000090   00 20 60 38 00 06 02 00 00 18 50 48 00 20 02 00
000000A0   00 40 F0 00 00 18 02 00 00 40 70 28 00 14 02 00
000000B0   00 60 60 38 00 40 02 00 00 58 50 48 00 48 02 00
000000C0   00 68 D0 10 00 42 02 00 00 68 50 48 00 44 02 00
000000D0   00 78 90 30 00 50 02 00 00 78 50 48
```

Offset	Size	Description	Data as Shown
00	2 Bytes	Start Flag (No importance)	8000
02	2 Bytes	CRam Size	0052
04	2 Bytes	Number of Tile Data Definitions	000D
06	2 Bytes	Number of Tile Group Definitions	000D
08	4 Bytes	Lookup Block Address	014D0000

At the sprite's address, the tile header is within the first 12 bytes (Street Fighter 3) or 8 bytes (Jojo's Bizarre Adventure). The Jojo series does not define the Lookup Block in the sprite header

Tile Data Definition



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   80 00 00 52 00 0D 00 0D 01 4D 00 00 01 4D 26 29
00000010   00 1F 00 00 01 4D 21 3B 00 0F 00 20 01 4D 24 C2
00000020   00 3F 00 40 01 4D 26 4B 00 1F 00 80 01 4D 26 E6
00000030   00 0F 00 30 01 4D 21 53 00 FF 01 00 01 4D 25 9E
00000040   00 3F 00 C0 01 4D 26 9E 00 1F 00 A0 01 4D 27 4C
00000050   00 0F 02 00 01 4D 25 3E 00 3F 02 40 01 4D 27 1D
00000060   00 0F 02 10 01 4D 26 36 00 1F 02 20 01 4D 27 69
00000070   00 0F 02 80 00 00 02 00 00 08 90 10 00 04 02 00
00000080   00 08 50 48 00 08 02 00 00 18 D0 10 00 10 02 00
00000090   00 20 60 38 00 06 02 00 00 18 50 48 00 20 02 00
000000A0   00 40 F0 00 00 18 02 00 00 40 70 28 00 14 02 00
000000B0   00 60 60 38 00 40 02 00 00 58 50 48 00 48 02 00
000000C0   00 68 D0 10 00 42 02 00 00 68 50 48 00 44 02 00
000000D0   00 78 90 30 00 50 02 00 00 78 50 48
```

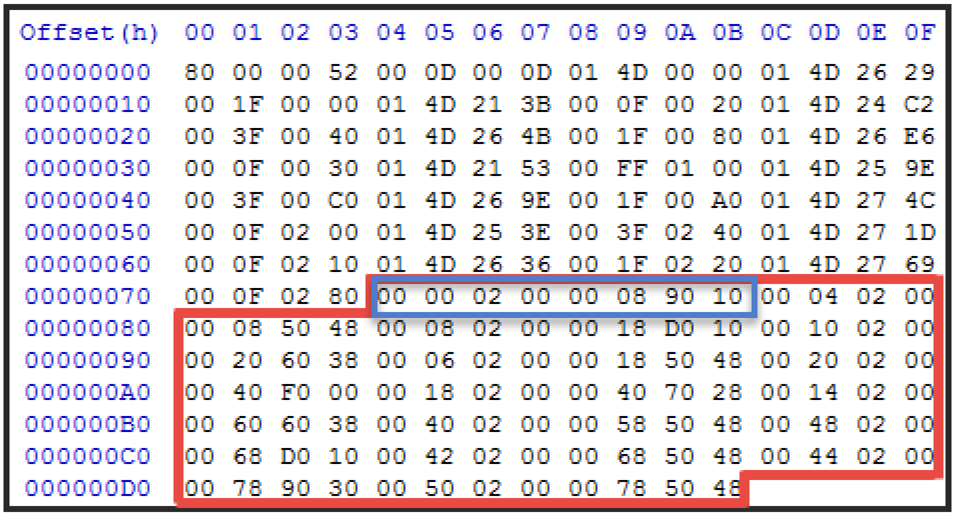
Offset	Size	Description	Data as Shown
00	4 Bytes	Tile Data Address	014D213B
04	2 Bytes	Size of Raw Tile Data	0F
06	2 Bytes	Start Position in CRam	20

After the sprite header, the sprite definition will contain each tile data definition (the 2nd one is highlighted in blue). Tile data definitions are in 8 byte lengths and are used to define where tile data is pulled from, and where it is decompressed to in the CRam. The number of tile data definitions are at offset 4 in the tile header.

Note: the size of raw tile data is a value that does not directly represent the number of bytes. It's a value that must go under a small formula

(Size * 2 + 2)* 8 = Number of bytes
(0x0F * 2 + 2) * 8 = 0x100

Tile Group Definition



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  80 00 00 52 00 0D 00 0D 01 4D 00 00 01 4D 26 29
00000010  00 1F 00 00 01 4D 21 3B 00 0F 00 20 01 4D 24 C2
00000020  00 3F 00 40 01 4D 26 4B 00 1F 00 80 01 4D 26 E6
00000030  00 0F 00 30 01 4D 21 53 00 FF 01 00 01 4D 25 9E
00000040  00 3F 00 C0 01 4D 26 9E 00 1F 00 A0 01 4D 27 4C
00000050  00 0F 02 00 01 4D 25 3E 00 3F 02 40 01 4D 27 1D
00000060  00 0F 02 10 01 4D 26 36 00 1F 02 20 01 4D 27 69
00000070  00 0F 02 80 00 00 02 00 00 08 90 10 00 04 02 00
00000080  00 08 50 48 00 08 02 00 00 18 D0 10 00 10 02 00
00000090  00 20 60 38 00 06 02 00 00 18 50 48 00 20 02 00
000000A0  00 40 F0 00 00 18 02 00 00 40 70 28 00 14 02 00
000000B0  00 60 60 38 00 40 02 00 00 58 50 48 00 48 02 00
000000C0  00 68 D0 10 00 42 02 00 00 68 50 48 00 44 02 00
000000D0  00 78 90 30 00 50 02 00 00 78 50 48
```

Offset	Size	Description	Data as Shown
00	1 Byte	Unknown	00
01	1 Byte	Tile Start Offset in CRam	00
02 (Bit 4)	1 Bit	Flip Group Horizontal	0
02 (Bit 5)	1 Bit	Flip Group Verfical	0
03	1 Byte	Unknown	0
04 (First 3 Nibbles)	12 Bits (2 bytes & 0x0FFF)	Group X Offset (Signed)	8
06 (First 3 Nibbles)	12 Bits (2 bytes & 0x0FFF)	Group Y Offset (Signed)	10
06 (Last Nibble, First 2 Bits)	2 Bits (1 byte & 0x30 >>4)	Number of Tiles Y	1
06 (Last Nibble, Last 2 Bits)	2 Bits (1 byte & 0xC0 >> 6)	Number of Tiles Y	2

After the list of tile data definitions, the sprite groups are defined. Each group is 8 bytes in length and outline what tiles to pull from the CRam, how big the group is and where it is placed relative to the position of the sprite. The number of tile group definitions are at offset 6 in the tile header.

Sprite Formation

Now we have all of the definitions for what the data represents in each portion of the sprite itself, we can go over how the sprite is formed.

Prepare the CRam

The CRam is the main buffer for the raw tile data for the sprite. It is where the raw tile data is decompressed to and stored **for the sprite as a whole**. Each tile group refers to the CRam data to determine what tile to pull and how many tiles from the CRam.

The CRam is a buffer where the size is a multiple of 256 bytes (the number of bytes for a tile). In this example, the CRam for this sprite is defined in the tile header as 0x52. It's real size is 0x2900. To get it's real size, perform the following formula:

(CRam Size / 2) * 256 = Real CRam Size
(0x52 / 2) * 0x100 = 0x2900

Tile data de-compression

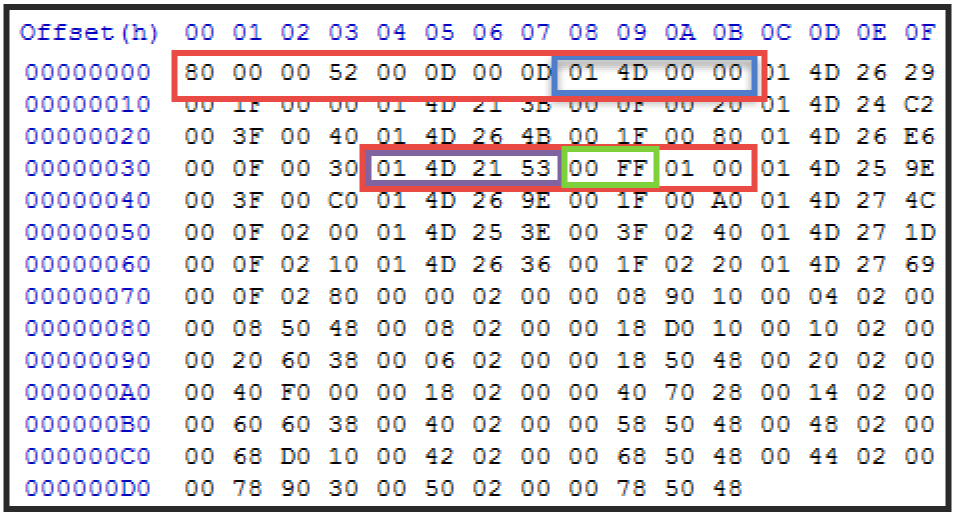
Compressed tile data is a sequence of bytes containing a real byte, an RLE byte, and a Lookup byte. The amount of bytes in the compressed tile data for the tile data definition is not defined explicitly. It is implied based on the length field in the tile data definition. Once the length is filled during decompression, the decompression is complete.

- 0x00 to 0x3F: Real byte
    - This is a direct reference to an index of the pallet
- 0x40 to 0x7F: RLE byte.
    - When Bit 7 is set and bit 8 is not set, the RLE byte fills the buffer at its current position with the last real byte, where the amount of the last real byte is filled at a count from the 6 first bits, (or by subtracting 0x40 from the RLE byte)
- 0x80 to 0xFF: Lookup byte
    - When Bit 8 is set, the first 7 bits are an index to the lookup table. The two bytes from the index of the lookup table are processed to fill data into the buffer's current position.

Decompress the Tile data definition, an example

Since the source of the tile data is in our tile data definition, we will outline the process of tile decompression from a tile data definition in our sample sprite. Lets revisit a tile data definition and the header:

The address of the lookup block is in the sprite header (0x14D0000 or 0x5A0000 in file 50) ,
The first 4 bytes of the tile data definition contains the location of the compressed data (0x14D2153 or 0x5A42A6 in file 50)
The length is at offset 4 of the tile data definition. (0xFF in this example, or a real length of 0x1000)

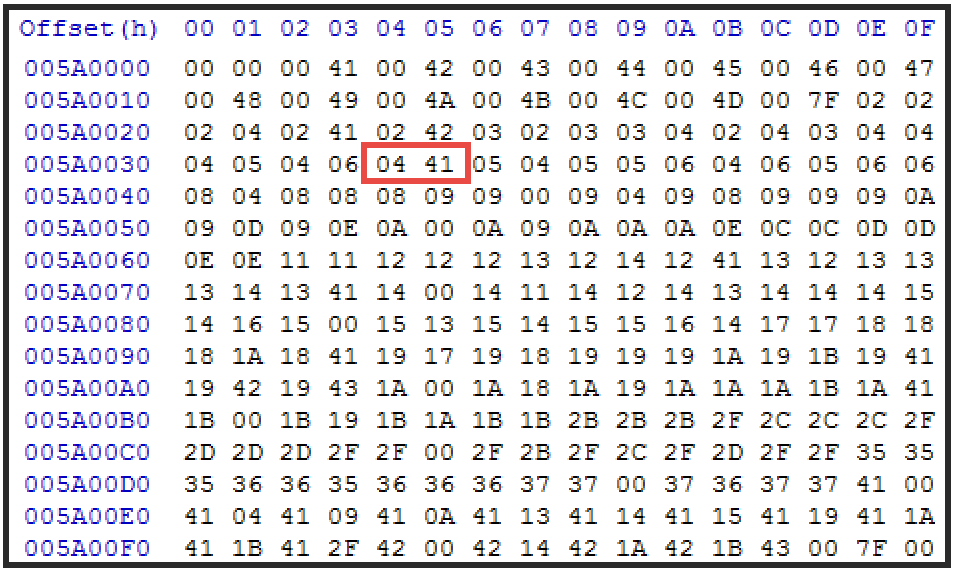


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   80 00 00 52 00 0D 00 0D 01 4D 00 00 01 4D 26 29
00000010   00 1F 00 00 01 4D 21 3B 00 0F 00 20 01 4D 24 C2
00000020   00 3F 00 40 01 4D 26 4B 00 1F 00 80 01 4D 26 E6
00000030   00 0F 00 30 01 4D 21 53 00 FF 01 00 01 4D 25 9E
00000040   00 3F 00 C0 01 4D 26 9E 00 1F 00 A0 01 4D 27 4C
00000050   00 0F 02 00 01 4D 25 3E 00 3F 02 40 01 4D 27 1D
00000060   00 0F 02 10 01 4D 26 36 00 1F 02 20 01 4D 27 69
00000070   00 0F 02 80 00 00 02 00 00 08 90 10 00 04 02 00
00000080   00 08 50 48 00 08 02 00 00 18 D0 10 00 10 02 00
00000090   00 20 60 38 00 06 02 00 00 18 50 48 00 20 02 00
000000A0   00 40 F0 00 00 18 02 00 00 40 70 28 00 14 02 00
000000B0   00 60 60 38 00 40 02 00 00 58 50 48 00 48 02 00
000000C0   00 68 D0 10 00 42 02 00 00 68 50 48 00 44 02 00
000000D0   00 78 90 30 00 50 02 00 00 78 50 48
```

Notes on the lookup block:

The lookup block is a 128 word array. The lookup block is comprised of the 128 most common 2 byte pairs found in decompressed tile data (this includes the RLE byte as well) for a group of sprites . Compressed tile data that that has the bit 7 set (or, if it is greater than 0x7F) is a lookup byte. The value to look up in the table can simply be obtained by subtracting the lookup byte by 0x80. The pair of bytes obtained from the lookup table is then placed into the tile data buffer at the time of decompression, saving 1 byte in the process.

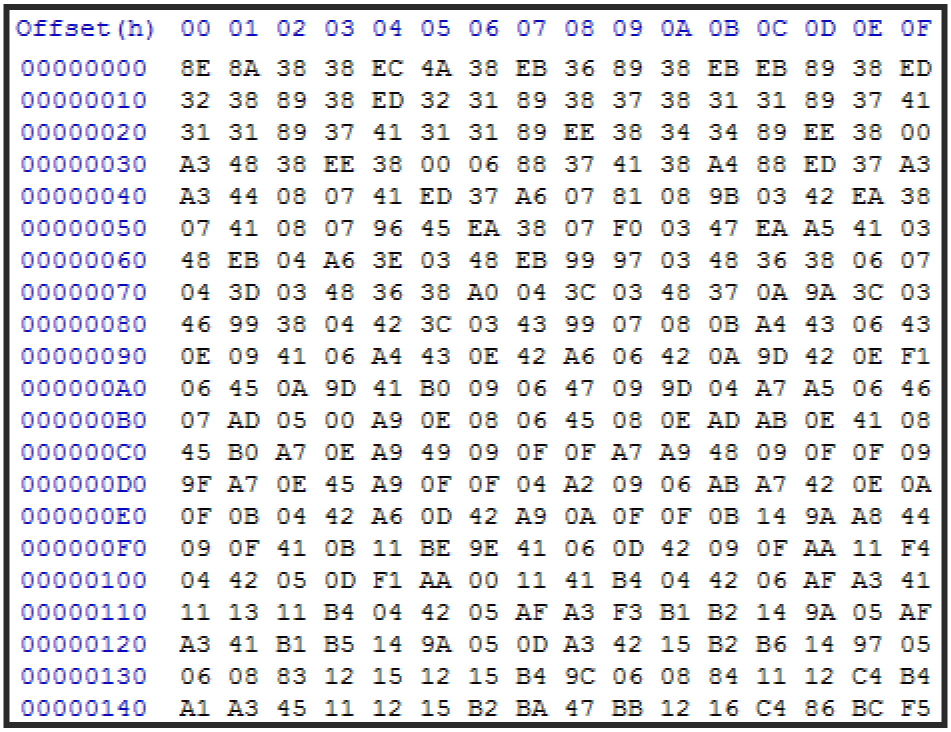
Since words are 2 byte values, the lookup index value in the compressed tile data must be multiplied by 2. For example, 0x9A - 0x80 = 0x1A. 0x1A * 2 = **0x34**



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

005A0000   00 00 00 41 00 42 00 43 00 44 00 45 00 46 00 47
005A0010   00 48 00 49 00 4A 00 4B 00 4C 00 4D 00 7F 02 02
005A0020   02 04 02 41 02 42 03 02 03 03 04 02 04 03 04 04
005A0030   04 05 04 06 04 41 05 04 05 05 06 04 06 05 06 06
005A0040   08 04 08 08 08 09 09 00 09 04 09 08 09 09 09 0A
005A0050   09 0D 09 0E 0A 00 0A 09 0A 0A 0A 0E 0C 0C 0D 0D
005A0060   0E 0E 11 11 12 12 12 13 12 14 12 41 13 12 13 13
005A0070   13 14 13 41 14 00 14 11 14 12 14 13 14 14 14 15
005A0080   14 16 15 00 15 13 15 14 15 15 16 14 17 17 18 18
005A0090   18 1A 18 41 19 17 19 18 19 19 19 1A 19 1B 19 41
005A00A0   19 42 19 43 1A 00 1A 18 1A 19 1A 1A 1A 1B 1A 41
005A00B0   1B 00 1B 19 1B 1A 1B 1B 2B 2B 2B 2F 2C 2C 2C 2F
005A00C0   2D 2D 2D 2F 2F 00 2F 2B 2F 2C 2F 2D 2F 2F 35 35
005A00D0   35 36 36 35 36 36 36 37 37 00 37 36 37 37 41 00
005A00E0   41 04 41 09 41 0A 41 13 41 14 41 15 41 19 41 1A
005A00F0   41 1B 41 2F 42 00 42 14 42 1A 42 1B 43 00 7F 00
```

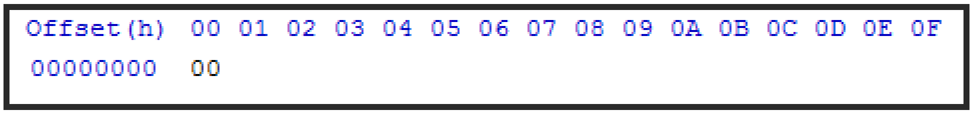
Decompression in action

The compressed tile data:



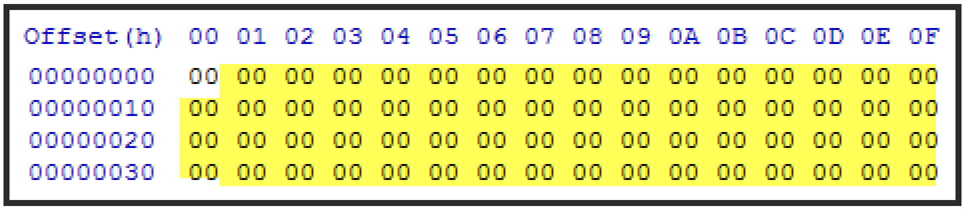
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   8E 8A 38 38 EC 4A 38 EB 36 89 38 EB EB 89 38 ED
00000010   32 38 89 38 ED 32 31 89 38 37 38 31 31 89 37 41
00000020   31 31 89 37 41 31 31 89 EE 38 34 34 89 EE 38 00
00000030   A3 48 38 EE 38 00 06 88 37 41 38 A4 88 ED 37 A3
00000040   A3 44 08 07 41 ED 37 A6 07 81 08 9B 03 42 EA 38
00000050   07 41 08 07 96 45 EA 38 07 F0 03 47 EA A5 41 03
00000060   48 EB 04 A6 3E 03 48 EB 99 97 03 48 36 38 06 07
00000070   04 3D 03 48 36 38 A0 04 3C 03 48 37 0A 9A 3C 03
00000080   46 99 38 04 42 3C 03 43 99 07 08 0B A4 43 06 43
00000090   0E 09 41 06 A4 43 0E 42 A6 06 42 0A 9D 42 0E F1
000000A0   06 45 0A 9D 41 B0 09 06 47 09 9D 04 A7 A5 06 46
000000B0   07 AD 05 00 A9 0E 08 06 45 08 0E AD AB 0E 41 08
000000C0   45 B0 A7 0E A9 49 09 0F 0F A7 A9 48 09 0F 0F 09
000000D0   9F A7 0E 45 A9 0F 0F 04 A2 09 06 AB A7 42 0E 0A
000000E0   0F 0B 04 42 A6 0D 42 A9 0A 0F 0F 0B 14 9A A8 44
000000F0   09 0F 41 0B 11 BE 9E 41 06 0D 42 09 0F AA 11 F4
00000100   04 42 05 0D F1 AA 00 11 41 B4 04 42 06 AF A3 41
00000110   11 13 11 B4 04 42 05 AF A3 F3 B1 B2 14 9A 05 AF
00000120   A3 41 B1 B5 14 9A 05 0D A3 42 15 B2 B6 14 97 05
00000130   06 08 83 12 15 12 15 B4 9C 06 08 84 11 12 C4 B4
00000140   A1 A3 45 11 12 15 B2 BA 47 BB 12 16 C4 86 BC F5
```

- The first byte is 0x8E. Since 0x8E is a byte with bit 7 set (or greater than 0x80), it is a reference to the lookup index. 0x8E - 0x80 = 0x0E. 0x0E * 2 = **0x1C**. 0x1C in the lookup table is 0x04 0x7F
    - 0x00 is less than 0x80 and less than 0x40. It is a real byte. 0x00 at this time is the last real byte



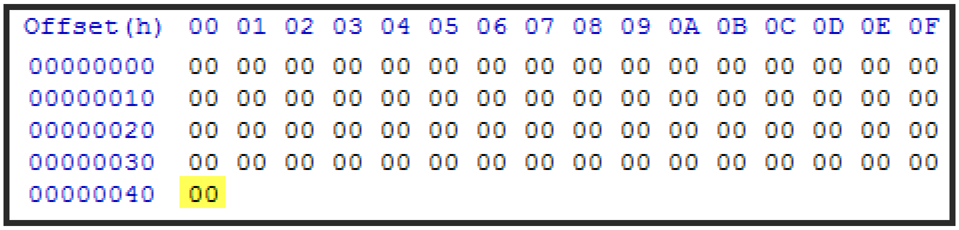
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   00
```

    - 0x7F is an RLE byte, as bit 7 is not set and 8 is set (or, it is less than 0x40). 0x7F - 0x40 = 0x3F. Since the last real byte is 0x00, we fill the buffer at its current position 0x3F (or 63) times.



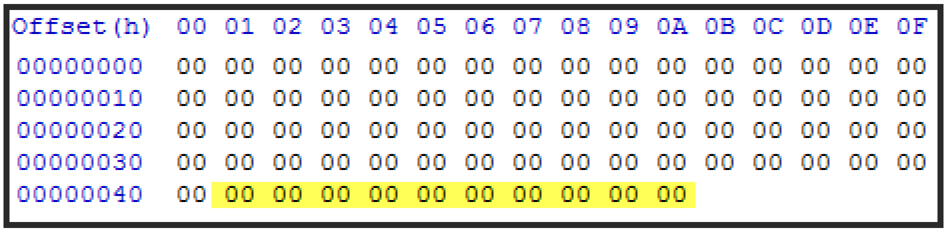
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- The next byte is 0x8A. Another lookup byte. 0x8A - 0x80 = 0x0A. 0x0A * 2 = **0x14**. 0x14 in the lookup table is 0x00 0x4A
    - 0x00 is another real byte. Our last real byte is set to 0x00



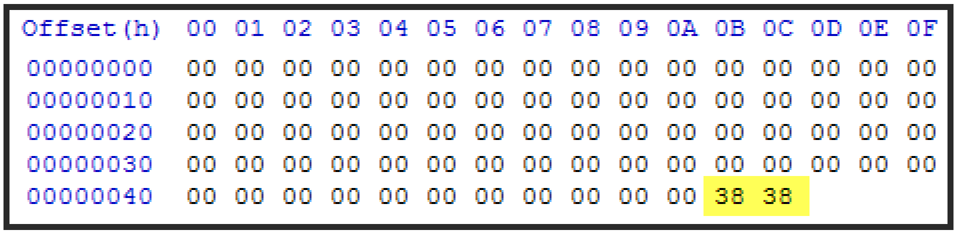
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040   00
```

- 0x4A is an RLE byte. 0x4A - 0x40 = 0x0A. Repeat the last real byte 0x0A times (or 10)



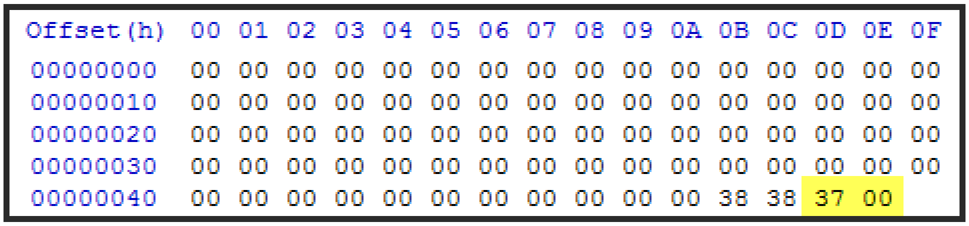
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040  00 00 00 00 00 00 00 00 00 00 00
```

- The next byte is 0x38. 0x38 is a Real byte.
- The next byte is another real byte, 0x38



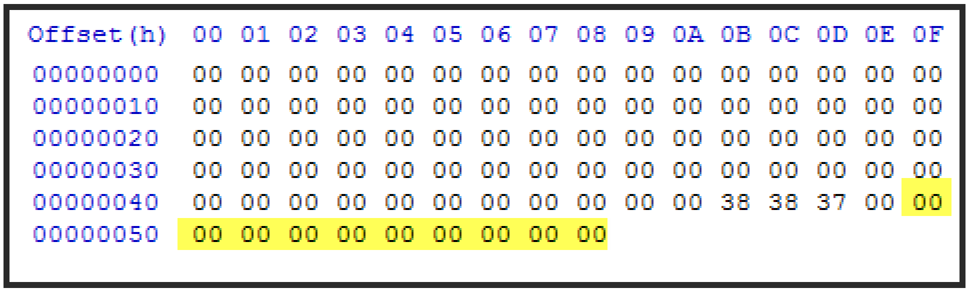
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040  00 00 00 00 00 00 00 00 00 00 00 38 38
```

- The next byte is 0xEC. 0xEC is a Lookup byte. 0xEC - 0x80 = 0x6C. 0x6C * 2 = **0xD8**. 0xD8 in the lookup table is 0x37 0x00. These are both real bytes, with 0x00 being the last real byte.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040  00 00 00 00 00 00 00 00 00 00 00 38 38 37 00
```

- The next byte is 0x4A. 0x4A is an RLE byte. 0x4A - 0x40 = 0x0A. Since the last real byte is 0x00, it is repeated 0x0A times (or 10)



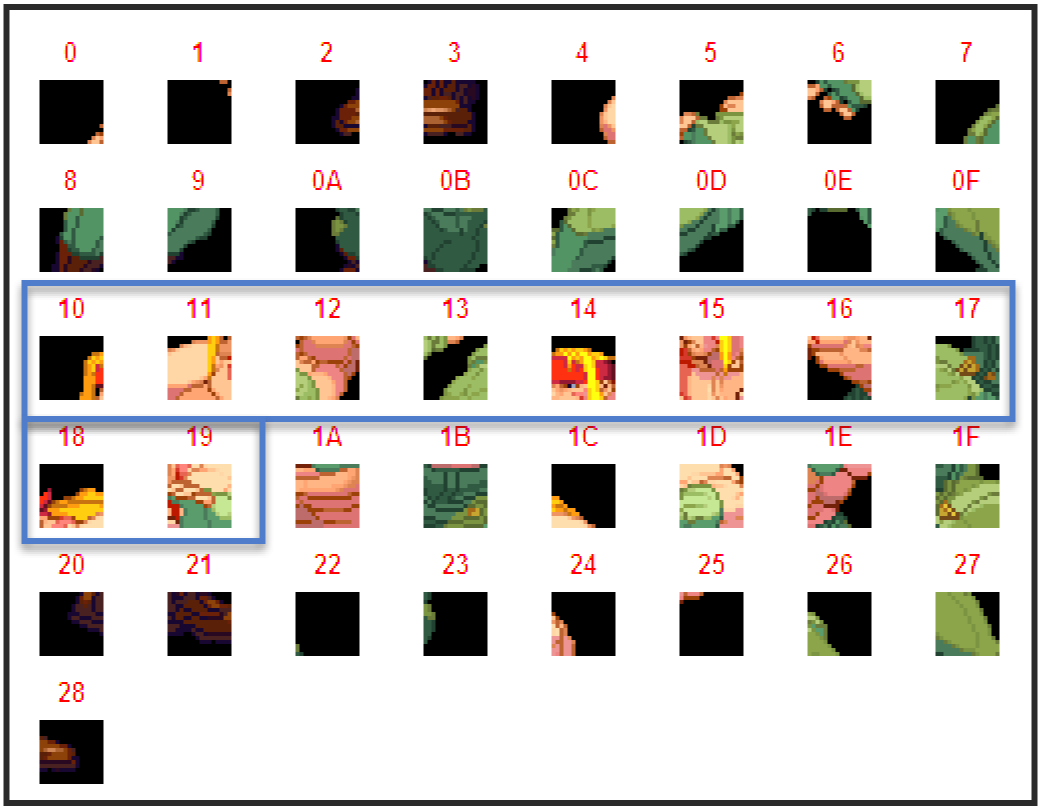
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040  00 00 00 00 00 00 00 00 00 00 00 38 38 37 00 00
00000050  00 00 00 00 00 00 00 00 00
```

Repeat the process of decompression until the buffer is filled up to the real length.

Tile data placement

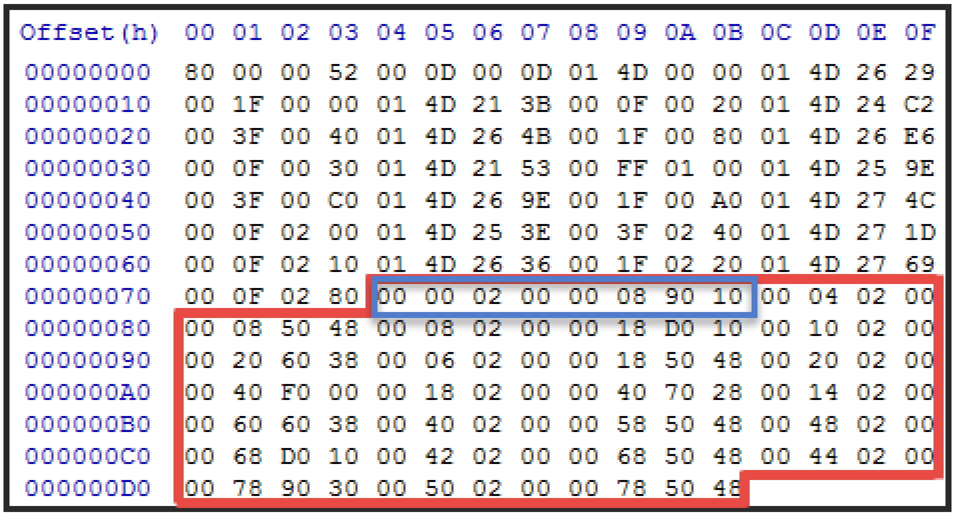
In our tile data definition, it is actually placed in the CRam buffer at a different position. Remember, the real CRam size is 0x2900 and the size of a tile is 0x100 bytes (or 256 bytes). That gives us 0x29 (or 41) tile slots. For simplicity's sake, we will use the first two bytes of the word to reference which slot we're using.

The Start Position in CRam value is 0x100 in the tile data definition. It's real position can be obtained by multiplying the value by 0x10, which results to 0x1000. Slot # 10 is where the data is placed. Since the real length is 0x1000 as well, it will fill up 10 slots.



Tile Group Formation

Let’s go by the first definition we see here:



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   80 00 00 52 00 0D 00 0D 01 4D 00 00 01 4D 26 29
00000010   00 1F 00 00 01 4D 21 3B 00 0F 00 20 01 4D 24 C2
00000020   00 3F 00 40 01 4D 26 4B 00 1F 00 80 01 4D 26 E6
00000030   00 0F 00 30 01 4D 21 53 00 FF 01 00 01 4D 25 9E
00000040   00 3F 00 C0 01 4D 26 9E 00 1F 00 A0 01 4D 27 4C
00000050   00 0F 02 00 01 4D 25 3E 00 3F 02 40 01 4D 27 1D
00000060   00 0F 02 10 01 4D 26 36 00 1F 02 20 01 4D 27 69
00000070   00 0F 02 80 00 00 02 00 00 08 90 10 00 04 02 00
00000080   00 08 50 48 00 08 02 00 00 18 D0 10 00 10 02 00
00000090   00 20 60 38 00 06 02 00 00 18 50 48 00 20 02 00
000000A0   00 40 F0 00 00 18 02 00 00 40 70 28 00 14 02 00
000000B0   00 60 60 38 00 40 02 00 00 58 50 48 00 48 02 00
000000C0   00 68 D0 10 00 42 02 00 00 68 50 48 00 44 02 00
000000D0   00 78 90 30 00 50 02 00 00 78 50 48
```

If you go back to refer how the values of the groups are pulled, you will determine that the sequence of bytes highlighted in blue consist of the following (ignoring most 0 values):

Tile Start Offset in CRam: 00
Group X Offset: 8
Group Y Offset: 10
Number of Tiles X: 1
Number of Tiles Y: 2