VDPの HMMM を用いた巨大キャラのサンプル

【前知識】
V9938/V9958 には、VDPコマンドと呼ばれる画像操作機能がある。
CPUからVDPに対してVDPコマンドを要求すると、VDPがそのコマンドを処理する。
処理には多少の時間がかかるが、CPUはその間別の演算を実施することができる。

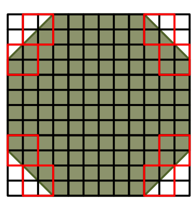
これを利用して、CPUとVDPの両方の稼働率が高くなるような順番で処理させると、演算性能が向上する。

VDPコマンドは、Sprite表示をOFFにすると、全体的に高速になる。
これは、VDPがSpriteのために確保しているDRAMアクセスタイミングを、VDPコマンドプロセッサに解放するためだと思われる。

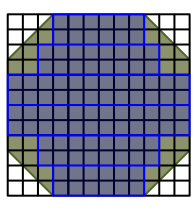
VDPコマンド実行中も VDPレジスタにアクセスは可能であるが、R#8 や R#18 等の「書き込むとVDPコマンドの挙動を乱すレジスタ」
が存在し、それらはVDPコマンド実行中に書き換えてはならない。このような動作はエミュレータでは実装されていない可能性が高く、
実機ではVDPコマンドの挙動がおかしくなって表示が崩れるが、エミュレータでは崩れないという現象が発生するので要注意である。

【このサンプルでやること】
(1) ゼビ〇スのアンド〇ジェネシスを浮遊させる
(2) SCREEN5 でやる

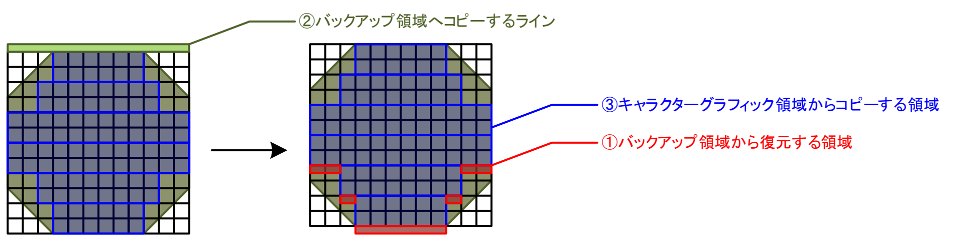
【巨大キャラの移動】
1フレームは、NTSCでは 1/59.94 であるが、この間に HMMM で転送できるのは 6000dot 程度(Sprite on時)である。
しかしながら、HMMM を使うとロジカルオペレーションが使えないため、
巨大キャラを 96x96dot で表現するとすれば、下記の赤枠部分を 16x16dot のスプライトで表現すれば、
残りは矩形ブロックの集合となり、HMMM 転送でコピー可能となる。



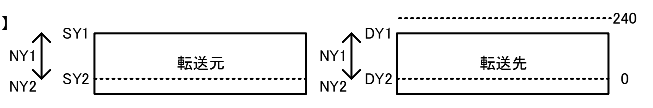
転送画素数も、赤い領域と絵のない領域を除けば、8x8 のブロックが100個になるため、6400dot となる。
これらを、数回の HMMM転送で描画することになるが、R#23 でスクロールするならば、line#255→line#0 の境界を
またがるケースが発生する。その場合、line#255側と、line#0側は分けて転送する必要があるため、
小分けのブロックは水平には分割しない方がその判定処理にかかる負荷を軽減できる。
下記の青い領域に分割して処理することとする。



【スクロール】
R#23 によるスクロールは、R#23 に表示画面の一番上に来る走査線番号を指定することによって行われる。
そのため、画像が上から下へ流れるタイプの垂直スクロールは、0→255→254 ... とデクリメント方向に変化させることになる。
巨大キャラは、見かけ上停止して見えるため、VRAM上では上に移動する形となる。



【HMMMコマンドの分割】



```
if DY1 > 240 then
  HMMMコマンド2回で転送
else
  HMMMコマンド1回で転送
endif
```

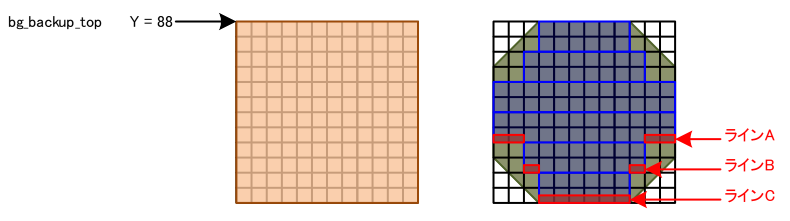
HMMMコマンド1回で転送の場合
SY1 .... 所望のパーツが置かれている page1 の Y座標
NY1 .... 16固定

HMMMコマンド2回で転送の場合
SY1 .... 所望のパーツが置かれている page1 の Y座標
NY1 .... neg DY1
SY2 .... (neg DY1) + SY1
NY2 .... DY1 & 15
DY2 0

【note】
241 は、8bit 2の補数表現と見なすと −15 になる。
neg −15 = 15
242 は、8bit 2の補数表現と見なすと −14 になる。
neg −14 = 14
.. となるため、NY1 は上記のようになる。

【端の画素のバックアップ】

巨大キャラの背景部分バックアップ領域は、この中でスクロールに合わせて画像をずらす時間はもったいないので、Y座標をラウンドロビンで利用する。



巨大キャラの背景部分バックアップ領域

```
' 巨大キャラの背景部分バックアップ領域上のラインA の位置
line_a = bg_backup_top + 64
if line_a >= 184 then
  line_a = line_a − 96
endif
```

```
' 巨大キャラの背景部分バックアップ領域上のラインB の位置
line_b = bg_backup_top + 80
if line_b >= 184 then
  line_b = line_b + 96
endif
```

```
' 巨大キャラの背景部分バックアップ領域上のラインC の位置
line_c = bg_backup_top − 1
if line_c < 88 then
  line_c = 88 + 96 − 1
endif
```

```
' 次のバックアップ位置算出
if bg_backup_top = 88 then
  bg_backup_top = 88 + 104 − 1
else
  bg_backup_top = bg_backup_top − 1
endif
```

ラインA, B, C のバックアップからの復元は、1ラインなので HMMM分割には該当せず、普通に HMMMすれば良い。巨大キャラの一番上のラインを big_char_y とすれば、下記の値となる。

ラインA 左

```
R#32 = 0
R#33 = 0
R#34 = line_a
R#35 = 1
R#36 = 80
R#37 = 0
R#38 = big_char_y + 63
R#39 = 0
R#40 = 16
R#41 = 0
R#42 = 1
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

ラインA 右

```
R#32 = 80
R#33 = 0
R#34 = line_a
R#35 = 1
R#36 = 160
R#37 = 0
R#38 = big_char_y + 63
R#39 = 0
R#40 = 16
R#41 = 0
R#42 = 1
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

ラインB 左

```
R#32 = 16
R#33 = 0
R#34 = line_b
R#35 = 1
R#36 = 96
R#37 = 0
R#38 = big_char_y + 79
R#39 = 0
R#40 = 8
R#41 = 0
R#42 = 1
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

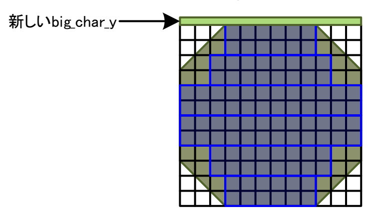
ラインB 右

```
R#32 = 72
R#33 = 0
R#34 = line_b
R#35 = 1
R#36 = 152
R#37 = 0
R#38 = big_char_y + 79
R#39 = 0
R#40 = 8
R#41 = 0
R#42 = 1
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

ラインC

```
R#32 = 24
R#33 = 0
R#34 = line_c
R#35 = 1
R#36 = 104
R#37 = 0
R#38 = big_char_y + 95
R#39 = 0
R#40 = 48
R#41 = 0
R#42 = 1
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

次に巨大キャラを上書きしてしまう背景領域を、巨大キャラ背景部分バックアップ領域へコピーする

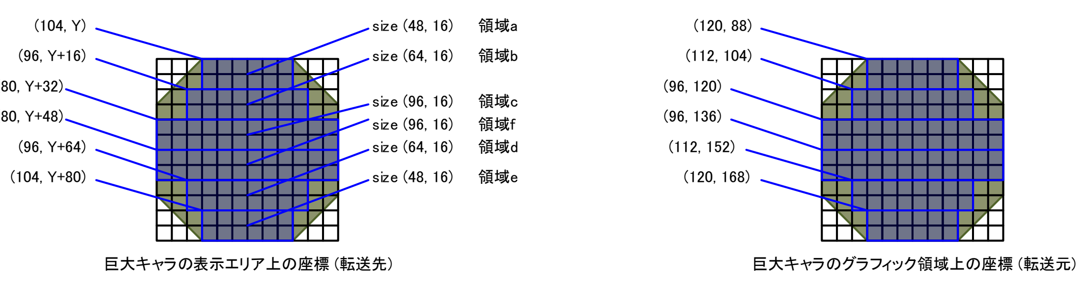


```
' 巨大キャラの新しい表示位置を求める
big_char_y = (big_char_y − 1) and 255
```

バックアップするライン

```
R#32 = 80
R#33 = 0
R#34 = big_char_y
R#35 = 0
R#36 = 0
R#37 = 0
R#38 = bg_backup_top
R#39 = 1
R#40 = 96
R#41 = 0
R#42 = 1
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

巨大キャラそのものを描画する



巨大キャラの表示エリア上の座標(転送先)　　巨大キャラのグラフィック領域上の座標(転送元)

領域a

```
R#32 = 120
R#33 = 0
R#34 = 88
R#35 = 1
R#36 = 104
R#37 = 0
R#38 = big_char_y
R#39 = 0
R#40 = 48
R#41 = 0
R#42 = 16
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

領域b

```
R#32 = 112
R#33 = 0
R#34 = 104
R#35 = 1
R#36 = 96
R#37 = 0
R#38 = big_char_y + 16
R#39 = 0
R#40 = 64
R#41 = 0
R#42 = 16
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

領域c

```
R#32 = 96
R#33 = 0
R#34 = 120
R#35 = 1
R#36 = 80
R#37 = 0
R#38 = big_char_y + 32
R#39 = 0
R#40 = 96
R#41 = 0
R#42 = 16
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

領域f

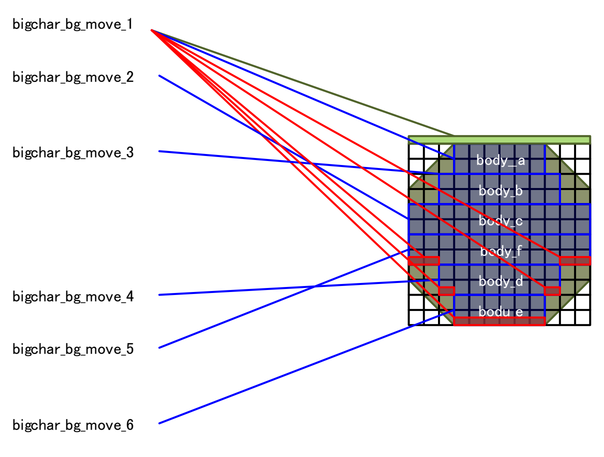
```
R#32 = 96
R#33 = 0
R#34 = 138
R#35 = 1
R#36 = 80
R#37 = 0
R#38 = big_char_y + 48
R#39 = 0
R#40 = 96
R#41 = 0
R#42 = 16
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

領域d

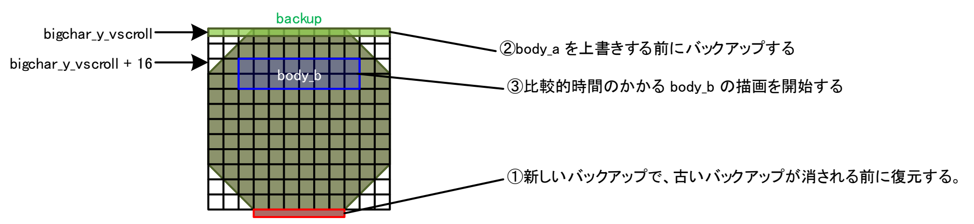
```
R#32 = 112
R#33 = 0
R#34 = 152
R#35 = 1
R#36 = 96
R#37 = 0
R#38 = big_char_y + 64
R#39 = 0
R#40 = 64
R#41 = 0
R#42 = 16
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

領域e

```
R#32 = 120
R#33 = 0
R#34 = 168
R#35 = 1
R#36 = 104
R#37 = 0
R#38 = big_char_y + 80
R#39 = 0
R#40 = 48
R#41 = 0
R#42 = 16
R#43 = 0
R#44 = 0
R#45 = 0
R#46 = 0b11010000
```

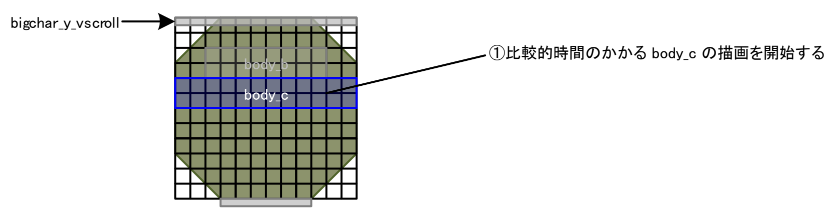


bigchar_bg_move_intr1_0



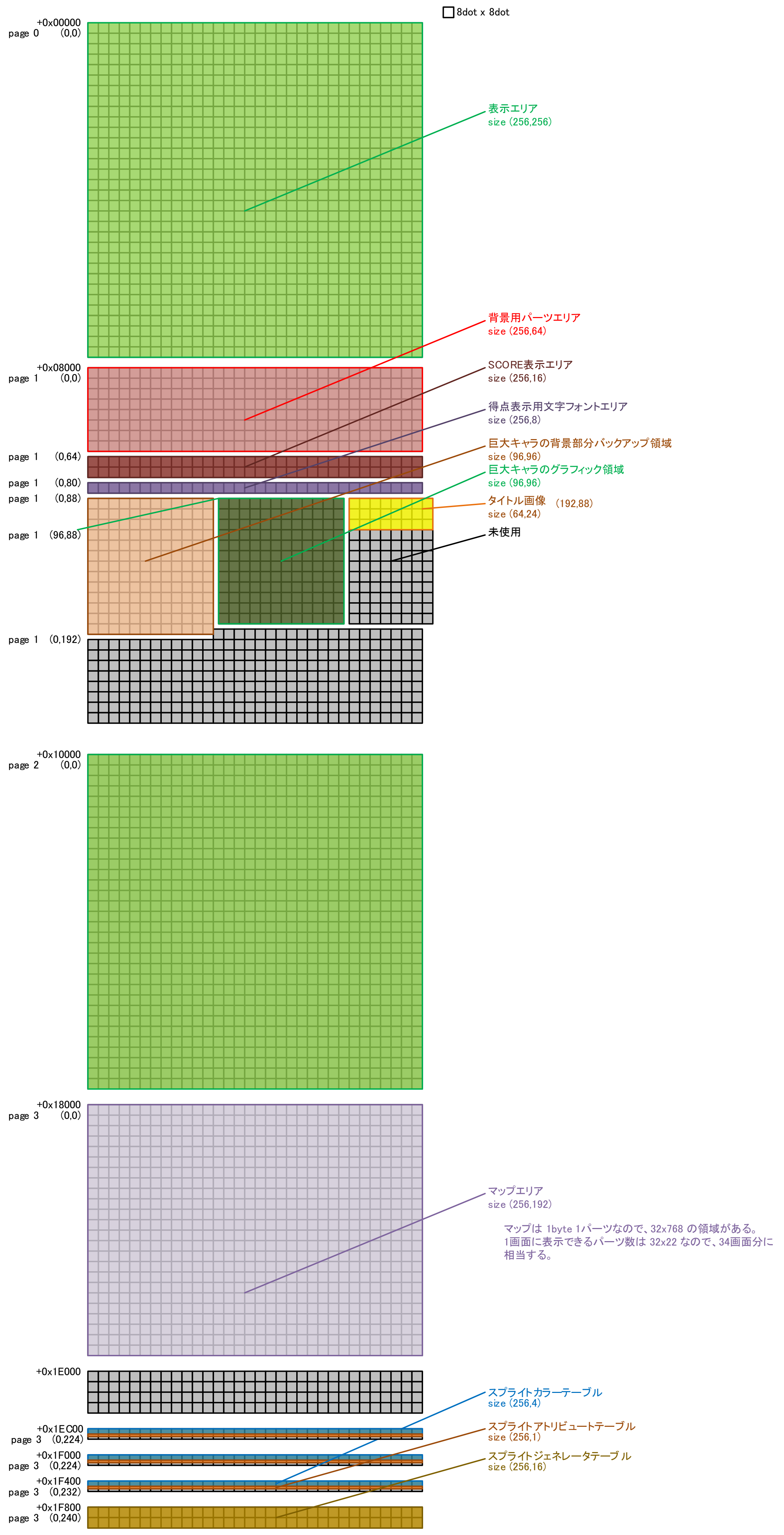
②body_a を上書きする前にバックアップする
③比較的時間のかかる body_b の描画を開始する
①新しいバックアップで、古いバックアップが消される前に復元する。

bigchar_bg_move_intr2_0



①比較的時間のかかる body_c の描画を開始する

【VRAMマップ】



マップは 1byte 1パーツなので、32x768 の領域がある。
1画面に表示できるパーツ数は 32x22 なので、34画面分に
相当する。

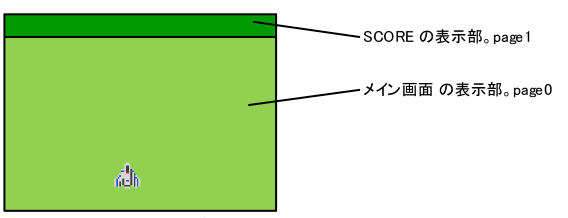
Sprite#
0: 自機1
1: 自機2
2: 弾1
3: 弾2
4: 弾3
5: 弾4
6: 弾5
7: 弾6
8: 弾7
9: 弾8
10: 弾9
11: 弾10
12: 弾11
13: 弾12
14: 弾13
15: フラッグ
16: 巨大キャラ 部分スプライト①-1
17: 巨大キャラ 部分スプライト①-2
18: 巨大キャラ 部分スプライト②-1
19: 巨大キャラ 部分スプライト②-2
20: 巨大キャラ 部分スプライト③-1
21: 巨大キャラ 部分スプライト③-2
22: 巨大キャラ 部分スプライト④-1
23: 巨大キャラ 部分スプライト④-2
24: 巨大キャラ 部分スプライト⑤-1
25: 巨大キャラ 部分スプライト⑤-2
26: 巨大キャラ 部分スプライト⑥-1
27: 巨大キャラ 部分スプライト⑥-2
28: 巨大キャラ 部分スプライト⑦-1
29: 巨大キャラ 部分スプライト⑦-2
30: 巨大キャラ 部分スプライト⑧-1
31: 巨大キャラ 部分スプライト⑧-2

【VDPコントロールレジスタ設定】

R#2: Pattern name table base address register

bit	7	6	5	4	3	2	1	0
	0	A16	A15	1	1	1	1	1

SCREEN5 では表示画面のページ番号を A16, A15 の 2bit に指定する。



画面表示のイメージ

R#5: Sprite attribute table base address register low

bit	7	6	5	4	3	2	1	0
	A14	A13	A12	A11	A10	A9	1	1

SCREEN1～12 のスプライトアトリビュートテーブルのアドレスを指定するレジスタ。
A8～A0 は 0b000000000 固定。A16～A15 は R#11 で指定。

R#11: Sprite attribute table base address register high

bit	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	A16	A15

SCREEN1～12 のスプライトアトリビュートテーブルのアドレスを指定するレジスタ。
A8～A0 は 0b000000000 固定。A14～A9 は R#5 で指定。

スプライトアトリビュートテーブルは、0x1EE00 と 0x1F200 と 0x1F600 と 0x0F200 と 0x0F600 の5カ所に設定するので、
0b1_1110_1110_0000_0000 と 0b?_1111_0010_0000_0000 と 0b?_1111_0110_0000_0000 のいずれかを指定することになる。
A14～A9 は 0b110_111 または 0b111_001 または 0b111_011
A16,～A15 は 0b?1
従って
R#5　に設定する値は 0b1101_1111 または 0b1110_0111 または 0b1110_1111
R#11 に設定する値は 0b0000_00?1
となる。
? は、表示ページ。

R#6: Sprite pattern generator table base address register

bit	7	6	5	4	3	2	1	0
	0	0	A16	A15	A14	A13	A12	A11

SCREEN1～12 のスプライトパターンジェネレータテーブルのアドレスを指定するレジスタ。
A10～A0 は 0 固定。

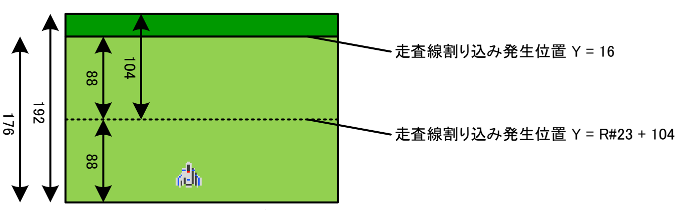
スプライトジェネレーターテーブルは、0x1F800 に固定するので、0b1_1111_1000_0000_0000 を指定することになる。
A16～A11 は 0b1_1111_1
従って
R#6　に設定する値は 0b0011_1111
となる。

スプライトカラーテーブルは、自動的にスプライトアトリビュートテーブルの直前の 512byte に割り付けられるため、アドレス設定レジスタは存在しない。

R#19: Interrupt line register

bit	7	6	5	4	3	2	1	0
	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0

R#0 の bit4 (IE_1) が 1 の場合に、割り込みが発生するライン番号を指定する。
画面表示上のライン番号ではなく、VRAM上のライン番号を指定するため、
画面上のどのあたりで割り込みが発生するかは、R#23 に設定している内容に影響されて上下することに注意すること。



画面表示のイメージ

MSXでは、Z80は割り込みモード1（IM 1）で動作しているため、CPU周辺回路からの割り込みは 0x0038番地に飛んでくる。
MAIN-ROM の 0x0038番地には、割り込み処理ルーチンへのジャンプコードが書かれており、
その割り込み処理ルーチンでは、裏レジスタを含む全レジスタをスタックへ退避した後に、すぐさま H.KEYIフックを呼ぶように記述されている。
H.KEYIをフックした独自の割り込み処理ルーチン冒頭で、page の切り替え（R#2書き換え）を実施すると、R#19 に記述したライン番号あたりから切り替わるわけだが、MAIN-ROM内の割り込み処理を実施する間に VDPは 3ライン程度出力が進んでしまうことがわかっている。（※Z80の場合）

そのため、16ライン目で切り替えるために、R#19 には 16-3 の値を指定する。

R#0: Mode register 0

bit	7	6	5	4	3	2	1	0
	0	DG	IE_2	IE_1	M_5	M_4	M_3	0

画面モードの指定レジスタ

000: SCREEN0 WIDTH40 (TEXT1)
010: SCREEN0 WIDTH80 (TEXT2)
000: SCREEN1 (GRAPHIC1)
001: SCREEN2 (GRAPHIC2)
000: SCREEN3 (MULTI COLOR)
010: SCREEN4 (GRAPHIC3)
011: SCREEN5 (GRAPHIC4)
100: SCREEN6 (GRAPHIC5)
101: SCREEN7 (GRAPHIC6)
111: SCREEN8 (GRAPHIC7)
111: SCREEN10(GRAPHIC7)
111: SCREEN11(GRAPHIC7)
111: SCREEN12(GRAPHIC7)

水平帰線割込許可レジスタ

0: 水平帰線割り込み禁止(通常の状態)
1: 水平帰線割り込み許可

いわゆる「走査線割り込み」の許可指定。
割り込む走査線番号は R#19 に指定する。

ライトペン割込許可レジスタ(MSX2では未使用, V9958ではこの機能は削除)

0: ライトペン割り込み禁止(通常の状態)
1: ライトペン割り込み許可

カラーバスの入出力方向指定レジスタ

0: 出力(通常の状態)
1: 入力(デジタイズ機能を持ったMSX2でのみ使用可能)

SCREEN5なので
　M5,M4,M3 = 0b011

走査線割り込みを使うので
　IE1 = 1

ライトペン割り込みは使わないので
　IE2 = 0

カラーバスは出力なので
　DG = 0

これにより、R#0 に書き込む値が決定される。
　R#0 = 0b0001_0110

この書き込みを終えた直後から、走査線割り込みが有効になる。
走査線割り込みが入ってきても良い準備が終わってから、このレジスタ設定を書き込む必要がある。
BIOS によって設定されている内容は REG0SAV に保存されているため、それを読みだして IE1 だけ 1 にすやり方もある。
VDPコマンド稼働中にこのレジスタをいじると、VDPコマンドの挙動が乱れる恐れがある。

R#23: Display offset register

bit	7	6	5	4	3	2	1	0
	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

表示開始のライン番号を指定する。

ハードウェア垂直スクロールのスクロール量を指定するレジスタである。
スコア表示部は 0、メイン画面はスクロール状況に応じて刻々と変化する設定となるため、

H.TIMI割り込み内で R#23 = 64 にする
H.KEYIの走査線割り込み内で R#23 = スクロール値 にする

画像が上から下へ流れるスクロールを実施するので、スクロール値はデクリメントする値となる。
このレジスタを書き換えると、画面表示上における R#19 の割り込み発生位置も変化するため、R#19 との連携を考慮する必要がある。

R#9: Mode register 3

bit	7	6	5	4	3	2	1	0
	LN	0	S1	S0	IL	EO	NT	DC

DLCLKモード

0: DLCLK端子を出力モードにする
1: DLCLK端子を入力モードにする

NTSC/PALモード

0: NTSC(262line)
1: PAL(313line)

RGB出力のみ有効。MSXはRGB出力を変換してコンポジット出力を精製しているため、この設定で NTSC, PAL を切り替えられる。

Even/Odd 交互表示

0: Even field/Odd Field で同じ画像を表示
1: Even filed/Odd Field で2枚の画像を交互表示

インターレース表示

0: ノンインターレース
1: インターレース

同期モード(YS信号)

00: VDPの画面を表示(常にYS=0)
01: 表示画面の透明部分で同期信号(YS=1)発生
10: 常に外部信号を選択(常にYS=1)
11: 予約

表示ライン数

192ラインモードで利用すると、VDPコマンドがやや速くなる(と思う)。
そのため、LN=0 にする。

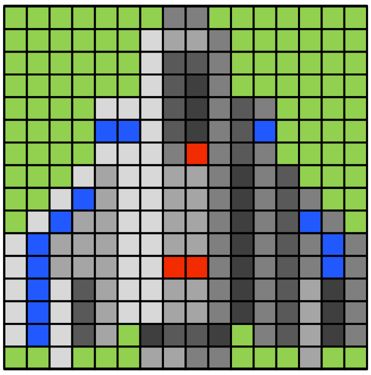
REG9SAV に元の値が保存されているので、これを元に LN だけ修正して R#9 に書き込む。

R#14: VRAM Access base address register

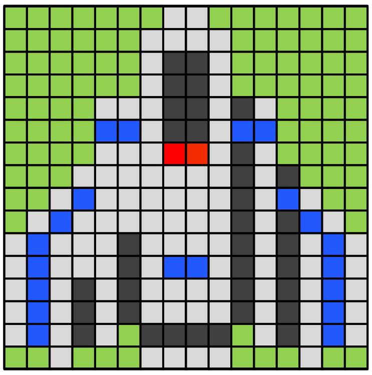
bit	7	6	5	4	3	2	1	0
	0	0	0	0	0	A16	A15	A14

VRAMにアクセスするときに、アドレスの上位3bit をこのレジスタにセットする。
この値は、SCREEN0～3 の時には、VRAMアクセスによるアドレスインクリメントの影響を受けない。(TMS9918との互換のため)
SCREEN4以上では A13 のキャリーを受けて自動的にインクリメントする。

【自機】

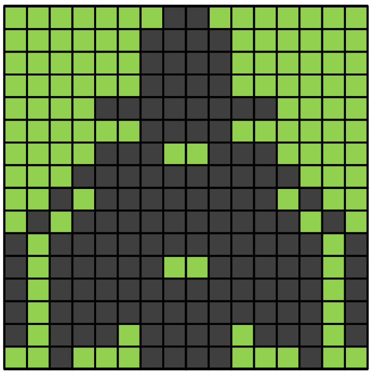


色を自由に使える場合



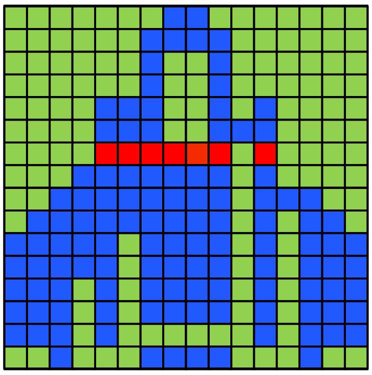
水平3色までに減色

Pattern#0



左	右	色
0x01	0x80	0x03
0x03	0xC0	0x03
0x03	0xC0	0x03
0x03	0xC0	0x03
0x0F	0xF0	0x03
0x03	0xC0	0x03
0x0E	0x70	0x03
0x1F	0xF8	0x03
0x2F	0xF4	0x03
0x5F	0xFA	0x03
0xBF	0xFD	0x03
0xBE	0x7D	0x03
0xBF	0xFD	0x03
0xBF	0xFD	0x03
0xBB	0xDD	0x03
0x23	0xC4	0x03

Pattern#4



0x01	0x80	0x46
0x03	0xC0	0x46
0x02	0x40	0x46
0x02	0x40	0x46
0x0E	0x50	0x46
0x0E	0x70	0x46
0x0F	0xD0	0x44
0x1F	0xD0	0x46
0x3F	0xDC	0x46
0x7F	0xD6	0x46
0xFB	0xD7	0x46
0xFB	0xD7	0x46
0xEB	0xD7	0x46
0xEB	0xD7	0x46
0xE8	0x17	0x46
0x23	0xC4	0x46

2つのスプライトに分離

	palette#	
🟥	0b0100	4
🟦	0b0110	6
⬜	0b0111	7
⬛	0b0011	3

⬛ or 🟦 = ⬜

⬛ or 🟥 = ⬜

[2] キー入力

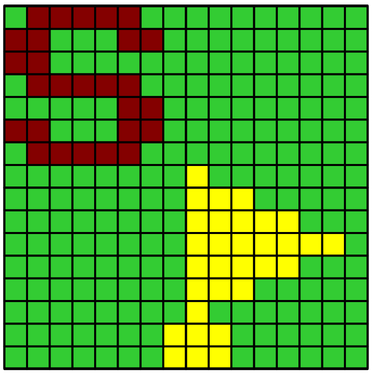
BIOS の GTSTICK を使うのが最も簡単なので、これを利用します。
GTSTICK は、MSX-BASIC の STICK(n) 関数に相当するルーチンで、Aレジスタに n を指定することになります。
結果は Aレジスタに返ってきますが、この値も STICK(n) と同じ値になります。

MSX-BASIC で、A = STICK(0) OR STICK(1) としていたなら、下記のプログラムで同様の結果を得られます。

```
xor   a, a
call  gtstick      ; get stick(0)
push  af
ld    a, 1
call  gtstick      ; get stick(1)
pop   bc
or    a, b         ; a = stick(0) or stick(1)
```

n	means
0	cursor keys
1	joystick 1
2	joystick 2

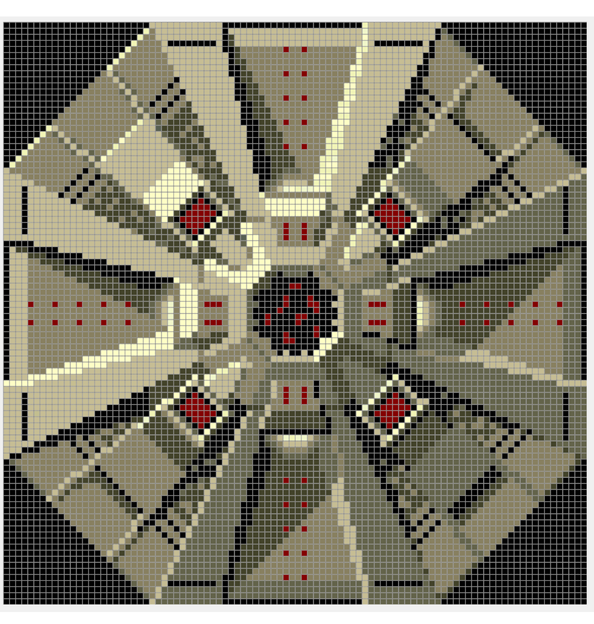
返値	means
0	押されていない
1	上
2	右上
3	右
4	右下
5	下
6	左下
7	左
8	左上



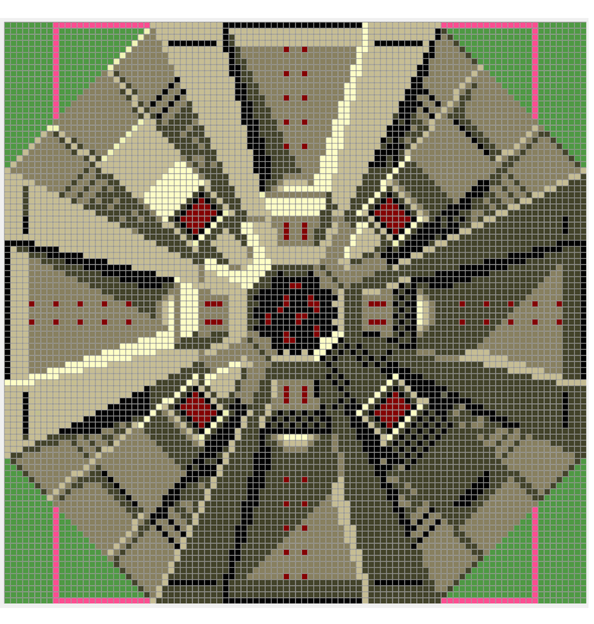
0x7C	0x00	0x04
0xC6	0x00	0x04
0xC0	0x00	0x04
0x7C	0x00	0x04
0x06	0x00	0x04
0xC6	0x00	0x04
0x7C	0x00	0x04
0x00	0x80	0x01
0x00	0xE0	0x01
0x00	0xF8	0x01
0x00	0xFE	0x01
0x00	0xF8	0x01
0x00	0xE0	0x01
0x00	0x80	0x01
0x01	0xC0	0x01
0x01	0xC0	0x01

スペシャル旗

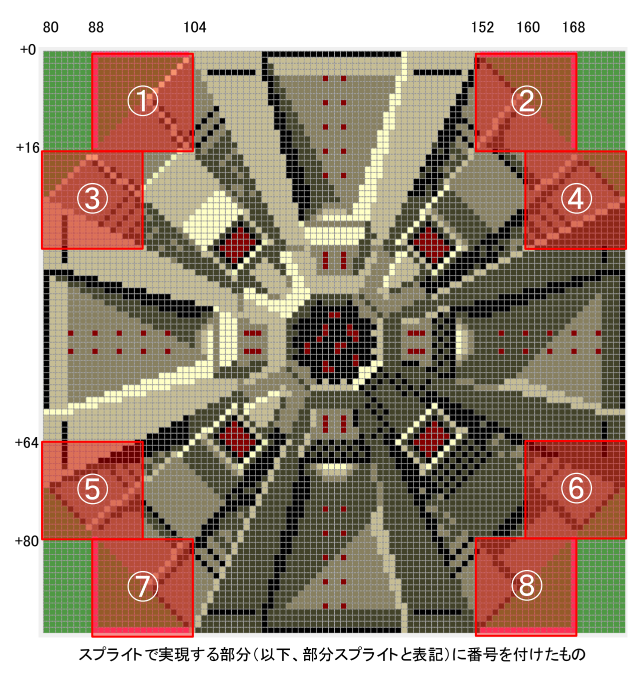
【巨大キャラ】



とりあえず自由に描いてみた版



少し色数を調整してみた版



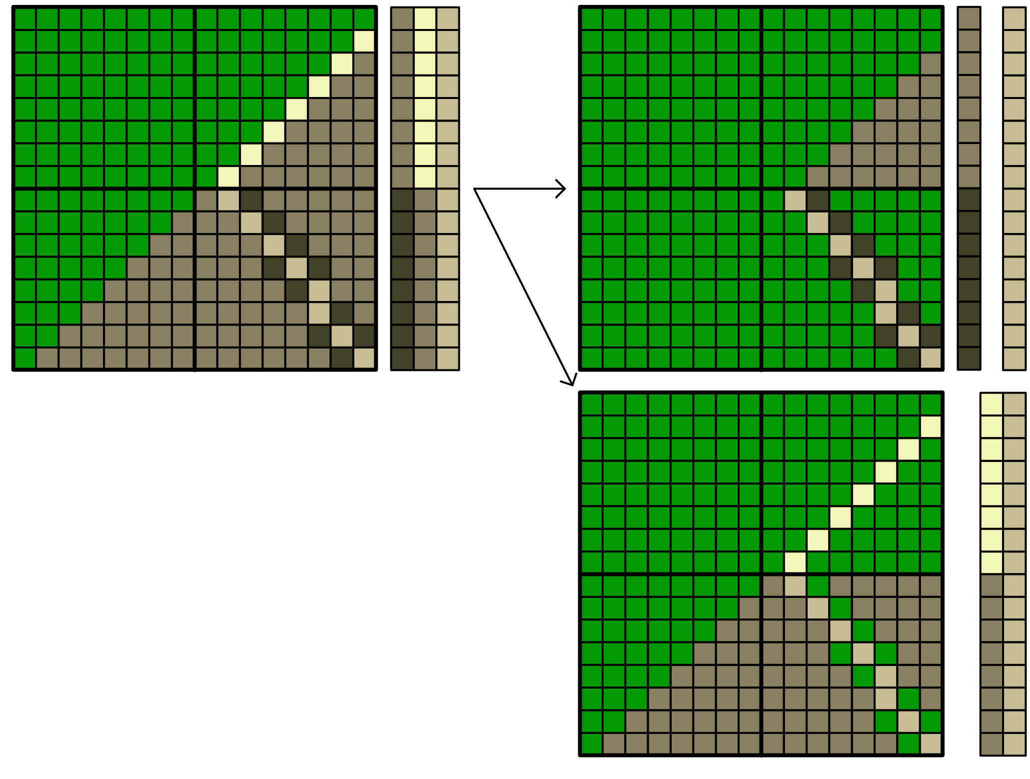
スプライトで実現する部分（以下、部分スプライトと表記）に番号を付けたもの

スプライト 1ライン内に使用しているpaletteの組み合わせ

第1色	第2色	OR色	
■ 0b1101	■ 0b1110	■ 0b1111	■■■
■ 0b1011	■ 0b1101	■ 0b1111	■■■
■ 0b1100	■ 0b1011	■ 0b1111	■■■

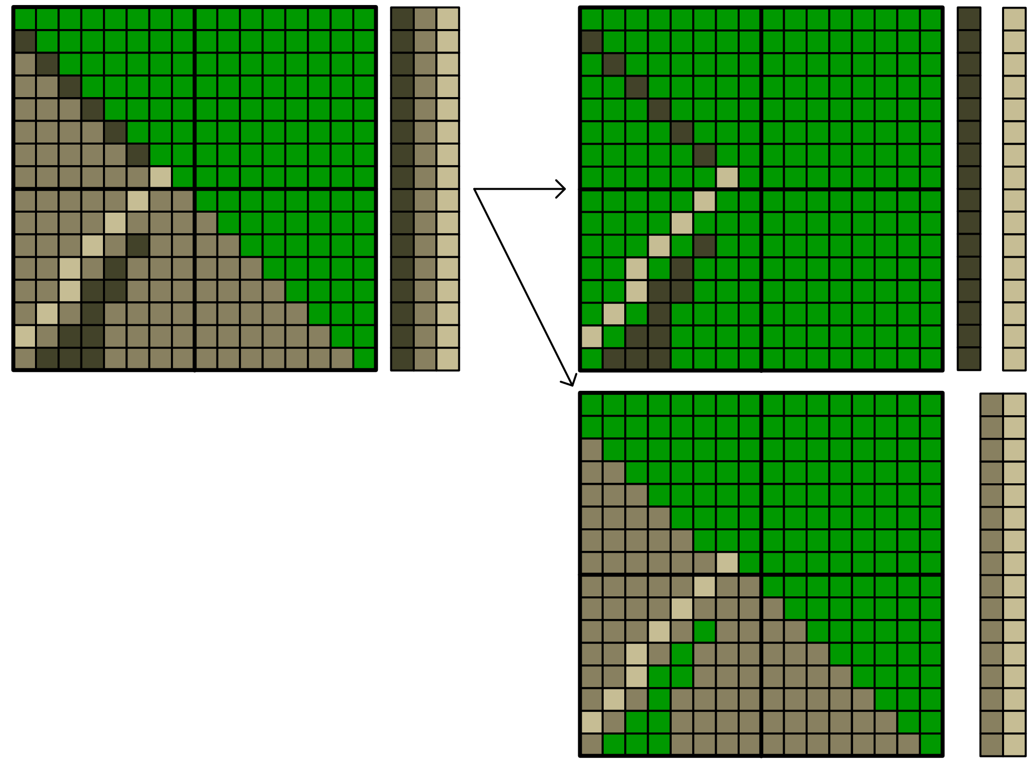
■	0b0000	0（透明）
■	0b1011	11
■	0b1100	12
■	0b1101	13
■	0b1110	14
■	0b1111	15

【部分スプライト①】



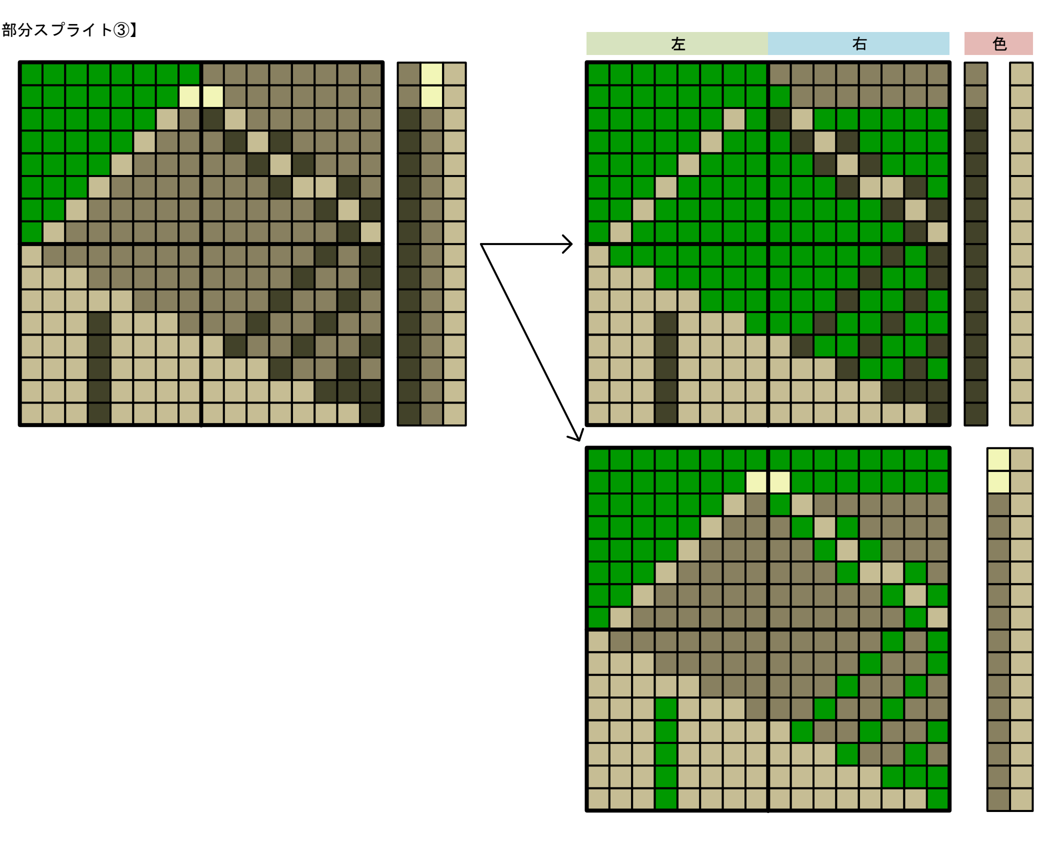
左	右	色	
0x00	0x00	0x0D	Pattern#8
0x00	0x00	0x0D	
0x00	0x01	0x0D	
0x00	0x03	0x0D	
0x00	0x07	0x0D	
0x00	0x0F	0x0D	
0x00	0x1F	0x0D	
0x00	0x3F	0x0D	
0x00	0x60	0x0B	
0x00	0x30	0x0B	
0x00	0x18	0x0B	
0x00	0x1C	0x0B	
0x00	0x0C	0x0B	
0x00	0x06	0x0B	
0x00	0x07	0x0B	
0x00	0x03	0x0B	
0x00	0x00	0x4E	Pattern#12
0x00	0x01	0x4E	
0x00	0x02	0x4E	
0x00	0x04	0x4E	
0x00	0x08	0x4E	
0x00	0x10	0x4E	
0x00	0x20	0x4E	
0x00	0x40	0x4E	
0x00	0xDF	0x4D	
0x01	0xEF	0x4D	
0x03	0xF7	0x4D	
0x07	0xEB	0x4D	
0x0F	0xF7	0x4D	
0x1F	0xFD	0x4D	
0x3F	0xFA	0x4D	
0x7F	0xFD	0x4D	

【部分スプライト②】



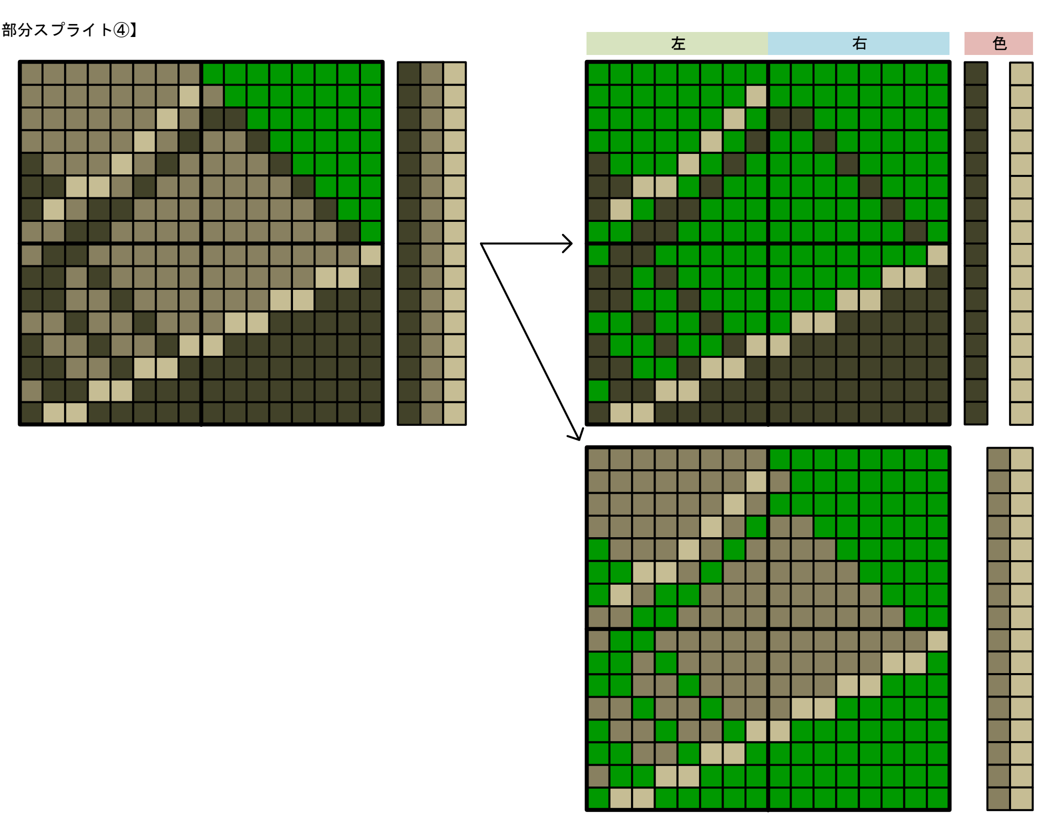
左	右	色	
0x00	0x00	0x0B	Pattern#16
0x80	0x00	0x0B	
0x40	0x00	0x0B	
0x20	0x00	0x0B	
0x10	0x00	0x0B	
0x08	0x00	0x0B	
0x04	0x00	0x0B	
0x02	0x00	0x0B	
0x04	0x00	0x0B	
0x08	0x00	0x0B	
0x14	0x00	0x0B	
0x28	0x00	0x0B	
0x38	0x00	0x0B	
0x50	0x00	0x0B	
0xB0	0x00	0x0B	
0x70	0x00	0x0B	
0x00	0x00	0x4D	Pattern#20
0x00	0x00	0x4D	
0x80	0x00	0x4D	
0xC0	0x00	0x4D	
0xE0	0x00	0x4D	
0xF0	0x00	0x4D	
0xF8	0x00	0x4D	
0xFE	0x00	0x4D	
0xFF	0x00	0x4D	
0xFF	0x80	0x4D	
0xFB	0xC0	0x4D	
0xF7	0xE0	0x4D	
0xE7	0xF0	0x4D	
0xEF	0xF8	0x4D	
0xCF	0xFC	0x4D	
0x8F	0xFE	0x4D	

【部分スプライト③】



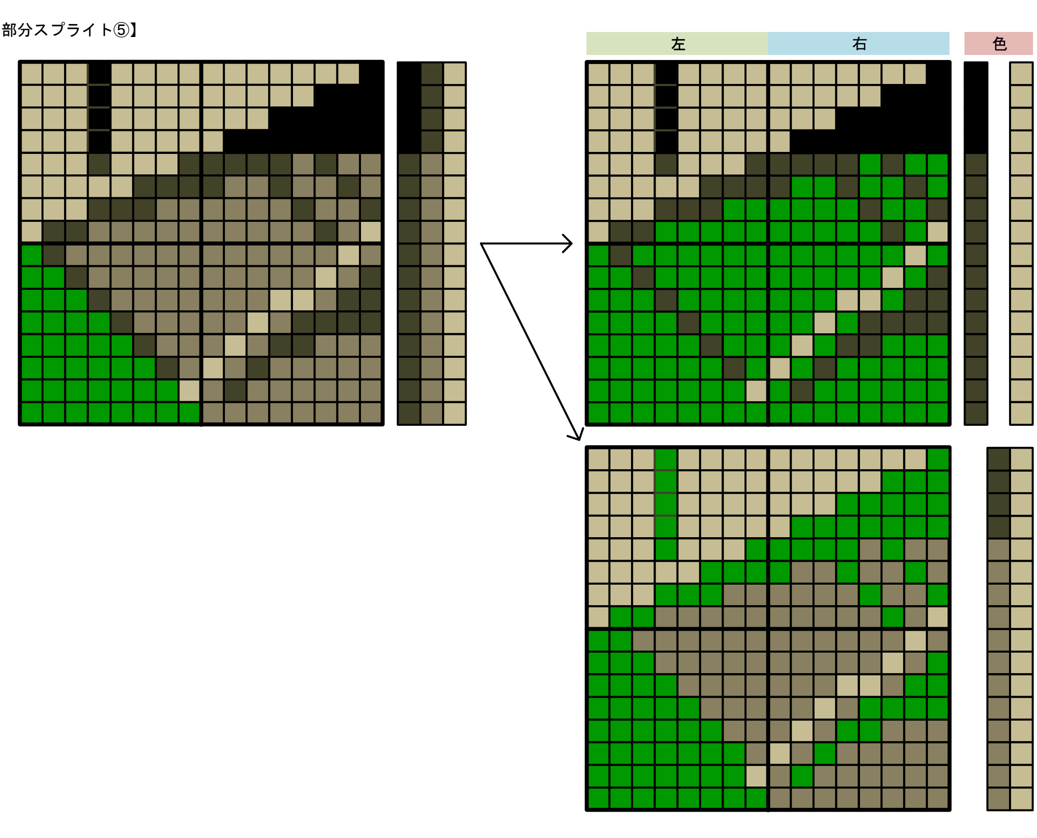
左	右	色	
0x00	0xFF	0x0D	Pattern#24
0x00	0x7F	0x0D	
0x02	0xC0	0x0B	
0x04	0x70	0x0B	
0x08	0x38	0x0B	
0x10	0x1E	0x0B	
0x20	0x07	0x0B	
0x40	0x03	0x0B	
0x80	0x05	0x0B	
0xE0	0x09	0x0B	
0xF8	0x12	0x0B	
0xFE	0x24	0x0B	
0xFF	0xC9	0x0B	
0xFF	0xF2	0x0B	
0xFF	0xFF	0x0B	
0xFF	0xFF	0x0B	
0x00	0x00	0x4E	Pattern#28
0x01	0x80	0x4E	
0x03	0x7F	0x4D	
0x07	0xAF	0x4D	
0x0F	0xD7	0x4D	
0x1F	0xED	0x4D	
0x3F	0xFA	0x4D	
0x7F	0xFD	0x4D	
0xFF	0xFA	0x4D	
0xFF	0xF6	0x4D	
0xFF	0xED	0x4D	
0xEF	0xDB	0x4D	
0xEF	0xB6	0x4D	
0xEF	0xED	0x4D	
0xEF	0xF8	0x4D	
0xEF	0xFE	0x4D	

【部分スプライト④】



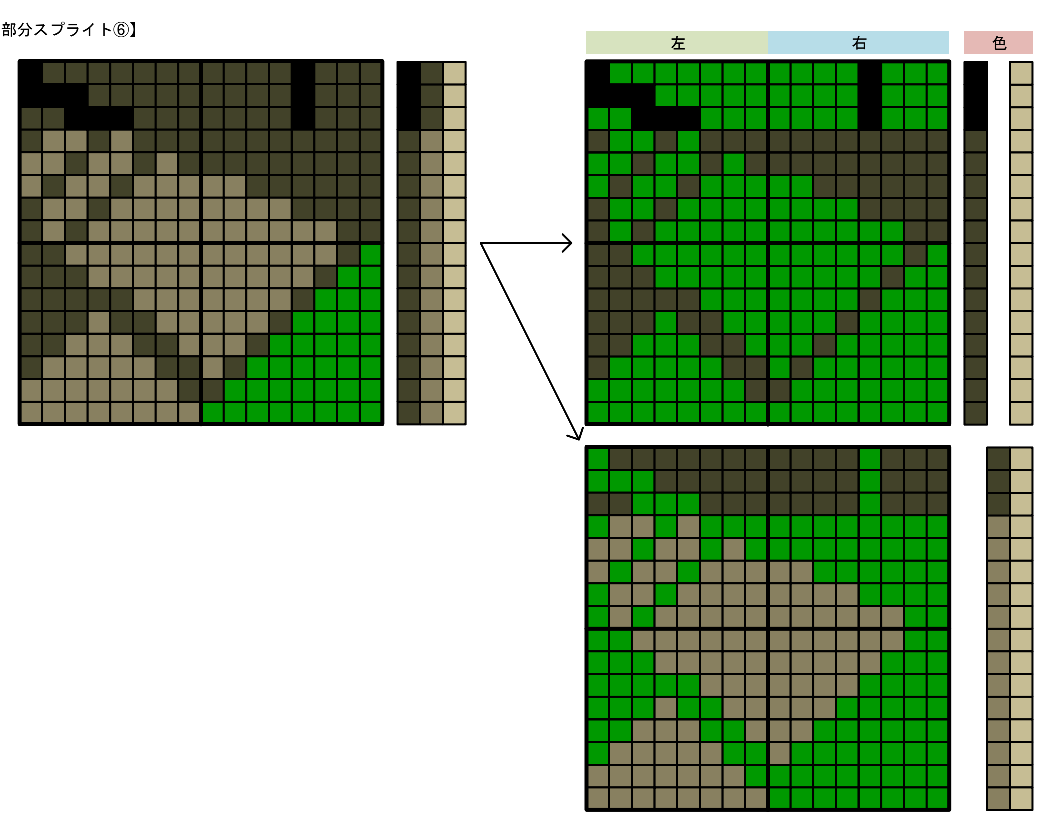
左	右	色	
0x00	0x00	0x0B	Pattern#32
0x01	0x00	0x0B	
0x02	0xC0	0x0B	
0x05	0x20	0x0B	
0x8A	0x10	0x0B	
0xF4	0x08	0x0B	
0xD8	0x04	0x0B	
0x30	0x02	0x0B	
0x60	0x01	0x0B	
0xD0	0x07	0x0B	
0xC8	0x1F	0x0B	
0x24	0x7F	0x0B	
0x93	0xFF	0x0B	
0xCF	0xFF	0x0B	
0x7F	0xFF	0x0B	
0xFF	0xFF	0x0B	
0xFF	0x00	0x4D	Pattern#36
0xFF	0x80	0x4D	
0xFF	0x00	0x4D	
0xFE	0xC0	0x4D	
0x7D	0xE0	0x4D	
0x3B	0xF0	0x4D	
0x67	0xF8	0x4D	
0xCF	0xFC	0x4D	
0x9F	0xFF	0x4D	
0x2F	0xFE	0x4D	
0x37	0xF8	0x4D	
0xDB	0xE0	0x4D	
0x6D	0x80	0x4D	
0x36	0x00	0x4D	
0x98	0x00	0x4D	
0x60	0x00	0x4D	

【部分スプライト⑤】



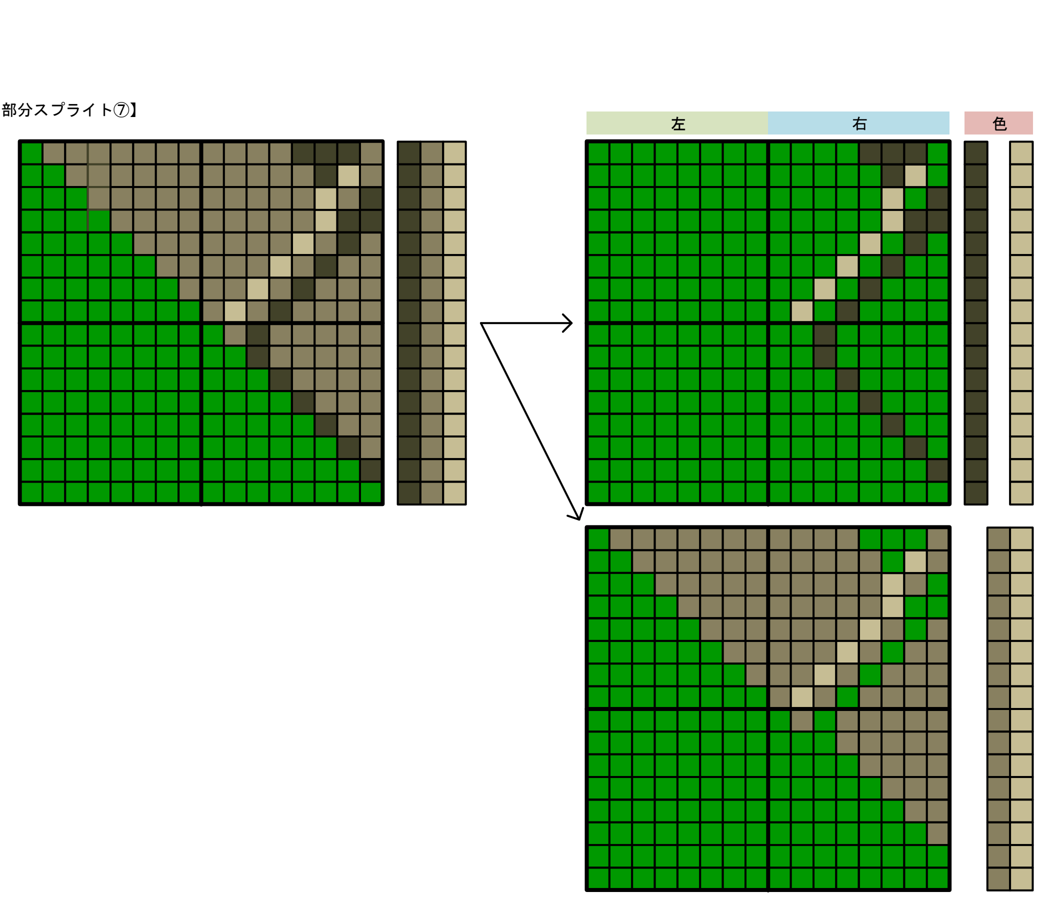
左	右	色	
0xFF	0xFF	0x0C	Pattern#40
0xFF	0xFF	0x0C	
0xFF	0xFF	0x0C	
0xFF	0xFF	0x0C	
0xFF	0xF4	0x0B	
0xFF	0x92	0x0B	
0xFC	0x09	0x0B	
0xE0	0x05	0x0B	
0x40	0x02	0x0B	
0x20	0x05	0x0B	
0x10	0x1B	0x0B	
0x08	0x2F	0x0B	
0x04	0x58	0x0B	
0x02	0xA0	0x0B	
0x01	0x40	0x0B	
0x00	0x00	0x0B	
0xEF	0xFE	0x4B	Pattern#44
0xEF	0xF8	0x4B	
0xEF	0xE0	0x4B	
0xEF	0x80	0x4B	
0xEE	0x0B	0x4D	
0xF8	0x6D	0x4D	
0xE3	0xF6	0x4D	
0x9F	0xFB	0x4D	
0x3F	0xFF	0x4D	
0x1F	0xFE	0x4D	
0x0F	0xFC	0x4D	
0x07	0xF0	0x4D	
0x03	0xE7	0x4D	
0x01	0xDF	0x4D	
0x01	0xBF	0x4D	
0x00	0xFF	0x4D	

【部分スプライト⑥】



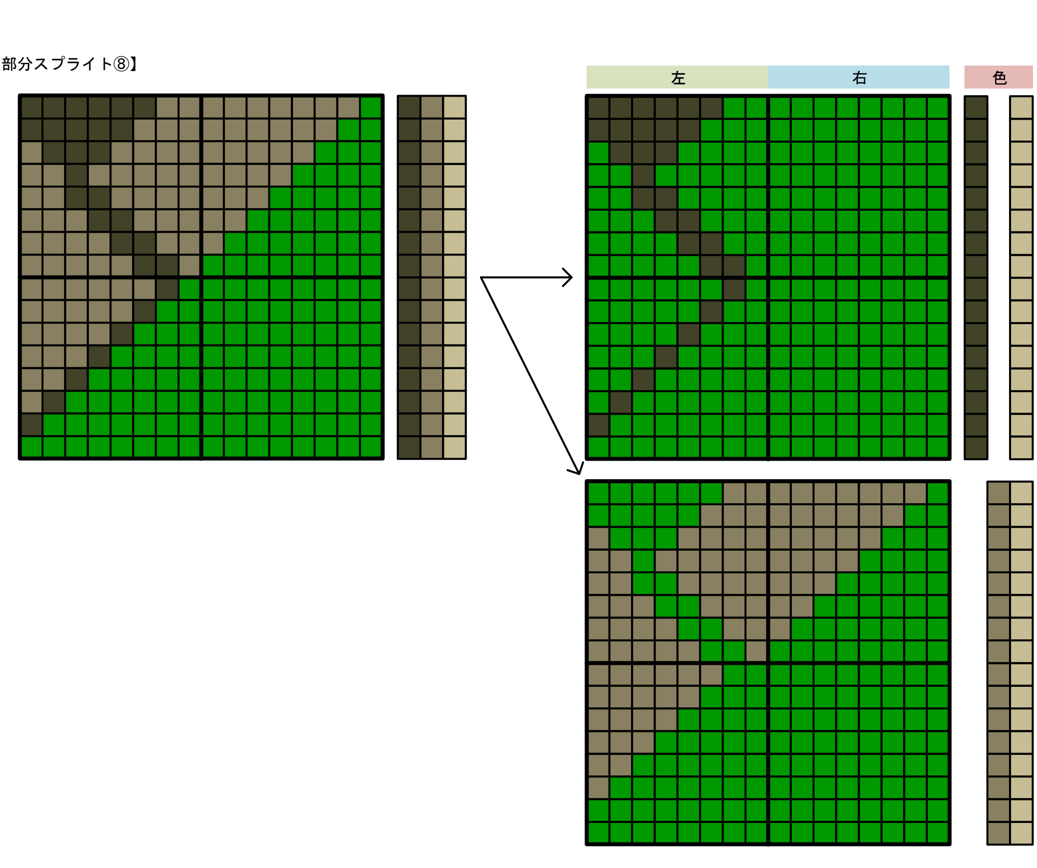
左	右	色	
0x80	0x08	0x0C	Pattern#48
0xE0	0x08	0x0C	
0x38	0x08	0x0C	
0x97	0xFF	0x0B	
0x25	0xFF	0x0B	
0x48	0x3F	0x0B	
0x90	0x0F	0x0B	
0xA0	0x03	0x0B	
0xC0	0x02	0x0B	
0xE0	0x04	0x0B	
0xF8	0x08	0x0B	
0xEC	0x10	0x0B	
0xC6	0x20	0x0B	
0x83	0x40	0x0B	
0x01	0x80	0x0B	
0x00	0x00	0x0B	
0x7F	0xF7	0x4B	Pattern#52
0x1F	0xF7	0x4B	
0xC7	0xF7	0x4B	
0x68	0x00	0x4D	
0xDA	0x00	0x4D	
0xB7	0xC0	0x4D	
0x6F	0xF0	0x4D	
0x5F	0xFC	0x4D	
0x3F	0xFC	0x4D	
0x1F	0xF8	0x4D	
0x07	0xF0	0x4D	
0x13	0xE0	0x4D	
0x39	0xC0	0x4D	
0x7C	0x80	0x4D	
0xFE	0x00	0x4D	
0xFF	0x00	0x4D	

【部分スプライト⑦】



左	右	色	
0x00	0x0E	0x0B	Pattern#56
0x00	0x06	0x0B	
0x00	0x05	0x0B	
0x00	0x07	0x0B	
0x00	0x0A	0x0B	
0x00	0x14	0x0B	
0x00	0x28	0x0B	
0x00	0x50	0x0B	
0x00	0x20	0x0B	
0x00	0x20	0x0B	
0x00	0x10	0x0B	
0x00	0x08	0x0B	
0x00	0x04	0x0B	
0x00	0x02	0x0B	
0x00	0x01	0x0B	
0x00	0x00	0x0B	
0x7F	0xF1	0x4D	Pattern#60
0x3F	0xFB	0x4D	
0x1F	0xFE	0x4D	
0x0F	0xFC	0x4D	
0x07	0xFD	0x4D	
0x03	0xFB	0x4D	
0x01	0xF7	0x4D	
0x00	0xEF	0x4D	
0x00	0x5F	0x4D	
0x00	0x1F	0x4D	
0x00	0x0F	0x4D	
0x00	0x07	0x4D	
0x00	0x03	0x4D	
0x00	0x01	0x4D	
0x00	0x00	0x4D	
0x00	0x00	0x4D	

【部分スプライト⑧】



左	右	色	
0xFC	0x00	0x0B	Pattern#64
0xF8	0x00	0x0B	
0x70	0x00	0x0B	
0x20	0x00	0x0B	
0x30	0x00	0x0B	
0x18	0x00	0x0B	
0x0C	0x00	0x0B	
0x06	0x00	0x0B	
0x02	0x00	0x0B	
0x04	0x00	0x0B	
0x08	0x00	0x0B	
0x10	0x00	0x0B	
0x20	0x00	0x0B	
0x40	0x00	0x0B	
0x80	0x00	0x0B	
0x00	0x00	0x0B	
0x03	0xFE	0x4D	Pattern#68
0x07	0xFC	0x4D	
0x8F	0xF8	0x4D	
0xDF	0xF0	0x4D	
0xCF	0xE0	0x4D	
0xE7	0xC0	0x4D	
0xF3	0x80	0x4D	
0xF9	0x00	0x4D	
0xFC	0x00	0x4D	
0xF8	0x00	0x4D	
0xF0	0x00	0x4D	
0xE0	0x00	0x4D	
0xC0	0x00	0x4D	
0x80	0x00	0x4D	
0x00	0x00	0x4D	
0x00	0x00	0x4D	

【カラーパレット】

🟩	0b0000	0 (透明)

背景を緑にするため

🟫	0b1011	11
⬛	0b1100	12
🟫	0b1101	13
🟨	0b1110	14
🟫	0b1111	15

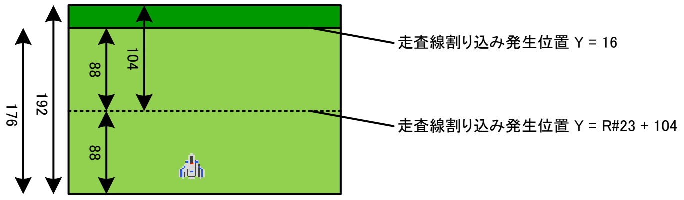
巨大キャラの都合で決まるもの

🟥	0b0100	4
🟦	0b0110	6
⬜	0b0111	7
⬛	0b0011	3

自機の都合で決まるもの

🟩	0b0000	0 (透明)
🟨	0b0001	1
🟨	0b0010	2
⬛	0b0011	3
🟥	0b0100	4
⬛	0b0101	5
🟦	0b0110	6
⬜	0b0111	7
🟩	0b0101	8
🟦	0b0110	9
⬜	0b0111	10
🟫	0b1011	11
⬛	0b1100	12
🟫	0b1101	13
🟨	0b1110	14
🟫	0b1111	15

【Sprite Doubler】



画面表示のイメージ

Sprite Attribute Table を2つ用意する。
　0xF200 .. 1つ目
　0xF600 .. 2つ目

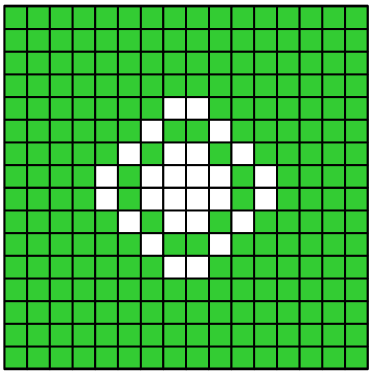
H.TIMI の中で Sprite Attribute Table を1つ目に設定する。
2回目の走査線割り込みの中で Sprite Attribute Table を2つ目に設定する。

継ぎ目の部分にまたがるスプライトも表示できないといけないので、
またがるスプライトは、1つ目と2つ目の両方で配置されなければならない。
Y = 72 ～ 88 のスプライトがまたがるスプライトである。
それより上は1つ目のみ、下は2つ目のみに配置する。
1つ目と2つ目は、それぞれ 32個のスプライトを表示できるので、
最大で 64個同時表示できるようになる。

走査線割り込み回数を増やせば、96個でも128個でも表示できるようになるが、
水平に並べられるのが 8個という制約は変わらないため、あまり増やしすぎても意味が無い。

自機 2枚
巨大キャラ 16枚
とすれば、残りは 64 − (2 + 16) = 46枚。

敵弾は、46個表示することにする。



左	右	色1	色2
0x00	0x00	0x0A	0x06
0x00	0x00	0x0A	0x06
0x00	0x00	0x0A	0x06
0x00	0x00	0x0A	0x06
0x01	0x80	0x0A	0x06
0x02	0x40	0x0A	0x06
0x05	0xA0	0x0A	0x06
0x0B	0xD0	0x0A	0x06
0x0B	0xD0	0x0A	0x06
0x05	0xA0	0x0A	0x06
0x02	0x40	0x0A	0x06
0x01	0x80	0x0A	0x06
0x00	0x00	0x0A	0x06
0x00	0x00	0x0A	0x06
0x00	0x00	0x0A	0x06
0x00	0x00	0x0A	0x06

敵弾

Sprite Doubler には、下記のサブルーチンを用意する。

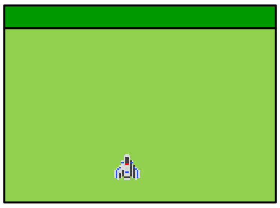
sd_initialize
　これからスプライトの配置を開始する前に実行する初期化ルーチン。

sd_put_sprite_pair
　d .... X座標
　e .... Y座標
　b .... パターンテーブルの番号
　hl ... カラーテーブルの座標
　2枚重ねのスプライトを配置する

sd_put_sprite_single
　d .... X座標
　e .... Y座標
　b .... パターンテーブルの番号
　hl ... カラーテーブルの座標
　単独のスプライトを配置する

sd_finalize
　現フレームのスプライト配置を終了する。
　このルーチンは、ゴミスプライトが表示されないようにケアをする。

【背景の描画】



8x8ドットのパーツを水平32個並べて描画する。
連続で描画すると、ほかの処理に影響を及ぼすので、
1フレームあたり 2個。16フレームかけて描画する。
2フレームに1回 1ドット垂直スクロールするため、16フレームで 8ドットスクロールして、次のラインに
対応可能。

しかしながら、実際に描画するのは page0 と page2 の両方になる。
実質 4個のパーツを描画することになる。

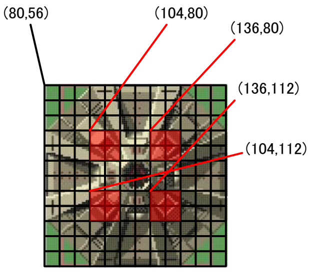
```
vscroll =   0, update_phase = 0 → parts#0,  #1  を Y = 0 に描画。
vscroll =   0, update_phase = 1 → parts#2,  #3  を Y = 0 に描画。
vscroll = 255, update_phase = 0 → parts#4,  #5  を Y = 0 に描画。
vscroll = 255, update_phase = 1 → parts#6,  #7  を Y = 0 に描画。
vscroll = 254, update_phase = 0 → parts#8,  #9  を Y = 0 に描画。
vscroll = 254, update_phase = 1 → parts#10, #11 を Y = 0 に描画。
vscroll = 253, update_phase = 0 → parts#12, #13 を Y = 0 に描画。
vscroll = 253, update_phase = 1 → parts#14, #15 を Y = 0 に描画。
vscroll = 252, update_phase = 0 → parts#16, #17 を Y = 0 に描画。
vscroll = 252, update_phase = 1 → parts#18, #19 を Y = 0 に描画。
vscroll = 251, update_phase = 0 → parts#20, #21 を Y = 0 に描画。
vscroll = 251, update_phase = 1 → parts#22, #23 を Y = 0 に描画。
vscroll = 250, update_phase = 0 → parts#24, #25 を Y = 0 に描画。
vscroll = 250, update_phase = 1 → parts#26, #27 を Y = 0 に描画。
vscroll = 249, update_phase = 0 → parts#28, #29 を Y = 0 に描画。
vscroll = 249, update_phase = 1 → parts#30, #31 を Y = 0 に描画。
vscroll = 248, update_phase = 0 → parts#0,  #1  を Y = 248 に描画。
```

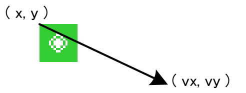
以後、繰り返し。

【各種更新タイミング】

```
vscroll     : 現在の R#23 の値
vscroll_next: 次の R#23 の値
bigchar_y   : 巨大キャラの次の表示位置

update_phase = 0
  表示画面: Page 0
  描画画面: Page 2
  draw_page    = 2
  vscroll      = N
  vscroll_sp   = N
  vscroll_next = N - 1
  bigchar_y    = M
  SpriteAttrib: page 1
  SpriteDoubler更新対象: page 3

update_phase = 1
  表示画面: Page 0
  描画画面: Page 2
  draw_page    = 2
  vscroll      = N
  vscroll_sp   = N - 1
  vscroll_next = N - 1
  bigchar_y    = M
  SpriteAttrib: page 3
  SpriteDoubler更新対象: page 1

update_phase = 0
  表示画面: Page 2
  描画画面: Page 0
  draw_page    = 0
  vscroll      = N - 1
  vscroll_sp   = N - 1
  vscroll_next = N - 2
  bigchar_y    = M + 1
  SpriteAttrib: page 1
  SpriteDoubler更新対象: page 3

update_phase = 1
  表示画面: Page 2
  描画画面: Page 0
  draw_page    = 0
  vscroll      = N - 1
  vscroll_sp   = N - 2
  vscroll_next = N - 2
  bigchar_y    = M + 1
  SpriteAttrib: page 3
  SpriteDoubler更新対象: page 1
```

【SHOTの発射位置】





ベクトル値 (vx, vy) の成分 vx, vy は、それぞれ 2byte の符号付き小数である。
2byte のうち、小数部 8bit とし、絶対値は 1.0 未満である。
従って上位 1byte は、0x00 か 0xFF のどちらかとなる。

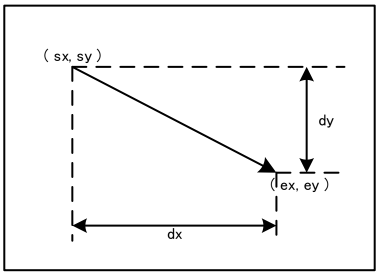
座標値 (x, y) の成分 x, y も、それぞれ 2byte の符号付き小数である。
2byte のうち、小数部 8bit とする。

x	0x00	上位8bit	下位8bit	
vx	上位8bit	上位8bit	下位8bit	(+
new x	上位8bit	中位8bit	下位8bit	

この加算の結果、上位8bit には、0xFF, 0x00, 0x01 のいずれかが現れる。
0xFF か 0x01 の場合、画面外に出たことになる。

```
; ahl = x
xor   a, a
ld    hl, [x]
; bbc = vx
ld    bc, [vx]
; ahl = ahl + bbc
add   hl, bc
adc   a, b
; check
jp    nz, outside
```

【自機に向かってくるタイプのSHOT】



```
( sx, sy ) は、発射開始位置
( ex, ey ) は、発射時の自機の位置

dx = | ex - sx |
dy = | ey - sy |

if( dx > dy ) {
    deno = dx
    nume = dy
}
else {
    deno = dy
    nume = dx
}

for( i = 0; i < 7; i++ ) {
    if( deno & 0x80 ) {
        break;
    }
    deno <<= 1;
    nume <<= 1;
}
if( dx >= dy ) {
    vx = sign( ex - sx ) * deno;
    vy = sign( ey - sy ) * nume;
}
else {
    vx = sign( ex - sx ) * nume;
    vy = sign( ey - sy ) * deno;
}
```

ex = 0〜255
sx = 0〜255

ex − dx = −255〜255

例えば、sx = 104; ex = 240; の場合
ex − dx = 104 − 240 = −136

−136 & 255 = 120

【TITLE】