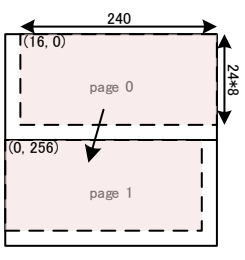
MSX2 SCREEN5 における横スクロール実現方法

【ポイント】
(1) VDPコマンドによる画面のコピーを実行中にも VDPレジスタへの書き込みは可能
(2) スプライトONの時の VDPコマンドは遅いが、裏画面でコピーしていればコピーの過程が見えることはない
(3) VDP R#18 を利用すれば -8～+7 の範囲でドット単位の表示位置調整が可能
(4) VDPコマンド実行中に書き換えると VDPコマンドの動作が乱れる VDPレジスタがある(少なくとも R#8 や R#18)

【実現方法】
[1] VDPコマンドの HMMM を使って 表画面から裏画面へ 16ドットずらした位置へコピーを実行する
[2] VDP R#18 を使って -8～+7 の範囲のドット単位スクロールを実行する
[3] HMMM が終わった頃に 裏画面の端にできている隙間に必要なパーツを敷き詰める (これも HMMM)
[4] 最後に 表画面と裏画面 を入れ替える (VDP R#2)

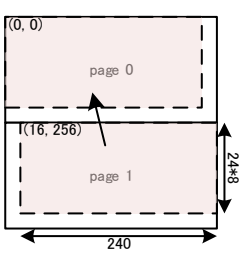
【具体的な設定方法】
[1] VDPコマンドの HMMM を使って 表画面から裏画面へ 16ドットずらした位置へコピーを実行する
→ しかし、下図のピンクの領域を一気に転送すると 16.6msec 以上かかってしまうため R#18 書き換えタイミングとバッティングする。
→ これを回避するために、ピンクの領域を 垂直24dot単位に分割して、8回に分けてコピーする。
240x24 のコピーであれば1回 16.6msec に収まるため、毎フレームの R#18 書き換えとバッティングしないように実施可能である。
x32 の方が計算しやすいが、240x32 だと 16.6msec に収まらない。

(a) 表画面が page 0, 裏画面が page 1 の場合



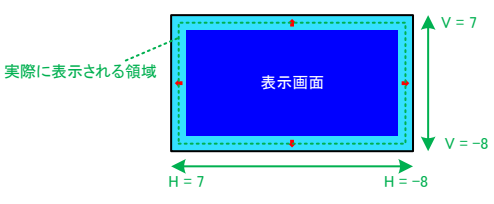

```
R#32 (SX low) : 16
R#33 (SX hi ) : 0
R#34 (SY low) : N * 24
R#35 (SY hi ) : 0
R#36 (DX low) : 0
R#37 (DX hi ) : 0
R#38 (DY low) : N * 24
R#39 (DY hi ) : 1
R#40 (NX low) : 240
R#41 (NX hi ) : 0
R#42 (NY low) : 32
R#43 (NY hi ) : 0
R#44          : 0
R#45 (ARG   ) : 0
R#46 (CMR   ) : 0b11010000
```

(b) 表画面が page 1, 裏画面が page 0 の場合




```
R#32 (SX low) : 16
R#33 (SX hi ) : 0
R#34 (SY low) : N * 24
R#35 (SY hi ) : 1
R#36 (DX low) : 0
R#37 (DX hi ) : 0
R#38 (DY low) : N * 24
R#39 (DY hi ) : 0
R#40 (NX low) : 240
R#41 (NX hi ) : 0
R#42 (NY low) : 32
R#43 (NY hi ) : 0
R#44          : 0
R#45 (ARG   ) : 0
R#46 (CMR   ) : 0b11010000
```

[2] VDP R#18 を使って -8～+7 の範囲のドット単位スクロールを実行する



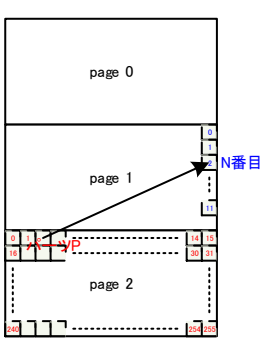
水平スクロールとして利用するので、プログラム上の循環ステートを phase0～15 とすれば、下記の値を設定する。

phase	設定したいH	R#18に設定する値
0	-8	0x08
1	-7	0x09
2	-6	0x0A
3	-5	0x0B
4	-4	0x0C
5	-3	0x0D
6	-2	0x0E
7	-1	0x0F
8	0	0x00
9	1	0x01
10	2	0x02
11	3	0x03
12	4	0x04
13	5	0x05
14	6	0x06
15	7	0x07

[3] HMMM が終わった頃に 裏画面の端にできている隙間に必要なパーツを敷き詰める (これも HMMM)

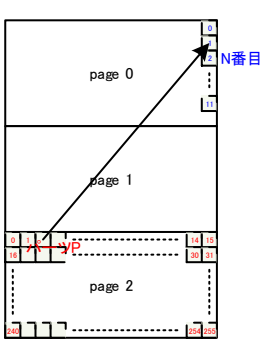
16[dot] x 16[dot] のパーツ 256種類を page2 に配置しておき、これを隙間に詰めていく。パーツ番号は P とする。
詰める隙間の上から何番目かを示す値は N とする。

(a) 表画面が page 0, 裏画面が page 1 の場合



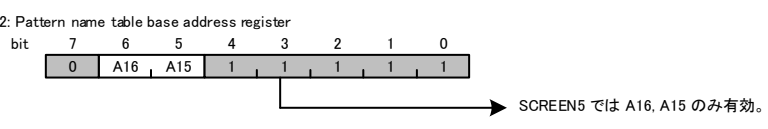

```
R#32 (SX low) : P << 4
R#33 (SX hi ) : 0
R#34 (SY low) : P & 0xF0
R#35 (SY hi ) : 2
R#36 (DX low) : 240
R#37 (DX hi ) : 0
R#38 (DY low) : N << 4
R#39 (DY hi ) : 1
R#40 (NX low) : 16
R#41 (NX hi ) : 0
R#42 (NY low) : 16
R#43 (NY hi ) : 0
R#44          : 0
R#45 (ARG   ) : 0
R#46 (CMR   ) : 0b11010000
```

(b) 表画面が page 1, 裏画面が page 0 の場合




```
R#32 (SX low) : P << 4
R#33 (SX hi ) : 0
R#34 (SY low) : P & 0xF0
R#35 (SY hi ) : 2
R#36 (DX low) : 240
R#37 (DX hi ) : 0
R#38 (DY low) : N << 4
R#39 (DY hi ) : 0
R#40 (NX low) : 16
R#41 (NX hi ) : 0
R#42 (NY low) : 16
R#43 (NY hi ) : 0
R#44          : 0
R#45 (ARG   ) : 0
R#46 (CMR   ) : 0b11010000
```

[4] 最後に 表画面と裏画面 を入れ替える (VDP R#2)



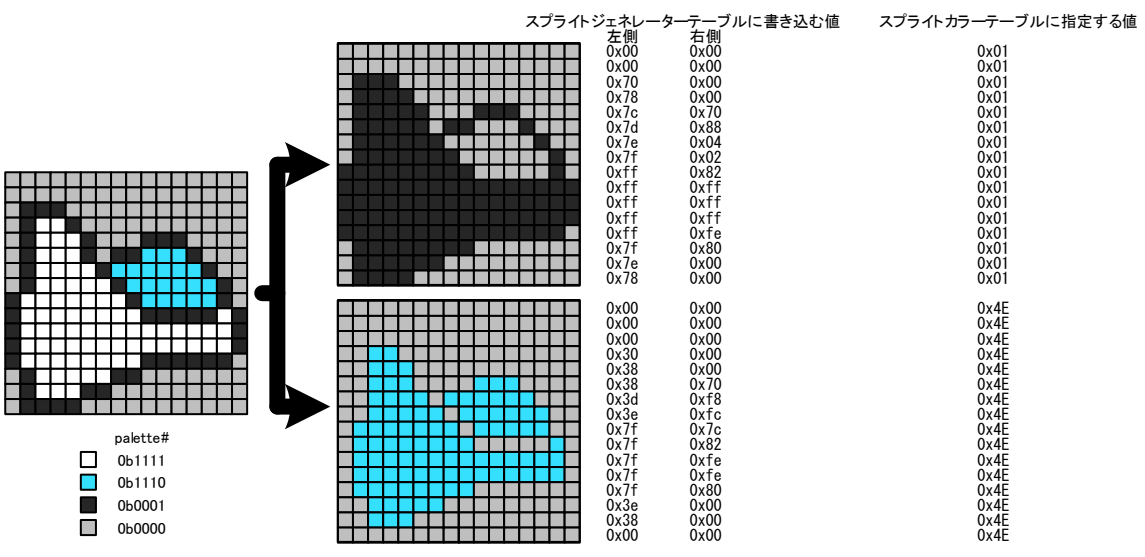
【その他】
[1] スプライトの表示

(a) スプライトの形状パターンをスプライトジェネレータテーブルに書き込んで定義する

スプライトは、ビットパターンによる二値画像で形状定義する。
SCREEN5 では、その1ラインごとにビットパターンが 1 である部分のカラーパレット番号を指定できる。
さらに、カラーテーブルの CCビットが 1 の場合、自分より優先度が高く、最も近い番号のスプライトと重なった部分が、
そのスプライトのカラーパレット番号と OR をとったカラーパレット番号で表示されるようになる。
2枚重ねることによって、透明＋3色の表現が可能になる。(3枚・4枚重ねることもできる。)

スプライトカラーテーブルに設定する値 (ライン単位に指定)

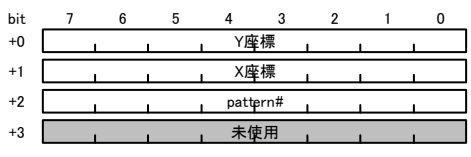
EC: 1にすると左へ32ドットずらす
CC: 1にすると重ね合わせする
IC: 1にすると衝突判定外になる
palette#: カラーパレット番号



スプライトジェネレーターテーブルに書き込む値 左側	右側	スプライトカラーテーブルに指定する値
0x00	0x00	0x01
0x00	0x00	0x01
0x70	0x00	0x01
0x78	0x00	0x01
0x7c	0x70	0x01
0x7d	0x88	0x01
0x7e	0x04	0x01
0x7f	0x02	0x01
0xff	0x82	0x01
0xff	0xff	0x01
0xff	0xff	0x01
0xff	0xff	0x01
0xff	0xfe	0x01
0x7f	0x80	0x01
0x7e	0x00	0x01
0x78	0x00	0x01
0x00	0x00	0x4E
0x00	0x00	0x4E
0x00	0x00	0x4E
0x30	0x00	0x4E
0x38	0x00	0x4E
0x38	0x70	0x4E
0x3d	0xf8	0x4E
0x3e	0xfc	0x4E
0x7f	0x7c	0x4E
0x7f	0x82	0x4E
0x7f	0xfe	0x4E
0x7f	0xfe	0x4E
0x7f	0x80	0x4E
0x3e	0x00	0x4E
0x38	0x00	0x4E
0x00	0x00	0x4E

VRAM上のアドレス　　msx.asmでの定義名
スプライトジェネレータテーブル　0x7800　SC5_SPR_GEN
スプライトアトリビュートテーブル　0x7600　SC5_SPR_COL
スプライトカラーテーブル　0x7400　SC5_SPR_ATTR

スプライトアトリビュートテーブルに設定する値 (スプライト面単位に指定, 連続4byte で一組)



スプライトアトリビュートテーブルのアドレス計算

スプライト面番号を N (0～31) とすると、下記の計算式で求められる。

スプライトアトリビュートテーブル +(N << 2)

スプライトカラーテーブルのアドレス計算

スプライト面番号を N (0～31) とすると、下記の計算式で求められる。

スプライトカラーテーブル +(N << 4)

スプライトジェネレーターテーブルのアドレス計算

pattern# を P (0～255) とすると、下記の計算式で求められる。

スプライトジェネレーターテーブル +(P << 3)

pattern# は、8x8 のパターンにつけられる番号で、スプライトモードが 16x16 の場合には、
アトリビュートテーブルで指定した pattern# の 8x8 から連続する 4 つが使われる。

[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

[3] 下準備

スクリーンモードの変更 (SCREEN5 にする、スプライトを 16x16 にする等) や、背景パーツを page2 へ読み込む処理は、BASICでやってしまいます。

```
100 DEFAINT A-Z
110 COLOR 15,0,0:SCREEN 5,1,0
120 SET PAGE 0,2
130 BLOAD "BACK.SC5",S
```

[4] 状態変数

垂直同期割り込みが入るたびに phase をインクリメントする。15 の次は 0 に戻る。
H.TIMI割り込みをフックするのではなく、HALTで割り込みが入って復帰するのを待つ。

phase	R#18に設定する値	16画素ずらし発動	パーツコピー(Nの値)	phase判定条件	R#18に設定する値	スプライトX加算値
0	0x07	✓	-	phase == 0	8	0
1	0x06	✓	-	bit(phase,3) == 0	phase xor 8	1
2	0x05	✓	-	bit(phase,3) == 0	phase xor 8	2
3	0x04	✓	-	bit(phase,3) == 0	phase xor 8	3
4	0x03	✓	-	bit(phase,3) == 0	phase xor 8	4
5	0x02	✓	-	bit(phase,3) == 0	phase xor 8	5
6	0x01	✓	-	bit(phase,3) == 0	phase xor 8	6
7	0x00	✓	-	bit(phase,3) == 0	phase xor 8	7
8	0x0F	-	0, 1	bit(phase,3) == 1	phase xor 8	8
9	0x0E	-	2, 3	bit(phase,3) == 1	phase xor 8	9
10	0x0D	-	4, 5	bit(phase,3) == 1	phase xor 8	10
11	0x0C	-	6, 7	bit(phase,3) == 1	phase xor 8	11
12	0x0B	-	8, 9	bit(phase,3) == 1	phase xor 8	12
13	0x0A	-	10, 11	bit(phase,3) == 1	phase xor 8	13
14	0x09	-	-	bit(phase,3) == 1	phase xor 8	14
15	0x08	-	-	bit(phase,3) == 1	phase xor 8	15

2019年8月12日 t.hara