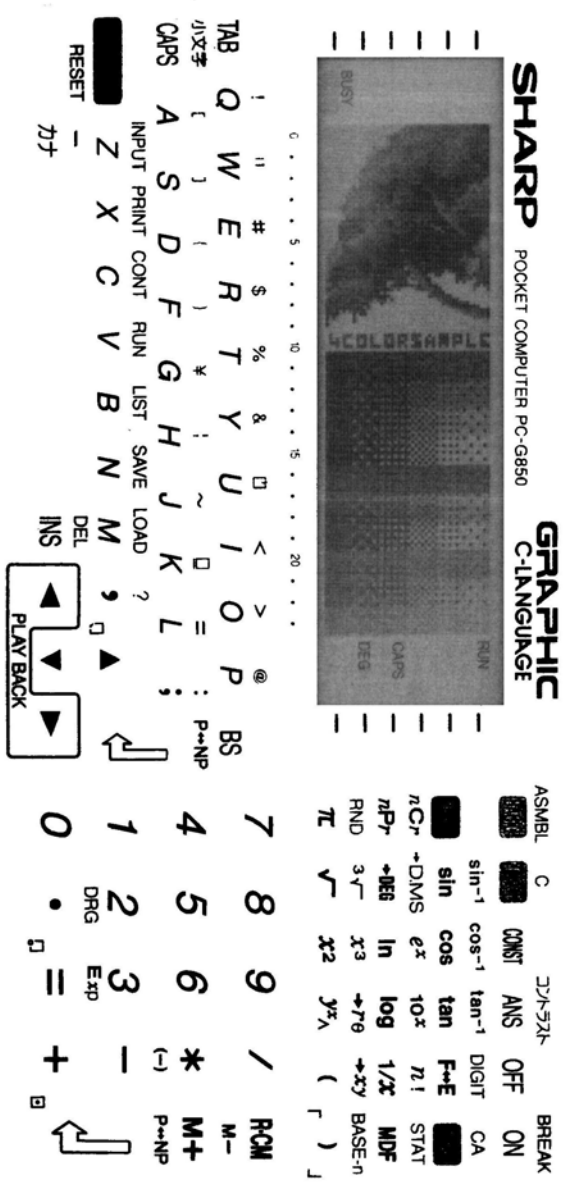
- SHARP 製 ポケットコンピュータ PC-G850 対象 -

ぽけこんのほん２



第１特集

「PC-G850 のマシン依存ルーチンの定石」

第２特集

「中間色のススメ２」

第３特集

「巷で見かけたポケコン」

著：単色ドット絵向上委員会(C66)

～ 前書 ～

どうも。

「あきひ」といいます。はじめまして　or お世話になっております。

前回 C64 にて出展したところ、まさか全部は捌けないだろうと思っていた本がすべて売れてしまいました。しかも昼ちょっと過ぎに。このジャンル「ポケコン」は小さいながらも需要があることを認識させられました。たかだか 20 冊という少ない冊数ではありましたが、手に入らなかった方もいらっしゃったようで、もうしわけないやらなにやら・・・コミケってすごい。

相変わらずポケコン単体としてのサークルとしては私しか参加していない、という現状ではありますが・・・参加しつづける意味が出来ました。私には何が出来るんだろう、私は何が残せるんだろう、と考えると、やはり私はポケットコンピュータに行き着きます。思い出に残すだけではなく、いつまでも残っていてほしいものがあります。ゲームがないなら作りましょう。ハードでわからない事があったら調べましょう。私は、ポケットコンピュータのサークルがいないので自ら参加しました。

アウトプットはインプットされた量が多いほど良い物を生み出す――――

BASIC は無視して完全にマシン語一本に絞った中級者以上向けの本になりましたが、ポケコンを愛する方々の知識の一片となれば幸いです。

～ 目次 ～

企画１：PC-G850 のマシン依存ルーチンの定石

ゲームなどのアプリケーションを作る際、構成されるアルゴリズムはある程度定石があります。処理の仕方が解っていれば、大概のマシンや言語で同じ(ような)動きをするものが作成できます。しかし、マシン毎に異なる動作をせざるをえないものがあります。「画面描画」「キー判定」「同期の取り方」「音の出し方」が代表的なものと思います。今回は「画面描画」「キー判定」について触れていきたいと思います。

さて、PC-G850 等の E200/G800 系のマシンには IOCS(Input Output Control Sysytem)というルーチン群がメーカーから公表されています。これらについては Z80 ポケコン入門(工学社刊)に詳しく掲載されています。Web で情報を得ることも容易に出来ます。このルーチン群には「文字列表示」「ビットパターン表示」「キー入力判定」等、マシン語でゲームを作成するのに必要最低限の機能が用意されています。しかし、それらのルーチンは汎用性がそれなりにあるため、速度を犠牲にしたり、コールした際に要するステップ数が解らなかったり、同時にキー判定が出来なかったりします。まあ、個人的に IOCS でゲームを組むのはイヤだったのですが、とりあえず周りから固めていこうかなーとか思ったのが運の尽きでした。コレまでゲームらしいゲームを組んだことがありません。

＜高速 LCD 描画ルーチンの作成＞

まずは IOCS を使用しないで画面を描画することを考えます。前回の本「ぽけこんのほん１」で画面描画に使用する I/O は個人的な解釈ではありますが解明してあります。手抜きっぽいですが、再度(ちょっと更新して)掲載します。

G850 LCD 主要 I/O マップ

40H(OUT) : LCD コントロール

```
MSB          LSB
□□□□□□□□
[      ]
   +―――――0H:横アドレスの下位 4 ビット
          1H:横アドレスの上位 4 ビット
          BH:縦アドレス
```

上記を踏まえた上で下記のようになる

```
MSB          LSB
□□□□□□□□
００００[      ]
            +―――横アドレスの下位 4 ビット指定

MSB          LSB
□□□■□□□□
０００１[      ]
            +―――横アドレスの上位 4 ビット指定

MSB          LSB
■□■■□□□□
１０１１０[    ]
            +―――縦アドレス(3 ビット)
```

40H(IN) : LCD ステータス

```
MSB          LSB
□□□□□□□□
[ ]
+―――――0:アクセス可能
       1:BUSY
```

41H(OUT) : LCDRAM データ・ライト

```
MSB          LSB
□□□□□□□□
[            ]
      +―――――RAM データ(書き込み後、横アドレスがインクリメントされる)
```

41H(IN) : LCDRAM データ・リード

```
MSB          LSB
□□□□□□□□
[            ]
      +―――――RAM データ(読み込み後、横アドレスがインクリメントされる)
```

画面アドレスイメージ(脇の数字(0～5)が縦アドレス、箱の中の 00H～90H が横アドレス)

```
  ―――――――――――――――――――――――――――――――――
0 |00H|01H|――――――――――――――→|8EH |8FH||90H| 90H は各種ステータス部(前号に詳細を記載)
1 |00H|01H|――――――――――――――→|8EH |8FH||90H|
2 |00H|01H|――――――――――――――→|8EH |8FH||90H|
3 |00H|01H|――――――――――――――→|8EH |8FH||90H|
4 |00H|01H|――――――――――――――→|8EH |8FH||90H|
5 |00H|01H|――――――――――――――→|8EH |8FH||90H|
  ―――――――――――――――――――――――――――――――――
```

6、7 行目もあるが、頭の中のイメージでは画面は 6 行と思っていてよいと個人的に思う。

・・・前回、画面描画関係の I/O が解った後、「41H に出力した後、横アドレスは自動加算されるか?」「横アドレスが加算されるとして、画面外アドレスまで到達した場合はどのような動作をするか?(縦アドレスも自動加算されるか?)」という 2 点から調査を開始しました。
具体的には、初期縦アドレスを 0 に指定、横アドレスを 0 に指定した後、41H に FFH を連続出力させるだけのプログラムを組み、どのようになるかを調べました(調査に使ったプログラムは割愛させていただきます)。なんでいきなりこんなことをやるのかというと、今までの E200/G800 系の画面描画系の I/O がそうなっていたからです。さて、結果としてステータス部を含む 1 行目が真っ黒に塗りつぶされました。これにより

「41H にデータを出力すると横アドレスが自動的に+1 される」
「目に見える画面の外に出たとしても縦アドレスの自動加算は行わない」

ことがわかりました。ここまで解れば最適化した画面描画ルーチンが組めそうです。

試行錯誤のうえ、下記のようなリストを作成しました。これは技術の流布のため、2ch の某掲示板にも投稿したので、見たことがある人はあるでしょう。850V での動作確認報告も受けております。

リスト１：G850 の仮想 VRAM→実 VRAM 転送ルーチン

```
 ORG 0100H              ; 開始アドレスは 100H
 VRMCLS:LD A,00H        ; VRAM クリア(埋め)開始。本当は XOR A
  LD HL,VRAM            ; HL が示すアドレスから
  LD DE,VRAM+1          ; DL が示すアドレスに
  LD BC,863             ; 863 回(=144*6 回)
  LD (HL),A             ;A レジスタで示した値で
  LDIR                  ; 埋め尽くします
 VRMWRT:LD A,(790DH)    ; VRAM 転送開始。実行時縦アドレスを取得
  LD D,A                ; 実行時縦アドレスを保存
  LD E,6                ; 6 回(6 行)ここのルーチンを繰返す
  LD C,41H              ; 出力ポートは 41H
  LD HL,VRAM            ; VRAM の先頭アドレスを読む
 TATEAD:LD A,0B0H       ; 縦アドレス基本値代入
  ADD A,D               ; 実行時縦アドレスを足しこむ
  AND 0B7H              ; 下位 3 ビットが縦アドレス
  OUT (40H),A           ; 縦アドレス指定、出力
  LD A,010H             ; 横アドレス上位 4 ビットを 0 に指定
  OUT (40H),A           ; 横アドレス上位指定、出力
  XOR A                 ; 横アドレス下位 4 ビットを 0 に指定
  OUT (40H),A           ; 横アドレス下位指定、出力
  LD B,144              ; １行ずつ書くため、144 回出力
  OTIR                  ;41H に 144 回 VRAM の値を出力だ！
  LD A,(TATEAD+1)       ; 縦アドレス基本値に+1 する(１行加算)
  INC A                 ; 次は１行下にいくよー
  LD (TATEAD+1),A       ;+1 した値を格納終了
  DEC E                 ;E は書いた行の回数でしたね。ここで-1
  JR NZ,TATEAD          ; 書いた行の回数が 6 回ではないときは TATEAD に戻る
  LD A,0B0H             ; 全部書いたので、縦アドレス規定値を初期値に戻す
  LD (TATEAD+1),A       ; それを格納終了(ここまでで VRAM 描画ルーチン終了)
 STPKEN:IN A,(1FH)      ; ストップキーを検出しますよー(MSB がそれだ！)
  RLCA                  ; １個左にずらして
  RET C                 ; キャリーが発生してたら終了だ！
  JP VRMWRT             ; １回だけ表示するとすぐに終わるので描画の最初に戻ろう
 VRAM:DS 144*6          ; 仮想 VRAM
```

テキストモードで(コメントを省き、行番号を入れつつ)打ち込んでアセンブルモードでアセンブルしてマシン語モニタから G100 と打ち込むと動作します。マシン語領域は 0FFFH あれば十分でしょう。
このリストはサンプルとして作成したものなので VRAM に値が入っておらず、そのまま実行すると何も表示されませんが、2 行目の LD A,00H の部分に好きなパターンを入れるとそのパターンで画面を埋めます。

リストのキモは VRAMWRT～STPKEN の間の処理です。わきのコメントを見れば動きはわかると思います。1 行ずつ何をしているかを追うのもよいでしょう。量が少ないので。「何をしているのかわからない～」という人は修行が足りません。精進してください。

このリスト中では 1 LINE(144 ドット)分を 6 回描画してます。ポケコンの画面が 144x48 だから当然と言えば当然ですが。
しかし、145 ドット目のパラメータ部(RUN とか BATT の部分)は今回は考慮してないので、そこも描画したい方は OTIR のループを一回増やしてください。
高速化を望むのであれば、B レジスタに 144 入れて OTIR する部分を OUTI×144 回とすれば 0.54msec ほど早くなります。算出式は(1/8000000Hz * 144 回 * 5 命令あたり節約サイクル * 6 ライン = 0.00054sec)のような感じです。しかし、約 280 バイト無駄にする価値があるかどうかは不明。よっぽどの事がない限りこのような高速化はいりません。また、6 行分の処理をこのリストでは繰返しをさせていますが、それを繰返させずにべた書きすると刹那にも満たない時間を節約できます。ほぼ無駄ですが。速度を得るには繰返し命令を極限まで減らすのはセオリーですが、メモリ容量を食うのが難点ですよねー。

リストでわかりにくいところと言えば「790DH に保存してある数値を足す」処理だと思います。普段のシステムでは 790DH に現在の行番号が記録されています。ここの数値を足さないと指定した開始行にならないことがあります。ここは IOCS を解析して、その中で同じように処理していた部分を流用しました。具体的には ROM00H 番の 8549H 付近や 8712H 付近です。B7H で AND をとる処理も超重要で、これを入れないと 8 LINE 以降に書こうとしてずれます。

伝統的に LCD 関係の I/O をいじるときは LCDWAIT(IN A,40H した最上位ビット)をチェックし、LCD コントローラに指示が出せるかどうかをチェックするんですが、特に入れないでも問題なく描画できていると思います。

捕捉として仮想 VRAM とは何かについて簡単に触れておきます。
ゲーム等で一番時間を食うのは画面描画です。G850 では 144x48 と言う画面を搭載していますが、それのひとつひとつのドットに対して何らかの処理を行わなくてはならないと考えると、膨大な量の処理が必要になることがわかると思います。ポケコンの BASIC ではよく PRINT 文でキャラクタを描いていますが、そのような処理だとキャラクタ毎にいちいち「消して描く」作業をしなくてはならず大変効率が悪いです。また、画面に描く量が一定でないのでちらつき等が発生します。そこで「画面に描く前にメモリに全部キャラクタを描きこんでから一気に描画」する方式をとります。その画面状態を描いておくメモリ領域を仮想 VRAM と呼んでいます。仮想スクリーンとも言われます。実 VRAM に直で書くには OUT 命令を駆使するなどしなくてはならず時間がかかるのでメモリに書くわけです。実 VRAM に書いた内容は勝手にハード側で画面に出してくれます。

メモリに書く際、座標系はどうするんだと考えなくてはなりませんが、これにも処理のセオリーがあります。Web を検索すれば出てくるとは思いますが・・・簡単に説明すると、仮想 VRAM の先頭アドレスに横座標と 1 行の大きさ×縦座標(行数)を足せば横座標は 1 ドット単位に、縦座標は行単位にメモリに書き出す位置を特定できます。縦座標を 1 ドット単位に操作したい場合は・・・シフトでずらして 2 度書きこむ、とかするんだったと思う(まだプログラムを書いたことがない)。縦の座標が 0～47 であるとすれば、まず 1 行のドット数である 8 で割って解と余りを出す。この時の解が書くべき行数、余りが書きこむデータをシフトする回数。2 回描画するわけですが、まず、ひとつ目のデータは先ほどのシフトする回数で右シフトします。シフトのたび、キャリーフラグの内容をふたつ目のデータに入れます(キャリーを含めた右シフト)。ひとつ目のデータは書くべき行数に書き、ふたつ目のデータは書くべき行数－1 に書く。という感じでよいと思う・・・次回はこの辺をどうにかするところから始めないといけないようです。

さて、仮想 VRAM→実 VRAM 転送ルーチンを使用したサンプルプログラムを見てみましょう。

リスト２：星のスクロール

```
;--------------------------
;STAR SCROLL
;--------------------------
  ORG 0100H         ; 100H よりプログラム配置
  DI                ; 一応、割り込みを禁止
  CALL STINIT       ; 画面外のステータス部を塗る(見栄えのため)
  LD A,0A7H         ; 40H に A7H を出力すると･･･
  OUT (40H),A       ; 画面反転します(コレも見栄えのため)
;--------------------------
;INIT (星データの初期化)
;--------------------------
  LD B,255          ; 星を 255 個作ります
  LD IX,STWORK      ; インデックスレジスタ IX は星データの先頭
INITLP:             ;
  CALL MKSTAR       ; 星作成
  CALL RND2         ; 始めに横アドレスが一緒だとカッコ悪いので
  AND 07FH          ; ランダムに 0-127 で書き直す
  LD (IX+1),A       ; 横アドレス設定しなおし
  INC IX            ; 次のデータのため IX レジスタをデータ分足す
  INC IX
  INC IX
  INC IX
  INC IX
  DJNZ INITLP       ; 255 回ループ
```

```
;----------------------------
;MAIN LOOP
;----------------------------
MAINST:
  CALL VRMCLS    ; 仮想 VRAM クリア
  LD B,255       ; 255 回星データの処理を行う
  LD IX,STWORK   ; IX に星データの先頭を入れる
MAINLP:          ;
; 横アドレス読み込み&セット
  LD E,(IX+1)    ; 横アドレスを読む
  XOR A          ; DE レジスタの上位を 0 で初期化
  LD D,A
  LD HL,VRAM
  ADD HL,DE      ; VRAM の先頭アドレス+横アドレス
; 縦アドレス読み込み&セット
  LD A,(IX+2)    ; 縦アドレスを読む
  OR A           ; 縦アドレスは 0 ですか?
  JR Z,READAD    ; 0 ならば次の足しこみは飛ばす
  PUSH BC        ; ループ用レジスタ保存
  LD B,A         ; 縦アドレスの回数分をループ用変数に入れる
  LD DE,144      ; 144(=画面の横幅)を入れる
ADDTAT:
  ADD HL,DE      ; 縦アドレス分 144 を足していく
  DJNZ ADDTAT
  POP BC         ; ループ用変数を戻す
READAD:
  LD A,(HL)      ; 現在 HL が指し示す仮想 VRAM のデータを読み
  LD C,(IX+3)    ; 星データのデータを取得し・・・
  OR C           ; それぞれを OR で足しこみ・・・
  LD (HL),A      ; 仮想 VRAM に書き込み直す
; カウンタ値を参照し星データを移動させるかどうか判定
  LD L,(IX)      ; 星データのスピード値を読む
  LD A,(IX+4)    ; 星データのカウンタを読む
  SUB L          ; カウンタからスピード値を減算
  LD (IX+4),A    ; 減算した値を星データに書き込み戻す
  JR NZ,EDLOOP   ; カウンタが 0 でなければ横アドレス減算しない
; 横アドレス値をデクリメントします
  LD A,(IX+1)    ; 横アドレス値を読んで
  SUB 01H        ; ひとつ減らす(キャリーフラグのため SUB 使用)
  LD (IX+1),A    ; 横アドレス値を書き込み戻す
  CALL C,MKSTAR  ; キャリーが発生したら星データを作成し直す
; 次のデータへ
EDLOOP:
  INC IX         ; IX レジスタをデータ要素分足す
  INC IX
  INC IX
  INC IX
  INC IX
  DJNZ MAINLP    ; ループ先頭に戻る
; 仮想 VRAM のデータを書き出す
  CALL VRMWRT    ;
; ストップキー検出
  IN A,(1FH)     ; 1FH の最上位が STOP キー
  RLCA           ; 1 個左にずらしてキャリーを
  JR C,ENDSUB    ; 押されてたら ENDSUB へ飛びましょう
; MAIN の先頭に戻る
  JP MAINST
;----------------------------
;MAKE ONE STARDATA(星データをひとつ作成)
;----------------------------
MKSTAR:
  PUSH BC        ; B レジスタはループで使っているので保存
  CALL RND2      ; <星の行内位置作成開始>
  AND 07H        ; 0-7 の値が入りますが・・・
  ADD A,01H      ; 1 を足すことによって 1-8 のランダム値を作成
  LD B,A         ; コレをシフト回数とします
  XOR A          ; A レジスタを初期化して・・・
  SCF            ; キャリーフラグのみセット
SFTRT1:
  RRA            ; 1-8 回キャリーをシフトします
  DJNZ SFTRT1    ; レジスタの何れか 1bit のみ立たせるのです
  LD D,A         ; D レジスタに退避(星の行内位置)
SETTAT:          ;
  CALL RND2      ; <星の縦アドレス作成開始>
AND 07H          ; 0-7 の値が入りますが・・・
```

```
  CP 6              ; ほしいのは 0-5 なので、値が 6 以上の場合・・・
  JR NC,SETTAT      ; ランダム値作成ルーチンを呼びなおす(手抜き)
  LD H,A            ; H レジスタに退避(星の縦アドレス)
  CALL RND2         ; <星の進むスピード作成開始>
  AND 03H           ; 0-3 の値が入りますが・・・
  ADD A,01H         ; 1 を足すことによって 1-4 のランダム値を作成
  LD B,A            ; コレをシフト回数とします
  LD A,01H          ; A レジスタに 01H を入れて・・・
  AND A             ; キャリーフラグリセット
SFTRT2:
  RRA               ; 0 ビット目を 1-4 回右シフトして
  DJNZ SFTRT2       ; 0,32,64,128 の値を得ます
  LD L,A            ; L レジスタに退避(星のスピード値)
; A レジスタに星の横アドレス初期値を入れる
  LD A,143          ;
; 今まで保存していた値を代入
  LD (IX),L         ; 星の進むスピード
  LD (IX+1),A       ; 星の横アドレス
  LD (IX+2),H       ; 星の縦アドレス
  LD (IX+3),D       ; 星の行内の位置(1bit のみ 1)
  LD (IX+4),00H     ; スピード用カウンタは 0 を入れておく
  POP BC            ; ループ用変数復活
  RET
;--------------------------
;RND2(ランダムな数値を返す)
;--------------------------
RND2:PUSH HL        ; 書籍「Z80 マシン語秘伝の書」参考
  PUSH DE           ; 2 個目のランダムルーチンなので
  LD HL,0           ; ルーチン名が RND2 なのです
  LD D,H            ; 深くは解説しません
  LD E,L            ; コールすると A レジスタに
  ADD HL,HL         ; ランダム(っぽい)値が入ります
  ADD HL,HL
  ADD HL,DE
  LD DE,3711H
  ADD HL,DE
  LD (RND2+3),HL
  LD A,H
  POP DE
  POP HL
  RET
;--------------------------
;END (終了時に呼ばれるルーチン)
;--------------------------
ENDSUB:
  CALL VRMCLS       ; 仮想 VRAM クリア
  CALL VRMWRT       ; クリアした仮想 VRAM を実画面に書き込み
  EI                ; 割り込み許可
  LD A,0A6H         ; A6H を 40H に出力すると・・・
  OUT (40H),A       ; 反転していたのが元に戻ります
  RET               ; プログラム終了
;--------------------------
;CLS ALL KASO VRAM (仮想 VRAM クリア)
;--------------------------
VRMCLS:XOR A
  LD HL,VRAM        ; コピー元アドレス
  LD DE,VRAM+1      ; コピー先アドレス
  LD BC,863         ; コピー回数
  LD (HL),A         ; 0 を仮想 VRAM の先頭にセット
  LDIR              ; LDIR でそれを終端まで(863 回)コピー
  RET
;--------------------------
;WRITE RAM TO VRAM (仮想 VRAM ->実 VRAM 転送)
;--------------------------
VRMWRT:             ; LCD 描画ルーチン
  LD A,(790DH)
  LD D,A
  LD E,6
  LD C,41H
  LD HL,VRAM
TATEAD:LD A,0B0H
  ADD A,D
  AND 0B7H
  OUT (40H),A
  LD A,010H
```

```
 OUT (40H),A
  XOR A
  OUT (40H),A
  LD B,144
  OTIR
  LD A,(TATEAD+1)
  INC A
  LD (TATEAD+1),A
  DEC E
  JR NZ,TATEAD
  LD A,0B0H
  LD (TATEAD+1),A
  RET
 ;---------------------------
 ;CREAR STATUS AREA (画面外のステータス部塗り)
 ;---------------------------
 STINIT:              ; LCD 描画ルーチンの応用編(?)
  LD A,(790DH)        ; 実行時の開始アドレスを読んで・・・
  LD D,A              ; D レジスタに退避
  LD E,00H            ; 縦アドレス値初期化(0-5 に変化,6 で終了)
 SILOOP:              ;
  LD A,0B0H           ; 縦アドレス基本値
  ADD A,D             ;  +開始時のアドレス加算
  ADD A,E             ;  +縦アドレス値加算
  AND 0B7H            ; 縦アドレス必要なとこだけにする
  OUT (40H),A         ; 縦アドレス値を出力(設定)
  LD A,19H            ; 横アドレス値(上位)128+16=144 ドット目
  OUT (40H),A         ; 横アドレス値(上位)を出力
  XOR A               ; 横アドレス値(下位)は足しこまなくてよい
  OUT (40H),A         ; 横アドレス値(下位)を出力
  LD A,0FFH
  OUT (41H),A         ; 出力先を全部(FFH なので)埋めましょう
  INC E               ; 縦アドレス値をインクリメント
  LD A,E              ; 判定のため、A レジスタに代入しましょう
  CP 6                ; 6 行書いてなかったら・・・
  JR NZ,SILOOP        ; SILOOP へ戻りましょう
  RET                 ; ルーチン終了
 ;---------------------------
 ;WORK AREA
 ;---------------------------
 STWORK:DS 255*5; 星データの領域確保
 VRAM:DS 144*6    ; 仮想 VRAM 領域確保
```

テキストモードで(コメントを省き、行番号を入れつつ)打ち込んでアセンブルモードでアセンブルしてマシン語モニタから G100 と打ち込むと動作します。1FFFH くらい領域確保をしておいてください。

画面イメージは下記の画像参考。

各点が右から左へ 4 段階の速度で動きます。
仕様上 8 段階も出来たのですが、画面内に常に 255 個星を存在させているために遅い星が画面上に残ってしまって見た目が悪いので 4 段階で。

リスト・・・そんなに大した量ではないと思っていましたが、紙面に写すと結構なページ食いますね。コメントを制限してふたつに分けないとかさばってしょうがありません。昔のプログラミング雑誌でアセンブラ言語のソースリストが載らないわけがよくわかります・・・

見たことあると思った人がいるかもしれません。これの元ネタはソフトバンク刊「マシン語プログラミングテクニック リアルタイムゲームを作ろう」の冒頭の方にあります。この本は PC-8801 用のため、ポケコンに合うように変えています。思想は同じ、実現方法は異なる、と言った感じです。

解説ですが、わきに書いてあるコメントで事足りると思います。特記事項を下記より書いていきます。

インデックスレジスタについて。
IX と IY レジスタなんですが、今回、生まれてはじめて使用しました。データを扱う際に便利ですね。って常識かもしれませんが。

スピードの原理は「星のカウンタ(初期値 00H)から星自身が持つスピード分引いて、0 になったら移動」。
コレはゆうき。さんのアイデアです。いろいろやったけど、結局コレになった。
 00H-00H=00H(毎回移動)
 00H-128H-128H=00H(二回に一回移動)
 00H-64H-64H-64H-64H=00H(四回に一回移動)
 00H-32H-32H-32H-32H-32H-32H-32H-32H=00H(八回に一回移動)

レジスタ中のいずれか 1 ビットだけ立たせる方法は適当に考えたものですが、00H、32H、64H、128H の数値を作ったりする方法が他に思いつきませんでした。なにか、他によいやり方があるのならば教えていただきたいものです・・・

星データは横座標が 1 ドット単位、縦座標が行単位で持たせています。さらに行の中の 1 点だけビットを立たせることによって、複合で縦座標としています。星データは今回、ひとつの点として扱うのでこうしたほうが楽。きちんとしたキャラクタを移動させるのであれば縦座標も 1 ドット単位で持ったほうがよいでしょう。

サブルーチン STINIT で画面外のステータス部(RUN とか DEG とか BATT とかの部分)を塗りつぶしています。いままでどうやっていじるのかわからなかった人は参考に。特定のステータスを表したり、デバッグ用途で使えそうです。いままでデバッグというと「ストップキー検出&RET ルーチン」を気になる個所に突っ込んで、抜けるかどうかで判断していましたが、特定のルーチンに移動したらステータス点灯とかするとよさげです。

ランダム値作成ルーチン。書籍そのままですが、原理は「こう足したりしていくとランダムっぽい値になる」と言うものです。コンピュータで完全なランダムを作ることは出来ないので「っぽい」んです。

星データの初期化について。星データを作るルーチンは初期化時と画面外に消えた時で同じルーチンを使っていますが、初期化時に横アドレス値が全部の星で 0 だと当たり前ですがすべての星が同じ位置から動き出します。これが、ばらけるのにえらい時間がかかります。10 分待ってもばらけません。よって、初期値としてしょうがなく 0〜127 を与えています。なので、実行時の一瞬は 128〜143 ドット目には星が存在しません。よく見ないとわからないけど、製作者的にはちょっと嫌。

初めてこのプログラムが動いた時、私は結構感動しました。ポケコンでもここまで出来るんだ、と。あんまりうれしいのでずーっと眺めていたり人に見せたりしました(すごいの?コレ?と言われましたが・・・)。2ch の某掲示板にも話題を振りました。このサンプルプログラムで G850 の能力の片鱗を見出していただけると幸いです。さすが 8MHz のことだけはあるっ！とか。応用すると、RPG とかで派手な画面効果を作れるので挑戦してみてはいかがでしょうか。

＜IOCS に頼らないキー検出法＞

キー入力の検出法は「PC-G850 解析資料 第 6 版」を見ればわかります。多分。毎度引用してすいませんが、ネットで検索すれば資料は出て来るでしょう。この資料上では「シフトキーの検出法」「実際のマシン語ルーチンの組み方」を提示していないので、今回したいのはそれの捕捉です。いや、ただ、シフトキーの検出法が語れればよいのですが。

通常のキーの入力は I/O ポートの 11H や 12H にキーに対応する値を出力し、I/O ポート 10H を読むことによって検出します。シフトキーの検出は、

「I/O の 11H に 08H 出力後、I/O の 13H の最下位ビットが立っていたら入力アリ」

と判断します。Z80 ポケコン入門(工学社刊)にはシステムポート表に 13H はシフトキー入力であり、キーストローブ(I/O の 11H や 12H のこと)が必要とは書いていましたが、どのストローブにどの値を入れればよいのかは記載されていませんでした。そこで調査したわけですが、11H に値を出力し、13H の最下位ビットが立っていたらプログラムを抜ける、というプログラムを作成して 11H に 01H、02H、04H・・・のように入れてみて調べました。

さて、実際にキーを検出するルーチンを作成してみましょう。
まずは検出したいキーを決め、その対応を表からみつけます。

ゲームでキー入力に必要なキーと言えば「上下左右」の方向用のキー、それと入力とキャンセルを行わせるためのキーが必要です。ファミコン系、ですね。これだけで 6 つもキーを使用します。PC-G850 でオーソドックスなキーと言えば「きちんと並んでいる方向キー」「リターンキー」「スペースキー」だと思います。「きちんと並んでいる」と言ういいかたに疑問を抱く方もいらっしゃると思いますが、ポケコンはそのサイズの制約からか方向キーが方向どおりに並んでいるとは限りません。UNIX ライクと言えばわかるでしょうか。手元で確認できるもので PC-U6000 は方向どおりに方向キーが並んでいませんでした。あとスペースキーですが、BASIC で軽く何か作るときは多用するのですが打鍵音がうるさいです。なので嫌いです。個人的にポケコンは授業中に遊べるツールとして使用されるものと認識しているので、先生ににらまれることがあってはいけません。

下記にこれから作成するもので欲しいボタンと PC-G850 の対応を記します

（欲しいボタン）					:	(対応するポケコンのボタン)						
		上			:				↑			
ボタン１	ボタン２	左	下	右	:	ＳＨＩＦＴ	カナ	←	↓	→		

方向キーに関してですが、当初は上下左右に２．０＝と割り振る気でいました(お手持ちのＧ850 と見比べてください)。そのほうが方向キーと違ってキーが大きく隙間もあり、打鍵しやすかったからです。しかし、ルーチンを作成してみるとテンキーに方向キーを割り当てる実装は向かないことがわかったため、妥協しまくって方向キーを使用しています。ボタン１、２は最近の DOS/V パソコンでいうところの SHIFT、Z、X、C あたりをあてがいたいのですが、キーが小さくて SHIFT はともかく Z キーなんて連続で打鍵するのには

向きません。それで近くで打ちやすいキーを探すとカナキーがありましたのでそれを使用します。
先ほどのキー配置であれば I/O の 11H だけ(12Hは使わない)でプログラムが書けそうです。
下記に記した対応表より必要な値を抜き出していきましょう。SHIFT は前に示したものを使用します。

＜I/O 11H キーストローブ と 10H キーコモン入力の対応表 by 解析資料 6 版 (C)akiyan 氏＞

＼	7	6	5	4	3	2	1	0
7	)	R･CM	M+	Enter	↑	,	K	U
6	／	＊	－	＋	↓	M	J	Y
5	9	6	3	＝	SPACE	N	H	T
4	8	5	2	.	TAB	B	G	R
3	7	4	1	0	カナ	V	F	E
2	π	INS	CON	ON	CAPS	C	D	W
1	BS	O	;	→	TXT	X	S	Q
0	P	I	L	←	BASIC	Z	A	OFF

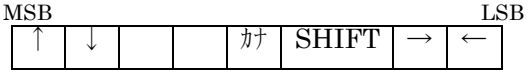
→11H に出力する値(いずれか 1BIT のみ入力)
↓10H から返却される値(入力があると対応する BIT が立つ)

↑　　:　I/O の 11H に 08H を出力、I/O の 10H より返る値の 7 ビット目
↓　　:　I/O の 11H に 08H を出力、I/O の 10H より返る値の 6 ビット目
←　　:　I/O の 11H に 10H を出力、I/O の 10H より返る値の 0 ビット目
→　　:　I/O の 11H に 10H を出力、I/O の 10H より返る値の 1 ビット目
カナ　:　I/O の 11H に 08H を出力、I/O の 10H より返る値の 3 ビット目
SHIFT　:　I/O の 11H に 08H を出力、I/O の 13H より返る値の 0 ビット目

設計として、キー入力を検知するサブルーチンを呼んだら A レジスタに必要なキーの状態が返ってくるものを想定します。いちいち何のキーが押されたら何をする的なことをやっていると無駄です。A レジスタは 8bit あるので、8 つのキーの状態を ON/OFF の状態でもてます。このやりかただとレジスタに欲しいキーの状態が全て入るので、BIT をチェックするだけで容易に同時押しなんかもチェックできます。2 つ余りますが、予備としておきましょう。

A レジスタに入れるには返る値のビットが重複するため、キースキャン後にビットをずらして格納する必要があります。シフトキーと←キーがそのままだと競合するので、シフトキーのスキャン結果は 2 回左シフトしましょう。また、関係ないキーの状態を読むのを防ぐため、キースキャンの結果にいらないキーを強制的に 0 を入れるようにしましょう。

A レジスタに入れる内容を以下のように対応付けます



いままでの事をまとめると以下のようにプログラムが組めると思います

リスト３：キースキャン途中版

```
KEYSC:
  LD A,08H          ; 08H のキーの状態だけ頂戴
  OUT(11H),A
  IN A,(10H)        ; ほい、08H のキーの状態だけ取得。
  AND 0C8H          ; 必要なのは 3,6,7 ビットだけ
  LD B,A            ; B レジスタに退避
  LD A,10H          ; 10H のキーの状態だけ頂戴
  OUT (11H),A
  IN A,(10H)        ; ほい、10H のキーの状態だけ取得。
  AND 03H           ; 必要なのは 0,1 ビットだけ
  OR B              ; 退避していたデータと足し合わせます
  LD B,A            ; 足し合わせた結果を B に退避
  LD A,08H          ; 08H のキーの状態だけ頂戴
  OUT(11H),A        ;
  IN A,(13H)        ; 13H からシフトキーの状態が入る
  SLA A             ; 結果を 2 回左にずらしてから・・・
  SLA A
  AND 04H           ; それ以外のビットを強制 0 にして
  OR B              ; 退避していたデータと足し合わせます
  RET
```

しかし、これだけでは足りません。それでは、いったい何が足りないのか見ていきましょう。
補足①：
先ほど提示したプログラムでもキーの状態の取得はできます。しかし、検出するはずのない BASIC キーを押すと←キーが検出されたように見えたり、0 キーを押したらカナキーが検出されたように見えてしまいます。これはキーを取得する際の I/O の使い方に問題があります。I/O の 11H に欲しいキー列を示す値を入れた後 I/O の 10H にキーの状態値が入りますが、次々と入力しているためか別のキー列の情報も取得してしまうようです。I/O の 11H には 10H と 08H という値を入れていますが、18H を入れているのと同じような状態になってしまっています(実際、複数のキー列を指定する事は出来ます)。これを回避するにはどうすればよいのでしょうか・・・？

しょうがないので IOCS で提供されるキー取得(BE53H,BCFDH)のルーチンを逆アセンブルしたものを見て参考にしようとしました。ROM 内のルーチンを見ることは最高の教科書なのです。すると、各所で「I/O の 11H に 00H を出力する」処理が見受けられました。確かに、00H を指定すればキーの状態はいっさいいらないと言っているようなものなのでよさそうです。そして、キーの取得処理の前に I/O の 11H に 00H を出力する処理を入れてみると「関係ないキーを押しているのに反応する」現象は起きなくなりました。ただし、その処理事態には意味はなく、適切な WAIT を取っているだけというオチなのかもしれませんが・・・

単発でキーを押したときに、別なキーで反応すると言うことはなくなりました。しかし、3 つのキーを押すと反応してしまう、ということがあります。後に掲載するプログラムで確認していただきたいのですが、「→、BASIC、TEXT を同時に押すと←キーが検出」されてしまいます。おそらくこれはソフト的なものではなく、ハード的な物でしょうがないものだと思います。I/O の 11H と 10H の対応表を見てみると、I/O の 10H に返る値で同じビットの重みにあるキーの同時押しがいけないことのように思われます。このことから、同じビットの重みに位置するキーの同時検出は正確性のためにはすべきではない、という法則が生まれます。この法則によって始めに上下左右に２．０＝を割り振るということはあきらめました。上下左右キーのビットがかぶっていないところを見ると、設計した人もその辺を考えていたのだと思います。

さて、先ほどの法則からするとキー取得のプログラムに追加できるキーの種類が絞られることになります。現在使っていないのは以下になります｛BIT5(９,６,３,＝,SPACE,N,H,T)、BIT4(８,５,２,．,TAB,B,G,R)、BIT2(π,INS,CON,ON,CAPS,C,D,W)｝。シフトで 1BIT 使っているため、実際に検出に加えられるのはこの中から 2 つだけです。シフトキーの状態はシフトさせればどうとでもなります。キーの配置からするとSPACE,TAB,CAPS あたりが第 3 のボタンとして使えるのではないでしょうか。ON キーを検出したいときは I/O ポートの 1FH を使用するのが常識的なので ON キーはここで取得しなくてもよいと思います。

補足②：
IOCS のキー取得ルーチン内では I/O の 11H や 12H に出力した後、0.3msec 程の WAIT を取ってから I/O の 10H より値を取得していました。おそらく、出力してから結果が返るまでにそれくらいかかるのでしょう。もしくはチャタリング防止なのかもしれません。正確にキー取得をしたいのであれば呼んだほうがよさそうです。しなくても影響はないと思いますが。CALL ８AADH とすればそのルーチンを呼べます。ただしこれは G850 でのアドレスであり、G850S や G850V ではアドレスが変わっている可能性があります。よって、下記に書き写しておきます。

リスト４：G850 では８AADH から始まっているキー取得用の WAIT 用ルーチン(解説抜き)

```
KEYWAIT:
  PUSH BC
  LD BC,050H
  CALL W1
  POP BC
  RET
W1:
  PUSH AF
W2:
  DEC BC
  LD A,C
  OR B
  JP NZ,W2
  RET
```

補足③：
リスト３を見て、「同じ 08H を出力しているのだから、一気に I/O の 10H と 13H の値を取ったほうがおいのではないか？」と疑問を持った方がおられると思います。しかし、そのように組んでみたら「検出しないはずのキーで取得してしまう現象」が再発しました。なので、正直に分けて考えました。

補足④：
私はプログラムの最初と最後に DI、EI 命令(割り込み禁止、解除)を行いますが、キー取得単品の処理として考えると、念のためにメソッドの最初に割り込み禁止して最後に解除するとよいと思う。

直すべき点はわかりました。次に、それらを盛り込んだキー取得ルーチンをサンプルに組み込んだものを示します。

プログラムリスト５：キー取得ルーチンと動作サンプル同梱リスト

```
 ORG 0100H          ; 100H よりプログラムを配置
 CLS:XOR A          ; 画面消去開始
  LD B,24*6         ; 画面全部分書きます
  LD D,A            ; 開始行 0
  LD E,A            ; 開始列 0
  CALL 0BFEEH       ; IOCS n 文字表示で処理
 START:XOR A        ; ステータス表示させる部分を消去
  LD B,8            ; 8 文字分
  LD D,A            ; 開始行 0
  LD E,A            ; 開始列 0
  CALL 0BFEEH       ; IOCS n 文字表示で処理
  CALL KEYSC        ; キースキャン処理コール
  BIT 7,A           ; ビット 7 が立っていたら↑処理へ
  CALL NZ,UP
  BIT 6,A           ; ビット 6 が立っていたら↓処理へ
  CALL NZ,DN
  BIT 0,A           ; ビット 0 が立っていたら←処理へ
  CALL NZ,LF
  BIT 1,A           ; ビット 1 が立っていたら→処理へ
  CALL NZ,RI
  BIT 2,A           ; ビット 2 が立っていたらシフト処理へ
  CALL NZ,SFT
  BIT 3,A           ; ビット 3 が立っていたらカナ処理へ
  CALL NZ,KAN
  IN A,(1FH)        ; ストップキー検出
  RLCA
  RET C             ; ストップが検出されたら終了
  CALL WAIT         ; WAIT ルーチンで速度調整
  JR START          ; 始めに戻る
 UP:PUSH AF         ; ＜↑＞
  LD DE,0000H       ; 縦 0 横 0 に
  LD A,55H          ; 文字「U」を
  CALL 0BE62H       ; 書く
  POP AF
  RET
 DN:PUSH AF         ; ＜↓＞
  LD DE,0001H       ; 縦 0 横 1 に
  LD A,44H          ; 文字「D」を
  CALL 0BE62H       ; 書く
  POP AF
  RET
 LF:PUSH AF         ; ＜←＞
  LD DE,0002H       ; 縦 0 横 2 に
  LD A,4CH          ; 文字「L」を
  CALL 0BE62H       ; 書く
  POP AF
  RET
 RI:PUSH AF         ; ＜→＞
  LD DE,0003H       ; 縦 0 横 3 に
  LD A,52H          ; 文字「R」を
  CALL 0BE62H       ; 書く
  POP AF
  RET
 SFT:PUSH AF        ; ＜シフト＞
  LD DE,0004H       ; 縦 0 横 4 に
  LD A,53H          ; 文字「S」を
  CALL 0BE62H       ; 書く
  POP AF
  RET
 KAN:PUSH AF        ; ＜カナ＞
  LD DE,0005H       ; 縦 0 横 5 に
  LD A,4BH          ; 文字「K」を
  CALL 0BE62H       ; 書く
  POP AF
  RET
 WAIT:XOR A         ; 汎用 WAIT ルーチン
 L1:LD B,5
 L2:DJNZ L2
  DEC A
  JR NZ,L1
  RET
 KEYSC:DI           ; キースキャン開始&割り込み禁止
  XOR A             ; 00H を
  OUT (11H),A       ; I/O の 11H に出して初期化
  LD A,08H          ; 08H のキーの状態だけ頂戴
  OUT(11H),A        ;
  CALL 8AADH        ; 結果が返るまで 0.3msec 待ちましょう
  IN A,(10H)        ; ほい、08H のキーの状態だけ取得。
  AND 0C8H          ; 必要なのは 3,6,7 ビットだけ
  LD B,A            ; B レジスタに退避
  XOR A             ; 00H を
  OUT (11H),A       ; I/O の 11H に出して初期化
```

```
LD A,10H        ; 10H のキーの状態だけ頂戴
OUT (11H),A     ;
CALL 8AADH      ; 結果が返るまで 0.3msec 待ちましょう
IN A,(10H)      ; ほい、10H のキーの状態だけ取得。
AND 03H         ; 必要なのは 0,1 ビットだけ
OR B            ; 退避していたデータと足し合わせます
LD B,A          ; 足し合わせた結果を B に退避
XOR A           ; 00H を
OUT (11H),A     ; I/O の 11H に出して初期化
LD A,08H        ; 08H のキーの状態だけ頂戴
OUT(11H),A      ;
CALL 8AADH      ; 結果が返るまで 0.3msec 待ちましょう
IN A,(13H)      ; 13H からシフトキーの状態が入る
SLA A           ; 結果を 2 回左にずらしてから・・・
SLA A
AND 04H         ; それ以外のビットを強制 0 にして
OR B            ; 退避していたデータと足し合わせます
EI              ; 割り込みオーケーです
RET
```

おなじみですが、テキストモードで(コメントを省き、行番号を入れつつ)打ち込んでアセンブルモードでアセンブルしてマシン語モニタから G100 と打ち込むと動作します。マシン語領域も確保を忘れずに。

取得しようとしているキーを押すと次の対応で文字が出力されます。

↑　　：　U (UP)
↓　　：　D (DOWN)
←　　：　L (LEFT)
→　　：　R (RIGHT)
シフト　：　S (SHIFT)
カナ　　：　K (KANA)

やっていることは紹介済みなので再度説明はしません。
また、この程度のプログラムでは完璧なキー取得ルーチンであるかどうかの判断が難しいため、何かおかしなところがあるかもしれません。その点は了承願います。

＜最後に＞

いかがだったでしょうか。自己流の点が多く(さらに流用多し)、テストも十分にしているとは言いがたいので完璧ですとは言いがたいですが、ゲームを作る上での基礎の部分が構築できたと思います。

何か意見や、間違いの指摘などがありましたら遠慮なくお送りください。なにぶん、個人でやっているもので、そう言う意見を言ってもらえる機会がないもので・・・

私はゲイムを作るにあたって、基礎の部分の構築から入ってしまいました。そのため、ゲイムの製作能力と言うのはほとんど身につかなかったのが現状です。ここまで来るのに――――G850 を手に入れたのが 1996 年ですから、実に 8 年もかかっています。我ながら何をやっていたのやら、という感じでありますが・・・

今度はまともにゲイムの作成が出来ます。いままで溜め込んだ知識、書籍等を総動員して作品を作り上げていこうと思っております。なんというか、「過去の知識の発掘」作業みたいですけど。これからの目標は、授業中に遊べて、休み時間にみんなでスコアの報告会なんかをするような――――そんなゲイムを作ることです。稼ぎ要素入りの STG なんかよいですね。

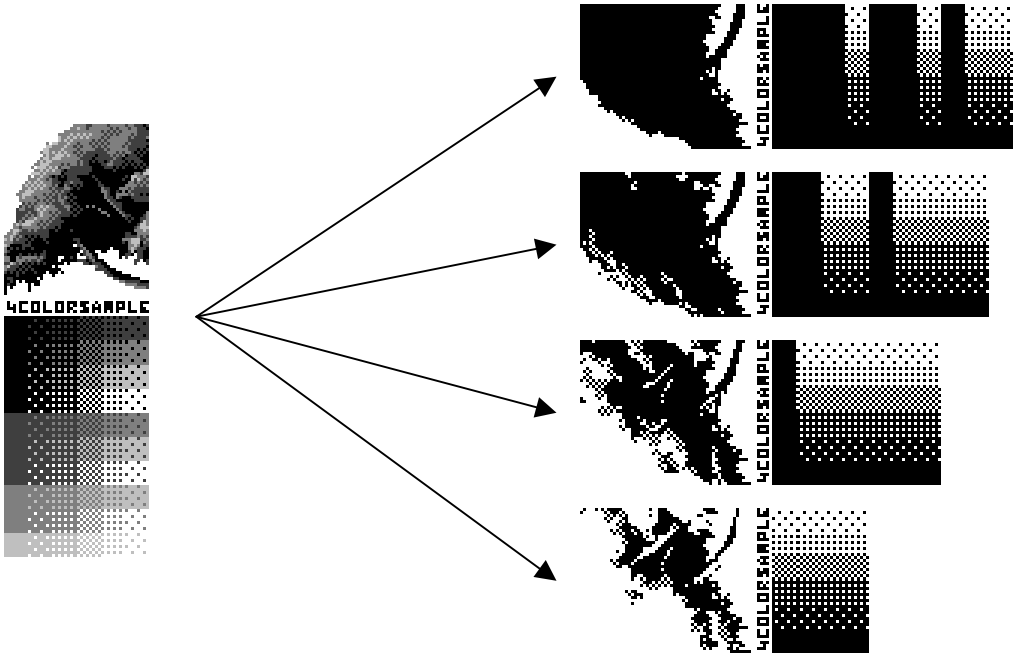
今回、星のスクロールルーチンで画面の取得に使った仕組みを載せようかとも思いましたが、紙面も時間も尽きてしまいました。そもそも、この本に書いてあることは Web ページ上への公開や 2ch への投稿をしたものなので、ここに書かずとも知り得ることは出来るのですが・・・機会があればこういう本の形式で触れていきたいと思います。

ここで書いたことが、マシン語でプログラムを組もうとしている方の手助けになるのでしたらそれほどうれしいことはありません。どうか、そうなりますように――――

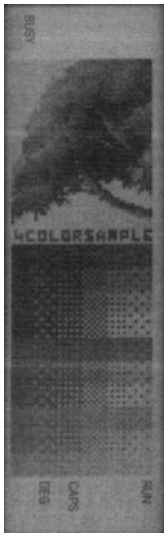
企画２：中間色のススメ２

前回の「ぽけこんのほん１」では２枚の画像を交互に高速に描画することによって「灰色に見える」プログラムを作成しました。今回はさらに色を増やすべく、４枚の画像を交互に高速に描画することを実践しました。

やっていることは前回と同じなのであまり説明をしません。
前回使用したプログラムの描画の部分を２回増やして WAIT 調整をしただけです。
今回も Windows 上のペイントソフトで(白を除いて)4 階調の色(0,0,0：63,63,63：127,127,127：191,191,191：255,255,255)を使用して画像を書き、前回の中間色のススメで提示した HSP のプログラムを使用して画像データを作成しました。



今回、色が増えたことによって表現できることが大幅に広がったのですが、私の腕が追いつかず、当初予定していた画像とは大きく違う画像を作成してしまいました。当初は女の子らしきものの絵を書いていましたが、どうしてもかわいく仕上がりません。色を載せるとどうもおかしくなる―というのも、いままで着色の技術をおざなりにしていたのが原因なのですが。しかしながらテーマが「日差しで透けたワンピースを着ている少女＋大木＋4 色であることを示す文字」だったので、それらのなかから使えそうな物を使いまわしました。下の方のパタン表は苦肉の策。



一応、次項にプログラム・リストを掲載しますが、紙面の都合上データ部等を飛ばします。全体を掲載と出来ないことはご了承ください。

実行すると左記のようになります。正直、ポケットコンピュータでここまでの表現が出来るとは思いませんでした

プログラム･リストの中で変わったところは OUTI の 144 回繰返しの部分と､WAIT 値の変更と、もちろんですがデータの描画とデータの増加です。

どうも WAIT 値の調整が微妙で、なるたけ早い LCD 描画をさせるために OTIR でループさせずに、ちょっと高速な OUTI×144 を採用しました。

WAIT 値の調整自体はもう、トライ＆エラーで何回もやって決めました。

問題は､「ゲームに使えるかどうか微妙」な事です。WAIT 調整が微妙過ぎ、前回のものレベルならまだしもゲームに組み込む自身がまったくありません。工夫次第でしょうか？

ポケコンでえっちなゲイムを作ることは工業高校生時代から妄想していたことなので、次は萌え絵を表示しつつゲイムのようになっているものを作るのを目標としたいと思います。

リスト：サンプル４色表示ルーチン(紙面の都合上の略アリ、前回のものを流用したため行番号が残ってしまった)

```
10 ;---------------------------
20 ;4 Color Write
30 ;---------------------------
40 ORG 0100H
50 DI
60 ;---------------------------
70 ;MAIN LOOP
80 ;---------------------------
90MAINST:
100 LD HL,SCR0
110 CALL VRMWRT
120 CALL WAIT
130 LD HL,SCR1
140 CALL VRMWRT
150 CALL WAIT
160 LD HL,SCR2
170 CALL VRMWRT
180 CALL WAIT
190 LD HL,SCR3
200 CALL VRMWRT
210 CALL WAIT
220 IN A,(1FH)
230 RLCA
240 JP C,ENDSUB
250 JP MAINST
260 ;---------------------------
270 ;WRITE RAM TO VRAM
280 ;---------------------------
290VRMWRT:
300 LD A,(790DH)
310 LD D,A
320 LD E,6
330 LD C,41H
340TATEAD:
350 LD A,0B0H
360 ADD A,D
370 AND 0B7H
380 OUT (40H),A
390 LD A,010H
400 OUT (40H),A
410 XOR A
420 OUT (40H),A
430 OUTI
440 OUTI
450 OUTI
460 OUTI
470 OUTI
480 OUTI
490 OUTI
500 OUTI
      ・
      ・
      ・        1 行分(144 回)OUTI を行う
      ・
      ・
      ・
1770 OUTI
1780 OUTI
1790 OUTI
1800 OUTI
1810 OUTI
1820 OUTI
1830 OUTI
1840 OUTI
1850 OUTI
1860 OUTI
1870 LD A,(TATEAD+1)
1880 INC A
1890 LD (TATEAD+1),A
1900 DEC E
1910 JP NZ,TATEAD
1920 LD A,0B0H
1930 LD (TATEAD+1),A
1940 RET
1950 ;---------------------------
1960 ;WAIT
1970 ;---------------------------
1980WAIT:
1990 LD A,2
2000L1:LD B,140
2010L2:DJNZ L2
2020 DEC A
2020 DEC A
2030 JR NZ,L1
2040 RET
2050 ;---------------------------
2060 ;END
2070 ;---------------------------
2080ENDSUB:
2110 EI
2140 RET
2150 ;---------------------------
2160 ;DATA
2170 ;---------------------------
2180SCR0:
2190 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
2200 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
2210 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
2220 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
2230 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
2240 DW 0FFFFH,0EFFFH,01B8FH,00011H
2250 DW 00000H,0F8C0H,0FFFFH,0001FH
2260 DW 00000H,04700H,04644H,00077H
2270 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
2280 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
      ・
      ・
      ・        1 枚目の残りのデータ
      ・
      ・
      ・
3270SCR1:
3280 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
3290 DW 0FFFFH,0FFFFH,0FFFFH,0FEFFH
3300 DW 0CFFCH,0FFD7H,0FFFFH,0FFFFH
3310 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
3320 DW 0F9FDH,0FDF8H,0FFFFH,0C7C7H
3330 DW 0DFEFH,0EFDFH,01B8FH,00011H
3340 DW 00000H,0F8C0H,0FFFFH,0001FH
3350 DW 00000H,04700H,04644H,00077H
3360 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
3370 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
      ・
      ・
      ・        2 枚目の残りのデータ
      ・
      ・
      ・
4360SCR2:
4370 DW 04F9FH,02F0FH,07F5EH,07FBFH
4380 DW 03F7FH,0FFFFH,0FFFFH,0FEFFH
4390 DW 0C1E4H,095D2H,05F0AH,05FBFH
4400 DW 0FFFFH,0FFFFH,0FEFFH,0FEFEH
4410 DW 0F8FCH,0E0F0H,0F5F0H,08506H
4420 DW 08F87H,0EFDFH,01B8FH,00011H
4430 DW 00000H,0F8C0H,00FFFH,00001H
4440 DW 00000H,04700H,04644H,00077H
4450 DW 0FFFFH,0FFFFH,0FFFFH,0FFFFH
4460 DW 00088H,00022H,00088H,00022H
      ・
      ・
      ・        3 枚目の残りのデータ
      ・
      ・
      ・
5450SCR3:
5460 DW 00102H,00000H,00100H,00100H
5470 DW 0150AH,0078BH,055A3H,0FCEAH
5480 DW 040E0H,080C0H,00000H,00580H
5490 DW 0E0C0H,0FFFFH,0FEFFH,0583CH
5500 DW 0F0B8H,0E0E0H,00030H,08400H
5510 DW 08184H,02393H,00203H,00000H
5520 DW 00000H,0F8C0H,00007H,00000H
5530 DW 00000H,04700H,04644H,00077H
5540 DW 00088H,00022H,00088H,00022H
5550 DW 00088H,00022H,00088H,00022H
      ・
      ・
      ・        4 枚目の残りのデータ
      ・
      ・
      ・
```

企画 3：巷で見かけたポケコン

前回、漫画の中でポケコンが出ているものを紹介しました。
それから、私がメデイアの中で見かけたポケコンを紹介します。
(といっても今回はひとつだけですが・・・)

アルルゥとあそぼ！！（Leaf、18 禁パソコンゲイム）
いわずと知れたリーフのゲイムです。
うわ、この本の気質に合っていない気がする・・・
このゲイムのどこにポケコンが出るのかというと、本編ではなく、スタッフのコーナーで YUM いわきさんというプログラマさんが「初めてのコンピュータ」として触れられています。


○初めてのコンピュータPC1245　岩城 猫

一番最初に手にしたコンピュータはポケットコンピュータ名前のとおりにシャツの胸ポケットに入れることができる大きさの電卓の親戚のようなものだった。

中学二年生の当時、パソコンなど高価でとても買えなかった私がなんとか購入したのがシャープのPC－1245、本当は処理速度が速いカシオのPB－100が欲しかった。PB－100が定価12，800円に対して手持ちが一万円ぽっきり予算が足らず、売りのPC－1245を購入した。実はこの機種は処理速度が遅く表示機能も劣っていた。しかし自分のコンピュータを手に出来る誘惑には勝てなかった。9，999円だった。その後隠し命令で機械語が実行できる事が判り速度や表示の問題が解決した。一気に逆転勝ち組となった。

授業中も平気でポケコンを触っていたので成績はガタ落ち高校受験も失敗した。

ポケコンの利点はどこにでも持ち歩けて、どこでもプログラミング出来てしまう処。学校の教室で、誰にでも自分の作ったゲームを遊んでもらえた。直接生の反応を貰える事は今思えば貴重だった。パソコンではこうはいかない。P/ECEを見たときにキーボードがあったらなぁと思ったものだ。

PC-1245
すごく小さいキー

<パッケージ画像>　<リーフのプログラマは PC-1245 ユーザだった！>

うわ、画像がこの本の気質に合っていない気がする・・・
このゲイムのどこにポケコンが出るのかというと、本編ではなく、スタッフのコーナーで YUM いわきさんというプログラマさんが「初めてのコンピュータ」として触れられています。

PC-1245 は昭和 58 年 3 月発売、定価 17800 円、(見たところ) 1 行 15 文字程度の表現力を有してそうだ。周辺機器として CE-124P(カセット・インターフェイス)と CE-125(S)(24 桁サーマル・プリンタ/マイクロ・カセット・レコーダ)と CE-126P(24 桁サーマル・プリンタ/カセット・インターフェイス)が使えるマシンです。シャープ製。
手持ちの資料でわかることはコレくらいです・・・
入門機の結構古い部類に入りますが、私の家にある一番古い資料「ポケコンジャーナル 88'1 号(創刊号)」に載っている広告にすでに載っていません・・・

PB-100 は手持ちの資料では詳しい性能がわかりません・・・PC-1245 とほぼドッコイと思うのですが、ネットで調べればわかるかもしれません。PB-100 は PJ の後期までプログラムが載るなど、愛用するユーザが多いんだなぁと思っていた記憶があります。カシオ製。

ポケコンの利点として、いつでもどこでもプログラミングできるという点を挙げられていますが、私もそう思います。というか、ポケコンユーザーの人は「その場でプログラミングできること」に喜びを見出すのではないでしょうか。学校で作ったプログラムが学校内を回りまわっていろんな改造を受けた状態で戻ってくるとかするのは、今考えると、すごい。

キーボード。最近はこのようなサイズのキーボードを持つ製品が PDA くらいでしかありえませんが、P/ECE とかワンダーウィッチとか携帯電話のプログラミングに手を出して結局またポケコンに戻ったのはやはりキーボードがあるからでしょう。

プログラマさんを好きになったのは Littlewitch のプログラマさん(アザナシさん・・・だっけ？)以来なので貴重な存在です　・　・　・と言ったところで今回はお開き。

～ あとがき ～

今は 8/8 の朝 4 時・・・7 月の中旬からやっていたから、作業日数は 3 週間というところでしょうか。これからコピーのために仙台七夕で混んでいる街中を通るのかと思うとぞっとしますが・・・

しかし・・・解析ネタと言うのは書くのにえらい時間がかかりますね・・・。1 ページ書くのに 3 時間とかかかっていました。プログラムのコメントとか入れるのにもえらい時間がかかりました。アセンブラのリストで解説入れようとすると、大した事やってないのに解説をたくさん入れないとわかんないとか紙面をいっぱい食うとかいいことがあまりないですね・・・

今回のネタでストックしてあった 4 年分くらい作業していた解析ネタがなくなりました。まだやることとしてはタイマ関係調査とかあるのですが、すっぱり作業が止まっています。

解析ネタはしばらくやめて、念願のゲイム作成に舞台を移していきたいです。そもそも解析に走ったのがゲイムの作成の土台を作るためですし。集めた過去の資料の勉強の進み具合がまだ 3 割程度なので、それを埋めつつ、来年の夏に向けてがんばっていきたいです。

連続で冬コミも申し込んでみようと思います。解析(済み)ネタが尽きたので、なにか軽いことを扱いたいですね。「私はこうやってポケコンのプログラムを開発している！」とか。期間も短いですし。ゲームのプロトタイプとか出来てたらいいなぁ。

ポケコンサークルが私しかいないというのは寂しいですね・・・今回もカタログを見て落胆してしまいました(自分のカットを見て悶絶もした)。私以外のサークルさんが現れる日は来るのでしょうか・・・？

2004 年 8 月 8 日朝方(やっぱり寝ていない)　あきひ(首謀者＆執筆者)

＜今回のカタログ用カット＞

奥付

誌名　：「ぽけこんのほん２」

制作　：単色ドット絵向上委員会(あきひ)

印刷　：オフィスベンダー南仙台店

発行日：2004 年 8 月 8 日　初版

ＨＰ　：「秋日和」

http://ha9.seikyou.ne.jp/home/akihi/

メアド：akihi@ma9.seikyou.ne.jp

技術は一代で終わるものにあらず。

次代にひきつぎ、よりたしかな技術としてさらに新しい技術を生み出す礎となれ。

人･･･それをして進歩となす･･･

２００４年 夏 単色ドット絵向上委員会