Tokuma Intermedia Mook

PC Engine

DEVELO でべろ

PCエンジンユーザー開発システム

SUPER CD-ROM²

でべろスターターキット・アセンブラ編

出荷台数300万台といわれるPCエンジンのソフトがこれ1冊で自分で作れるようになるゲームクリエイターをめざす人のための本

PCエンジン用
スーパーCD-ROM²
付録

PC-98・MSX共用
3.5インチFD

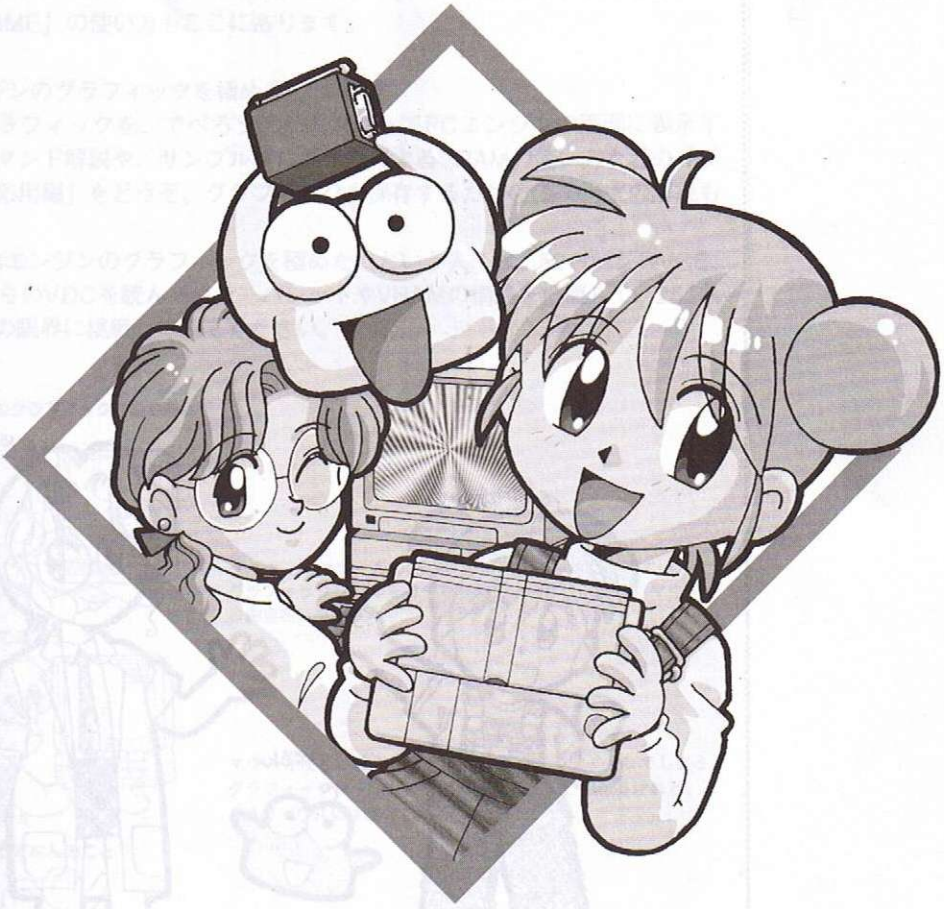
PC Engine FAN 編集部 編・著

Tokuma Intermedia Mook

DEVELO でべろ

PCエンジンユーザー開発システム

スターターキット・アセンブラ編



本書の利用の手引き

PCエンジンファン
弊社発行のPCエンジンの専門誌です。

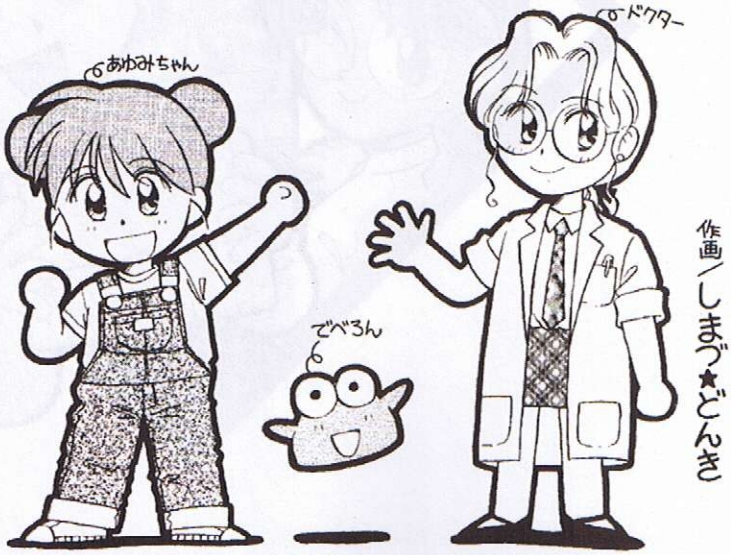
Developer
ディベロッパー。ゲームを作る制作者は今まではゲーム制作会社しかいませんでしたが、これからは、このでべろを買ったあなたもディベロッパーです。

でべろんからごあいさつ

『でべろ』という商品名は、制作者を意味する英語のDeveloperからとりました。でべろという親しみやすい名称から、下のようなイメージキャラクタを作りました。本書の表紙や付録のCD-ROMのタイトル画面やメニュー画面など随所に現れますので、かわいがってやってください。

このでべろのキャラクタたちは、月刊PCエンジンファン誌でコミックとして連載されています。ちなみにコミックの設定では、でべろんという謎の生物は、博士が発見したことになっているのですが、どこで発見したのかは不明です。

というわけで、博士は超頭のいいベテランプログラマ、あゆみちゃんはゲームが大好きな少女でプログラムを覚えようとしている初心者プログラマ、でべろんは……。次ページからの手引きでは、あなたのレベルに合わせて最適な本書のページを示します。あなたのレベルは、どのキャラクタに近いですか？



あゆみちゃん
好奇心旺盛な元気な女のコ。
ドクターの助手をつとめる

でべろん
あゆみちゃんといつも
いっしょの謎の生き物

ドクター
でべろんを発見(?)した、
かしこいおねーさん

グラフィックに興味がある人へ

グラフィックに興味のある人へむけた、本書の読み進め方の手引きです。グラフィックに関する知識・技術の上達具合によって、読み進めるべきページを絞ってあります。

●グラフィックってなんのこと

PCエンジンではどのようなグラフィックが扱えるのでしょうか。16ページのように、さっそくPCエンジンに付録CD-ROMをセットして、CD-ROMの中に収録されている数多くのグラフィックを見てみましょう。

●とにかく描いてみよう

自分でもグラフィックを描いてみたいという人は、22ページからの「コンピュータで絵を描く」を読んでください。グラフィックツールという絵を描くためのツールで、マウスという道具を使ってグラフィックを描いていく過程を説明しています。

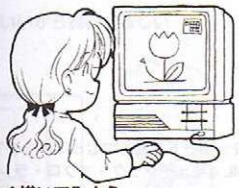
PC-98用グラフィックツール『マルチペイント』、MSX用のグラフィックツール『SII-ANIME』の使い方もここにあります。

●PCエンジンのグラフィックを極める

描いたグラフィックを、でべろシステムを使ってPCエンジンの画面に表示するためのコマンド解説や、サンプルプログラムによるVRAMの使い方などの「グラフィック応用編」をどうぞ。グラフィックを保存するためのGP0形式の解説もあります。

さらにPCエンジンのグラフィックを極めたいという人には、52ページのVCE、53ページからのVDCを読んでカラーパレットやVRAMの構造を把握して、256色の表示能力の限界に挑戦してみてください。

●PCエンジンのグラフィックを極める



グラフィックツールの友「マウス」を日常的に使っている人。グラフィックツールを自分なりにカスタマイズして、マウスを自由自在に操れる人

●とにかく描いてみよう

マウスを使うと思うように動かせなくてイライラしてしまう。グラフィックツールという言葉は聞いたことがあるけれど、あんまりマウスには慣れていないという人

●グラフィックってなんのこと

マウスってなーに？　マウスという言葉はもちろん、グラフィックツールという言葉すら聞いたことがないという人

グラフィック
PCエンジンからテレビに映しだされる画面に描かれた絵のこと。

サンプルプログラム
プログラムの具体的な書き方の例として、PCエンジンを動作させるためのプログラムをCD-ROMに収録したり、このマニュアルに掲載したりしてあります。自分がプログラムを作るときの参考にしてください。

VCE
ビデオ・カラー・エンコーダ。機能は52ページの解説を参照。

VDC
ビデオ・ディスプレイ・コントローラ。機能は53ページからの解説を参照。

カスタマイズ
自分が使いやすいようにツールの操作環境や動作などを変更すること。

サウンドに興味津々の人へ

サウンドに興味のある人へむけた、本書の読み進め方の手引きです。サウンドに関する知識・技術の上達具合によって、読み進めるべきページを絞ってあります。

●コンピュータって音がでるの？

PCエンジンが奏でるサウンドをとりあえず聴きたいという人は、16ページを見てCD-ROMに収録の全5曲を聴いてみてください。

●曲を鳴らしてみよう

PCエンジンの曲はミュージックトラックデータという形になっています。パソコンからミュージックトラックデータを演奏するための方法を、28ページの「コンピュータで音を奏でる」で解説しています。

●オレの曲を聴け～

自分で曲を作りたいんですけどォ～。というあなたに、MMLという手法で音楽を作るPCエンジンでの曲づくりの解説が、172ページからの「サウンド応用編」です。MMLコンバータを使ってのPSG音源の曲を作るときの手順を解説しています。

さらにMMLコンバータはPSGの機能のすべてを活かしきっていな～いと豪語されるあなたには、228ページからのPSGドライバーをどうぞ。PCエンジンに用意されているPSGのすべてのサービスを網羅しています。

●オレの曲を聴け～

音楽は大好き。どちらのキーボードも目をつむってタッチできる人。コンピュータの楽譜「MML（ミュージック・マクロ・ランゲージ）」で曲が作れる

●曲を鳴らしてみよう

カラオケは、へたなんだけど大好きな人。いつも人の曲を聴いているだけ、自分でも作れたらなー、なんて思っている

●コンピュータって音がでるの？

かんたんに音楽が作れるわけがないと思っている。とうぜんのようにパソコンで音楽なんて作れっこないと思っている人

プログラムを作ってみたい人は……

PCエンジンのプログラムを作ってみたい人へむけた、本書の読み進め方の手引きです。プログラミングに関する知識・技術の上達具合によって、読み進めるべきページを絞ってあります。

●プログラムってなに？

プログラミングの経験がほとんどない人は、30ページからの入門編「プログラミングとは」からを読みはじめましょう。ここでは、プログラムを組む際に使用するテキストエディタやアセンブラについての説明もしています。

●とにかくやってみたい

とりあえず実際にやりながら覚えていきたい人というは、132ページからの第3部「プログラミング一般」を読んでみましょう。いくつかのサンプルを交えたプログラミングの解説があります。

また、156ページからには付録CD-ROMに収録したサンプルプログラムの解説があります。実際に動かして試せるこれらサンプルプログラムのリストを見ながら解説を読めば、プログラミングの世界を身近なものとして感じられるかもしれません。

64ページからの部分には、マシン語の命令（ニーモニック）の解説がありますので、サンプルを改造したりしながら覚えていくといいでしょう。

●理屈はいいから資料がほしい

プログラミングについてはある程度感触がつかめているという人は、184ページからに第4部「データ＆資料」としてBIOSやPSGドライバー、でべろ外部コマンドなどをまとめてあります。

また、36ページからの第1部には、VDCやVCE、PSGのレジスタやVRAMの構造などの概説もありますので、本格的にプログラミングを始める前に、一度読んでおくといいでしょう。

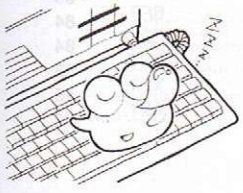
●理屈はいいから資料がほしい

PCエンジンのことは知らなくても、だいたい頭の中でプログラムが組み立てられるという人

●とにかくやってみたい

パソコンにさわるのはソフトを使うときだけ、という人。プログラムなんてこわごわ

●プログラムってなに？



パソコンとか、プログラムとか、むずかしいことはわからないけど、自分でゲームが作ってみたいという人

上達具合
プログラミングを始める（覚えていく）準備が整っていない人、準備は整ったが自力ではプログラムが組めない人、自力でプログラムが組める人の3段階にわけてあります。

リストを見ながら
『SPEEDER』と『BENDBENT』の2つのサンプルプログラムのソースリストは掲載していません。

ソフト
ソフトウェアのこと。プログラムやデータなどの情報のこと。反対にPCエンジン本体などの機械のことをハードウェア（ハード）という。

CONTENTS もくじ

ページ数の横についているマークは難易度を示しています

 やさしい

 ちょっと努力が必要

 ハイレベル

はじめに

でべろ入門

第1部　ハードウェア解説

第２部　ソフトウェア解説

第３部　アセンブラプログラミング解説

第４部　データ&資料

第５部　付録

※Hu7SYSTEM2およびHu7CD SYSTEM3より以下のページを転載・流用しました。
P 36～ 62（第1部　ハードウェア解説）
P 64～131（第2部　ソフトウェア解説）
P188～215（第4部　BIOS）
P228～270（第4部　PSGドライバー、グラフィックドライバー）

「でべろ」ユーザーになったみなさんへ

私たちは、この「でべろ」をつうじて、みなさんにぜひすばらしいゲーム、音楽やＣＧなどの作品を作る楽しさを知ってもらいたいと思っています。この企画は、そういった願いのもとに日本電気ホームエレクトロニクス株式会社さま、株式会社ハドソンさまの協力を得て実現しました。

そしていつの日か、みなさんがこの「でべろ」を足掛かりとして、プロのゲームデザイナーやプログラマーとして巣立ってほしいと思っています。

なぜ、このようにゲームデザイナーやプログラマーにこだわるのかというと、今の世の中では、アマチュアからスターへの道がほとんどないからなのです。ゲームを作りたいと思ったときは、ゲームを作っている会社に入るしか方法がないのが実情なのです。私たちはそういった道をぜひ作りたいと考えています。

私たちのよく知っている、現在スタープログラマーと呼ばれている人たちも、スタートは自分のパソコンで自作した作品で、デビューしています。最初からゲーム機で作品を作っていたわけではありません。その誰もが、パソコン（当時はマイコンと呼ばれたＮＥＣ製ＴＫ-80、ＰＣ-8001 等）で、アマチュアとして作品を作っていたのです。ゲーム機は現在64ビットの高機能のマシンとなり、開発環境も1000万円もするハードを必要としています。そういった状況のなかで、アマチュアの人たちの力を発揮する場がなくなってきていることは否定できません。特に、ゲームに関しては、現在のMacintoshやWindowsでは、本来のゲームらしいゲームには機能的に向いていない部分もあります。そこで、みなさんにゲーム機の開発を体験し、力を発揮してもらえるように、という願いがこの「でべろ」には込められています。アセンブラというマシンにもっとも近い言語を選んだのもそのためです。

もちろん、誰もかれもにプロになれ、とまでは思っていません。が、せめてゲームを作る楽しさを感じてほしいと思っています。

300 万台も普及しているＰＣエンジンを利用して、どんどん作品を作ってください。私たちは月刊ＰＣエンジンファン（弊社刊行）の雑誌を通じて、作品の投稿の受付とそれをＣＤ－ＲＯＭに収録して、作品の発表の機会を提供し、より多くのみなさんに「でべろ」で作った作品をご紹介したいと思っています。もちろん市販ソフトへの道やいろいろなプロへの登竜門としての場を提供していきたいと思っています。

今回の「でべろ」の企画にあたり、株式会社ハドソンさま、日本電気ホームエレクトロニクス株式会社さまにはＰＣエンジンの技術の開示を、こころよく承諾してくださいました。ほんとうにありがとうございました。

また、ハドソンコンピュータデザイナーズスクール株式会社さま、未来計画株式会社さまにも多大なるご協力をいただきました。ありがとうございました。

さらに、たくさんの方々にもご協力をいただいたこと、感謝し、御礼申し上げます。

1996年1月10日　月刊ＰＣエンジンファン　編集長　北根紀子／副編集長　福成雅英

はじめに

C O N T E N T S

セットの中身について

PC-98シリーズ
VX以降の機種、286以上のCPU、Ver.3.3C以降のDOSが必要。

MSXシリーズ
MSX2以降の機種。

でべろBOX
1万円で販売中。詳しくは294ページを参照。

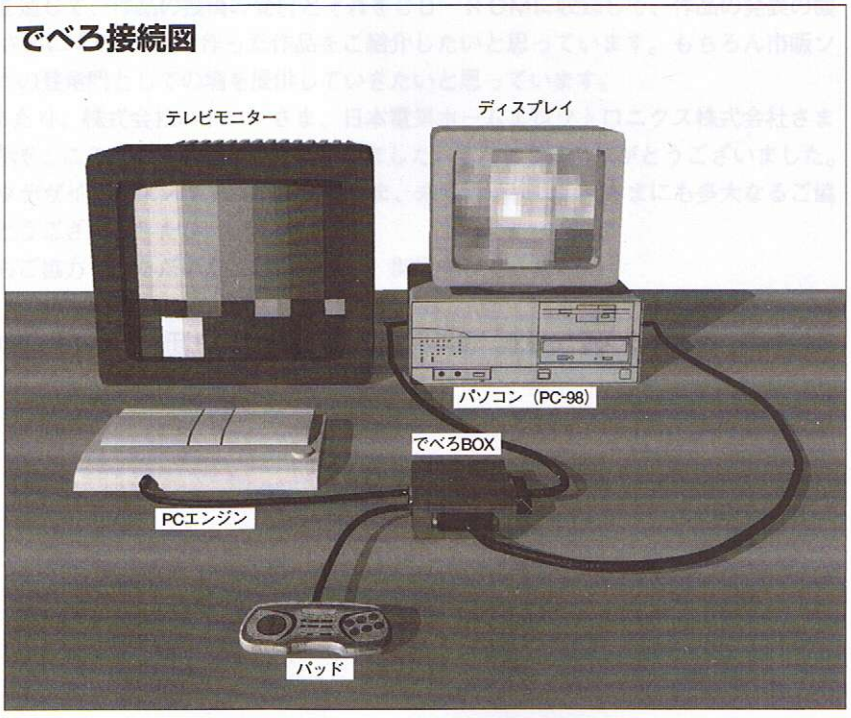
ケーブル
でべろBOXとパソコンを接続するケーブルは2種類必要。PC-98シリーズは、プリンターケーブルとRS-232Cケーブル。RS-232Cケーブルは、かならず全結線のストレートのものをご用意ください。MSXシリーズは、プリンターケーブルとジョイポート専用ケーブル。MSXのジョイポート専用ケーブルは、市販品が手に入りにくいため、弊社で通信販売しております（2000円）。ご利用ください。

でべろでソフト開発をするには

でべろでは、PCエンジンとパソコンを接続してプログラムやデータをやりとりします。接続図は、下図のようになります。PCエンジンは、スーパーCD-ROM²システムが動作する機種に限ります。パソコンは、PC-98シリーズか、MSXシリーズがご使用になれます。

PCエンジンとパソコンとを接続するには、でべろBOXというハードが必要になります。弊社で通信販売しておりますので、294ページの案内をお読みになり、お申し込みください。でべろBOXには、パソコンとつなぐためのケーブルは付属しておりませんので、市販のケーブルを別にご用意ください。

でべろ接続図



技術情報は本書に、ツールは付録に収録

本書には、PCエンジンのソフト開発をおこなうための技術資料が詳細に解説されています。従来は、ソフトメーカーにのみ開示されていた資料で、一般に公開されるのははじめてのことです。

本書の構成は、BIOSやでべろの外部コマンドに代表されるソフトウェアの解説と各CPUの構成やレジスタなどに代表されるハードウェアの解説とに大別されます。どちらの資料もプロ用の開発マニュアル（Hu7システム）から一部引用させていただきました。

開発システムのツール類は、Hu7システムとは異なるため新たに開発しました。PCエンジンとパソコンとで通信するための基本システムをはじめ、でべろで使用するDOSの外部コマンドやアセンブリ言語、ツール類などは、付録のCD-ROMに収録されています。

付録のフロッピーディスクには、PCエンジン・スーパーCD-ROM²専用の付録CD-ROMから必要なツール類をパソコンのディスクにダウンロードするためのプログラムが入っています。まずは、このプログラムをインストールすることからはじめてください。

BIOS
188ページ以降を参照してください。

各CPUの構成
35ページ以降を参照してください。

レジスタ
35ページ以降を参照してください。

Hu7システム
プロ用の開発システム。PCエンジンのソフトはすべてこのシステムを使用しています。Huカード用の開発システム「Hu7システム」とCD-ROM²用の開発システムHu7CDシステム」の2種類の開発システムがあります。でべろの開発システムは、このプロ用の開発システムにくらべて機能は大きく劣るものの、でべろ用の開発ツールを多くのユーザーが作るので、優れたツールが生み出される可能性があります。

DOSの外部コマンド
でべろの開発ツール類は、ほぼDOSのコマンドとして収められています。コマンドライン上で外部コマンドを実行してください。詳細は271ページ以降を参照してください。

アセンブリ言語
でべろの開発言語は、現在までのところアセンブリ言語のみです。詳細は132ページ以降を参照してください。

ツール類
でべろによるプログラム開発をサポートするためのツール類。フリーソフトウェアや製品のお試し版等も収録しています。

ダウンロードするためのプログラム
外部コマンドのslaveコマンドのこと。パソコン側でこのコマンドを実行しておいてからCD-ROMから必要なファイルを転送します。

インストール
この場合は、slaveコマンドを自分のパソコンに作ること。付録のフロッピーを入れて、DOSのコマンドライン上でinstallと打ち込めば、PC-98かMSXかを判断してファイルを作ります。

でべろシステムの起動方法

付録CD-ROM
詳細は、284ページからの「付録CD-ROM・FDの使い方」の項をご覧ください。

MSX・FAN誌
1995年8月号で休刊したMSX専門のパソコン誌。読者の投稿プログラムに支えられ、MSX自体の生産が打ち切られてからも雑誌を発刊しつづけました。MSXは、アスキーが提唱した統一規格パソコン。

PCエンジン単体で遊ぶとき

でべろシステムは、PCエンジンとパソコンをでべろＢＯＸでつなぐことによってソフト開発の環境を実現しますが、付録のCD-ROMはPCエンジン単体でも動作します。

起動の方法は、一般のスーパーCD-ROM²システムのゲームソフトと同じです。CD-ROMドライブに付録のCD-ROMをマウントしたら、本体の電源を入れてください。パッドのRUNボタンを押すとCD-ROMが立ち上がります（写真左上）。

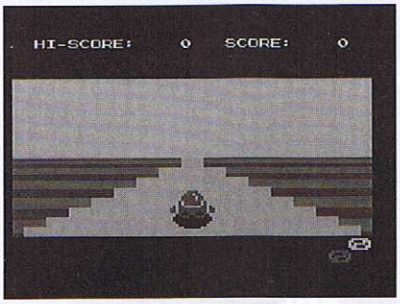
でべろの概要が知りたいときは、メニュー画面（写真右上）から「説明」のコーナーを選んでください。スライドとボイスででべろについて説明します。

また、でべろで制作した「（ゲーム）プログラム」（写真左下）や「ＣＧ」、「サウンド」は、各コーナーに収録されています。

「でべろ」（写真右下）と「転送」のコーナーは、でべろＢＯＸによってパソコンとつながっていないと使用することができません。

付録CD-ROMのタイトル画面。本体の電源を入れてRUNボタンを押すとこの画面になります

メニュー画面。点滅カーソルを移動し、選択



でべろで制作したゲーム『SPEEDER』。MSX・FAN誌の投稿作品をPCエンジンで再現

でべろシステムの画面。この状態でPCエンジンにプログラムやデータを送ることができます

パソコンとつなげるには

でべろでソフト開発をおこなうためには、でべろＢＯＸを使ってパソコンと接続する必要があります。ここでは、PC-98を例に、でべろＢＯＸとの接続について説明します。

PCエンジンと接続するためのケーブルが、でべろＢＯＸ本体から出ています。このケーブルをPCエンジンのパッド端子に接続します。つぎに、PCエンジンのパッドをでべろＢＯＸのパッド端子に接続します。マルチタップ等をお使いの方は、はずしてご使用ください。

パソコンとの接続は、2つの端子を使用します。まず、パソコン本体のプリンター端子とでべろBOXのプリンター端子を接続します。用意するプリンターケーブルは、でべろBOXの接続側が、セントロニクス仕様の36ピン端子を持つものです。つぎに、パソコンのRS-232C端子とでべろBOXのRS-232C端子を接続します。でべろBOX側の接続には、25ピン全結線のRS-232Cストレートケーブルをご用意ください。でべろには、ケーブル類は付属していません。

MSXとの接続は、プリンター端子はPC-98と同様ですが、RS-232C端子が標準装備されていませんので、ジョイポート２端子と接続します。ケーブルは、一端がメス、もう一端がオスのジョイポートケーブルが必要ですが、市販品が手に入りにくいケーブルですので、１本2000円で販売しております。ご希望の方は、294ページの案内をお読みになり、お申し込みください。

マルチタップ

PCエンジンのパッドを複数個接続するためのオプション。でべろでは、マルチタップによる動作を完全には確認していませんので、はずしての使用を推奨します。

プリンターケーブル

でべろＢＯＸ側の端子、セントロニクス仕様の36ピンは、多くのプリンターで採用されている一般的なものです。ただし、パソコン本体のプリンター端子には、様々な規格がありますので、お持ちのパソコンの取り扱い説明書をご熟読のうえ、ご用意ください。

RS-232Cケーブル

でべろＢＯＸ側の端子は、多くのモデムやパソコンで採用されている一般的なものです。かならず、25ピン全結線のRS-232Cストレートケーブルをご用意ください。RS-232Cケーブルには、ストレートのほかに、リバース（クロス）がありますので、お間違いにならないようお気をつけください。また、一部のケーブルで全結線ではないもの（すべてのピンが出ていないもの）を見かけます。全ピンが出ているかどうか、ご確認ください。

今キットでできること

peas
132ページ以降を参照してください。

CD-ROMそのものを制作することはできません
市販ソフトでは、ライセンス契約を結んだ会社以外はPCエンジンのソフトを製造、開発、販売することはできません。でべろで作ったプログラムをユーザー自身の手でCD-ROM化することはできませんが、弊社では投稿プログラムを募集しております。優秀作品はCD-ROMに収録して発表するという計画もありますので、294ページの案内をご覧のうえ、参加してください。

オンラインソフトウェア
パソコン通信など、パッケージによらない流通ソフトのこと。その種別により、フリーソフトウェア、シェアウェアなどがあります。

全ソース
でべろのシステムを構成する全ソースプログラムは、公開されています。このソースを参考にして自分のプログラムにいかすことも自由ですし、新しい開発環境を作り出すことも可能です。

パソコンでプログラミング

今回リリースしました本書『でべろスターターキット・アセンブラ編』と『でべろＢＯＸ』を利用しておこなえることを説明します。

パソコンでプログラミングし、でべろＢＯＸをつうじてPCエンジンにプログラムを送り込み、自作のプログラムを走らせることができます。そのためのアセンブリ言語用に、peasというアセンブラを用意しました。下のリストがpeas用に書かれたプログラムソースです（sound1.asm）。

でべろでPCエンジンのプログラムが作れますが、PCエンジンのCD-ROMそのものを制作することはできません。パソコンのフロッピーディスクなどを利用してプログラムを保存してください。オンラインソフトウェア等で配布することは自由ですが、実行するためには相手もでべろＢＯＸを含むでべろのシステムを持っている必要があります。

でべろシステムの全ソースは、付録CD-ROMの中に収められています。自作して、オリジナルのシステムを作ることも可能です。

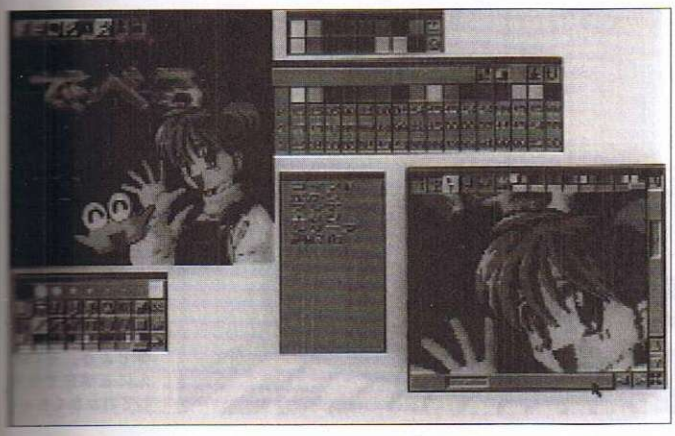
```
    include "develo.inc"

    org     $8000

Sound_Ch        equ     $0800
Sound_Vol       equ     $0801
Sound_FrqL      equ     $0802
Sound_FrqH      equ     $0803
Sound_ChVol     equ     $0804
Sound_Pan       equ     $0805
Sound_Wave      equ     $0806
Sound_Noise     equ     $0807
Sound_LFO       equ     $0808
Sound_LF_Ctrl   equ     $0809

    cla
    sta     Sound_Vol
    cla
    sta     Sound_LF_Ctrl

    cla
clear:
    sta     Sound_Ch
    stz     Sound_ChVol
    inc     a
    cmp     #6
    bne     clear

    cla
    sta     Sound_Ch
    lda     #$ff
    sta     Sound_Vol
    sta     Sound_Pan

    jmp     loop4

loop0:
    stwi    _ax,$e000

loop1:
    cla
loop2:
    dec     a
    bne     loop2

    cla
    jsr     ex_joysns
    tst     #1,joytrg
    bne     loop0
    tst     #2,joytrg
    bne     quit

    lda     (_ax)
    sta     Sound_Wave

    lda     #$ff
    sec
    sbc     _ah
    ora     #$c0
    sta     Sound_ChVol

    incw    _ax
    cmpwi   _ax,0
    bne     loop1

loop4:
    cla
    jsr     ex_joysns
    tst     #1,joytrg
    bne     loop0
    tst     #2,joytrg
    beq     loop4

quit:
    cla
    sta     Sound_Vol

    jmp     cd_boot
```

パソコンでCG作成

PCエンジンのグラフィックは、パソコンのグラフィックと異なり、特殊な方式を採用しています。同時発色512色相当というポテンシャルを持っていますが、8×8ドットの中には16色までしか使用することができません。この16色の集まりをパレットと呼びますが、PCエンジンではこのパレットを16まで持つことができます。つまり、16色の集まりが16パレットぶんで（16×16＝256）256色（実際にはパレットの一部が共通色のため241色）。8×8ドットの中で16色以上使いたい場合は、BG（バックグラウンド）とスプライトとの重ね合わせにより512色相当を実現することになります。

でべろでは、開発ツールとしてパソコン用のグラフィックツールを用意しました。PC-98では、C-Lab社のご厚意によりベストセラーソフト『マルチペイント』のお試し版が収録されています。MSXでは、MSX・FAN誌の投稿作品から『SII-ANIME』を収録しました。これら、パソコン用のグラフィックツールは16色までの対応になりますので、16色のデータコンバートツールを用意しています。『gp0.exe（PC-98）』、『gracon.com（MSX）』というツールで16色までのコンバートが可能です。

また、でべろ発売に先がけてモニター募集をしたところ、モニターの方から『でべろヴュアー（dv.exe）』というPC-98用ツールが送られてきました。グラフィックツールと組み合わせて使えば、グラフィックツールで作業中にPCエンジンへデータを転送してPCエンジン上で確認することができます。



マルチペイント。パソコンで絵を描いてからPCエンジンへコンバートして使用します

色

1つの色には、それぞれに512色から選ぶことが可能です。

BG（バックグラウンド）

PCエンジンでは、フル画面で表示できるグラフィックの表示色は256色相当までです。このグラフィックをBGと呼び、パソコンで描いた絵をそのままコンバートすることができるグラフィックです。ただし、8×8ドット中は16色しか使用できません。

スプライト

PCエンジンのスプライトは64個まで表示できます。256色以上使いたいところにスプライトを重ね合わせ、512色相当の表示能力を作り出せます。

グラフィックツール

お絵かきソフトのこと。どちらのソフトとも、マウスを使用します。使い方等は、ソフトウェアに付属のドキュメントファイルを参照してください。また、詳細は22ページを参照してください。

パソコンで作曲

PCエンジンは、6声からなるPSG音源と1声のADPCMを持っています。さらに、市販ソフトの場合は、CD-ROMによる音源、CD-DAも使用することができます。でべろでは、このうちPSGとADPCMが使用可能です。CD-DA音源を使用するには、CD-ROMを制作する必要がありますので、現時点ではサポートしていません。

PSG音源は、おもにゲーム中の効果音として使用されることが多く、コンピュータらしい特性の音といえるでしょう。初期のゲーム音楽を思い浮かべれば、想像できるでしょうか。PCエンジンのPSG音源は、かなり充実しており、波形データと組み合わせれば、FM音源並みの音色をカバーできます。ですから工夫しだいでBGMとして使用しても遜色ない曲が作れます。

PCエンジンには、比較的容易に音楽を演奏できるPSGドライバーが搭載されており、MML（ミュージック・マクロ・ランゲージ）のような手軽さで演奏させることが可能です。詳しくは第4部PSGドライバーの解説や第3部サウンドの解説などを参照してください。

ADPCMは、おもにボイスやドラム音などサンプリング音源として使用します。

いろいろな音源を駆使して、自分の思い通りの音楽を奏でよう。そのまえに五線符で作曲しておかなくちゃ

PSG音源
プログラマブル・サウンド・ジェネレータ。PCエンジンでは6声を持つうえに、波形データ（ウェーブ・サウンドと呼ばれる）と組み合わせて、音色を変化させることができます。PSGとは思えない豊かな音色が再現できるのはこのためです。

ADPCM
PCMとは、サンプリング音源と呼ばれる音源方式で、デジタルで録音したデータを再生して使用します。ADPCMとは、PCMデータを圧縮して録音しておく一方式のことです。

FM音源
PSGと並んで代表的なコンピュータ音源です。周波数変調（Frequency Moduration）によって豊富な音色を作成できるという特徴を持ち、多くのパソコンで採用されています。

CD-DA音源
CDに録音された音声（Digital Audio）のことです。市販の音楽CDと同等の音質を持っていますが、プロ用の開発環境がないとプログラムで使用できません。

でべろ入門

C O N T E N T S

コンピュータで絵を描く

パソコンでグラフィックを作る

PCエンジンのグラフィックはパソコンを使って描かれています。パソコン上でグラフィックツールというソフトウェアを動かして、その上でグラフィックを作っていきます。グラフィックツールとして、PC-98用には、C-Labから『マルチペイント』のお試し版を収録させていただきました。このツールを使って説明をしていきます。

ここで注意しておくことは、PCエンジンの画面のサイズです。PC-98は横640×縦400ドットが標準ですが、PCエンジンの表示画面はそれよりもかなりせまく、だいたい横は240～330ドットで縦は200～240ドット程度です。ここでは、でべろ外部コマンドのdspgrpがサポートしている横256ドットで描くことにします。

●マウスってなんだ

マウスというのは、ジョイパッドと同じ入力装置の一種です。形は楕円形でボタンが2つあり、頭からコードが出ていて、ちょうどネズミのような格好をしています。これを右手で軽く握り、人差し指を左のボタン、中指を右のボタンの上に置くのが一般的な持ち方です。こうして机の上などでマウスを滑らせると、お腹の部分にくっついているボールが、動いた方向や距離を読み取って、パソコン上のマウスカーソルを動かすというわけです。ジョイパッドが8方向、動いた距離はわからないのに比べると、とても細かく動きを伝えられるので、絵を描くという用途にはぴったりです。

それではここで、マウスを使うときの用語を説明しておきましょう。

マウスではボタンを押して離すことを「クリック」するといいます。右ボタンを押して離せば「右クリック」左ボタンなら「左クリック」です。また、押したままの状態にすることを「ドラッグ」するといいます。よく使うので、覚えておきましょう。

●PCエンジンでグラフィックを表示するには

マルチペイントを使って描いたグラフィックは、拡張子「.gp0」のGP0形式でセーブしておきます。PC-98ではgp0.exeを使ってください。GP0形式のファイルならばdspgrpを使用して、でべろに転送して表示することができます。

●タイルパターン使用上の注意

16色のグラフィックでは、足りない色数を補うために、タイルパターンという手法をよく使います。これは違う色を市松模様に組み合わせて、見かけ上の色を増やす方法です。しかし色の系統によってはPCエンジンで表示したときに、タイルパターンがちらついて表示されてしまいます。赤やオレンジ系の色で作ったタイルパターンのときに起きやすいようです。

マウスについて

形は楕円形だけではなく、円形などさまざまなものがあります。使いやすい形のものを選びましょう。また、ボタンが2つのものが主流ですが、3ボタンのものも存在します。

GP0でセーブ

マルチペイントの中からGP0でセーブしたいときは、mps.cfgをテキストエディタで開いて、「*CHILD MENU」以降に、
GP0セーブ=gp0 -s -fk
という行を加えてください(fkはかならず小文字)。そして、MPS¥UTYのディレクトリの中にgp0.exeを入れておきます。
これでマルチペイントのチャイルドプロセスに「GP0セーブ」という項目が増えます。これを選択すればセーブするファイル名の入力になり、GP0形式でセーブできます。セーブが無事終了してもマルチペイントがチャイルドプロセスエラーを出しますが、気にしないでください。

チャイルドプロセス

マルチペイント上から、別のプログラムを呼び出す機能です。

グラフィックツールの使い方

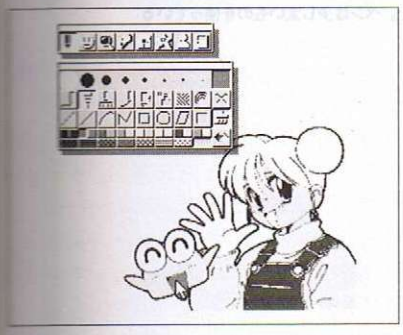
マルチペイントを起動すると、左上に小さいウィンドウが出て、いくつかアイコンが並んでいます。このアイコンをクリックして、いろいろな命令を伝えるのです。マウスを動かしてみてください。マウスカーソルが動きますね。そして今度は、マウスの左ボタンをドラッグしたまま、マウスを動かしてみてください。うまく線が描けましたか。これがグラフィックを描く第一歩です。

マルチペイントについての詳しい説明は、マルチペイントに同梱のドキュメントファイルをテキストエディタなどで読んでいただくとして、これからはCD-ROMのタイトルグラフィックを例にとって、描き方の実際を見ていくことにします。

まずは黒色のペンで、あゆみちゃん(女の子)や、でべろん(謎の生物)の輪郭を描いていきます。コネクトラインを使って少しずつ線を引いていくと、直線のガイドが出るのでやりやすいというテクニックもあります。(写真上)

大まかに線が描けたら、次は色を塗ります。塗りつぶす領域がハッキリしているときは、シードフィル(塗りつぶし)を選んで、その領域の一点をクリックすると、領域全体の色が変わります。(写真下)

一番細いペン先とコネクトラインで線を引いていく



大きい範囲からどんどん色を塗っていこう

ドキュメント

マルチペイント試供版のドキュメント(説明)ファイルは、マルチペイントを解凍してできるディレクトリ「MPS」のなかにある、「試供版.DOC」です。

マスク
変更したくない色をマスクウィンドウでチェックしておくと、別の色で上書きしてもその部分だけ変更されないという機能です。

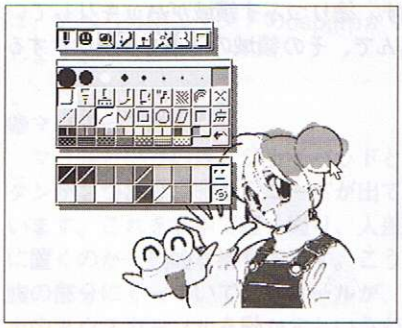
パレット
マルチペイントのパレットはRGBそれぞれ16段階ずつありますが、PCエンジン用にGP0形式でセーブすると各8段階にカットされてしまいます。パレットを変更するときは、PCエンジンで表示するときにそなえて、RGBの数値が偶数になるように変更するとよいでしょう。

マスク機能とルーペ機能

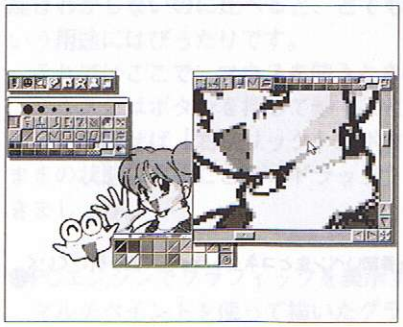
塗りつぶす領域の境界がはっきりしていないときは、シードフィルを使うと、色がはみ出て余計な部分まで塗られてしまいます。このときは、輪郭線の色をマスクしておいて、太いペン先を使って塗っていきます。(写真上)

大体を塗り終わったら、影をつけていきます。欲しい色がない場合は、パレットウィンドウを出して、使っていない色のパレットを変更して欲しい色を作ります。細かい部分はルーペを開いてその中で作業をするとやりやすいです。(写真中)

キャラクタができたら背景や文字を入れて完成です。(写真下)



輪郭線に使っている黒の他に、顔の肌色もマスクウィンドウでチェックしておくといい



顔の部分に影をつける。濃い肌色を暗い部分に塗っていく。ペンは少し太いものを使っている

でべろ付録CD-ROMのタイトル画面もこのようにしてグラフィックツールを使って作られたのだ

グラフィックツール作業中にPCエンジンで確認

今回収録したツールの中にdv(でべろビュアー)という便利なツールがあります。これは、PC-98で表示したグラフィックをPCエンジンに転送して表示するというものです。マルチペイントと組み合わせて使用すれば、作業中のグラフィックをPCエンジンに転送して確認することができます。このソフトは常駐パレットとマウスドライバがないと動作しませんが、マルチペイントと組み合わせて使用する場合は、マルチペイントにマウスドライバが内蔵され常駐パレットも作成されていますので、そのまま動作します。

使い方は、DOSのコマンドライン上で

dv

と入力します。このときにdv.exeとdvbat.tmpがカレントディレクトリにないと正常に動きませんのでご注意ください。

dvが起動すると画面にマウスカーソルが表示されます。マウスカーソルでPCエンジンに送りたい絵の左上端(転送開始位置)をクリックしてください。パレット、アトリビュートテーブル、キャラクタジェネレータの順に転送します。転送にはsetpltやsetvramなどのでべろ外部コマンドも使用していますので、これらのでべろ外部コマンドの存在するディレクトリにパスを通しておいてください。

また、オプションを設定することによって、パレットだけ、キャラクタだけの転送など限定されたデータの転送ができたり、転送ブロックの先頭を指定することによってマウスドライバがなくても動かすことができたりします。使い方は「DV/?」として画面に出るヘルプを参照してください。

●dvsize.exe

dvは画面のサイズは変更してくれませんので、でべろの基本画面でdvを使用したときには縦長の画像になり、上下が画面の外にはみ出してしまいます。

dvsize.exeはそんな画面を画像の縦のサイズに合わせて画面モードを設定します。コマンドラインから

dvsize〈画像の縦サイズ〉

と入力してください。

マルチペイントへの登録

mps.cfgをテキストエディタで編集します。「*CHILD MENU」以降に、

ビュアー=dv

などいう行を追加します。するとマルチペイントのチャイルドプロセスメニューに「ビュアー」と表示され、この文字をクリックすることにより、マルチペイント内からdvが起動されます。

カレントディレクトリ

現在作業中のディレクトリのことです。

でべろビュアー作業ファイル

dvbat.tmp:dv050.lzhに同梱され、アトリビュートテーブルとして転送されます。dv.exeと同じディレクトリに置いておきます。

dvpal.tmp:転送されるパレットファイルです。このファイルが作られるためには常駐パレットが必要です。dv.exeがカレントディレクトリにないと、うまく転送されません。

dvchar.tmp:キャラクタジェネレータとして転送されます。

パスを通す

DOSの環境変数PATHにディレクトリを登録することにより、カレントディレクトリ以外からも実行ファイルを起動できるようにすることです。DOSのコマンドラインから「SET PATH=」のあとに探索するディレクトリを「;」(セミコロン)で区切って書き並べます。詳しくはDOSのマニュアルなどを見てください。

gracon.com
CD-ROMの「転送」コーナーから、「周辺ツール個別転送」の「GRACON」を選んで転送し、PMextでgracon.lzhを解凍してください。

MSXでの描き方

MSX用のグラフィックツールとして『SII-ANIME』を収録してあります。MSXのスクリーン5専用で、市販されている『グラフサウルス』のスクリーン5と同じ形式の画像ファイルを扱うことができます。グラフィックを描く方法はPC-98の『マルチペイント』となんら変わりません。ただ、MSXのスクリーン5は256×212ドットまでの画像しか扱うことができません。

絵を描いたあとは、忘れずにセーブしましょう。『SII-ANIME』はSC5形式でグラフィックを保存します。この保存したファイルをPCエンジンで表示するためには、GP0という形式にコンバートします。

gracon.comというMSX-DOS用のグラフィックコンバータがありますので、これを使い、SR5形式からGP0形式にグラフィックを変更することができます。SR5形式だけでなく、圧縮していないGE5形式やBASICのBSAVE形式（拡張子SC5）にも対応しています。

GP0形式のファイルは、でべろのdspgrpというコマンドでPCエンジンに表示することができます。

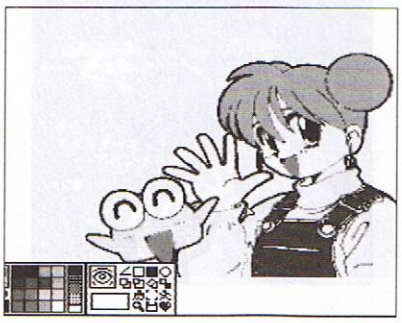
●SII-ANIME起動

MSXでBASICから「siianime.bas」を起動すると、タイトル画面が表示され、しばらくするとウィンドウが2つとマウスカーソルが表示されます。

SII-ANIMEには3つのウィンドウがあります。コマンドウィンドウ、パレットウィンドウ、そしてルーペコマンドで現れるルーペウィンドウです。コマンドアイコンを左クリックすると、マウスカーソルが尻尾付きの形に変わり、それぞれのコマンドを実行します。ウィンドウは左上のボタンで、場所を移動することができます。

●コマンドウィンドウ

左下の長方形をクリックするとコマンドウィンドウのボタンが簡易ルーペになり、マウスカーソルの周囲を拡大表示します。コマンドウィンドウ上で右クリックするとトレースモードの変更です。TPSETはコピーの時にパレットの0番に対して透明色の扱いをします。



MSX・FAN1993年2月情報号に掲載されたグラフィックツールだ。より詳しい説明が出ているので、持っている人は読んでみて欲しい。

SII-ANIMEの14個のコマンドアイコンを左上から説明します。
・連続線……始点と終点を指定して直線を引きます
・ボックス……対角の頂点を指定して長方形を描きます
・ボックスフィル……中身を塗り潰したボックスのを描きます
・サークル……円を描きます
・当倍コピー……対角で指定した長方形のパターンをコピーします
・自由倍コピー……対角で指定した長方形のパターンを別の場所に対角を指定して、好きな大きさの長方形にコピーします
・回転コピー……長方形を自由な角度に回転してコピーします
・カラー変更……上段を下段の色に変更します。色を指定後、変更する領域を選択します
・ペイント……描画色で指定した領域を塗り潰します
・画面補正……画面の位置の微調整です。カーソルキーを使用
・アニメーション……簡易アニメーション機能
・ルーペ……ルーペウィンドウを開き、細部の編集をおこないます
・ディスク……画像のロード、セーブをおこないます。テキストの画面になるので、キーボードから入力します
　1）LOAD　現在のページに画像とパレットをロードします
　2）SAVE　現在のページの画像とパレットをセーブします
　3）FILES　ディスク内の指定した拡張子のファイルの一覧です
　4）XXXX　作業対象の拡張子を変更します
　5）RET　キャンセルです
・アンドゥ……コマンドを実行する直前の状態に戻します

●パレットウィンドウ
・パレットウィンドウボタン……左クリックでウィンドウの移動です。右クリックはパレットのコピーをします
・PG……2画面あるページを切り換えます
・パレットナンバー……クリックでパレットのセットを変更します。各ページに0～7まであります
・真ん中の16のパレット……左クリックで描画色を選択します。右クリックで色を変更します。現在のRGBが表示されるので、RGBごとにクリックして変更してください
・タイルパターン……上下に好きな色を入れ、3段階のタイルパターンを作ります

●ルーペウィンドウ
拡大されたグラフィックが表示され直接編集できます。拡大元をクリックすると、編集領域を変更できます。ルーペウィンドウの外で右クリックすることで、ルーペの作業を終了します。
・カーソル……矢印のボタンは、拡大された編集領域を移動します
・ドロー……ルーペ内を自由線で編集します
・ペイント……ルーペ画面内の塗り潰しです
・倍率変更……右下の数字で拡大倍率が2、4、8倍と変わります

アニメーション
アニメーションさせたい絵と、表示させる場所を、コピーと同じ方法で指定します。すると「No.n: WAIT XXXX～」と表示されますので、選択した絵をどのくらい表示するか、WAITの値を左右クリックで増減させ、「OK」をクリックしてください。
この要領で次々にアニメーションさせたい絵を指定していき、すべて指定し終わったところで右クリックすると「実行・中止」が表示されます。実行をクリックすると今まで指定した順番にアニメーションを始め、右クリックでストップ、中止のクリックでキャンセルです。

アンドゥ
サークル、3種類のコピー、カラー変更、ペイントは1手前、それ以外はコマンドを選択した時点へ戻ります。ただしページを切り換えたときは、その時点の状態が保存されますので注意してください。アンドゥのアンドゥ（リドゥ）はできません。

コンピュータで音を奏でる

getfil
でべろ外部コマンドのひとつで、CD-ROM内のファイルをパソコンに転送する。

pc.mx
CD-ROMに収録した曲は、チャンネル6をパーカッションモードにして演奏しています。そのため、CD-ROM内のpc.mxを一緒にロードしてやる必要があります。pc.mxの中では、休符をパーカッションデータテーブルのPC0（Cの音）に割り当てたり、それ以降は順番にバスドラム、スネアドラム、ミッドタムなどの音色が配置してあります。詳しくはpc.mxのソースである、pc.asmを見てください。
また、250ページ、トラックデータのモード説明内の、パーカッションモードでの休符の指定の仕方や、261ページのパーカッションデータも合わせて参照してください。

PCエンジンの音楽を聴こう

でべろ付録CD-ROMに収録されているPSGの曲は、メニューのサウンドコーナーから聴くことができますが、でべろ外部コマンドplytrkでパソコンから演奏させることもできます。

演奏する方法は、DOSのコマンドラインから、

plytrk〈曲のファイル名〉pc.mx

と入力してください。CD-ROM内に収録されているPSGの曲とそのファイル名は次の通りです。

ファイル名	曲名
labylnt.mx	「The Labyrinth Of Devil」
step.mx	「Step !」
beast.mx	「メカナイズド・ビースト」
saphire.mx	「砂漠のサファイア」
tako.mx	「タコ&イカ　SUPER」

この他にパーカッションのデータとして、同じCD-ROM内のpc.mxというファイルが必要です。CD-ROMメニューの転送コーナーを利用するか、外部コマンドのgetfilを使用するかして、パソコンに転送しておいてください。

plytrkを実行すると、曲データ、パーカッションデータの順に転送後、PCエンジンでの演奏が始まります。曲が終わるか、IIボタンを押すかで終了しますが、「STEP！」以外の曲は、データの終わりがないエンドレスの曲ですので、IIボタンで終了させてください。

でべろでの音楽づくり

PSG音源はプログラマブル・サウンド・ジェネレータの頭文字をとったもので、プログラムでいろいろな音を発生させることができる装置です。音の長さや音の高さ、音色までもプログラムでコントロールできるので、音楽、ゲームのBGMづくりに最適な音源です。ウェーブテーブルというところに音色を登録しておき、エンベロープで音の鳴り方を設定したり、LFOでビブラートなどの効果を付けたりといった、芸の細かいこともできます。

ところで、PSGを使うにしても、まず作曲、入力するもとの曲を作らないといけないのですが、こればかりはなかなか解説が難しいところです。

心の中に浮かんだちょっとしたメロディをいろいろ展開させていって作るとか、好きな曲のコード進行やリズムパターンなどを借りて、それにメロディを付けたりするとか、曲の進行に起承転結を入れてメリハリを付けながら作るなど、いろいろなやりかたがあるようですが、はっきりこうだという作り方はないようです。収録した曲のソースは、拡張子がMMLというファイル名でCD-ROM内にありますので、参考にしてみてください。

ただし、PCエンジンのPSGは、6チャンネルしかありません。これは楽器が6つしかないようなものですから、フルオーケストラの演奏をすることはできません。しかもそのうち2チャンネルしかパーカッションに割り当てられませんから、同時に3音以上のドラムを鳴らすなどということもできません。この限られた制約の中でどのような曲が作れるのかが、腕の見せ所なのかもしれません。

PSG音源を使って作曲し演奏するためには、テキストエディタ、MMLコンバータ、peas、plytrkが必要になります。CD-ROMの中からパソコンに転送しておいてください。

音色

音色はウェーブテーブルを使って変更します。これは音を時間と振幅に分けたテーブルのことです。時間ごとの電圧の変化を表し、これが人間の耳には音色の違いに聞こえます。

PCエンジン内蔵のウェーブテーブルは、1周期を32に分けた単位時間あたりに32段階（5ビット）の振幅で表現し、HuC6280のR6波形レジスタで操作しています。この中で自由に波形を設定して、目的の音色を作っていくわけです。PSGドライバーが使えるときはトラックデータのウェーブ（$E5）を使って変更します。

プログラミングとは

マシン語とアセンブリ言語

PCエンジンで動作するプログラムを作るには、プログラムがどんなものかを知る必要があるでしょう。プログラムとは、演劇でいうシナリオみたいなもので、PCエンジンにどこで何をさせるかといった情報の集まりです。

PCエンジンのような機械にも言葉があり、その機械が理解できる言葉を使って何をどうするかを指示していきます。この言葉をマシン語といいます。マシン語は0〜255の数字の集まりなので、人間には理解するのが難しいという難点があります。そこでアセンブリ言語です。

アセンブリ言語とは、人間に理解しやすく作られた言葉のことで、これをアセンブラというプログラムでマシン語に変換することができます。

PCエンジンのプログラムを組んで、何をどのようにおこなうかを指示するためには、まずアセンブリ言語を覚えなければいけません。

アセンブリ言語のプログラムは、命令（ニーモニック）と、命令で用いるレジスタ名や数値などといったパラメータ（オペランド）から構成されています。「何を」にあたるのがオペランドで、「どうするか」がニーモニックなわけですね。

ニーモニックに用意されているのは、レジスタやメモリにデータを転送したりする、単純で基本的な命令だけです。ニーモニックは、CPUというPCエンジンの脳にあたる部品に対する命令です。脳は目や耳や口を通して見たり聞いたり話したりができますが、脳単体ではそれらはおこなえません。同様に、CPUだけでは画面にグラフィックを表示したり音楽を演奏させたりはできないのです。

CPUに対して、PCエンジンの目や耳や口にあたるのが、画面表示をつかさどるVDCやVCE、音楽を演奏するPSGで、これらは自らの役割をマイペースでこなすCPUの協力者です。そしてこれらに対しては、BIOSを呼び出したり、特定のメモリにデータをセットしたりして動作の指示を出していきます。呼び出しやデータのセットは、ニーモニックで指示できます。

このように、プログラムでは、PCエンジンに何をどうするかの指示が記されていきます。

始めは1つの目的（例えばキャラクタを表示するなど）から、順に1つずつ試して覚えていきましょう。その際、付録CD-ROMのサンプルや、本書の解説や使用例などを参考にしてください。

数字の集まり
プログラミングの世界では、2進数という数字が用いられます。それぞれの桁が"0"と"1"の2種類しかなく、2まで進むと桁上がりする数です。2進数は通常4桁単位でまとめられた16進数という数で扱われます。16進数は16まで進むと桁上がりする数で、2進数4桁ぶんで1桁に対応しています。なお、16進数の各桁の10〜15にあたる数は、"A"〜"F"のアルファベットを用いて表現します。

マシン語に変換する
「アセンブルする」とも言う。付録CD-ROMに収録されているpeas.exe（MSXではpeas.com）がPCエンジン用のアセンブラプログラム。

ニーモニック
第2部を参照。

レジスタ
CPUの手足となって活躍するメモリ。データの仮置き場や、特定の情報を管理したりしている。

VDC
ビデオ・ディスプレイ・コントローラ。画面にどのような情報を表示するかなどを管理している。

VCE
ビデオ・カラー・エンコーダ。VDCから受けた画面への出力情報を、色コードに基づいて実際の表示色に変換する。

PSG
プログラマブル・サウンド・ジェネレータ。与えられた情報から音を作り出して鳴らす働きをする。

BIOS
BIOSは、予め用意されたプログラムの処理集のようなもので、とくにCPU単体ではおこなえない処理を中心に用意されている。それぞれの処理ごとに定められたデータを設定した後でサブルーチンコールすると、目的の動作をおこなってくれる。

アセンブラによって変換されるプログラムをソースプログラムもしくはアセンブリソースといいます。ソースプログラムはテキスト形式のファイルで、テキスト形式のファイルが作成できれば、ワープロソフトなどを使って書くこともできます。

ソースプログラムのうち、ニーモニックの命令が記述されている部分をコード部と呼びます。実際にマシン語のプログラムに変換される部分ですね。ソースプログラムには、他にラベルやマクロ命令を定義する定義部と、データやワークエリアを置くデータ部があります。定義部は、演劇を例にとれば役者の名前や大道具・小道具の名前を設定しているような部分で、データ部は舞台セットの配置だとか、色や役者のプロフィールなど、情報が記されている部分です。

通常は、定義部→コード部→データ部の順で記述します。コード部で使用されるラベルやマクロ命令は、あらかじめ定義部で設定されていないとアセンブルできないからです。ソースプログラムの記述などについては、第3部プログラミング解説の項を参照してください。

■ソースプログラムの例（wave.asm）

```
    org     $8000

    include "develo.inc"

Start:
    clr     _dh
    jsr     dv_Screen1

    lda     #5
    sta     _al
    lda     #7
    sta     _ah
    lda     #2
    sta     _dh
    jsr     dv_Screen1

    stwi    _bx,TitleStr
    lda     #4
    sta     _dh
    jsr     dv_Screen1
            :
            :
            :
```

ワープロソフト

いくつかのワープロソフトでは、編集情報などが付加された独自の保存形式をおこなうものもあります。このようなソフトの場合、ソースプログラムの作成には不適です。

付録CD-ROMにはソースプログラムの作成に使えるテキストエディタも収録されています。PC-98シリーズはEditEngine、MSXシリーズはv&z_smallです。

EditEngine
付録CD-ROMに収録されているee070.lzhがEditEngineです。でべろCD-ROMの転送メニューからPC-98上に読み込んで使用してください。

v&z_small
付録CD-ROMに収録されているv&zsmall.lzh がv&z_smallです。でべろCD-ROMの転送メニューからPC-98上に読み込んで使用してください。

CD-ROM収録のテキストエディタ

付録CD-ROMには、PC-98シリーズ用のテキストエディタ「EditEngine」とMSXシリーズ用のテキストエディタ「v&z_small」を収録してあります。

●EditEngine

PC-98シリーズ用に収録したEditEngineは、テキストエディタ本体と複数のデータファイルからなります。さまざまな機能をもっていますので、まずはee.doc（EditEngineの説明書）をよく読んでください。

起動方法は、DOSのコマンドラインから、以下のようにコマンド名を入力すればOKです。

A>EE

コマンド名の後、スペースを開けてオプションスイッチやファイル名の指定をすることもできます。これらの指定には順序などの制限はありません。なお、ファイル名が省略された場合には、ファイル選択画面になり、そこで編集するファイルを指定することになります。

●v&z_small

テキストエディタのv&z_smallは、vs.comとzs.comの2つからなります。vs.comは漢字（2バイト文字）を使用しない場合のエディタで、zs.comは漢字も扱えるエディタです。

起動方法は、DOSのコマンドラインから、以下のようにコマンド名を入力すればOKです。

A>vs　　……vsmallを起動する場合

または

A>zs　　……zsmallを起動する場合

また、起動時に編集ファイル名を指定することもできるので、まずは

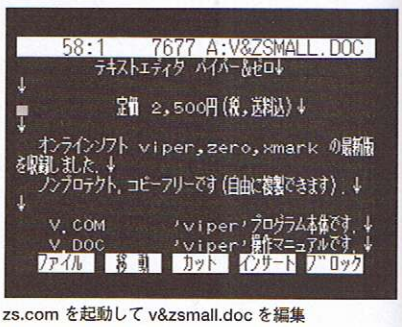
A>zs v&zsmall.doc

と実行して、v&z_smallのドキュメントファイルを読んでください。

■EditEngine

ee.exe を起動して ee.doc を編集

■v&z_small



zs.com を起動して v&zsmall.doc を編集

アセンブラ（peas.exe／peas.com）の使い方

アセンブリ言語で書かれたプログラムを、マシン語に変換してくれるのがアセンブラの役割です。せっかく作ったプログラムも、アセンブラでマシン語に変換しなければ動作させることができません。

PCエンジンのプログラムをアセンブルするには、付録CD-ROMに収録されているpeasという外部コマンドを使います。以下では、同じく付録CD-ROMに収録されているスプライト表示のサンプルを例にとり、アセンブルの手順を解説します。

まず、peasと同じディレクトリに、アセンブルする目的のソースプログラムファイルを置きます。peasでアセンブルできるのは、拡張子がasmのファイルに限られます。スプライト表示のサンプルでは、sprite.asm と develo.incの2つのファイルが必要なので、どちらもpeas と同じディレクトリに置いておきます。

次に、PC-98シリーズの場合は、DOSのコマンドラインから

```
>peas sprite
```

のように入力して実行します。MSXシリーズの場合には、MSX-DOS上で

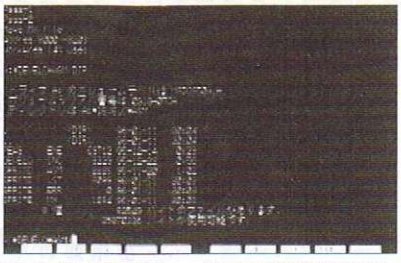
```
>peas sprite /o
```

のように入力して実行します。

正しくアセンブルされた場合には、実行したディレクトリ内に、同じファイル名で拡張子がmxのファイルが作成されます（下の写真参照）。

■PC-98シリーズのアセンブル

「peas sprite」を実行してアセンブルする



ディレクトリを見ると、sprite.mx が作成されている

■MSXシリーズでのアセンブル

MSX版では /o などのスイッチを忘れずに

RAMディスクのほうが早くアセンブルできる

マシン語に変換

アセンブラのpeasでは、完全にマシン語に変換されるわけではありません。peasで変換した後に作成されるファイルは、拡張子がmx のテキストファイルです。このファイルをマシン語のバイナリファイルに変換したい場合には、mx2bin コマンドを使用します。なお、でべろ外部コマンドでは、拡張子mxのファイルの状態で、PCエンジンに転送して実行することができます。

スプライト表示のサンプル

156ページのソースプログラムリストです。付録CD-ROMの中のsprite.asm がそれです。

中身がない
拡張子errのエラーファイルのサイズ表が「0byte」になっている状態を意味します。

どんなエラー
エラーファイルに記録されるエラーメッセージの一覧が153ページに掲載されています。

デバッグ作業
プログラムに存在するミスをバグといい、それを取り除いて修正する作業をデバッグといいます。

MSXでアセンブルする場合
MSXでpeasを使ってアセンブルする場合、変換作業に若干の時間を要します。2度3度と同じ作業を繰り返さないためにも、実行する際は /o と /e のオプションは付けておきましょう。

アセンブラで出力されるファイルのうち、拡張子がerrのファイル（エラーファイル）にはアセンブル時に見つかったエラーの情報が、拡張子がlstのファイル（リスティングファイル）には変換後のアドレスやマシン語コード、エラーの情報が記されます。

アセンブル作業が正常におこなわれてmxファイルが作成された場合は、エラーファイルには中身がありませんが、もしソースプログラムにミスがあり、アセンブル時にエラーが発生した場合には、エラーファイルにソースプログラムの何行目でどんなエラーが発生したかの情報が入ります。アセンブルがうまくいかなかったときは、エラーファイルの中を調べて、ソースプログラムを修正しましょう。

アセンブルが正常におこなわれた後も、プログラムにミスがあるかもしれません。アセンブル後のプログラムを実行したとき、期待した結果が得られなかった場合には、リスティングファイルの情報を参考に、デバッグ作業をおこないましょう。

MSXシリーズのパソコンを使ってアセンブルする場合は、オプションを付加してアセンブルしないとエラーファイルやリスティングファイルが作成されません。それぞれ、エラーファイルを出力するときは /eを、リスティングファイルを出力するときは/lを付けて実行します。

また、MSXで大きなプログラムをアセンブルするときは、ディスクの残り容量に注意してください。容量が足りないと、これらのファイルを作成できなくなります。

■リスティングファイルの中身

```
アドレス  マシン語コード          ソースプログラム
8000 :                  org     $8000
8000 : 62               cla
8001 : 206CE0           jsr     ex_dotmod
8004 : 62               cla
8005 : 2069E0           jsr     ex_scrsiz
8008 : 62               cla
8009 : A220             ldx     #32
800B : A01E             ldy     #30
800D : 206FE0           jsr     ex_scrmod
8010 :                  stwi    sat_adr,$2000
8010 : A920             lda # high ( $2000 )
8012 : 8D1522           sta sat_adr + 1
8015 : A900             lda # low ( $2000 )
8017 : 8D1422           sta sat_adr
801A : 20A2E0           jsr     ex_satclr
                 ⋮
                 ⋮
                 ⋮
```

第1部
ハードウェア解説

CONTENTS

ハードウェア概説

概要

HuC62シリーズのICは、CRT画面上に表示するカラーアニメーションを主体とした画像を、従来にはない多色を用いて表現させるばかりでなく、パターンの高速移動性による動きの速い画面の表現、そしてパターンの反転や画面のスクロールを簡単な手続きで表示できる機能、さらにはこれらのダイナミックにあふれた画面の動きを多彩な効果音でサポートできる音声発生部の充実など、豊富な機能を持ったICセットです。

特長

●各機能を有機的に分担する3つのICの組み合せ
　①HuC6280……CPU+PSG（プログラマブル・サウンド・ジェネレータ）
　②HuC6270……VDC（ビデオ・ディスプレイ・コントローラ）
　③HuC6260……VCE（ビデオ・カラー・エンコーダ）
●CRT（TV）接続の容易性……RGB出力+映像色信号出力
●多色表示機能……512色（RGB各色8段階で設定可能）
●画面構成方式……バックグラウンドとスプライト画面の組み合せ
●多彩な画面を作り出す強力なレジスタ機能……ブランキング
　水平、垂直方向へのスムーススクロール（バックグラウンド）
　スプライトの反転表示
　スプライトの合成
●画面の状況にすばやく対応する割込み機能
●低消費電力……すべてのICをCMOSで実現
●単一電源……5V（アナログ部含むすべてのIC）

システムの動作概要

HuC62システムは、基本的に3個の専用ICと専用メモリ、そしてクロック発生回路より構成されます。またこの基本システムをサポートするものとして、楽音用の増幅器とスピーカ、CRT画面表示用のRGB信号用I／F回路、およびコンポジットビデオ信号用I／F回路、カラーCRT部より成り立っています。

3個の専用ICとは、全体を制御するCPUと音楽を発生させるPSG（プログラマブル・サウンド・ジェネレータ）とを1個のICにまとめたHuC6280、そしてVDC（ビデオ・ディスプレイ・コントローラ）としてカラー画像データを発生するHuC6270、さらにVCE（ビデオ・カラー・エンコーダ）としてVDCから送られてくるカラー画像データをもとにTV画像用のRGB信号とさらに映像色信号を作り出すHuC6260です。

一方メモリについては、CPU用のRAMとROM、そしてVDC用外部メモリとしてカラー画像データを保持するVRAMが付随します。

CPU（HuC6280）

HuC62システムの核となる中央制御処理部です。

プログラムを読み出して画像構築に必要なデータを各ICに送り込むほか、画面と一体となる音楽データをPSGに送り込みます。

VDC（ビデオ・ディスプレイ・コントローラ、HuC6270）

バックグラウンドとスプライトの組み合わせで画面を構成します。VDCではこのバックグラウンドとスプライトのそれぞれについて、パターン宣言と色コード、画面のどの位置に表示するかのデータを定義します。このデータは逐次VCEに送られて最終画面情報となります。

また割込み端子（IRQ）が用意されているため、画面上に生じる各種の現象に対してCPUに随時その現象を知らせることが可能です。

現象例

①スプライトどうしの衝突が起きたとき
②目的のラスタ位置となったとき
③垂直帰線期間に入ったとき、ほか

・VRAMについて

VDCに付随する形でVRAMを使用します。

このVRAMにバックグラウンドの画面データ、スプライトの画面データをストアしておきます。

VCE（ビデオ・カラー・エンコーダ、HuC6260）

ビデオデータ出力ポート（VD0～VD8）を通じ、VDCから送られてくるデジタルカラー画像信号をもとに、VCE内部に持っているカラーパレット情報（これはカラーコード情報であり、あらかじめCPUより送られている）と突き合わせて、CRT用アナログRGB信号および映像色信号を作り出します。

PSG（プログラマブル・サウンド・ジェネレータ、HuC6280）

楽音（メロディ音、リズム音など）または擬音を発生させるブロックです。

CPUブロックと一体のICで構成されています。

メモリとD／Aコンバータを組み合せた音楽発生方式であり、CPUから送られてきたデータをもとにステレオ回路構成、ノイズ発生回路、低周波発生回路などの機能により画面と一体となった効果音を作り出すことができます。

CPU(HuC6280)解説

概要

HuC6280は、1チップ上に8ビット並列処理ALU、アドレス拡張のための8つのマッピングレジスタ、7ビットのインターバルタイマ、8ビットの入力ポート、8ビットの出力ポート、さらにPSG（プログラマブル・サウンド・ジェネレータ）を内蔵したCMOS8ビットマイクロプロセッサです。
2Mバイトのアドレス空間を持ちます。

特長

●モノリシックCMOS8ビット並列処理マイクロプロセッサ
●動作周波数……3.58MHz～21.48MHz（実使用時）
●最小インストラクション実行時間……276nsec
●インストラクションセット……89種234命令
●入出力ポート……入力ポート　1ポート×8ビット
　出力ポート　1ポート×8ビット
●割込み……外部割込み　NMI、IRQ1、IRQ2
　内部割込み　タイマ割込み、BRK命令
●7ビットのインターバルタイマ内蔵
●スタックエリア……256バイト（Max）
●単一電源……5V
●パッケージ……80ピンプラスチックフラットパッケージ
●アドレス空間……8つのマッピングレジスタにより
　物理アドレス 2Mバイト
　論理アドレス 64Kバイトの構成。

レジスタセット

HuC6280は以下の計10個の8ビットレジスタを持ちます。
①汎用レジスタであるアキュムレータ（ACC）
②Xレジスタ（X）
③Yレジスタ（Y）
④プログラムカウンタハイ（PCH）
⑤プログラムカウンタロー（PCL）
　※このPCHとPCLでプログラムカウンタを構成します。
⑥スタックポインタ（S）
⑦ステータスレジスタ（P）
⑧ソースハイ（SH）
⑨デスティネーションハイ（DH）
⑩レングスハイ（LH）
　※SH、DH、LHはブロック転送命令時に使用します。

アキュムレータ（ACC）

ACCは、8ビット構成の汎用レジスタです。ステータスレジスタのメモリ演算フラグ（T）が"0"のときは、算術論理演算はおもにACCを中心に実行されます。このとき、ACCのデータはALUの入力データとなり、また演算結果はACCにストアされます。

また、ACCはメモリからメモリへのデータ転送や、メモリと周辺回路とのデータ転送に使用されます。

さらに、ブロック転送命令（TII～TDD命令）の実行時には、その時点のデータをスタックに退避した後でレングスローとして機能し、ブロック長のカウントに使用されます。

XレジスタとYレジスタ

XレジスタとYレジスタは、共に8ビット構成の汎用レジスタで、おもにインデックス・アドレッシングに使用されます。

Xレジスタは、ステータスレジスタのメモリ演算フラグ（T）が"1"のとき、演算のデスティネーションとなるゼロページのメモリアドレスを指定するのに使用されます。

さらに、ブロック転送命令（TII～TDD命令）の実行時には、その時点のデータをスタックに退避した後でソースローとして機能し、ソースアドレスを指定するのに使用されます。

Yレジスタは、ブロック転送命令（TII～TDD命令）の実行時には、その時点のデータをスタックに退避した後でデスティネーションローとして機能し、デスティネーションアドレスを指定するのに使用されます。

プログラムカウンタ（PCH、PCL）

プログラムカウンタは、プログラムカウンタハイ（PCH）と、プログラムカウンタロー（PCL）から構成される16ビットのアップカウンタです。

プログラムカウンタは、命令の実行にしたがって自動的にインクリメントされ、次に実行する命令のアドレスや、オペランドのアドレスを指定します。

システムリセットの後、

PCLには、物理アドレスで、$001FFE番地のデータを

PCHには、物理アドレスで、$001FFF番地のデータを

設定して、CPUはスタートします。

●プログラムカウンタの退避と復帰

a. BSR命令、JSR命令実行時

プログラムカウンタの値は、BSR命令またはJSR命令実行時、スタックに退避されます。スタックには、PCH、PCLの順にストアされます。スタックに退避されるプログラムカウンタの値は、BSR命令または、JSR命令の最後のバイトのアドレスです。したがって、RTS命令では、スタックに退避されていたプログラムカウンタの値をロードした後、それに1を加えてサブルーチンよりリターンします。

b. 割込み発生時

プログラムカウンタの値は、割込みが発生してもスタックに退避されます。スタックには、PCH、PCL、Pの順にストアされます。スタックに退避されるプログラムカウンタの値は、割込み処理ルーチンが挿入された次の命令のアドレスです。したがって、RTI命令では、スタックに退避されていたプログラムカウンタの値をロードして、割込み処理ルーチンよりリターンします。ステータスレジスタ（P）の値も元に戻されます。

c. BRK命令実行時

プログラムカウンタの値は、BRK命令の実行後スタックに退避されます。スタックに退避されるプログラムカウンタの値は、（BRK命令+2）のアドレスです。スタックには、PCH、PCL、Pの順にストアされます。したがって、RTI命令により復帰した場合、プログラムは（BRK命令+2）のアドレスにリターンします。ステータスレジスタ（P）の値も元に戻されます。

アキュムレータ（ACC）

ACCは、8ビット構成の汎用レジスタです。ステータスレジスタのメモリ演算フラグ（T）が"0"のときは、算術論理演算はおもにACCを中心に実行されます。このとき、ACCのデータはALUの入力データとなり、また演算結果はACCにストアされます。

また、ACCはメモリからメモリへのデータ転送や、メモリと周辺回路とのデータ転送に使用されます。

さらに、ブロック転送命令（TII～TDD命令）の実行時には、その時点のデータをスタックに退避した後でレングスローとして機能し、ブロック長のカウントに使用されます。

XレジスタとYレジスタ

XレジスタとYレジスタは、共に8ビット構成の汎用レジスタで、おもにインデックス・アドレッシングに使用されます。

Xレジスタは、ステータスレジスタのメモリ演算フラグ（T）が"1"のとき、演算のデスティネーションとなるゼロページのメモリアドレスを指定するのに使用されます。

さらに、ブロック転送命令（TII～TDD命令）の実行時には、その時点のデータをスタックに退避した後でソースローとして機能し、ソースアドレスを指定するのに使用されます。

Yレジスタは、ブロック転送命令（TII～TDD命令）の実行時には、その時点のデータをスタックに退避した後でデスティネーションローとして機能し、デスティネーションアドレスを指定するのに使用されます。

プログラムカウンタ（PCH、PCL）

プログラムカウンタは、プログラムカウンタハイ（PCH）と、プログラムカウンタロー（PCL）から構成される16ビットのアップカウンタです。

プログラムカウンタは、命令の実行にしたがって自動的にインクリメントされ、次に実行する命令のアドレスや、オペランドのアドレスを指定します。

システムリセットの後、

PCLには、物理アドレスで、$001FFE番地のデータを

PCHには、物理アドレスで、$001FFF番地のデータを

設定して、CPUはスタートします。

●プログラムカウンタの退避と復帰

a. BSR命令、JSR命令実行時

プログラムカウンタの値は、BSR命令またはJSR命令実行時、スタックに退避されます。スタックには、PCH、PCLの順にストアされます。スタックに退避されるプログラムカウンタの値は、BSR命令または、JSR命令の最後のバイトのアドレスです。したがって、RTS命令では、スタックに退避されていたプログラムカウンタの値をロードした後、それに1を加えてサブルーチンよりリターンします。

b. 割込み発生時

プログラムカウンタの値は、割込みが発生してもスタックに退避されます。スタックには、PCH、PCL、Pの順にストアされます。スタックに退避されるプログラムカウンタの値は、割込み処理ルーチンが挿入された次の命令のアドレスです。したがって、RTI命令では、スタックに退避されていたプログラムカウンタの値をロードして、割込み処理ルーチンよりリターンします。ステータスレジスタ（P）の値も元に戻されます。

c. BRK命令実行時

プログラムカウンタの値は、BRK命令の実行後スタックに退避されます。スタックに退避されるプログラムカウンタの値は、（BRK命令+2）のアドレスです。スタックには、PCH、PCL、Pの順にストアされます。したがって、RTI命令により復帰した場合、プログラムは（BRK命令+2）のアドレスにリターンします。ステータスレジスタ（P）の値も元に戻されます。

スタックポインタ（S）

スタックポインタは、8ビット構成のレジスタです。スタックポインタは、スタックの空領域の最上位アドレスの下位8ビットを指定しています。

スタックポインタは、スタックにデータをプッシュした後デクリメントされ、プルする前にインクリメントされます。

システムリセット後、スタックポインタの値は不定となります。したがってイニシャルルーチンにて必ず設定しなければなりません。

スタックポインタがアドレスバスに出力される場合、上位バイトは必ず$21となります。したがってHuC6280のスタックエリアは、論理アドレスで、$21FF番地～$2100番地の256バイト（Max）となります。

ステータスレジスタ（P）

ステータスレジスタは、8ビット構成のレジスタです。ビット構成を図1-2-1に示します。

ステータスレジスタは文字どおりCPUの状態を示すレジスタで、各ビットごとに個別の意味を持ち、互いに独立した存在です。

ステータスレジスタは、割込みが発生したとき、またはBRK命令を実行したとき、自動的にスタックに退避され、RTI命令で元に戻されます。

(1) キャリーフラグ（C）

キャリーフラグは、ADC命令において、MSBからの桁上がりがあるとセットされます。また、SBC命令やCMP、CPX、CPY命令において、MSBへの桁下がりがない場合にセットされます。また、SEC命令によりセットされ、CLC命令によりリセットされます。

キャリーフラグは、シフトやローテイト命令によっても変化します。

(2) ゼロフラグ（Z）

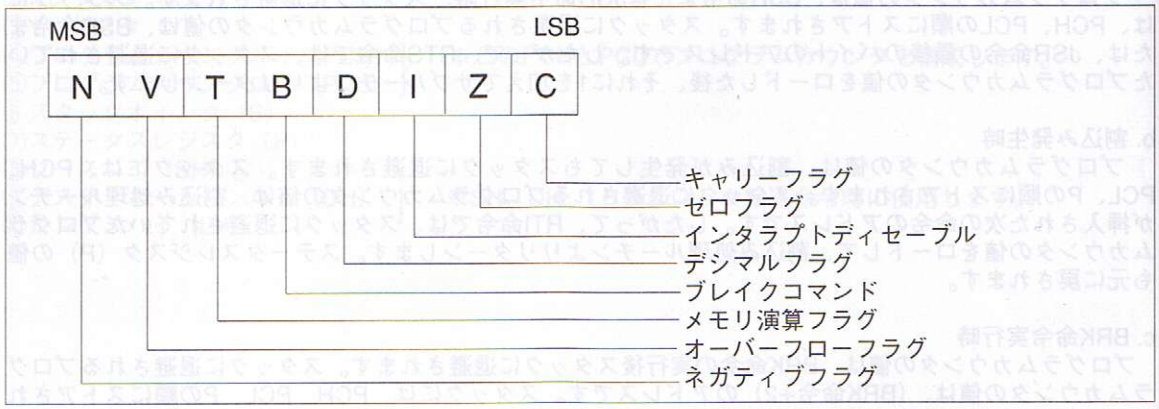
ゼロフラグは、算術論理演算、またはデータ転送の結果がゼロのときセットされます。

(3) インタラプトディセーブル（I）

インタラプトディセーブルは、システムリセットの後、または割込みが発生したとき、またはBRK命令を実行したときにセットされます。また、SEI命令によりセットされ、CLI命令によりリセットされます。

インタラプトディセーブルが"1"のとき、IRQ1入力またはIRQ2入力による外部割込み、またはタイマーによる内部割込みは発生しません。

図1-2-1 ステータスレジスタ（P）の構成



(4) デシマルフラグ（D）
デシマルフラグは、システムリセットの後、または割込みが発生したとき、またはBRK命令を実行したときにリセットされます。また、SED命令によりセットされ、CLD命令によりリセットされます。
デシマルフラグが"1"のとき、ADC命令およびSBC命令は10進演算となり、デシマルアジャストのために命令実行サイクルは+1サイクルされます。

(5) ブレイクコマンド（B）
ブレイクコマンドは、IRQ2入力による割込みが発生したとき、またはBRK命令を実行したときのみ意味を持つステータスです。
IRQ2入力により割込みが発生したとき、またはBRK命令を実行したときは、どちらの場合も論理アドレスで、
$FFF6番地から下位アドレスを
$FFF7番地から上位アドレスを
読み出して、サブルーチンコールします。このとき、スタックに退避されるステータスレジスタ（P）中のブレイクコマンドの値は、
0……IRQ2入力による割込みの場合
1……BRK命令による分岐の場合
のように設定されます。
ブレイクコマンドは、上記の場合以外では意味を持ちませんが、PHP命令ののちPLA命令などを使って読み出した場合は"1"となります。

(6) メモリ演算フラグ（T）
メモリ演算フラグは、SET命令によりセットされSET命令の次に実行される命令に影響を与えます。
SET命令の次に実行される命令が、
AND命令、OR命令、EOR命令、ADC命令
であった場合、これらの命令はACCの代わりに、Xレジスタにより間接指定されるゼロページのメモリが演算の対象になります。
SET命令の次に実行される命令が上記以外であった場合は、SET命令は何の意味も持ちません。
メモリ演算フラグは、SET命令でセットされ、SET命令の次の命令のフェッチサイクルでリセットされます。
メモリ演算フラグは、割込み発生時にスタックに退避されます。したがって、SET命令と次に置かれた演算命令との間に割り込み処理ルーチンが置かれた場合にも、演算命令はメモリ演算命令として動作します。
メモリ演算フラグは、ステータスレジスタをPHP命令ののちPLA命令などを使って読み出した場合、"0"となります。スタックに退避されたメモリ演算フラグが"1"となるのは、前述の特殊な割込みが発生した場合だけです。

(7) オーバーフローフラグ（V）
オーバーフローフラグは符号付演算を行ったとき、演算結果にしたがってセットまたはリセットされます。また、CLV命令によりリセットされます。
HuC6280は、加算、減算はすべてALUの加算器（アダー）を使っておこないますが、減算は引く側の数の2の補数をとってALUの加算器へ入力します。
ALUのアダーに入力される2つの数値が、片方が"正"で他方が"負"である場合、オーバーフローフラグはリセットされます。両値ともに"正"である場合は、加算結果のビット7が"1"のときオーバーフローフラグはセットされ、"0"のときリセットされます。また、両値がともに"負"の場合、加算結果のビット7が"0"のときオーバーフローフラグはセットされ、"1"のときはリセットされます。
BIT、TRB、TSB、TST命令を実行すると、メモリのビット6のデータがオーバーフローフラグにセットされます。

(8) ネガティブフラグ（N）
ネガティブフラグは、演算命令などの結果のビット7にしたがって変化します。
演算などの結果、ビット7が"1"の場合ネガティブフラグはセットされ、"0"の場合リセットされます。
BIT、TRB、TSB、TST命令を実行すると、メモリのビット7のデータがネガティブフラグにセットされます。

SH、DH、LHレジスタ

ソースハイ（SH）、デスティネーションハイ（DH）、レングスハイ（LH）は、ブロック転送命令（TII～TDD命令）の実行時にのみ機能する専用レジスタです。
ソースハイ（SH）は、Xレジスタをロー側8ビットとして、ペアで16ビットのソースアドレスを指定します。デスティネーションハイ（DH）は、Yレジスタをロー側8ビットとして、ペアで16ビットのデスティネーションアドレスを指定します。レングスハイ（LH）は、ACCをロー側8ビットとして、ペアで16ビットのダウンカウンタとなり、転送ブロック長のカウントをおこないます。なお、計数はバイトでおこないます。なお、SH、DH、LHレジスタは、HuC6280の命令により読み出したり、書き込んだりはできません。

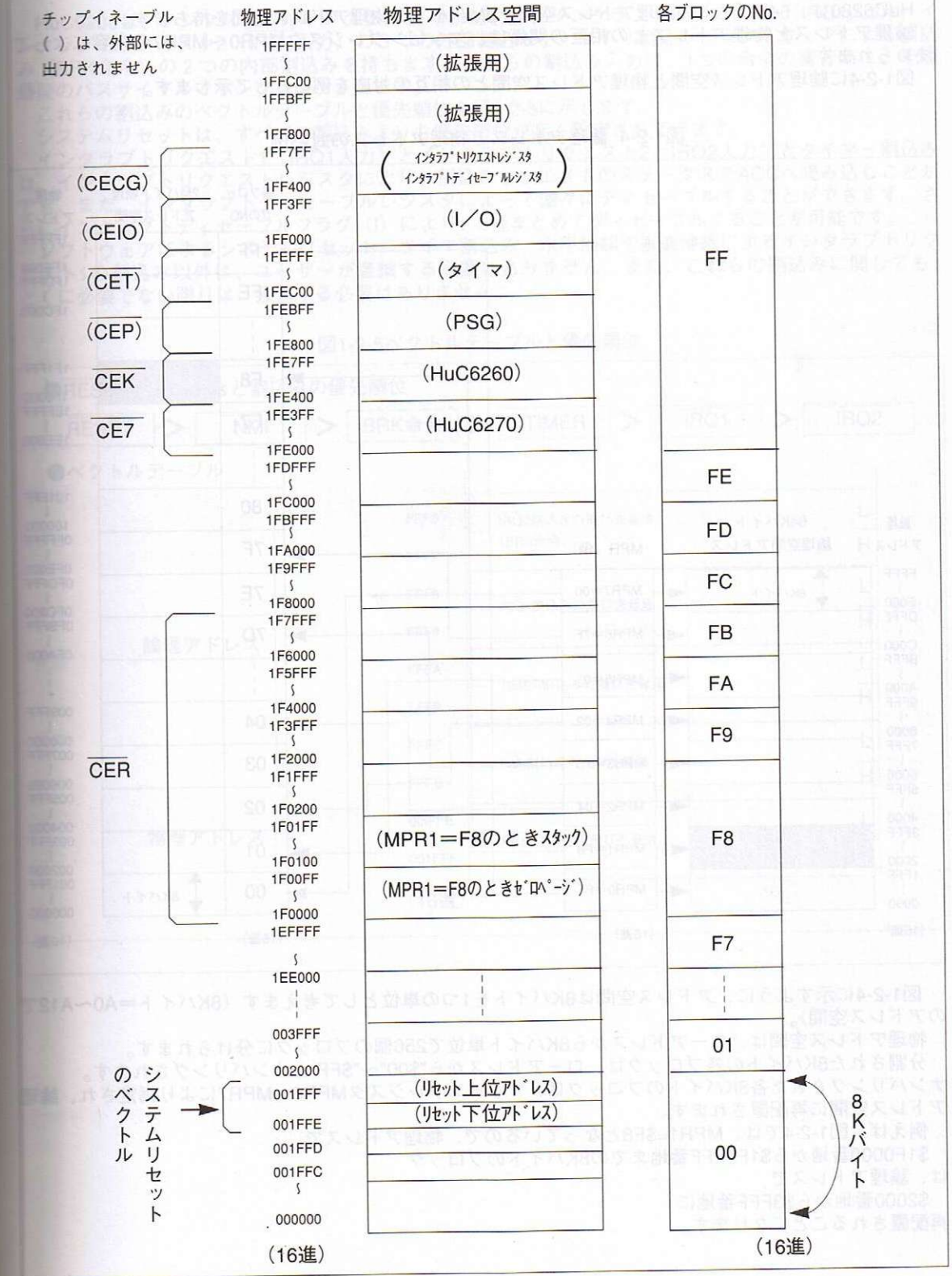
マッピングレジスタ（MPR）

HuC6280は、8個のマッピングレジスタ（MPR0～MPR7）を持ちます。
マッピングレジスタは8ビット構成のレジスタで、HuC6280の16ビットの論理アドレスを21ビットの物理アドレスに変換します。
8個のマッピングレジスタ（MPR0～MPR7）は、論理アドレスの上位3ビットによって選択されます。論理アドレスの上位3ビットは、選択されたマッピングレジスタが持っているデータ8ビットに置き換わり、物理アドレスの21ビットとなります。
システムリセットにより、マッピングレジスタのうちMPR7はオール0（$00）に設定されます。HuC6280はシステムリセット後、論理アドレスで、
$FFFE番地から下位アドレスを
$FFFF番地から上位アドレスを
読み出してスタートします。したがって、MPR7が"$00"のため、プログラムは物理アドレスで、
$001FFE番地から下位アドレスを
$001FFF番地から上位アドレスを
読み出してスタートします。
システムリセットにより、マッピングレジスタのほかのMPR0～MPR6の値は初期設定されません。イニシャルルーチンにて設定してください。
マッピングレジスタからのデータの読み出しは、TMAi命令により行います（i=0～7）。また、マッピングレジスタに対するデータの書き込みは、TAMi命令により行います（i=0～7）。
TMAi命令およびTAMi命令は、共に2バイト命令で、第2バイトでどのマッピングレジスタを選択するかを指定します。第2バイトのビット構成は、選択するマッピングレジスタのNo.に対応するビットを"1"とし、ほかのビットは"0"として指定します。表1-2-2に対応表を示します。

表1-2-2 TMAi、TAMiによるマッピングレジスタ指定

マッピングレジスタ	第2バイト（2進） MSB　　LSB
MPR0	00000001
MPR1	00000010
MPR2	00000100
MPR3	00001000
MPR4	00010000
MPR5	00100000
MPR6	01000000
MPR7	10000000

図1-2-3チップイネーブルの割り当て



メモリ空間

HuC6280は、64Kバイトの論理アドレス空間と2Mバイトの物理アドレス空間を持ちます。
論理アドレスと物理アドレスとの相互の関係は、マッピングレジスタMPR0～MPR7の内容によって決められます。
図1-2-4に論理アドレス空間と物理アドレス空間との相互の対応を例にとって示します。

図1-2-4 論理アドレスと物理アドレスの対応例

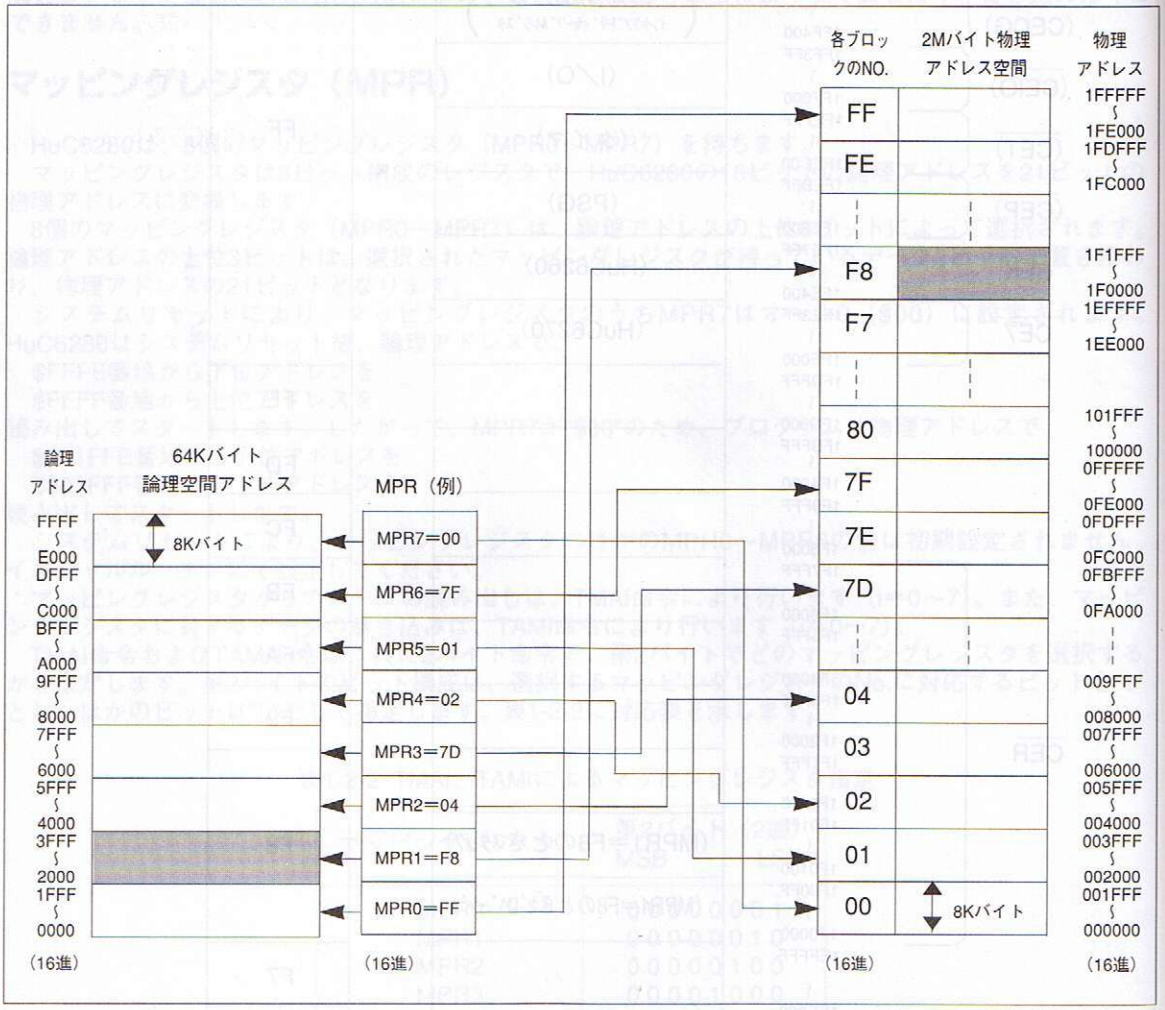


図1-2-4に示すように、アドレス空間は8Kバイトを1つの単位として考えます（8Kバイト＝A0～A12でのアドレス空間）。
物理アドレス空間は、ローアドレスから8Kバイト単位で256個のブロックに分けられます。
分割された8Kバイトの各ブロックは、ローアドレスから"$00"～"$FF"とナンバリングされます。
ナンバリングされた各8Kバイトのブロックは、マッピングレジスタMPR0～MPR7により指定され、論理アドレス空間に再配置されます。
例えば、図1-2-4では、MPR1=$F8となっているので、物理アドレスで
$1F0000番地から$1F1FFF番地までの8Kバイトのブロック
は、論理アドレスで
$2000番地から$3FFF番地に
再配置されることになります。

割込みとブレイク

HuC6280は、ノンマスカブルインタラプト（NMI入力）、インタラプトリクエスト1（IRQ1入力）、インタラプトリクエスト2（IRQ2入力）の3つの外部割込みと、タイマ割込み、ソフトウェアによる割込み（BRK命令）の2つの内部割込みを持ちます。これらの割込み入力は、1つの命令の実行時に、その最後のバスサイクルでサンプリングされます。

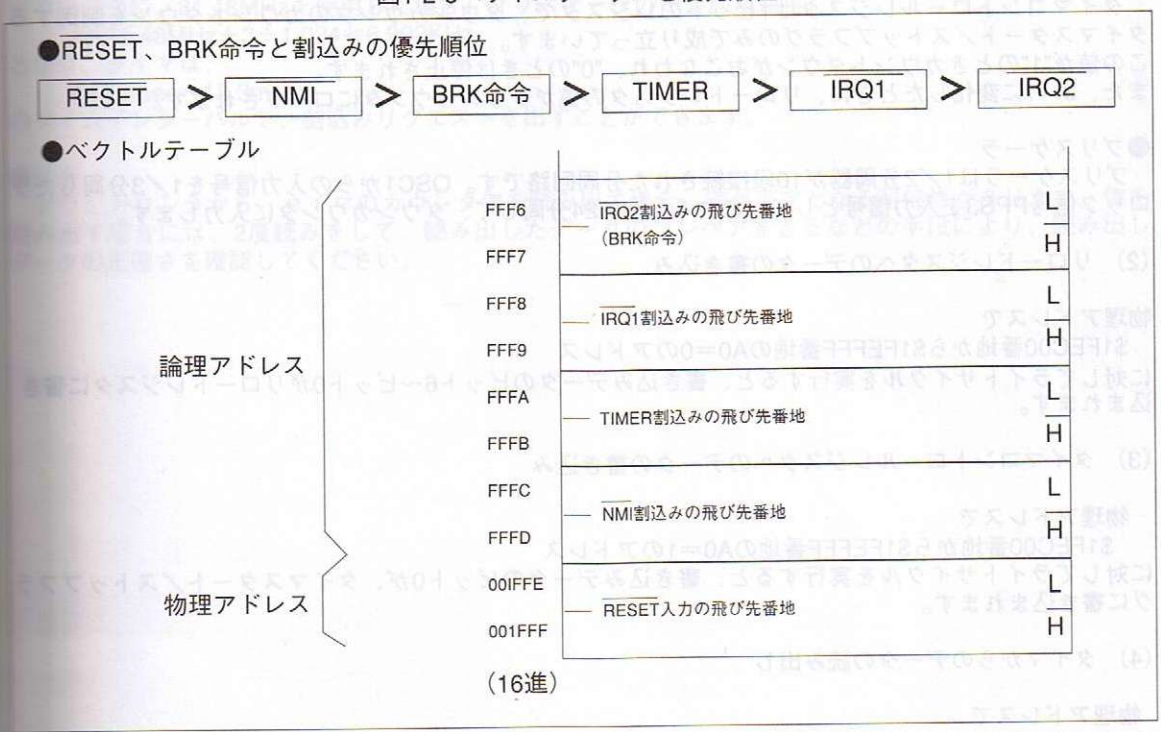
これらの割込みのベクトルテーブルと優先順位を図1-2-5に示します。

システムリセットは、すべての割込みよりも優先順位が高く設定されています。

インタラプトリクエスト1（IRQ1入力）とインタラプトリクエスト2（IRQ2入力）とタイマー割込みは、インタラプトリクエストレジスタにより、割込みリクエストのステータスをACCへ読み込むことができ、また、インタラプトディセーブルレジスタによって個々にディセーブルすることができます。さらにインタラプトディセーブルフラグ（I）により、3種まとめてディセーブルすることが可能です。

ソフトウェアによるシステムリセット、タイマ割込み、水平帰線や垂直帰線によるインタラプトリクエスト1の割込み以外は、ユーザーが意識する必要はありません。また、これらの割込みに関しても、とくに必要でない限りは、操作する必要はありません。

図1-2-5ベクトルテーブルと優先順位



タイマ

HuC6280は、7ビットのタイマを1本内蔵しています。

(1) タイマの構成

タイマは、7ビットのダウンカウンタ、7ビットのリロードレジスタ、タイマコントロールレジスタ、プリスケーラより構成されます。

●ダウンカウンタ
ダウンカウンタは2進のカウンタで、プリスケーラの出力を受けてカウントダウンします。

●リロードレジスタ
リロードレジスタは7ビットのレジスタで、タイマコントロールレジスタのスタート／ストップフラグの0→1の変化か、またはダウンカウンタにボローが発生すると、リロードレジスタの値がダウンカウンタにロードされます。

●タイマコントロールレジスタ
タイマコントロールレジスタは1ビットのレジスタで、ダウンカウンタのカウントダウンを制御するタイマスタート／ストップフラグのみで成り立っています。
この値が"1"のときカウントダウンがおこなわれ、"0"のときは停止されます。
また、0→1に変化したときに、リロードレジスタの値がダウンカウンタにロードされます。

●プリスケーラ
プリスケーラは1／2分周器が10段接続された分周回路です。OSC1からの入力信号を1／3分周したクロック信号PPS3を入力信号として、これを1／1,024分周して、ダウンカウンタに入力します。

(2) リロードレジスタへのデータの書き込み

物理アドレスで
$1FEC00番地から$1FEFFF番地のA0＝0のアドレス
に対してライトサイクルを実行すると、書き込みデータのビット6～ビット0がリロードレジスタに書き込まれます。

(3) タイマコントロールレジスタへのデータの書き込み

物理アドレスで
$1FEC00番地から$1FEFFF番地のA0＝1のアドレス
に対してライトサイクルを実行すると、書き込みデータのビット0が、タイマスタート／ストップフラグに書き込まれます。

(4) タイマからのデータの読み出し

物理アドレスで
$1FEC00番地から$1FEFFF番地
に対してリードサイクルを実行すると、読み出されたデータのビット6～ビット0が、ダウンカウンタから読まれた値となります。

5) タイマの動作

●システムリセット時
システムリセット後、リロードレジスタおよびダウンカウンタのデータは不定です。また、タイマコントロールレジスタのタイマスタート／ストップフラグはリセットされます。

●タイマの動作
リロードレジスタにタイムインターバルを設定した後、タイマコントロールレジスタのタイマスタート／ストップフラグをセットすると、リロードレジスタのデータがダウンカウンタにロードされてタイマはカウントダウンを開始します。
ダウンカウンタにボローが発生すると、インタラプトリクエストレジスタのタイマインタラプトリクエスト（TIQ）がセットされ、タイマインタラプトディセーブル（TIQD）が"0"で、かつインタラプトディセーブルフラグ（I）が"0"であれば、CPUにタイマ割込みが要求されます。
同時に、リロードレジスタのデータがダウンカウンタにロードされて、タイマはカウントダウンを続けます。
一度割込み要求が発生した場合には、その後インタラプトディセーブルレジスタに対してライトサイクルを実行し、タイマインタラプトリクエスト（TIQ）のリセットをおこなう必要があります。

●インターバルの制御
タイマがカウントダウンしている間に、リロードレジスタの値を書き換えることが可能です。したがって、割込みのインターバルを制御することができきます。
タイマスタート／ストップフラグに"0"を書き込むと、タイマはストップし、プリスケーラはリセット状態となります。
クロックに、21.48MHzを使用した場合、プリスケーラの出力周波数は
f＝21.48MHz÷3÷1,024≒6.992KHz
となり、タイマは、
143 μsec～18.3msec
のタイムインターバルで、割込みリクエストを出すことができます。

●注意
ダウンカウンタから、タイマのカウンタ値を読み出す場合、値が不定となる可能性があります。値を読み出す場合には、2度読みをして、読み出したデータのコンペアをとるなどの手段により、読み出しデータの正確さを確認してください。

VCE(HuC6260)解説

概要

HuC6260は、1チップ上に同期信号発生回路、カラーテーブルRAM、アナログRGBビデオ信号用D／Aコンバータ、映像信号用D／Aコンバータを集積したCMOSビデオ・カラー・エンコーダです。

ビデオ・ディスプレイ・コントローラ（VDC）から出力された画像データを、カラーテーブルRAM（カラーパレット）で変換して、アナログRGB信号として出力するとともに、映像色信号としても出力する機能を持っています。

特長

●モノリシックCMOSビデオ・カラー・エンコーダ
●同期信号発生回路を内蔵
●表現できる色数……512色（RGB各色8段階）
●カラー画像信号の種類……アナログRGB信号
　映像色信号
●RGBカラー信号帯域……7MHz
●カラーパレット内容はデータバスからリード、ライト可能
●映像色信号回路は少ない外付回路で発生可能
●単一電源……5V（デジタル部分、アナログ部分とも）
●パッケージ……80ピンプラスチックフラットパッケージ

内部レジスタ

HuC6260は、内部に制御レジスタを持っています。これらに対してCPUからアクセスする事により、動作モードの設定、カラーデータのリード/ライトなど各種の機能を実現できます。

図1-3-1　内部レジスター覧表

$\overline{CS}$	A1	A2	R／W	記号	レジスタ名	A0													
						1								0					
						D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
1	—	—	—	—	—																
0	0	0	W	CR	コントロールレジスタ									DCC							
0	1	0	W	CTA	カラーテーブルアドレス									CTA							
0	0	1	W	CTW	カラーテーブルデータライト									G			R			B	
0	0	1	R	CTR	カラーテーブルデータリード									G			R			B	

※図中の■部については使用しません。

カラーテーブルRAM(カラーパレット)

図2-3-2 カラーテーブルRAMの構造とVD0～VD8データとの関係図

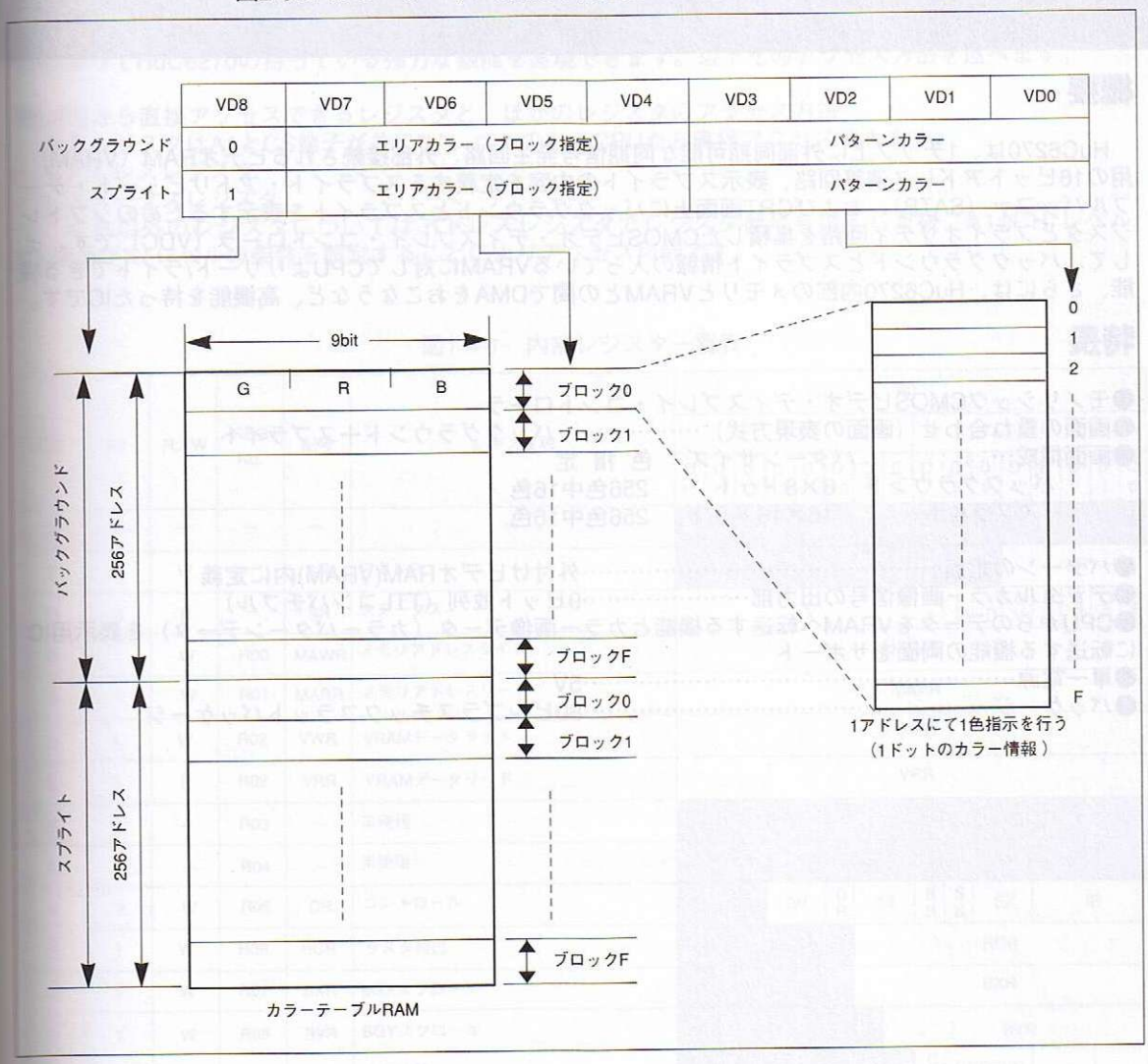


図2-3-2のようにカラーテーブルRAMはアドレス方向に512アドレス、データ長方向9bitにより構成されているカラー情報テーブルです。

カラーテーブルの持っている512のアドレスは、1アドレスで1色のG、R、B、1ドット分のカラー情報となります。そのため、3ビットずつに分けて、それぞれG(緑)、R(赤)、B(青)の色データとしています。

VDC(HuC6270)解説

概要

HuC6270は、1チップ上に外部同期可能な同期信号発生回路、外部接続されるビデオRAM（VRAM）用の16ビットアドレス演算回路、表示スプライトの内容を定義するスプライト・アトリビュート・テーブルバッファ（SATB）、およびCRT画面上にバックグラウンドとスプライトを表示するためのシフトレジスタとプライオリティ回路を集積したCMOSビデオ・ディスプレイ・コントローラ（VDC）です。そして、バックグラウンドとスプライト情報の入っているVRAMに対してCPUよりリード/ライトできる機能、さらには、HuC6270内部のメモリとVRAMとの間でDMAをおこなうなど、高機能を持ったICです。

特長

●モノリシックCMOSビデオ・ディスプレイ・コントローラ
●画面の重ね合わせ（画面の表現方式）……バックグラウンド十スプライト
●画面構成…

	パターンサイズ	色 指 定
バックグラウンド	8×8ドット	256色中16色
スプライト	16×16ドット	256色中16色

●パターンの定義……外付けビデオRAM(VRAM)内に定義
●デジタルカラー画像信号の出力部……9ビット並列（TTLコンパチブル）
●CPUからのデータをVRAMへ転送する機能とカラー画像データ（カラーパターンデータ）を表示用ICに転送する機能の両面をサポート
●単一電源……5V
●パッケージ……80ピンプラスチックフラットパッケージ

内部レジスタのアクセス

HuC6270は内部に各種の制御レジスタを有しており、これらに対してCPUからライト／リードすることによってHuC6270の持っている強力な機能を実現できます。以下そのアクセス方法を述べます。

●CPUから直接アクセスできるレジスタと、ほかのレジスタのアクセス方法

次のレジスタはA1とCS端子が共に"L"レベルのときCPUから直接アクセスできます。

a.アドレスレジスタ

b.ステータスレジスタ

これら以外のレジスタについてはアドレスレジスタでレジスタ番号を指定した後、A1が"H"レベル、CS端子が"L"レベルの条件を設定することによりアクセス可能です。

図1-4-1　内部レジスター覧表

$\overline{CS}$	A1	R／W	REG No.	記号	レジスタ名	A0															
						1								0							
						D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	—	—	—	—	—																
0	0	W	—	AR	アドレス												AR				
0	0	R	—	SR	ステータス	0								BSY	VD	DV	DS	RR	OR	CR	
0	1	W	R00	MAWR	メモリアドレスライト	MAWR															
0	1	W	R01	MARR	メモリアドレスリード	MARR															
0	1	W	R02	VWR	VRAMデータライト	VWR															
0	1	R	R02	VRR	VRAMデータリード	VRR															
0	1	—	R03	—	未使用																
0	1	—	R04	—	未使用																
0	1	W	R05	CR	コントロール			IW	DR		TE		BB	SB		EX			IE		
0	1	W	R06	RCR	ラスタ検出									RCR							
0	1	W	R07	BXR	BGXスクロール									BXR							
0	1	W	R08	BYR	BGYスクロール										BYR						
0	1	W	R09	MWR	メモリ幅											CM	SCREEN				
0	1	W	R0A	HSR	水平同期				HDS									HSW			
0	1	W	R0B	HDR	水平表示				HDE									HDW			
0	1	W	R0C	VPR	垂直同期					VDS								VSW			
0	1	W	R0D	VDW	垂直表示										VDW						
0	1	W	R0E	VCR	垂直表示エンド位置										VCR						
0	1	W	R0F	DCR	DMAコントロール											DV	DS	RR	OR	CR	
0	1	W	R10	SOUR	DMAソースアドレス	SOUR															
0	1	W	R11	DESR	DMAデスティネーション	DESR															

※図中の□部については使用しません。

内部レジスタの機能

HuC6270の内部レジスタのうち、おもにCD-ROM² BIOSなどからアクセスすることができないいくつかのレジスタについて解説します。

(1) アドレスレジスタ (AR)

アドレスレジスタ (AR) は、HuC6270内部のレジスタR00～R13を指定するためのライト専用レジスタです。
R00～R13にライトまたはリードをおこなうときは、まずARに該当するレジスタの番号をライトする必要があります。A1とCSが"L"レベルのとき、HuC6270にライトするとARが選択されます。
プログラムからのデータ設定にはST0命令を用います。
注) AR=04の禁止について
ARに04を設定すると正常な動作をしないことがありますのでご注意ください。

(2) ステータスレジスタ (SR)

ステータスレジスタは、HuC6270内部の状態を示すレジスタです。
コントロールレジスタ (CR) とDMAコントロールレジスタ (DCR) 割込み許可ビットにより許可された割込み原因が発生すると、ステータスレジスタの対応するビットがセットされ、割込み (IRQ) がアクティブになります。ステータスの内容は、ステータスを読むことにより自動的にクリアされます (BSYは除く)。

a. 衝突検出 (CR:bit0)
スプライト#0がスプライト#1～#63のいずれかと衝突したことを示します。

b. オーバー検出 (OR:bit1)
オーバー検出は次の3つの場合にセットされます。
1. 1ラスタライン上に17個以上のスプライトを表示しようとしたとき。
2. ヒットしたスプライトのデータを水平帰線期間中に取り込めないとき。
3. SATで示すCGXが立ったスプライトがあるため、ヒットしたスプライトのデータをすべてHuC6270内に取り込めないとき。
(SAT:スプライトアトリビュートテーブル)

c. ラスタ検出 (RR:bit2)
ラスタカウンタがラスタ検出レジスタの設定値になったことを示します。

d. DMA転送 (VRAM-SATB間) 終了検出 (DS:bit3)
VRAM-SATB間転送が終了したことを示します。

e. DMA転送 (VRAM-VRAM間) 終了検出 (DV:bit4)
VRAM-VRAM間転送が終了したことを示します。
垂直表示期間になるとVRAM-VRAM間のDMA転送は打ち切られますが、この場合終了検出は発生しません。

f. 垂直帰線期間検出 (VD:bit5)
垂直帰線期間になったことを示します。

g. BSY (BSY:bit6)
BSYは、CPUのアクセスにもとづいてVRAMがリード、ライト中であることを示します。
BSYがセットされても割込み (IRQ) は発生しません。
BSYが"1"の間IW (CR R05のbit11と12) の書き込みはしないでください。
(なお、このビット情報は基本的にBUSY端子情報となります。……ただし論理は逆です)

(3) コントロールレジスタ (CR R05)

コントロールレジスタは、HuC6270の動作モードを設定するレジスタです。
EX、TE、DRフィールドは書き換えると直ちに有効になりますが、これらのフィールドを0以外の値にした場合、動作は保証されませんので注意してください。

コントロールレジスタの各フィールドのうち、いくつかについて解説します。

a. 割込み要求許可（IE:R05 bit0～3）
IEの各ビットに対応した割込みを許可します。

IE bit	記号	内　容
0	CC	衝突検出
1	OC	オーバー検出
2	RC	ラスタ検出
3	VC	垂直帰線期間検出

b. スプライトブランキング（SB:R05 bit6）
画面にスプライトの表示をおこなうか否かを設定するビットです。
セットしたSBは次の水平表示期間から有効になります（バーストモードの解除は除きます）。
0:スプライトを消します
1:スプライトを表示します
c. バックグラウンドブランキング（BB:R05 bit7）
画面にバックグラウンドの表示をおこなうか否かを設定するビットです。
セットしたBBは次の水平表示期間から有効になります（バーストモードの解除は除きます）。
0:バックグラウンドを消します
1:バックグラウンドを表示します
SB＝BB＝0のときは、次のフレームからバーストモードになります。バーストモードでは
1. 表示のためのVRAMのアクセスを止めてCPUがVRAMをアクセスできます。
2. VRAM-VRAM間DMAが常時可能になります。
3. VD0～VS7は"L"レベル、SPBGは"H"レベルを出力します。
バーストモードはSB=1またはBB=1となった次のフレームで解除されます。

(4) ラスタ検出レジスタ（RCR R06）

ラスタ検出レジスタはCRT走査の何ラスタ目で割込みを発生するかを設定するレジスタです。この割込みは内部のラスタカウンタの値とラスタ検出レジスタの値を比較して一致したときに発生します。内部のラスタカウンタは表示開始の1ラスタ前で"64"にセットされ、ラスタごとにプラス1されます。

(5) BGXスクロールレジスタ（BXR R07）

BGXスクロールレジスタは、バックグラウンド画面の水平スクロールをおこなうときに用いるレジスタです。
このレジスタを書き換えると、次のラスタラインから有効になります。

(6) BGYスクロールレジスタ（BYR R08）

BGYスクロールレジスタは、バックグラウンド画面の垂直スクロールをおこなうときに用いるレジスタです。
表示中にこのレジスタを書き換えると、次のラスタラインからBYR+1として有効になります。

バックグラウンドの表示機能

バックグラウンド表示

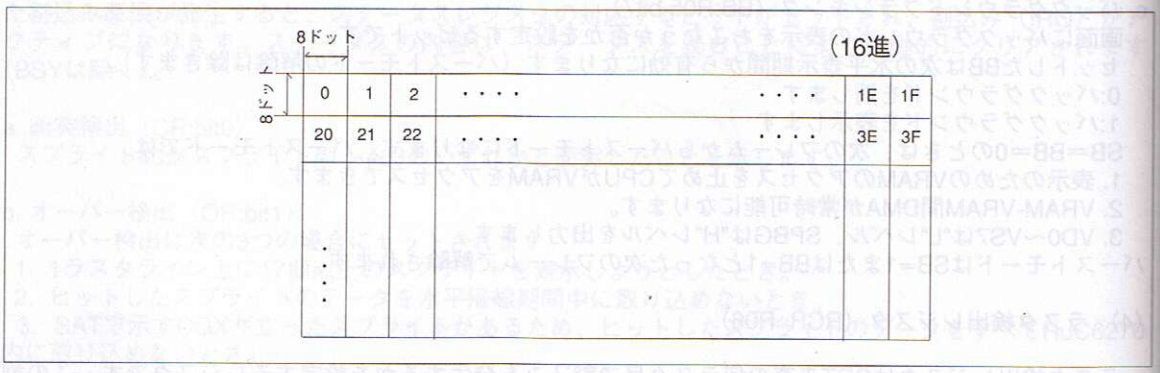
a. キャラクタサイズ……8×8ドット
b. キャラクタジェネレータに定義できるキャラクタの数……最大4,096個
c. 表示色……各キャラクタごとに256色中16色
（メモリ幅レジスタにて4ドットモードを選択したときは256色中4色）

バックグラウンドアトリビュートテーブル（BAT）

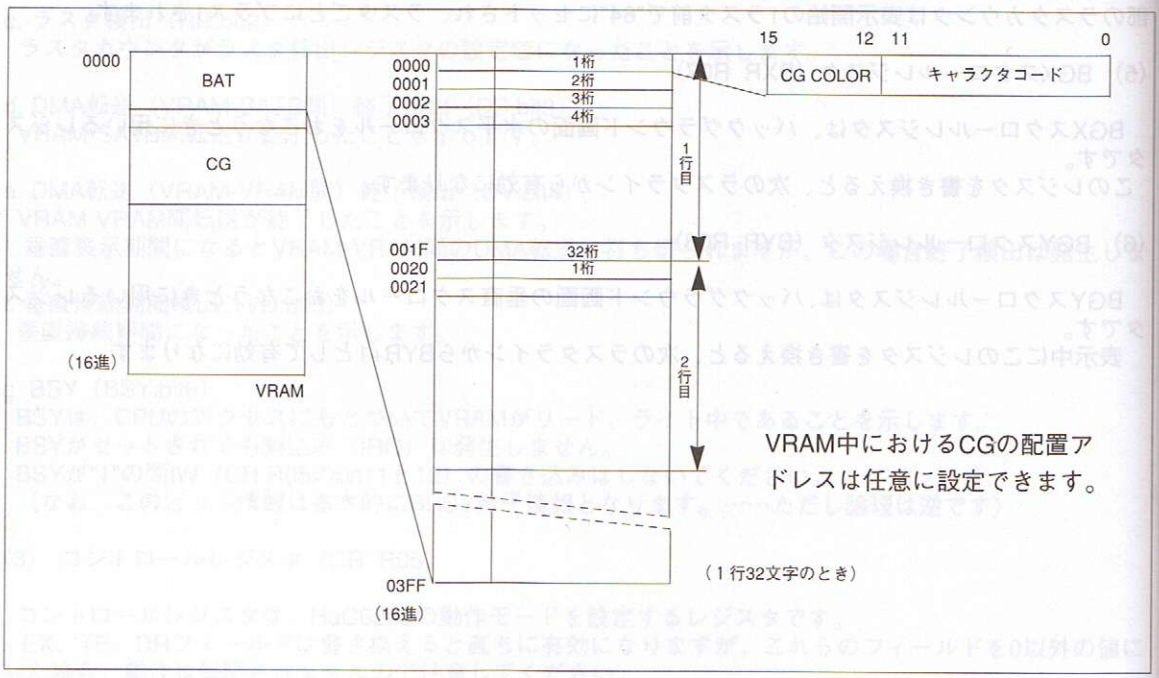
バックグラウンドアトリビュートテーブルは、仮想スクリーン上の各キャラクタ位置に、どのキャラクタをどの色で表示するかを指定するために、VRAM中に設定するテーブルです。テーブルは、VRAMの0番地から始まる領域におかれます。仮想スクリーンの左上端のキャラクタのデータが0番地、その右のキャラクタが1番地……のように対応しています。

(1) 仮想スクリーンのキャラクタのバックグラウンドアトリビュートテーブル中のアドレス

(32×32キャラクタの場合)



(2) BATのVRAM中での位置とその内容



(3) バックグラウンドアトリビュートテーブル中のデータ構成

LSB															LSB
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CG COLOR				キャラクターコード											

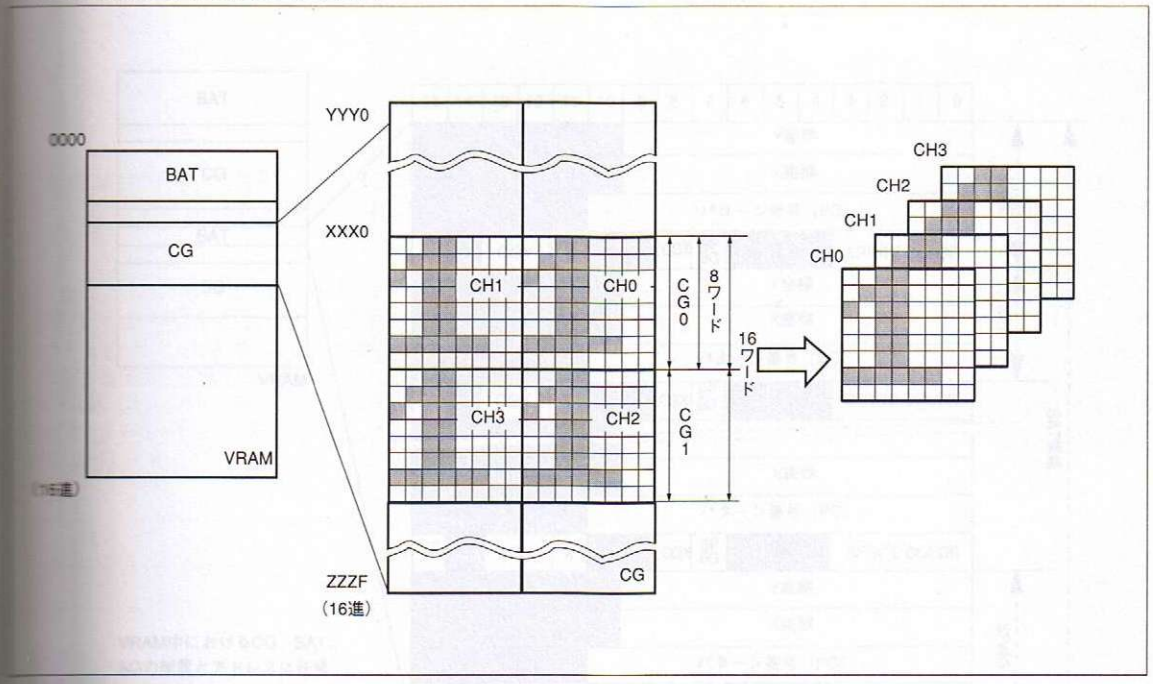
a.キャラクタコード
VRAM内のキャラクタジェネレータ（CG）で定義されるキャラクタのパターン番号を指定します。

b.CG COLOR
表示するキャラクタの色を、どのパレットセットの色で表示するかを、4ビットで指定します。

キャラクタジェネレータ（CG）

キャラクタジェネレータは、キャラクタのパターンを定義するVRAM内の領域です。
画面中での1個のキャラクタは8×8ドットで構成されています。各ドットを4ビットの色コードとして定義する必要上、キャラクタジェネレータ中では8×8ドットの面を4面使います。各面はCH0、CH1、CH2、CH3と呼ばれ、CH0とCH1で8ワード（CG0）、CH2とCH3で8ワード（CG1）、合計16ワードでパターンを定義します。
CH0の先頭のアドレスは、下位4ビットが0となるアドレスに配置します。そのアドレスの上位12ビットが、そのパターンのパターン番号になります。

CGのVRAM中での位置とその内容



バックグラウンドの表示色

BATに指定されたCG COLORの値にもとづき、そのパターンの各ドットの色が表示されます。CG COLORの値はHuC6260のカラーテーブルRAMのビット7～4に対応し、各ドットに設定された色がビット3～0に対応します。また、バックグラウンドの表示では、カラーテーブルRAMのビット8はつねに"0"になります

VD8	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0
0	*CG COLOR*				*CH3*	*CH2*	*CH1*	*CH0*
					CG1		CG0	

ただし、パターン中のCH0～CH3に共通して"0"となる部分の色は、全パレット共通でCG COLOR=0で表示されます。

また、画面周囲のボーダーカラーは、VD8=1、VD7～VD0=0の色で表示されます。

スプライト機能

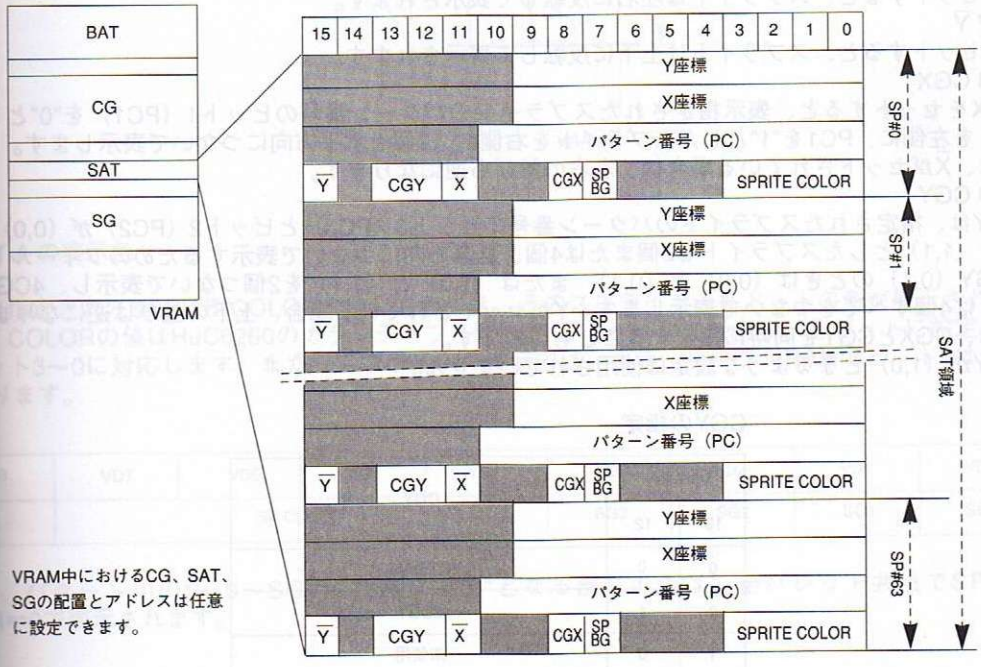
スプライト表示

a. 各スプライトは座標を変更することによって、表示画面内を自由に移動することができ、パターンの再定義は一切必要としません。
b. 64個のスプライトには優先順位があり、2つ以上のスプライトが重なっていく場合、スプライトの重なった部分は優先順位の低い順に隠されていきます。
c. スプライトのサイズ……16×16ドット
d. スプライトジェネレータ（SG）に定義できるスプライトの数……最大1,024個
（ただし64Kワード[1ワード16ビット]VRAMを搭載時にスプライトのみを定義したとき）
e. スプライトアトリビュートテーブル（SAT）に登録できるスプライトの数……64個
f. スプライトの表示制限
スプライトは、同一水平ライン上に16個まで存在することができます。同一水平ライン上に17個以上のスプライトを表示しようとすると、優先順位の高い16個のスプライトまでが表示され、17番目以降のスプライトは表示されません。
g. 色表示
各スプライトごとに256色のうち16色を指定。

スプライトアトリビュートテーブルバッファ（SATB）

スプライトアトリビュートテーブルバッファ（SATB）は、スプライトの表示位置（X,Y）、色、パターン番号などを登録するHuC6270内のメモリです。
SATBへの書き込みは、CPUから直接はできません。VRAM中にスプライトアトリビュートテーブル（SAT）領域を設け、VRAM-SATB間のDMA転送によりおこないます。

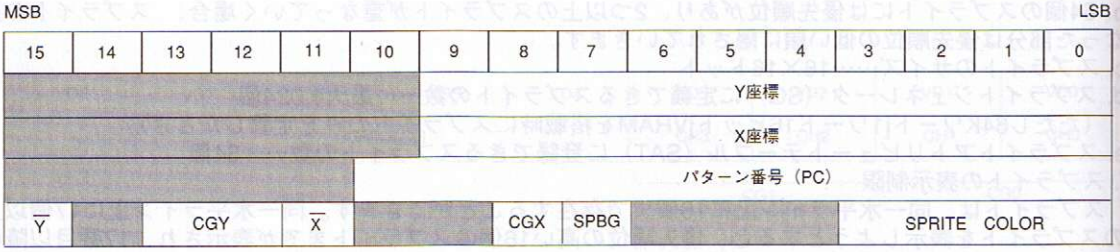
(1) VRAM中にSATを配置する例
SAT領域では、4ワードで1個のスプライトを定義し、256ワードで64個のスプライトを定義します。スプライトの優先順位は、このアドレス順になります。



VRAM中におけるCG、SAT、SGの配置とアドレスは任意に設定できます。

(2) SATの構成

SATの構造は次のようになっています。



(2) -1 Y座標

Y座標はスプライトの垂直位置を指定するフィールドで、画面上端が64となります。(0～1,023)

(2) -2 X座標

X座標はスプライトの水平位置を指定するフィールドで、画面左端が32となります。(0～1,023)

(2) -3 パターン番号（PC）

VRAM内にスプライトジェネレータ（SG）として定義されたスプライトのパターン番号を指定します。

パターン番号の上位10ビットはVRAMのアドレスの上位10ビットになり、その領域に定義されたスプライトのパターンデータがこのパターン番号で指定されるスプライトです。

(2) -4 SPRITE COLOR

SPRITE COLORはスプライトをどのパレットセットで表示するかを4ビットで指定するフィールドです。

(2) -5 SPBG

SPBGはバックグラウンド表示とスプライト表示の優先度を指定するビットです。

0:バックグラウンド表示
1:スプライト表示

(2) -6 $\overline{X}$

Xをセットすると、スプライトは左右に反転して表示されます。

(2) -7 $\overline{Y}$

Yをセットすると、スプライトは上下に反転して表示されます。

(2) -8 CGX

CGXをセットすると、表示指定されたスプライトのパターン番号のビット1（PC1）を"0"としたスプライトを左側に、PC1を"1"としたスプライトを右側に、2個を水平方向につないで表示します。

また、Xがセットされている場合は、左右の並びも逆になります。

(2) -9 CGY

CGYは、指定されたスプライトのパターン番号のビット3（PC3）とビット2（PC2）が（0,0）、（0,1）、（1,0）、（1,1）としたスプライトを2個または4個、垂直方向につないで表示するためのフィールドです。

2CGY（0,1）のときは（0,0）と（0,1）、または（1,0）と（1,1）を2個つないで表示し、4CGY（1,1）のときは4個すべてをつないで表示します。Yがセットされている場合、上下の並びは逆になります。

なお、CGXとCGYを同時に指定することも可能です。

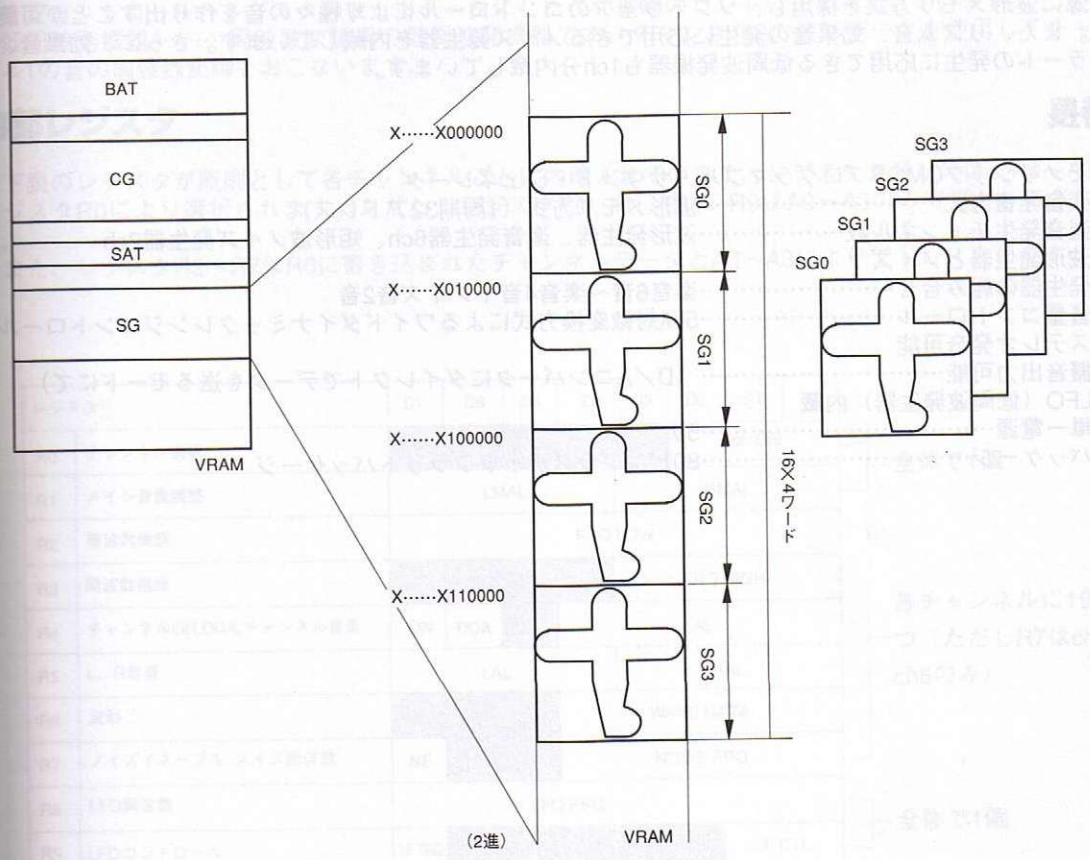
CGYが（1,0）となるような設定は使用されていません。

CGYの指定

CGY bit		CGY
13	12	
0	0	NORMAL
0	1	2CGY
1	0	未使用
1	1	4CGY

スプライトジェネレータ（SG）

スプライトジェネレータはVRAM中に設定され、スプライトのパターンを記憶します。1個のスプライトは16×16ドットの面が4面からなり、各面はSG0、SG1、SG2、SG3と呼ばれ、おのおのVRAMの16ワードを占めています。SG0の先頭のアドレスは、下位6ビットが0となるアドレスに配置します。そのアドレスの上位10ビットが、そのスプライトのパターン番号になります。



スプライトの表示色

SATに指定されたSPRITE COLORの値にもとづき、そのスプライトの各ドットの色が表示されます。SPRITE COLORの値はHuC6260のカラーテーブルRAMのビット7～4に対応し、各ドットに設定された色がビット3～0に対応します。また、スプライトの表示では、カラーテーブルRAMのビット8はつねに"1"になります。

VD8	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0
1	SP COLOR				SG3	SG2	SG1	SG0

ただし、パターン中のSG0～SG3に共通して"0"となる部分の色は、全パレット共通でSPRITE COLOR=0で表示されます。

PSG（HuC6280）解説

概要

このPSGは、HuC6280LSIに内蔵されたプログラマブル・サウンド・ジェネレータです。音源に波形メモリ方式を採用し、ソフトウェアのコントロールにより種々の音を作り出すことが可能です。また、リズム音、効果音の発生に応用できるノイズ発生器を内蔵しています。さらに、効果音、ビブラートの発生に応用できる低周波発振器も1ch分内蔵しています。

特長

●モノリシックCMOSプログラマブル・サウンド・ジェネレータ
●楽音発音方式……波形メモリ方式（1周期32アドレス）
●楽音発生チャンネル数……波形発生器、楽音発生器6ch、矩形波ノイズ発生器2ch
●波形発生器とノイズ
　発生器の組み合せ……楽音6音～楽音4音＋ノイズ音2音
●音量コントロール……5bit対数変換方式によるワイドダイナミックレンジコントロール
●ステレオ発音可能
●擬音出力可能……（D／Aコンバータにダイレクトでデータを送るモードにて）
●LFO（低周波発生器）内蔵
●単一電源……5V
●パッケージ……80ピンプラスチックフラットパッケージ

動作機能

PSGの全機能はPSG内部レジスタにデータを書き込むことによって制御されます。すなわちこれらレジスタの内容にしたがって発音がなされます（DDA:ダイレクトD／Aモードを除く）。

●波形発生器……各チャンネルごとに波形レジスタ（5bit×32wordで一周期分の波形を形成）の内容にしたがい波形を発生します。
●ノイズ発生器……擬似ランダム波形を発生します。出力波形は矩形波です。
●音量コントロール…レジスタのセットデータにしたがい振幅のコントロールをおこないます。
●低周波発振器……周波数変調用の低周波発振器です。チャンネル2の波形データを使用してチャンネル1の音の周波数変調をおこないます。

内部レジスタ

下表のレジスタが原則として各チャンネルごとに内蔵されています。レジスタアドレスは、A0～A3とレジスタR0により選択されます。ただし、レジスタR0、R1、R8、R9はA0～A3のみでアドレスされます。

また、レジスタR2～R7はR0に書き込まれたチャンネルデータとA0～A3によりアドレスされます。

レジスタ		D7	D6	D5	D4	D3	D2	D1	R0	
R0	チャンネル選択						ch SEL			全体で1個
R1	メイン音量調整	LMAL					RMAL			
R2	周波数微調	FRQ LOW								各チャンネルに1個ずつ（ただしR7はch5、ch6のみ）
R3	周波数粗調					FRQ HIGH				
R4	チャンネルON,DDA,チャンネル音量	ON	DDA			AL				
R5	L、R音量	LAL				RAL				
R6	波形				WAVE DATA					
R7	ノイズイネーブル ノイズ周波数	NE			NOISE FRQ					
R8	LFO周波数	LFO FRQ								全体で1個
R9	LFOコントロール	LF TRG						LF CTL		

内蔵される全レジスタ表を次に記します。

	ch1	ch2	ch2	ch4	ch5	ch6
R0	全体で1個 } (R0の値によらず選択される)			ch（annel）SELECT		
R1	全体で1個 }			M（ain） A（mplitude Level）		
R2	FRQ LOW	FRQ LOW	FRQ LOW	FRQ LOW	FRQ LOW	FRQ LOW
R3	FRQ HI	FRQ HI	FRQ HI	FRQ HI	FRQ HI	FRQ HI
R4	ON DDA AL	ON DDA AL	ON DDA AL	ON DDA AL	ON DDA AL	ON DDA AL
R5	LAL RAL	LAL RAL	LAL RAL	LAL RAL	LAL RAL	LAL RAL
R6	WAVE DATA	WAVE DATA	WAVE DATA	WAVE DATA	WAVE DATA	WAVE DATA
R7					NE NOISE FRQ	NE NOISE FRQ
R8	全体で1個 } (ch1をch2のデータでコントロールする)			LFO FRQ		
R9	全体で1個 }			LF TRG,LF CTL		

第2部
ソフトウェア解説

C O N T E N T S

ソフトウェア概要

概要

ここでいうソフトウェアとはハードウェアを動作させるための命令体系のことです。HuC6280のアセンブリ言語の各命令について説明します。

命令セット

HuC6280は基本命令として89種類の命令があり、それらの各命令は、お互いにプログラミングに最適になるように設計されています。

HuC6280の命令セットは
●ALU関係
●フラグ関係
●データ転送関係
●分岐関係
●サブルーチン関係
●テスト関係
●コントロール関係に分類されます。

表2-2-1から表2-2-3に、HuC6280の各命令とその機能との対応表を、上記の分類に従って示します。HuC6280の各命令の表記は、この規則に従っています。
表2-3-1から表2-3-3にはアルファベット順表を示します。

解説で使用する記号・用語の意味(定義)

●アドレッシングモード:各命令の持つアドレッシングモードを示します。記述方法は表2-1-1に従います。なお、一般のプログラミングにおいて、アドレッシングモードをとくに意識する必要はありません。そのため、当マニュアルではこれらの解説を省いてあります。

表2-1-1 アドレッシングモード

アドレッシングモード	記述法	アドレッシングモード	記述法
インプライド	IMPLID	アブソリュート・Xレジスタ・インデックス	ABS, X
イミディエイト	IMM	アブソリュート・Yレジスタ・インデックス	ABS, Y
ゼロページ	ZP	アブソリュート・インダイレクト	(ABS)
ゼロページ・Xレジスタ・インデックス	ZP, X	アブソリュート・インデックス・インダイレクト	(ABS, X)
ゼロページ・Yレジスタ・インデックス	ZP, Y	リラティブ	REL
ゼロページ・リラティブ	ZP, REL	イミディエイト・ゼロページ	IMM ZP
ゼロページ・インダイレクト	(IND)	イミディエイト・ゼロページ・インデックス	IMM ZP, X
ゼロページ・インデックス・インダイレクト	(IND, X)	イミディエイト・アブソリュート	IMM ABS
ゼロページ・インダイレクト・インデックス	(IND), Y	イミディエイト・アブソリュート・インデックス	IMM ABS, X
アブソリュート	ABS	アキュムレータ	ACC

● ニーモニック：各命令について、アセンブリ言語での記述方法を示します。記述の中で、記号は表2-1-2に従います。

表2-1-2 ニーモニック

記号	説明	記号	説明
A	アキュムレータ	hh	メモリのアドレスの上位バイト(16進)
X	Xレジスタ	ll	メモリのアドレスの下位バイト(16進)
Y	Yレジスタ	ZZ	ゼロページのアドレスの下位バイト(16進)
M	メモリ	∧	論理積
Ms	メモリ(スタック)	∨	論理和
Mi	メモリの指定ビット	∀	排他的論理和
M_6	メモリのビット6	+	加算
M_7	メモリのビット7	−	減算
$M_{(x)}$	Xレジスタで指定されるゼロページメモリ	#	イミディエトデータを示す※
IMM	イミディエイトデータ	nn	8bitデータ
MPR	マッピングレジスタ	i	ビットデータorマッピングレジスタNo.
MPRi	指定されたマッピングレジスタ	␣	タブまたはスペース
N	ネガティブフラグ	rr	相対分岐命令でのオフセット(16進)
V	オーバーフローフラグ	SH	ソースのアドレスの上位バイト
T	メモリ演算フラグ	SL	ソースのアドレスの下位バイト
B	ブレイクコマンド	DH	デスティネーションのアドレスの上位バイト
D	デシマルフラグ	DL	デスティネーションのアドレスの下位バイト
I	インタラプトディセーブル	LH	レングスの上位バイト
Z	ゼロフラグ	LL	レングスの下位バイト
C	キャリーフラグ	M_{DD}	DH、DLにより指定されるメモリ
$\bar{C}$	キャリーノット、ボロー	M_{SS}	SH、SLにより指定されるメモリ
PC	プログラムカウンタ	x	ブロック 転送バイト数
PCH	プログラムカウンタの上位バイト		
PCL	プログラムカウンタの下位バイト		
S	スタックポインタ		
P	ステータスレジスタ		

※) ニーモックとマシンコードに使用する。

● フラグ：各命令の実行により、ステータスレジスタがどのように変化するかを示します。記号は表2-1-3に従います。

表2-1-3 フラグ

記号	説明	記号	説明
N,V,Z,Cのいずれか	各フラグが命令の実行によって、変化することを示します※	M_7	メモリのビット7がセットされる
		M_6	メモリのビット6がセットされる
1	フラグがセットされる	(RESTORED)	スタックのデータが、ステータスレジスタにロードされる
0	フラグがリセットされる		
-	フラグは変化しない		

※) N：ネガティブフラグ、V：オーバーフローフラグ、Z：ゼロフラグ、C：キャリーフラグ

項目別分類表

表2-2-1 ニーモニックとその機能(1)

分類	ニーモニック	機能
ALU関係	ADC	キャリーフラグを含めた加算演算
	AND	論理積演算
	ASL	左方向シフト
	CLA	アキュムレータのクリア
	CLX	Xレジスタのクリア
	CLY	Yレジスタのクリア
	CMP	アキュムレータとの比較
	CPX	Xレジスタとの比較
	CPY	Yレジスタとの比較
	DEC	デクリメント
	DEX	Xレジスタのデクリメント
	DEY	Yレジスタのデクリメント
	EOR	排他的論理和演算
	INC	インクリメント
	INX	Xレジスタのインクリメント
	INY	Yレジスタのインクリメント
	LSR	右方向シフト
	ORA	論理和演算
	ROL	キャリーフラグを含めた左方向ローテーション
	ROR	キャリーフラグを含めた右方向ローテーション
	SBC	キャリーフラグを含めた減算演算
フラグ関係	CLC	キャリーフラグのクリア
	CLD	デシマルフラグのクリア
	CLI	インタラプトディセーブルのクリア
	CLV	オーバーフローフラグのクリア
	SEC	キャリーフラグのセット
	SED	デシマルフラグのセット
	SEI	インタラプトディセーブルのセット
	SET	メモリ演算フラグのセット
データ転送関係	LDA	アキュムレータへのロード
	LDX	Xレジスタへのロード
	LDY	Yレジスタへのロード
	SAX	アキュムレータとXレジスタとの交換
	SAY	アキュムレータとYレジスタとの交換
	ST0	HuC6270へのストア
	ST1	HuC6270へのストア
	ST2	HuC6270へのストア
	STA	アキュムレータのストア
	STX	Xレジスタのストア
	STY	Yレジスタのストア
	STZ	"00_{16}"のストア

表2-2-2 ニーモニックとその機能(2)

分類	ニーモニック	機能
データ転送関係	SXY	Xレジスタ とYレジスタとの交換
	TAI	ブロック転送(アルタネイトからインクリメントへ)
	TAMi	アキュムレータからマッピングレジスタへの転送
	TAX	アキュムレータからXレジスタへの転送
	TAY	アキュムレータからYレジスタへの転送
	TDD	ブロック転送(デクリメントからデクリメントへ)
	TIA	ブロック転送(インクリメントからアルタネイトへ)
	TII	ブロック転送(インクリメントからインクリメントへ)
	TIN	ブロック転送(インクリメントから固定へ)
	TMAi	マッピングレジスタからアキュムレータへ転送
	TSX	スタックポインタからXレジスタへ転送
	TXA	Xレジスタからアキュムレータへ転送
	TXS	Xレジスタからスタックポインタへ転送
	TYA	Yレジスタからアキュムレータへ転送
分岐関係	BBRi	ビットがリセットされている場合に分岐(相対アドレス)
	BBSi	ビットがセットされている場合に分岐(相対アドレス)
	BCC	キャリーフラグがクリアされている場合に分岐(相対アドレス)
	BCS	キャリーフラグがセットされている場合に分岐(相対アドレス)
	BEQ	ゼロフラグがセットされている場合に分岐(相対アドレス)
	BMI	ネガティブフラグがセットされている場合に分岐(相対アドレス)
	BNE	ゼロフラグがクリアされている場合に分岐(相対アドレス)
	BPL	ネガティブフラグがクリアされている場合に分岐(相対アドレス)
	BRA	常に分岐(相対アドレス)
	BVC	オーバーフローフラグがクリアされている場合に分岐(相対アドレス)
	BVS	オーバーフローフラグがセットされている場合に分岐(相対アドレス)
	JMP	分岐(絶対アドレス)
サブルーチン関係	BSR	サブルーチンコール(相対アドレス)
	JSR	サブルーチンコール(絶対アドレス)
	PHA	アキュムレータをスタックへプッシュ
	PHP	ステータスレジスタをスタックへプッシュ
	PHX	Xレジスタをスタックへプッシュ
	PHY	Yレジスタをスタックへプッシュ
	PLA	スタックからアキュムレータにプル
	PLP	スタックからステータスレジスタにプル
	PLX	スタックからXレジスタにプル
	PLY	スタックからYレジスタにプル
	RTI	割込み処理ルーチンからのリターン
	RTS	サブルーチンからのリターン

表2-2-3 ニーモニックとその機能(3)

分類	ニーモニック	機能
テスト関係	BIT	ビットのテスト(アキュムレータとメモリの論理積)
	TRB	ビットのリセット(アキュムレータの否定とメモリの論理積)
	TSB	ビットのセット(アキュムレータとメモリの論理和)
	TST	メモリのテスト(メモリと即値の論理積)
コントロール関係	BRK	ソフトウェア割込み
	NOP	ノーオペレーション
	RMBi	メモリのビットをリセット
	SMBi	メモリのビットをセット

アルファベット順表

表2-3-1

ニーモニック	機能	フラグ								参照ページ
		N	V	T	B	D	I	Z	C	
ADC	A←A+M+C(T=0のとき) $M_{(x)}$←$M_{(x)}$+M+C(T=1のとき)	N	V	0	-	-	-	Z	C	73
AND	A←A∧M(T=0のとき) $M_{(x)}$←$M_{(x)}$∧M(T=1のとき)	N	-	0	-	-	-	Z	-	74
ASL	C←[7 0]←0	N	-	0	-	-	-	Z	C	75
BBRi	Branch on Mi=0	-	-	0	-	-	-	-	-	76
BBSi	Branch on Mi=1	-	-	0	-	-	-	-	-	76
BCC	Branch on C=0	-	-	0	-	-	-	-	-	77
BCS	Branch on C=1	-	-	0	-	-	-	-	-	77
BEQ	Branch on Z=1	-	-	0	-	-	-	-	-	78
BIT	A∧M	M_7	M_6	0	-	-	-	Z	-	78
BMI	Branch on N=1	-	-	0	-	-	-	-	-	79
BNE	Branch on Z=0	-	-	0	-	-	-	-	-	79
BPL	Branch on N=0	-	-	0	-	-	-	-	-	80
BRA	Branch Always	-	-	0	-	-	-	-	-	80
BRK	Break	-	-	0	1	0	1	-	-	81
BSR	Branch Subroutine	-	-	0	-	-	-	-	-	82
BVC	Branch on V=0	-	-	0	-	-	-	-	-	83
BVS	Branch on V=1	-	-	0	-	-	-	-	-	83
CLA	A←00_{16}	-	-	0	-	-	-	-	-	84
CLC	C←0	-	-	0	-	-	-	-	0	84
CLD	D←0	-	-	0	-	0	-	-	-	85
CLI	I←0	-	-	0	-	-	0	-	-	85
CLV	V←0	-	0	0	-	-	-	-	-	86
CLX	X←00_{16}	-	-	0	-	-	-	-	-	86
CLY	Y←00_{16}	-	-	0	-	-	-	-	-	87
CMP	A−M	N	-	0	-	-	-	Z	C	88
CPX	X−M	N	-	0	-	-	-	Z	C	89
CPY	Y−M	N	-	0	-	-	-	Z	C	89
DEC	M←M−1 or A←A−1	N	-	0	-	-	-	Z	-	90
DEX	X←X−1	N	-	0	-	-	-	Z	-	91
DEY	Y←Y−1	N	-	0	-	-	-	Z	-	91

表2-3-2

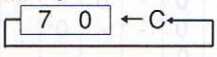
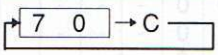
ニーモニック	機能	フラグ N	V	T	B	D	I	Z	C	参照ページ
EOR	A←A∀M	N	-	0	-	-	-	Z	-	92
INC	M←M+1 or A←A+1	N	-	0	-	-	-	Z	-	93
INX	X←X+1	N	-	0	-	-	-	Z	-	94
INY	Y←Y+1	N	-	0	-	-	-	Z	-	94
JMP	Jump to New Location	-	-	0	-	-	-	-	-	95
JSR	Jump to Subroutine	-	-	0	-	-	-	-	-	96
LDA	A←M	N	-	0	-	-	-	Z	-	97
LDX	X←M	N	-	0	-	-	-	Z	-	98
LDY	Y←M	N	-	0	-	-	-	Z	-	98
LSR	0→ [7 0] →C	0	-	0	-	-	-	Z	C	99
NOP	No Operation	-	-	0	-	-	-	-	-	100
ORA	A←A∨M	N	-	0	-	-	-	Z	-	101
PHA	Ms←A, S←S−1	-	-	0	-	-	-	-	-	102
PHP	Ms←P, S←S−1	-	-	0	-	-	-	-	-	102
PHX	Ms←X, S←S−1	-	-	0	-	-	-	-	-	103
PHY	Ms←Y, S←S−1	-	-	0	-	-	-	-	-	103
PLA	S←S+1, A←Ms	N	-	0	-	-	-	Z	-	104
PLP	S←S+1, P←Ms	(RESTORED)								104
PLX	S←S+1, X←Ms	N	-	0	-	-	-	Z	-	105
PLY	S←S+1, Y←Ms	N	-	0	-	-	-	Z	-	105
RMBi	Mi←0	-	-	0	-	-	-	-	-	106
ROL		N	-	0	-	-	-	Z	C	106
ROR		N	-	0	-	-	-	Z	C	107
RTI	Return from Interrupt	(RESTORED)								107
RTS	Return from Subroutine	-	-	0	-	-	-	-	-	108
SAX	A←→X	-	-	0	-	-	-	-	-	108
SAY	A←→Y	-	-	0	-	-	-	-	-	109
SBC	A←A−M−$\overline{C}$	N	V	0	-	-	-	Z	C	110
SEC	C←1	-	-	0	-	-	-	-	1	111
SED	D←1	-	-	0	-	1	-	-	-	111
SEI	I←1	-	-	0	-	-	1	-	-	112
SET	T←1	-	-	1	-	-	-	-	-	112
SMBi	Mi←1	-	-	0	-	-	-	-	-	113

表2-3-3

ニーモニック	機能	フラグ								参照ページ
		N	V	T	B	D	I	Z	C	
ST0	HuC6270 : (A1,A0)=(0,0)←IM	-	-	0	-	-	-	-	-	113
ST1	HuC6270 : (A1,A0)=(1,0)←IM	-	-	0	-	-	-	-	-	114
ST2	HuC6270 : (A1,A0)=(1,1)←IM	-	-	0	-	-	-	-	-	115
STA	M←A	-	-	0	-	-	-	-	-	116
STX	M←X	-	-	0	-	-	-	-	-	116
STY	M←Y	-	-	0	-	-	-	-	-	117
STZ	M←00_{16}	-	-	0	-	-	-	-	-	117
SXY	X←→Y	-	-	0	-	-	-	-	-	118
TAI	Transfer Block Data(INC←ALT)	-	-	0	-	-	-	-	-	119
TAMi	MPRi←A	-	-	0	-	-	-	-	-	120
TAX	X←A	N	-	0	-	-	-	Z	-	121
TAY	Y←A	N	-	0	-	-	-	Z	-	121
TDD	Transfer Block Data(DEC←DEC)	-	-	0	-	-	-	-	-	122
TIA	Transfer Block Data(ALT←INC)	-	-	0	-	-	-	-	-	123
TII	Transfer Block Data(INC←INC)	-	-	0	-	-	-	-	-	125
TIN	Transfer Block Data(FIX←INC)	-	-	0	-	-	-	-	-	126
TMAi	A←MPRi	-	-	0	-	-	-	-	-	127
TRB	M←$\bar{A}\wedge M$	M_7	M_6	0	-	-	-	Z	-	127
TSB	M←A∨M	M_7	M_6	0	-	-	-	Z	-	128
TST	M∧IM	M_7	M_6	0	-	-	-	Z	-	128
TSX	X←S	N	-	0	-	-	-	Z	-	129
TXA	A←X	N	-	0	-	-	-	Z	-	129
TXS	S←X	-	-	0	-	-	-	-	-	130
TYA	A←Y	N	-	0	-	-	-	Z	-	130

ニーモニック解説

命令の説明

本章では、HuC6280の持つ89種の命令のおのおのについて、その機能、アセンブリ言語での記述方法、ステータスレジスタへの影響などを記述しています。
各説明は、以下の表記方法に従がっており、本マニュアルにおいて統一されています。

Function: ——— 各命令の機能について説明しています。説明は、
- ●詳細機能説明
- ●概略機能説明

の2つのプログラムにわかれています。

Instruction: ——— 各命令について、そのアドレッシングモード、アセンブリ言語での記述方法(ニーモニック)、マシンコード、バイト数、およびサイクル数を記述しています。

- ●アドレッシングモード：各命令の持つアドレッシングモードを示します。記述方法は表2-1-1(64ページ参照)に従います。
- ●ニーモニック：各命令について、アセンブリ言語での記述方法を示します。記述の中で、記号は表2-1-2(65ページ参照)に従います。
- ●マシンコード：各命令について、そのマシンコードを示します。数値は16進表記で示します。マシンコードは先頭にオペコードを持ち、必要に応じてオペランドが続いて並びます。
- ●バイト数：各命令について、そのマシンコードのバイト数を示します。
- ●サイクル数：各命令についてその命令実行サイクル数を示します。ここでの1サイクルとは、1つのバスサイクル(リードサイクル、またはライトサイクル、またはダミーサイクル)です。

Flags: ——— 各命令の実行により、ステータスレジスタがどのように変化するかを示します。記号は表2-1-3(65ページ参照)に従います。

ADC (Add with Carry)

Function: Tにより、以下の2つの場合があります。

i) T=1のとき(直前にSET命令を実行したとき)
M(X)とMとCを加算して、結果をM(X)に入れます。サイクル数は+3されます。

$$M_{(X)} \leftarrow M_{(X)} + M + C$$

ii) T=0のとき(直前にSET命令を実行していないとき)
AとMとCを加算して、結果をAに入れます。

$$A \leftarrow A + M + C$$

Dにより、以下の2つの場合があります。

i) D=1のとき(事前にSED命令を実行したとき)
10進法の加算演算を行います。サイクル数は +1されます。またVは変化しません。

ii) T=0のとき
2進法の加算演算を行います。

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ ADC ␣ #nn	69 , nn	2	2
ZP	␣ ADC ␣ ZZ	65 , ZZ	2	4
ZP , X	␣ ADC ␣ ZZ , X	75 , ZZ	2	4
(IND)	␣ ADC ␣ (ZZ)	72 , ZZ	2	7
(IND , X)	␣ ADC ␣ (ZZ , X)	61 , ZZ	2	7
(IND) , Y	␣ ADC ␣ (ZZ) , Y	71 , ZZ	2	7
ABS	␣ ADC ␣ hhll	6D , ll , hh	3	5
ABS , X	␣ ADC ␣ hhll , X	7D , ll , hh	3	5
ABS , Y	␣ ADC ␣ hhll , Y	79 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	V	0	–	–	–	Z	C

AND (And)

Function: Tにより、以下の2つの場合があります。

i) T=1のとき(直前にSET命令を実行したとき)
M(X)とMとの論理積をとり、結果をM(X)に入れます。サイクル数は+3されます。

M(X) ← M(X)∧M

ii) T=0のとき(直前にSET命令を実行していないとき)
AとMとの論理積をとり、結果をAに入れます。

A←A∧M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣AND␣ #nn	29 , nn	2	2
ZP	␣AND␣ ZZ	25 , ZZ	2	4
ZP , X	␣AND␣ ZZ , X	35 , ZZ	2	4
(IND)	␣AND␣ (ZZ)	32 , ZZ	2	7
(IND , X)	␣AND␣ (ZZ , X)	21 , ZZ	2	7
(IND) , Y	␣AND␣ (ZZ) , Y	31 , ZZ	2	7
ABS	␣AND␣ hhll	2D , ll , hh	3	5
ABS , X	␣AND␣ hhll , X	3D , ll , hh	3	5
ABS , Y	␣AND␣ hhll , Y	39 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

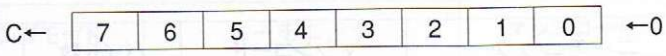
ASL(Shift Left)

Function: メモリまたはアキュムレータを、1ビット分左方向へシフトします。
このとき

M_0またはA_0には0が

CにはM_7またはA_7が

セットされます。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ASL␣ZZ	06 , ZZ	2	6
ZP , X	␣ASL␣ZZ , X	16 , ZZ	2	6
ABS	␣ASL␣hhll	0E , ll , hh	3	7
ABS , X	␣ASL␣hhll , X	1E , ll , hh	3	7
ACC	␣ASL␣A	0A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	C

※16bit以上のデータを左方向へシフトする場合は、ROL命令と組み合わせて使うとよい。この場合、下位バイトから行うのがポイント。

例：

```
asl   memory
rol   memory+1
 ⋮
 ⋮
```

BBRi (Branch on Bit Reset)

Function: 指定されたゼロページのビットが "0" のとき、プログラムは指定されたアドレスへ相対分岐し、サイクル数は+2されます。
上記の指定ビットが "1" のときは、プログラムカウンタを+3増加するのみです。

PC←PC+3+rr　if Mi=0

PC←PC+3　if Mi=1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP , REL	␣ BBRi ␣ ZZ , hhll	10i+F , ZZ , rr	3	6

NOTE : rr=hhll−(PC+3)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BBSi (Branch on Bit Set)

Function: 指定されたゼロページのビットが "1" のとき、プログラムは指定されたアドレスへ相対分岐し、サイクル数は+2されます。
上記の指定ビットが "0" のときは、プログラムカウンタを+3増加するのみです。

PC←PC+3+rr　if Mi=1

PC←PC+3　if Mi=0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP , REL	␣ BBSi ␣ ZZ , hhll	10i+8F , ZZ , rr	3	6

NOTE : rr=hhll−(PC+3)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BCC (Branch on Carry Clear)

Function: Cが "0" のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Cが "1" のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if C=0

PC←PC+2　　if C=1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BCC␣hhll	90 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BCS (Branch on Carry Set)

Function: Cが "1" のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Cが "0" のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if C=1

PC←PC+2　　if C=0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BCS␣hhll	B0 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BEQ (Branch on Equal)

Function: Zが "1" のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Zが "0" のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if Z=1

PC←PC+2　　if Z=0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BEQ␣ hhll	F0 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BIT (Bit Test)

Function: アキュムレータとメモリの論理積をとります。結果はどこにもストアされません。
メモリのビット7とビット6がそれぞれネガティブフラグとオーバーフローフラグにセットされます。

A∧M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣BIT␣ #nn	89 , nn	2	2
ZP	␣BIT␣ ZZ	24 , ZZ	2	4
ZP , X	␣BIT␣ ZZ , X	34 , ZZ	2	4
ABS	␣BIT␣ hhll	2C , ll , hh	3	5
ABS , X	␣BIT␣ hhll, X	3C , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
M_7	M_6	0	−	−	−	Z	−

BMI (Branch on Minus)

Function: Nが “1” のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Nが “0” のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if N=1

PC←PC+2　　if N=0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BMI␣hhll	30 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BNE (Branch on Not Equal)

Function: Zが "0" のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Zが "1" のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if Z=0

PC←PC+2　　if Z=1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BNE␣hhll	D0 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BPL (Branch on Plus)

Function: Nが “0” のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Nが “1” のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if N=0

PC←PC+2　　if N=1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BPL␣ hhll	10 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BRA (Branch Always)

Function: 指定されたアドレスへ相対分岐します。

PC←PC+2+rr

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BRA␣ hhll	80 , rr	2	4

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

FLAGS:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BRK (Break)

Function: スタックにプログラムカウンタと、ステータスレジスタの内容をPCH、PCL、Pの順に退避した後、論理アドレスで
FFF6$_{16}$番地から　下位アドレスを
FFF7$_{16}$番地から　上位アドレスを
読み出して、割込み処理ルーチンへサブルーチンコールします。このときスタックへ退避されるステータスレジスタ中のBは “1” にセットされます。またスタックに退避されるプログラムカウンタの値は(BRK命令+2)のアドレスです。

PC←PC+2
Ms←PCH　　, S←S−1
Ms←PCL　　, S←S−1
Ms←P　　, S←S−1
PCL←(FFF6$_{16}$)
PCH←(FFF7$_{16}$)

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	BRK	00	1	8

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	1	0	1	−	−

※デバッグなどの目的で使用される命令だが、とくにユーザーが使用する必要はない。
BRKによる割り込みからの復帰には、RTI命令を使用する。

BSR (Branch Subroutine)

Function: スタックにプログラムカウンタを、PCH、PCLの順に退避した後、指定されたアドレスへ相対分岐します。スタックに退避されるプログラムカウンタの値は、BSR命令のラストバイトのアドレスです。

PC←PC+1

Ms←PCH , S←S−1

Ms←PCL , S←S−1

PC←PC+2+rr

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BSR␣ hhll	44 , rr	2	8

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

※分岐先からの復帰には、RTS命令を使用する。

BVC (Branch on V Clear)

Function: Vが “0” のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Vが “1” のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if V=0

PC←PC+2　　if V=1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BVC␣hhll	50 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

BVS (Branch on V Set)

Function: Vが “1” のとき、指定されたアドレスへ相対分岐し、サイクル数は+2されます。
Vが “0” のときは、プログラムカウンタを+2増加するのみです。

PC←PC+2+rr　if V=1

PC←PC+2　　if V=0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
REL	␣BVS␣hhll	70 , rr	2	2

NOTE : rr=hhll−(PC+2)
−128(10進)≦rr≦127(10進)

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

CLA (Clear A)

Function: アキュムレータをクリアします。

$A \leftarrow 00_{16}$

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ CLA	62	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

CLC (Clear C)

Function: キャリーフラグをクリアします。

$C \leftarrow 0$

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ CLC	18	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	0

CLD (Clear D)

Function: デシマルフラグを、クリアします。

D←0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣CLD	D8	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	0	–	–	–

CLI (Clear I)

Function: インタラプトディセーブルフラグ(I)をクリアします。

I←0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣CLI	58	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	0	–	–

※インタラプトディセーブルフラグは、割込みに大きく関わるので、とくに操作する必要はない。

CLV (Clear V)

Function: オーバーフローフラグ(V)をクリアします。

V←0

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ CLV	B8	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	0	0	–	–	–	–	–

CLX (Clear X)

Function: Xレジスタをクリアします。

X←00_{16}

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ CLX	82	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

CLY (Clear Y)

Function: Yレジスタをクリアします。

$Y \leftarrow 00_{16}$

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ CLY	C2	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

CMP (Compare A with M)

Function: アキュムレータの内容とメモリの内容を減算により比較します。
結果は、どこにもストアしません。

A−M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣CMP␣ #nn	C9 , nn	2	2
ZP	␣CMP␣ ZZ	C5 , ZZ	2	4
ZP , X	␣CMP␣ ZZ , X	D5 , ZZ	2	4
(IND)	␣CMP␣ (ZZ)	D2 , ZZ	2	7
(IND , X)	␣CMP␣ (ZZ , X)	C1 , ZZ	2	7
(IND) , Y	␣CMP␣ (ZZ) , Y	D1 , ZZ	2	7
ABS	␣CMP␣ hhll	CD , ll , hh	3	5
ABS , X	␣CMP␣ hhll , X	DD , ll , hh	3	5
ABS , Y	␣CMP␣ hhll , Y	D9 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	−	0	−	−	−	Z	C

※Aレジスタの値とメモリまたはイミディエイトデータとを比較する場合に使用する。
例：AREG＞#20 かの判定

```
        cmp   #20
        bcc   False
        beq   False
        jmp   True        ;Aレジスタ＞#20 の処理へ分岐
False:                    ;そうでなかった時の処理
        ⋮
        ⋮
```

CPX (Compare X with M)

Function:Xレジスタの内容とメモリの内容を減算により比較します。
結果は、どこにもストアしません。

X−M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ CPX ␣ #nn	E0,nn	2	2
ZP	␣ CPX ␣ ZZ	E4,ZZ	2	4
ABS	␣ CPX ␣ hhll	EC,ll,hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	−	0	−	−	−	Z	C

CPY (Compare Y with M)

Function:Yレジスタの内容とメモリの内容を減算により比較します。
結果は、どこにもストアしません。

Y−M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ CPY ␣ #nn	C0 , nn	2	2
ZP	␣ CPY ␣ ZZ	C4 , ZZ	2	4
ABS	␣ CPY ␣ hhll	CC , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	−	0	−	−	−	Z	C

DEC (Decrement)

Function: メモリまたは、アキュムレータの内容を−1します。

M← M−1

または

A← A−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ DEC ␣ ZZ	C6 , ZZ	2	6
ZP , X	␣ DEC ␣ ZZ , X	D6 , ZZ	2	6
ABS	␣ DEC ␣ hhll	CE , ll , hh	3	7
ABS , X	␣ DEC ␣ hhll , X	DE , ll , hh	3	7
ACC	␣ DEC ␣ A	3A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	−	0	−	−	−	Z	−

※Aレジスタまたはメモリの値が$00になるとゼロフラグがセットされる。また、値が$00のときにDECを実行すると、$FFになるが、キャリーフラグは変化しない。

DEX (Decrement X)

Function: Xレジスタの内容を、−1します。

X←X−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ DEX	CA	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	−	0	−	−	−	Z	−

※Xレジスタまたはメモリの値が$00になるとゼロフラグがセットされる。また、値が$00のときにDEXを実行すると、$FFになるが、キャリーフラグは変化しない。

DEY (Decrement Y)

Function: Yレジスタの内容を、−1します。

Y←Y−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ DEY	88	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	−	0	−	−	−	Z	−

※Yレジスタまたはメモリの値が$00になるとゼロフラグがセットされる。また、値が$00のときにDEYを実行すると、$FFになるが、キャリーフラグは変化しない。

EOR (Exclusive OR)

Function: Tにより、以下の2つの場合があります。

i) T=1のとき(直前にSET命令を実行したとき)
M(x)とMとの排他的論理和をとり、結果をM(x)に入れます。
サイクル数は+3されます。

$$M_{(x)} \leftarrow M_{(x)} \forall M$$

ii) T=0のとき(直前にSET命令を実行していないとき)
AとMとの排他的論理和をとり結果をAに入れます。

$$A \leftarrow A \forall M$$

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	⌙ EOR⌙ #nn	49 , nn	2	2
ZP	⌙ EOR⌙ ZZ	45 , ZZ	2	4
ZP , X	⌙ EOR⌙ ZZ , X	55 , ZZ	2	4
(IND)	⌙ EOR⌙ (ZZ)	52 , ZZ	2	7
(IND , X)	⌙ EOR⌙ (ZZ , X)	41 , ZZ	2	7
(IND) , Y	⌙ EOR⌙ (ZZ) , Y	51 , ZZ	2	7
ABS	⌙ EOR⌙ hhll	4D , ll , hh	3	5
ABS , X	⌙ EOR⌙ hhll , X	5D , ll , hh	3	5
ABS , Y	⌙ EOR⌙ hhll , Y	59 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

INC (Increment)

Function: メモリまたはアキュムレータの内容を、+1します。

M←M+1

または

A←A+1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣INC ␣ZZ	E6 , ZZ	2	6
ZP , X	␣INC ␣ZZ , X	F6 , ZZ	2	6
ABS	␣INC ␣hhll	EE , ll , hh	3	7
ABS , X	␣INC ␣hhll , X	FE , ll , hh	3	7
ACC	␣INC ␣A	1A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

※Aレジスタまたはメモリの値が$00になればゼロフラグは変化するが、キャリーフラグは変化しない。また、値が$FFのときにINCを実行すると、$00になる。

INX (Increment X)

Function: Xレジスタの内容を、+1します。

X←X+1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ INX	E8	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

※Xレジスタが$00になればゼロフラグは変化するが、キャリーフラグは変化しない。
また、値が$FFのときにINXを実行すると、$00になる。

INY (Increment Y)

Function: Yレジスタの内容を、+1します。

Y←Y+1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ INY	C8	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

※Yレジスタが$00になればゼロフラグは変化するが、キャリーフラグは変化しない。
また、値が$FFのときにINYを実行すると、$00になる。

JMP (Jump to New Location)

Function: 指定されたアドレスに分岐します。

i) アドレッシングモードがABSのとき

PCL←ll
PCH←hh

ii) アドレッシングモードが(ABS)のとき

PCL←(hhll)
PCH←(hhll+1)

iii) アドレッシングモードが(ABS,X)のとき

PCL←(hhll+X)
PCH←(hhll+X+1)

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ABS	␣ JMP ␣ hhll	4C ,ll , hh	3	4
(ABS)	␣ JMP ␣ (hhll)	6C ,ll , hh	3	7
(ABS,X)	␣ JMP ␣ (hhll , X)	7C ,ll , hh	3	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

JSR (Jump to Subroutine)

Function: スタックにプログラムカウンタをPCH、PCLの順に退避した後、指定されたアドレスへ分岐します。
スタックに退避されるプログラムカウンタの値は、JSR命令のラストバイトのアドレスです。

PC←PC+2
Ms←PCH , S←S−1
Ms←PCL , S←S−1
PC←hhll

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ABS	␣ JSR ␣ hhll	20 , ll , hh	3	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
−	−	0	−	−	−	−	−

※分岐先からの復帰には、RTS命令を使用する。

LDA (Load A)

Function: メモリの内容を、アキュムレータにロードします。

A←M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ LDA ␣ #nn	A9 , nn	2	2
ZP	␣ LDA ␣ ZZ	A5 , ZZ	2	4
ZP , X	␣ LDA ␣ ZZ , X	B5 , ZZ	2	4
(IND)	␣ LDA ␣ (ZZ)	B2 , ZZ	2	7
(IND , X)	␣ LDA ␣ (ZZ , X)	A1 , ZZ	2	7
(IND) , Y	␣ LDA ␣ (ZZ) , Y	B1 , ZZ	2	7
ABS	␣ LDA ␣ hhll	AD, ll , hh	3	5
ABS , X	␣ LDA ␣ hhll , X	BD, ll , hh	3	5
ABS , Y	␣ LDA ␣ hhll , Y	B9 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

LDX (Load X)

Function: メモリの内容を、Xレジスタにロードします。

X←M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ LDX ␣ #nn	A2, nn	2	2
ZP	␣ LDX ␣ ZZ	A6, ZZ	2	4
ZP, Y	␣ LDX ␣ ZZ, Y	B6, ZZ	2	4
ABS	␣ LDX ␣ hhll	AE, ll, hh	3	5
ABS, Y	␣ LDX ␣ hhll, Y	BE, ll, hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

LDY (Load Y)

Function: メモリの内容を、Yレジスタにロードします。

Y←M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ LDY ␣ #nn	A0, nn	2	2
ZP	␣ LDY ␣ ZZ	A4, ZZ	2	4
ZP, X	␣ LDY ␣ ZZ, X	B4, ZZ	2	4
ABS	␣ LDY ␣ hhll	AC, ll, hh	3	5
ABS, X	␣ LDY ␣ hhll, X	BC, ll, hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

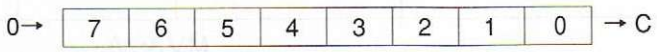
LSR (Logical Shift Right)

Function: メモリまたはアキュムレータを、1ビット分右方向へシフトします。
このとき

M_7またはA_7には0が

CにはM_0またはA_0が

セットされます。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣LSR␣ZZ	46 , ZZ	2	6
ZP , X	␣LSR␣ZZ , X	56 , ZZ	2	6
ABS	␣LSR␣hhll	4E , ll , hh	3	7
ABS , X	␣LSR␣hhll	5E , ll , hh	3	7
ACC	␣LSR␣A	4A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
0	–	0	–	–	–	Z	C

※16bit以上のデータを右方向へシフトする場合は、ROR命令と組み合わせて使うとよい。この場合、上位バイトから行うのがポイント。

例：

```
lsr    memory +1
ror    memory
 ⋮
 ⋮
```

NOP (No Operation)

Function: プログラムカウンタをインクリメントします（なにもせずに時間を費やす命令）。

PC←PC+1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	NOP	EA	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

ORA (OR)

Function: Tにより、以下の2つの場合があります。

i) T=1 のとき(直前にSET命令を実行したとき)
M(x)とMとの論理和をとり、結果をM(x)に入れます。サイクル数は+3されます。

M(x)←M(x)∨M

ii) T=0のとき(直前にSET命令を実行していないとき)
AとMとの論理和をとり、結果をAに入れます。

A←A∨M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ORA␣ #nn	09 , nn	2	2
ZP	␣ORA␣ ZZ	05 , ZZ	2	4
ZP , X	␣ORA␣ ZZ , X	15 , ZZ	2	4
(IND)	␣ORA␣ (ZZ)	12 , ZZ	2	7
(IND , X)	␣ORA␣ (ZZ , X)	01 , ZZ	2	7
(IND) , Y	␣ORA␣ (ZZ) , Y	11 , ZZ	2	7
ABS	␣ORA␣ hhll	0D , ll , hh	3	5
ABS , X	␣ORA␣ hhll , X	1D , ll , hh	3	5
ABS , Y	␣ORA␣ hhll , Y	19 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

PHA (Push A)

Function: アキュムレータの内容をスタックに退避します。

Ms←A
S←S−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PHA	48	1	3

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※退避した内容をAレジスタに戻すには、PLA命令を使う。

PHP (Push P)

Function: ステータスレジスタの内容をスタックに退避します。

Ms←P
S←S−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PHP	08	1	3

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※退避した内容をステータスレジスタに戻すには、PLP命令を使う。

PHX (Push X)

Function: Xレジスタの内容をスタックに退避します。

Ms←X
S←S−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PHX	DA	1	3

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※退避した内容をXレジスタに戻すには、PLX命令を使う。

PHY (Push Y)

Function: Yレジスタの内容をスタックに退避します。

Ms←Y
S←S−1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PHY	5A	1	3

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※退避した内容をYレジスタに戻すには、PLY命令を使う。

PLA (Pull A)

Function: スタックから、データをアキュムレータにプルします。

S←S+1
A←Ms

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PLA	68	1	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

PLP (Pull P)

Function: スタックから、データをステータスレジスタにプルします。

S←S+1
P←Ms

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PLP	28	1	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
(R	E	S	T	O	R	E	D)

PLX (Pull X)

Function: スタックから、データをXレジスタにプルします。

S←S+1
X←Ms

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PLX	FA	1	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

PLY (Pull Y)

Function: スタックから、データをYレジスタにプルします。

S←S+1
Y←Ms

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ PLY	7A	1	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

RMBi (Reset Memory Bit)

Function: ゼロページのメモリの指定されたビットをリセットします。

Mi←0

Instruction:

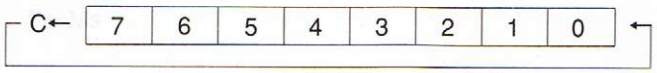
アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ RMBi␣ ZZ	10i+7,ZZ	2	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

ROL (Rotate Left)

Function: メモリまたはアキュムレータを、キャリーフラグを含めて1ビット分、左方向にローテイトします。



Instruction:

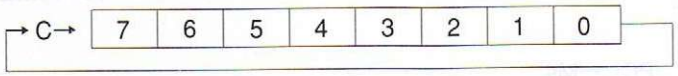
アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ ROL␣ ZZ	26 , ZZ	2	6
ZP , X	␣ ROL␣ ZZ , X	36 , ZZ	2	6
ABS	␣ ROL␣ hhll	2E , ll , hh	3	7
ABS , X	␣ ROL␣ hhll, X	3E , ll , hh	3	7
ACC	␣ ROL␣ A	2A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	C

ROR (Rotate Right)

Function: メモリまたはアキュムレータを、キャリーフラグを含めて1ビット分、右方向にローテイトします。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ ROR␣ ZZ	66 , ZZ	2	6
ZP , X	␣ ROR␣ ZZ , X	76 , ZZ	2	6
ABS	␣ ROR␣ hhll	6E , ll , hh	3	7
ABS , X	␣ ROR␣ hhll, X	7E , ll , hh	3	7
ACC	␣ ROR␣ A	6A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	C

RTI (Return from Interrupt)

Function: スタックから、データをステータスレジスタ、プログラムカウンタの下位バイト、プログラムカウンタの上位バイトの順にプルします。
プログラムは、プログラムカウンタにより指定されるアドレスに分岐します。

S←S+1 , P←Ms
S←S+1 , PCL←Ms
S←S+1 , PCH←Ms

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ RTI	40	1	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
(R	E	S	T	O	R	E	D)

※RTI命令は、タイマー割込みなどによって呼び出された処理から復帰する場合に使用する。

RTS (Return from Subroutine)

Function: スタックからデータを、プログラムカウンタの下位バイト、プログラムカウンタの上位バイトの順にプルします。
その後、プログラムカウンタを+1してから、プログラムカウンタにより指定されるアドレスに分岐します。

S←S+1 , PCL←Ms
S←S+1 , PCH←Ms
PC←PC+1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ RTS	60	1	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※RTS命令は、JSR命令やBSR命令で呼び出された処理から復帰する場合に使用する。

SAX (Swap A for X)

Function: アキュムレータと、Xレジスタの内容を入れ換えます。

A←→X

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ SAX	22	1	3

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

SAY (Swap A for Y)

Function: アキュムレータと、Yレジスタの内容を入れ換えます。

A⟷Y

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣SAY	42	1	3

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

SBC (Subtract with Carry)

Function: AからMと$\overline{C}$(キャリーフラグの値を反転させたもの）を減算して、結果をAに入れます。

$$A \leftarrow A - M - \overline{C}$$

Dにより、以下の2つの場合があります。

i) D=1のとき
10進の減算演算を行います。サイクル数は+1されます。また、Vは変化しません。

ii) D=0のとき
2進の減算演算を行います。

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ SBC ␣ #nn	E9 , nn	2	2
ZP	␣ SBC ␣ ZZ	E5 , ZZ	2	4
ZP , X	␣ SBC ␣ ZZ , X	F5 , ZZ	2	4
(IND)	␣ SBC ␣ (ZZ)	F2 , ZZ	2	7
(IND , X)	␣ SBC ␣ (ZZ , X)	E1 , ZZ	2	7
(IND) , Y	␣ SBC ␣ (ZZ) , Y	F1 , ZZ	2	7
ABS	␣ SBC ␣ hhll	ED, ll , hh	3	5
ABS , X	␣ SBC ␣ hhll , X	FD, ll , hh	3	5
ABS , Y	␣ SBC ␣ hhll , Y	F9 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	V	0	–	–	–	Z	C

※演算実行後、キャリーフラグが 0 であれば桁下がりがあったことを示す。
演算前にキャリーフラグをセットするには、SEC命令を使うとよい。

例：

```
lda   #10
sec
sbc   #5    ;Aレジスタ= #10 –#5
 ⋮
 ⋮
```

SEC (Set C)

Function: キャリーフラグを、セットします。

C←1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ SEC	38	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	1

SED (Set D)

Function: デシマルフラグを、セットします。

D←1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ SED	F8	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	1	–	–	–

SEI (Set I)

Function: インタラプトディセーブルフラグ(I)を、セットします。

I←1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ SEI	78	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	1	–	–

※インタラプトディセーブルフラグは、割込みに大きく関わるので、とくに操作する必要はない。

SET (Set T)

Function: メモリ演算フラグを、セットします。

T←1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ SET	F4	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	1	–	–	–	–	–

SMBi (Set Memory Bit)

Function: ゼロページのメモリの指定されたビットをセットします。

Mi←1

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ SMBi␣ ZZ	10i+87,ZZ	2	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

ST0 (Store HuC6270 No.1)

Function: HuC6270に、イミディエイトデータを転送します。
このとき

$\overline{CE7}$= “L” レベル
A1 = “L” レベル
A0 = “L” レベル

となります。

HuC6270:(A1,A0)=(0,0)←IM

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ ST0 ␣ #nn	03,nn	2	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※HuC6270の内部動作に詳しくないうちは使わないほうがよい。

ST1 (Store HuC6270 No.2)

Function: HuC6270に、イミディエイトデータを転送します。
このとき

$\overline{CE7}$= “L” レベル
A1 = “H” レベル
A0 = “L” レベル

となります。

HuC6270:(A1,A0)=(1,0)←IM

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ ST1␣ #nn	13,nn	2	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※HuC6270の内部動作に詳しくないうちは使わないほうがよい。

ST2 (Store HuC6270 No.3)

Function: HuC6270に、イミディエイトデータを転送します。
このとき

$\overline{CE7}$= “L” レベル
A1 = “H” レベル
A0 = “H” レベル

となります。

HuC6270:(A1,A0)=(1,1)←IM

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM	␣ ST2␣ #nn	23,nn	2	4

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※HuC6270の内部動作に詳しくないうちは使わないほうがよい。

STA (Store A)

Function: アキュムレータの内容を、メモリにストアします。

M←A

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ STA ␣ ZZ	85 , ZZ	2	4
ZP , X	␣ STA ␣ ZZ , X	95 , ZZ	2	4
(IND)	␣ STA ␣ (ZZ)	92 , ZZ	2	7
(IND , X)	␣ STA ␣ (ZZ , X)	81 , ZZ	2	7
(IND) , Y	␣ STA ␣ (ZZ) , Y	91 , ZZ	2	7
ABS	␣ STA ␣ hhll	8D , ll , hh	3	5
ABS , X	␣ STA ␣ hhll , X	9D , ll , hh	3	5
ABS , Y	␣ STA ␣ hhll , Y	99 , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

STX (Store X)

Function: Xレジスタの内容を、メモリにストアします。

M←X

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ STX ␣ ZZ	86 , ZZ	2	4
ZP , Y	␣ STX ␣ ZZ , Y	96 , ZZ	2	4
ABS	␣ STX ␣ hhll	8E , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

STY (Store Y)

Function: Yレジスタの内容を、メモリにストアします。

M←Y

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ STY ␣ ZZ	84 , ZZ	2	4
ZP , X	␣ STY ␣ ZZ , X	94 , ZZ	2	4
ABS	␣ STY ␣ hhll	8C , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

STZ (Store Zero)

Function: "00_{16}" を、メモリにストアします。

M←00_{16}

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ STZ ␣ ZZ	64 , ZZ	2	4
ZP , X	␣ STZ ␣ ZZ , X	74 , ZZ	2	4
ABS	␣ STZ ␣ hhll	9C , ll , hh	3	5
ABS , X	␣ STZ ␣ hhll, X	9E , ll , hh	3	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

SXY (Swap X for Y)

Function: Xレジスタと、Yレジスタの内容を入れ換えます。

X⟷Y

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	⌴ SXY	02	1	3

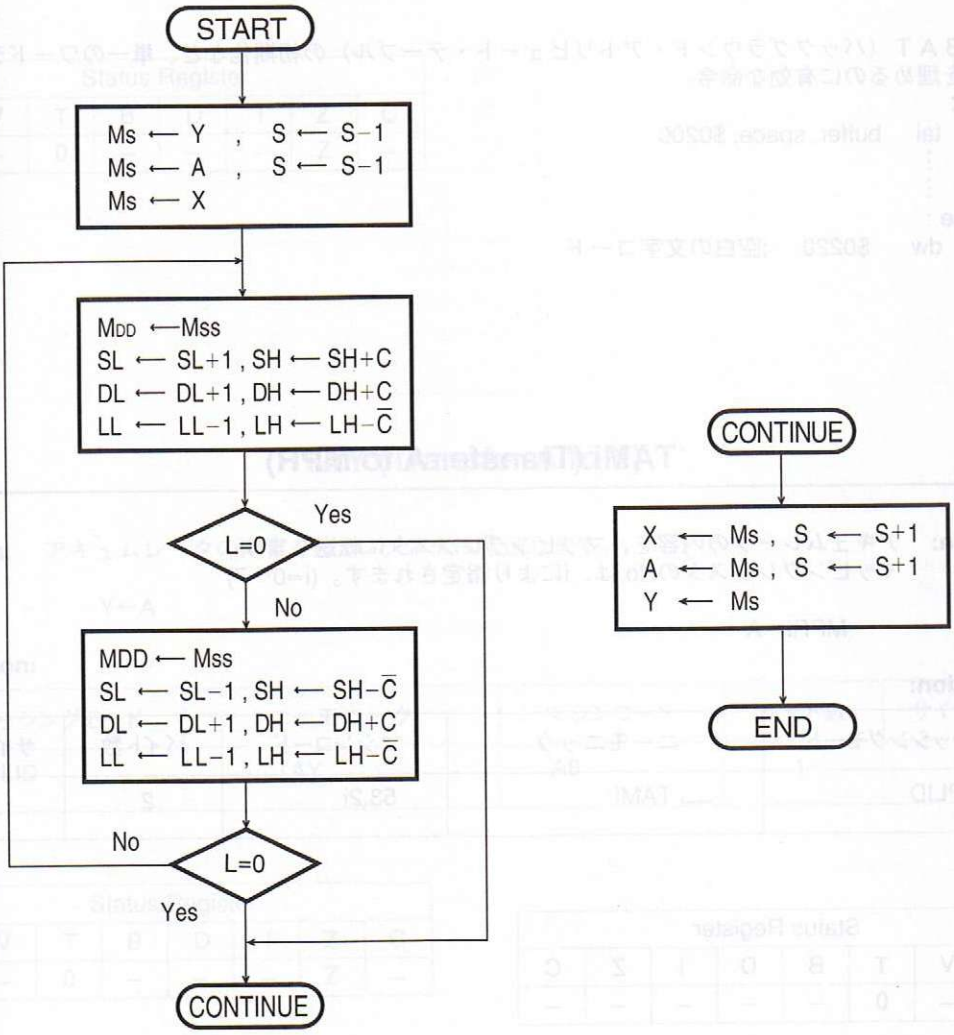
Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

TAI (Transfer Block Data)

Function: メモリからメモリへ、連続してデータを転送します。
ソースのメモリのアドレスは、1バイトの転送ごとに、インクリメントとデクリメントを交互に行います。
デスティネーションのメモリのアドレスは、1バイトの転送ごとにインクリメントします。
転送するデータのバイト数は、レングスにより指定します。
レングスが0のときは、65,536バイトを転送します。

TAI命令では命令の実行時に、内部レジスタ(A、X、Y)のデータ保存用にスタックを3レベル使用します。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TAI␣SHSL,DHDL, LHLL	F3,SL,SH,DL, DH,LL,LH	7	17+6x

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※仮想ＢＡＴ（バックグラウンド・アトリビュート・テーブル）の初期化など、単一のワードデータで領域を埋めるのに有効な命令。

例：

```
        tai     buffer, space, $0200
        :
        :
space :
        dw      $0220    ;空白の文字コード
```

TAMi (Transfer A to MPR)

Function: アキュムレータの内容を、マッピングレジスタに転送します。
マッピングレジスタのNo.は、iにより指定されます。(i=0～7)

MPRi←A

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TAMi	53,2i	2	5

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※マッピングレジスタの0～2および7番を不用意に変更すると不具合が生じる可能性があるので注意。

TAX (Transfer A to X)

Function: アキュムレータの内容を、Xレジスタに転送します。

X←A

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TAX	AA	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

TAY (Transfer A to Y)

Function: アキュムレータの内容を、Yレジスタに転送します。

Y←A

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TAY	A8	1	2

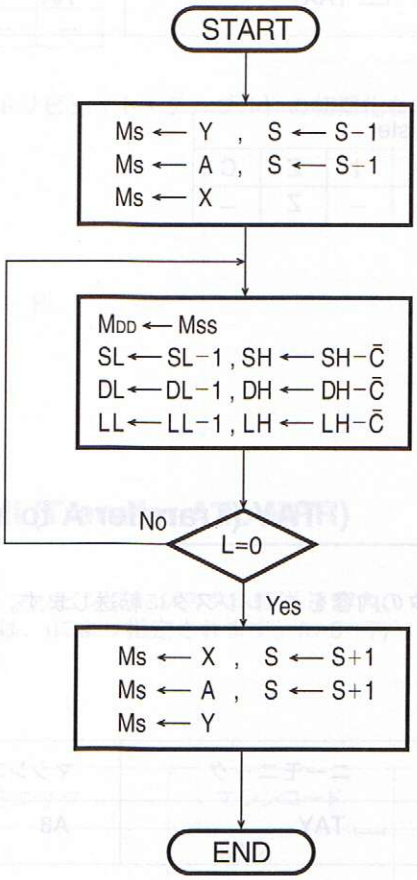
Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

TDD (Transfer Block Data)

Function: メモリからメモリへ、連続してデータを転送します。
ソースのメモリのアドレスは、1バイトの転送ごとに、デクリメントします。
デスティネーションのメモリのアドレスも、1バイトの転送ごとにデクリメントします。
転送するデータのバイト数は、レングスにより指定されます。
レングスが0のときは、65,536バイトを転送します。

TDD命令では、命令の実行時に、内部レジスタ(A、X、Y)のデータ保存用にスタックを3レベル使用します。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TDD ␣ SHSL, DHDL , LHLL	C3 , SL , SH , DL, DH, LL , LH	7	17+6x

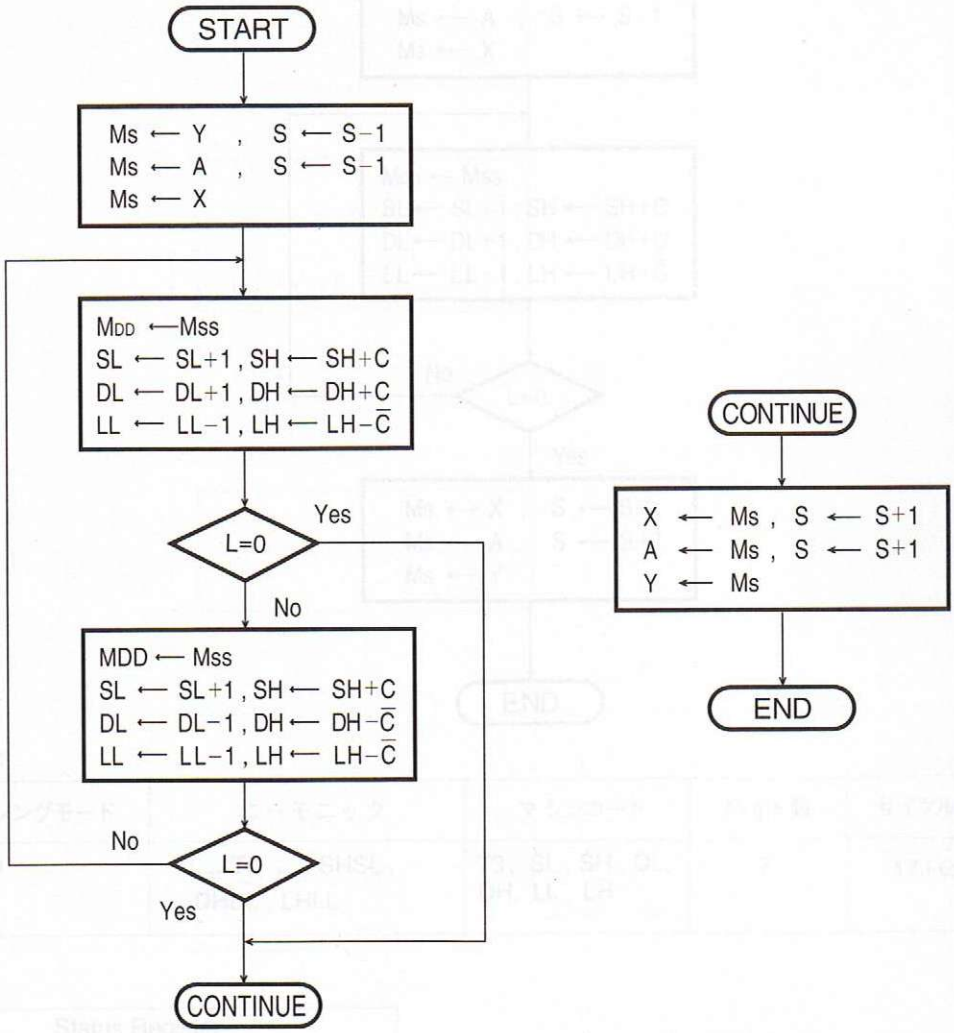
Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

TIA (Transfer Block Data)

Function: メモリからメモリへ、連続してデータを転送します。
ソースのメモリのアドレスは、1バイトの転送ごとに、インクリメントします。
デスティネーションのメモリのアドレスは、1バイトの転送ごとにインクリメントとデクリメントを交互に行います。
転送するデータのバイト数は、レングスにより指定されます。
レングスが0のときは、65,536バイトを転送します。

TIA命令では、命令の実行時に、内部レジスタ(A、X、Y)のデータ保持用にスタックを3レベル使用します。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TIA ␣ SHSL , DHDL , LHLL	E3 , SL , SH , DL, DH, LL , LH	7	17+6x

Flags:

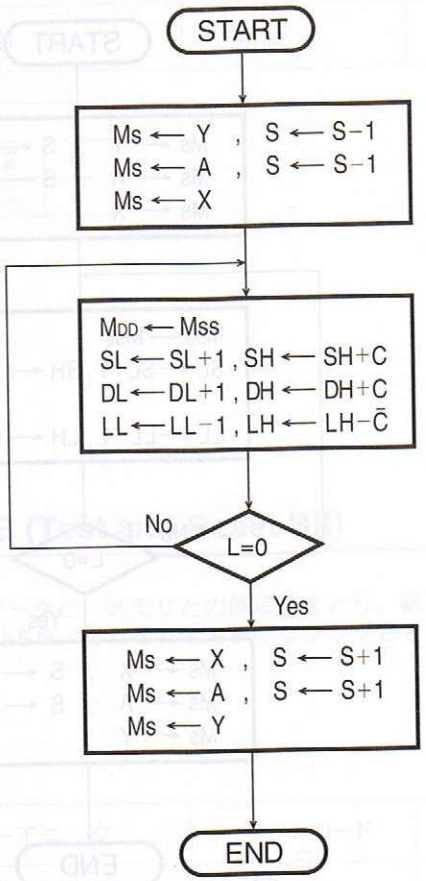
Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

※HuC6270からのデータ読み込みに有効な命令。

TII (Transfer Block Data)

Function: メモリからメモリへ、連続してデータを転送します。
ソースのメモリのアドレスは、1バイトの転送ごとにインクリメントします。
デスティネーションのメモリのアドレスも、1バイトの転送ごとにインクリメントします。
転送するデータのバイト数は、レングスにより指定されます。
レングスが0のときは、65,536バイトを転送します。

TII命令では、命令の実行時に、内部レジスタ(A、X、Y)のデータ保存用にスタックを3レベル使用します。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TII ␣ SHSL , DHDL , LHLL	73 , SL , SH , DL, DH, LL , LH	7	17+6x

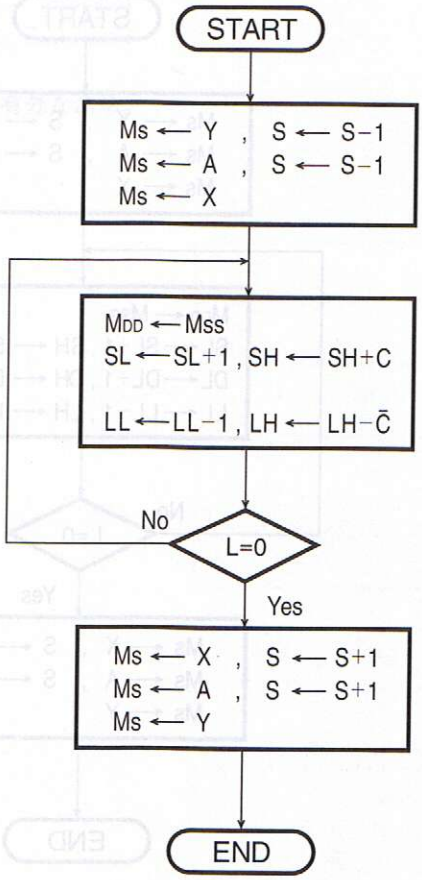
Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

TIN (Transfer Block Data)

Function: メモリからメモリへ、連続してデータを転送します。
ソースのメモリのアドレスは、1バイトの転送ごとに、インクリメントします。
デスティネーションのメモリのアドレスは、固定です。
転送するデータのバイト数は、レングスにより指定されます。
レングスが0のときは、65,536バイトを転送します。

TIN命令では、命令の実行時に、内部レジスタ(A、X、Y)のデータ保存用にスタックを3レベル使用します。



Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TIN ␣ SHSL , DHDL , LHLL	D3 , SL , SH , DL, DH, LL , LH	7	17+6x

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

TMAi (Transfer MPR to A)

Function: マッピングレジスタの内容を、アキュムレータに転送します。
マッピングレジスタNo.は、iにより指定されます。(i=0～7)

A←MPRi

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TMAi	43,2i	2	4

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

TRB (Test and Reset Bit)

Function: アキュムレータの反転データと、メモリとの論理積をとり、結果をメモリにストアします。
メモリのビット7とビット6が、それぞれネガティブフラグとオーバーフローフラグにセットされます。

$M \leftarrow \overline{A} \wedge M$

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣ TRB␣ ZZ	14, ZZ	2	6
ABS	␣ TRB␣ hhll	1C, ll, hh	3	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
M_7	M_6	0	–	–	–	Z	–

TSB (Test and Set Bit)

Function: アキュムレータと、メモリとの論理和をとり、結果をメモリにストアします。
メモリのビット7とビット6が、それぞれネガティブフラグとオーバーフローフラグにセットされます。

M←A ∨M

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
ZP	␣TSB ␣ZZ	04, ZZ	2	6
ABS	␣TSB ␣hhll	0C, ll, hh	3	7

Flags:

Status Register							
N	V	T	B	D	I	Z	C
M_7	M_6	0	–	–	–	Z	–

TST (Test Memory)

Function: メモリと、イミディエータとの論理積をとります。
結果は、どこにもストアしません。
メモリのビット7とビット6が、それぞれネガティブフラグとオーバーフローフラグにセットされます。

M∧IM

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMM ZP	␣TST ␣#nn, ZZ	83, nn, ZZ	3	7
IMM ZP, X	␣TST ␣#nn, ZZ, X	A3, nn, ZZ	3	7
IMM ABS	␣TST ␣#nn, hhll	93, nn, ll, hh	4	8
IMM ABS, X	␣TST ␣#nn, hhll, X	B3, nn, ll, hh	4	8

Flags:

Status Register							
N	V	T	B	D	I	Z	C
M_7	M_6	0	–	–	–	Z	–

TSX (Transfer S to X)

Function: スタックポインタの内容を、Xレジスタに転送します。

X←S

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TSX	BA	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

TXA (Transfer X to A)

Function: Xレジスタの内容を、アキュムレータに転送します。

A←X

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	␣ TXA	8A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

TXS (Transfer X to S)

Function: Xレジスタの内容を、スタックポインタに転送します。

S ← X

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	⌴ TXS	9A	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
–	–	0	–	–	–	–	–

TYA (Transfer Y to A)

Function: Yレジスタの内容を、アキュムレータに転送します。

A ← Y

Instruction:

アドレッシングモード	ニーモニック	マシンコード	バイト数	サイクル数
IMPLID	⌴ TYA	98	1	2

Flags:

Status Register							
N	V	T	B	D	I	Z	C
N	–	0	–	–	–	Z	–

第3部
アセンブラプログラミング解説

C O N T E N T S

プログラミング一般

注意
でべろのツールコマンドによって、システムが用意した環境を変更した場合はこの限りではありません。

CG
VRAMのキャラクタジェネレータ領域のこと。パターンの形状や色を定義する領域です。

BG
バックグラウンド。つねにVRAMの先頭からの領域にある。どの場所に、どんなパレットセットの色で、どのパターンを表示するかを管理する領域です。

ユーザープログラムの実行される環境

でべろシステムでは、システムの起動時にいくつかの初期化作業をおこなっています。ユーザープログラムは、でべろシステムによる共通の環境で実行されるので、システムによっておこなわれる初期化作業を省くことが可能です。

でべろシステムからユーザープログラムに処理が渡されるときの環境は以下のとおりです。

・CGへの8ドットフォントパターンの定義
・画面は32×32キャラクタサイズで表示は32文字×24行
・BGのみ表示可能。スプライトは非表示状態で初期化は一切おこなわれない
・パレットセット0とボーダーカラーの初期化
・割込みの初期化
・スタックポインタを$FFに初期化
・バンクRAMの設定
・PSGのメイン音量を初期化
・ユーザープログラムは先頭アドレスから実行開始

実行時に、でべろのシステムに依存しないプログラムを作成する場合でも、これらの初期環境は有効になります。
またユーザープログラムにおいて、システムが与える環境に依存することなく、独自に初期化作業をおこなってもまったく問題はありません。

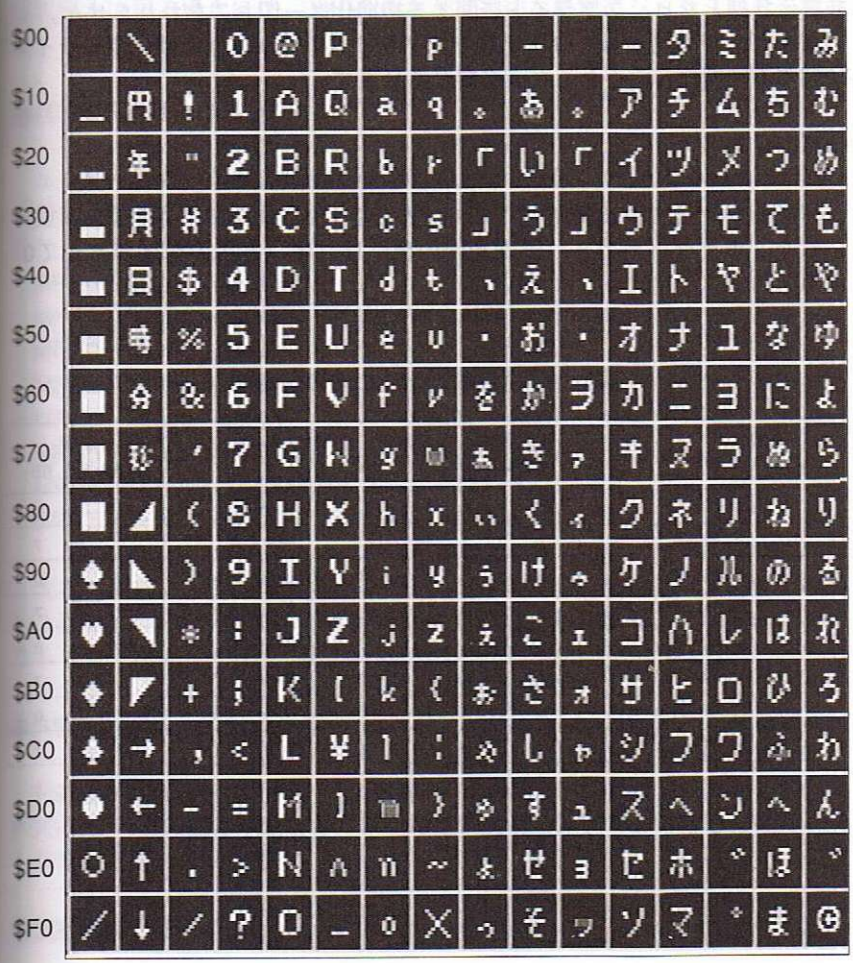
次からにシステムによっておこなわれる設定について、順に解説します。

●8ドットフォントパターン

でべろシステムによってCGに定義される8ドットサイズのフォントパターンは、表3-1-1の256種の文字となっています。

これらのパターンが定義される領域は、VRAMの$2000～$2FFFまでの4096ワードの領域で、どれも色はモノクロで定義されます。

表3-1-1　でべろシステムで定義される8ドットフォント



モノクロ表示

用意されたフォントの色を変更する場合には、パターンを再定義するか、パレットを工夫してください。

表3-1-1の見方

横の項目はパターンコードの上位4bitを、縦の項目は下位4bitを表します。例えば"A"というパターンのコードは$41になります。このコードはBGに設定する際の下位データになります。

仮想画面のサイズ
仮想画面のサイズを変更する場合には、CD-ROM² BIOSのEX_SCRSIZを呼び出しておこないます。仮想画面のサイズとは、VRAMのBG領域の大きさのことです。

表示画面のサイズ
表示画面のサイズを変更する場合には、CD-ROM² BIOSのEX_SCRMODを呼び出しておこないます。仮想画面のうち、どの範囲までを表示するかを意味します。

●画面サイズ

仮想画面のサイズは横32文字×32行の大きさで、VRAMの$0000～$03FFがBG領域として扱われ、BGデータは32ワードごとに行区切りされます。

表示画面のサイズは横32文字×24行の大きさで、モニターの違いによる画面の端の表示切れを考慮して、上下左右端に十分な余白が取られています。

●BGのみ表示可能

システムの初期状態ではBGのみ表示可能になっています。スプライトは非表示状態となり、スプライトに関する初期化は一切おこなわれません。

ユーザーが作成するプログラムでスプライトを扱う場合には、スプライトに関する初期化作業をおこなう必要があります。

●パレット

パレットは$0000～$000Fまでの16色が表3-1-2のように初期化されます。

また、ボーダーカラー（画面周囲の色）は、黒（各色の輝度がすべて0）に初期化されます。

表3-1-2　パレットセット0の設定値

パレット色番号	色	輝度 赤R 緑G 青B	パレット色番号	色	輝度 赤R 緑G 青B
0	黒	0 0 0	8	暗い灰色	3 3 3
1	暗い青	0 0 4	9	明るい青	0 0 7
2	暗い赤	4 0 0	10	明るい赤	7 0 0
3	暗いマゼンダ	4 0 4	11	明るいマゼンダ	7 0 7
4	暗い緑	0 4 0	12	明るい緑	0 7 0
5	暗いシアン	0 4 4	13	明るいシアン	0 7 7
6	暗い黄色	4 4 0	14	明るい黄色	7 7 0
7	明るい灰色	6 6 6	15	白	7 7 7

●割込み

割込みはすべて解除状態になります。ただし、でべろシステムでIRQ1の割込みを使用していますので注意してください。

ユーザーが割込みを使用する場合には、CD-ROM² BIOSのEX_SETVECルーチンを呼び出して、ユーザーベクトルを設定して使用してください。

●スタックポインタ

スタックポインタは$FFに初期化され、スタック領域の256バイトがすべて解放されます。したがって、ユーザープログラムの終了時などに、でべろシステムに戻る場合には、JMP命令などで適切な処理にジャンプしてください。

●バンクRAMとメモリ

ユーザープログラムに処理が渡されるとき、MPR（マッピングレジスタ）の値は以下のように設定されています。

MPR0（$0000～$1FFF）：I/O（$FF）
MPR1（$2000～$3FFF）：WORK RAM（$F8）
MPR2（$4000～$5FFF）：でべろ（$80）
MPR3（$6000～$7FFF）：ユーザ解放（$84）
MPR4（$8000～$9FFF）：ユーザ解放（$85）
MPR5（$A000～$BFFF）：ユーザ解放（$86）
MPR6（$C000～$DFFF）：ユーザ解放（$87）
MPR7（$E000～$FFFF）：CD-ROM² BIOS（$00）

また、現在システムで使用しているバンクRAMは、$80と$81だけですが、$82と$83は将来システムで使用される可能性があります。ユーザープログラムでは、なるべく$84～$87のバンクRAMを使用してください。

●PSG

PSGレジスタのR1（メイン音量調整）のみ$00に初期化します。その他のPSGレジスタの値は不定となりますので、適切な設定をおこなった後にメイン音量を設定してください。

●実行開始アドレス

ユーザーが作成したプログラムは、つねにその先頭から実行が開始されます。作成するプログラムでは、不都合がないように注意してください。

メモリマップについて

詳しくは、第4部「メモリマップ」を参照してください。

足し算と引き算

算術演算のうち、足し算と引き算はHuC6280の命令を使用します。掛け算および割り算はCD-ROM² BIOSの演算パッケージを利用すると便利です。

演算パッケージには三角関数の計算なども用意されているので、命令を使った演算は、足し算と引き算だけで済むことになります。

●足し算

足し算はADC命令を使用します。

ADC命令は、Aレジスタにメモリまたは定数とキャリーフラグの値を加える命令で、とくに必要のない場合にはキャリーフラグをクリアするCLC命令と組み合わせて使用します。

List3-1-1　8ビットの足し算

```
    CLC             ;キャリーフラグをクリア
    ADC    #39      ;Aレジスタに39を加える
      :
      :
```

演算の結果、Aレジスタが$FFを超える値になった場合はキャリーフラグがセットされ、Aレジスタには演算結果の下位8ビットが入ります。

したがって16ビット以上の足し算の場合はリスト3-1-2のようになります。

List3-1-2　16ビットの足し算

```
    LDA    _ax      ;Aレジスタに_axの値をロード
    CLC             ;キャリーフラグをクリア
    ADC    $FF      ;Aレジスタに$FFを加える
    STA    _ax      ;_axに演算結果の下位バイトをストア
    LDA    _ax+1    ;Aレジスタに_ax+1の値をロード
    ADC    $20      ;Aレジスタに$20を加える
    STA    _ax+1    ;_ax+1に演算結果の上位バイトをストア
      :
```

List3-1-1の動作

Aレジスタの内容に39を加えます。
このとき、演算結果が$FFを超えた場合、Aレジスタには下位8ビットの値が残り、キャリーフラグが1にセットされます。

List3-1-2の動作

_axのメモリの内容が$1234だった場合、最初のADC命令では$34 + $FF + 0(Carry=0) = $133となって、_axのアドレスに$33がストアされます。このとき、演算結果が$FFを超えるのでキャリーフラグは1にセットされます。
次のADC命令までの間にはキャリーフラグを変化される命令がないため、次の演算の結果は$12 + $20 + 1(Carry=1) = $33のようになり、16ビットの計算が正常におこなわれるわけです。

●引き算

引き算はSBC命令を使用します。

SBC命令は、Aレジスタからメモリまたは定数を引き、さらにキャリーフラグが0のときに1を引く命令で、とくに必要のない場合にはキャリーフラグをセットするSEC命令と組み合わせて使用します。

List3-1-3　8ビットの引き算

```
    SEC             ;キャリーフラグをセット
    SBC    #39      ;Aレジスタから39を引く
      :
      :
```

演算の結果、Aレジスタが0を下回る値になった場合は、Aレジスタは$100を加えた値に修正されてキャリーフラグがクリアされます。

したがって16ビット以上の引き算の場合はリスト4のようになります。

List3-1-4　16ビットの引き算

```
    LDA    _ax      ;Aレジスタに_axの値をロード
    SEC             ;キャリーフラグをセット
    SBC    $FF      ;Aレジスタから$FFを引く
    STA    _ax      ;_axに演算結果の下位バイトをストア
    LDA    _ax+1    ;Aレジスタに_ax+1の値をロード
    SBC    $20      ;Aレジスタから$20を引く
    STA    _ax+1    ;_ax+1に演算結果の上位バイトをストア
      :
      :
```

●ポイント

足し算と引き算とでは、キャリーフラグによる影響のしかたに違いがあるので注意が必要です。

16ビット以上の計算では、計算の始めにキャリーフラグの設定をおこない、下位バイトから順に続けて計算していくのがコツです。

List3-1-3の動作

Aレジスタから39を引きます。このとき、演算結果が0を下回った場合、Aレジスタには演算結果に$100を加えた値が残り、キャリーフラグが0にクリアされます。

List3-1-4の動作

_axのメモリの内容が$F000だった場合、最初のSBC命令では

$00 - $FF - 0(Carry=1) = -$FF

となり、結果が0を下回ったので

-$FF + $100 = $01

が_axのアドレスにストアされます。このとき、演算結果が0を下回ったのでキャリーフラグは0にクリアされます。

次のSBC命令までの間にはキャリーフラグを変化させる命令がないため、次の演算の結果は

$F0 - $20 - 1(Carry=0) = $CF

のようになり、16ビットの計算が正常におこなわれるわけです。

数値の比較と条件分岐

条件分岐のうち、基本となるのは数値の関係による分岐です。条件分岐を実現させるには、CMPなどの比較命令と、BEQなどの条件分岐命令を使用します。

比較命令とは、レジスタからメモリまたは定数を引き算して、フラグにその結果を反映させるものです。演算の結果はどこにも出力されません。

条件分岐命令とは、特定のフラグの状態によって処理を分岐する命令です。個々の命令が持つ条件にあう場合は指定されたアドレスに分岐し、あわない場合は直後の命令に進みます。

以下に、いくつかの例を示します。

●比較した2値が等しいかどうかの分岐

レジスタと値を比較して、等しいかどうかで指定したアドレスへ分岐させたい場合には、BEQ命令やBNE命令を使用します。

List3-1-5　2値が等しいときの分岐

```
    CMP     #0          ;Aレジスタと0を比べる
    BEQ     label       ;Aレジスタが0ならlabelへ分岐
      :                 ;Aレジスタが0でなかった場合の処理
      :
label:                  ;Aレジスタが0のときの処理
```

●比較した2値の大小関係による分岐

レジスタと値を比較して、大きいか小さいかで指定したアドレスへ分岐させたい場合には、BCC命令やBCS命令を使用します。

List3-1-6　値が大きいときの分岐

```
    CMP     #20         ;Aレジスタと20を比べる
    BCS     label       ;Aレジスタが20以上ならlabelへ分岐
      :                 ;Aレジスタが20未満の場合の処理
      :
label:                  ;Aレジスタが20以上のときの処理
```

●ポイント

16ビットの値を比較したい場合には、付属CD-ROMに収録された、DEVELO.INCにあるcmpwiのマクロ命令を利用すると便利です。

比較命令
比較命令には、Aレジスタまたはメモリに対しておこなうCMPと、Xレジスタに対しておこなうCPX、Yレジスタに対しておこなうCPYがあります。おもにレジスタやメモリの値を判定する場合に使用します。
比較命令の動作は引き算と似ていますが、直前にSECでキャリーフラグをセットする必要はありません。

条件分岐命令
Z(ゼロ)フラグの状態によって分岐するBEQとBNE、キャリーフラグの状態によって分岐するBCCとBCSなど、おもなフラグのそれぞれについて2つずつあります。

条件の設定
他にも、BIOSを呼び出した結果がフラグに反映される場合や、割込みの種別の判定などに条件分岐が利用されます。

List3-1-5の動作
Aレジスタが0であった場合、CMPによってZフラグが1にセットされます。0でなかった場合には、Zフラグは0にクリアされます。
BEQはZフラグが1だった場合に指定アドレスへ分岐する命令なので、結果としてAレジスタが0だった場合に分岐します。

List3-1-6の動作
Aレジスタが20以上であった場合、CMPによってキャリーフラグが1にセットされます。20未満のときは、キャリーフラグは0にクリアされます。
BCSはキャリーフラグが1だった場合に指定アドレスへ分岐する命令なので、結果としてAレジスタが20以上だった場合に分岐します。

キャリーフラグとZフラグの変化だけで、数値の大小関係のすべてを判定することが可能です。

表3-1-3 「CMP #100」での判定例

判定したい条件	条件が成立するときのフラグの値
Aレジスタ＝100	Zフラグが1
Aレジスタ≠100	Zフラグが0
Aレジスタ≧100	キャリーフラグが1
Aレジスタ＜100	キャリーフラグが0
Aレジスタ≦100	キャリーフラグが0　または　Zフラグが1
Aレジスタ＞100	キャリーフラグが1　かつ　Zフラグが0

数値の関係は表3-1-3にある6通りがありますが、例えば「等しい」という条件が成立しなかった場合は「等しくない」という条件が成立します。このように、それぞれ対になる条件があるので、Zフラグ単体、キャリーフラグ単体、そしてキャリーフラグとZフラグを組み合わせた場合の3つを覚えておけば、すべての関係が判定できるようになります。

プログラムでの表現

表3-1-3の判定をプログラムで表現すると以下のようになります。それぞれ、条件が成立する場合には、labelで指定される処理にジャンプするものとします。

・Aレジスタ＝100
```
CMP    #100
BEQ    label
```

・Aレジスタ≠100
```
CMP    #100
BNE    label
```

・Aレジスタ≧100
```
CMP    #100
BCS    label
```

・Aレジスタ＜100
```
CMP    #100
BCC    label
```

・Aレジスタ≦100
```
CMP    #100
BEQ    label
BCC    label
```

・Aレジスタ＞100
```
      CMP    #100
      BEQ    next
      BCS    label
next:
```

ループ処理

ループ処理とは、繰り返しておこなう処理のことで、条件分岐の応用にあたる処理です。プログラム作成での基本処理でもあります。

ループ処理には、大きくわけて以下の2つがあります。

●回数の決まっているループ

ある特定の領域を統一のデータで埋める場合など、あらかじめ繰り返す回数が決まっている場合の処理です。

基本的に、繰り返した回数を計るためのカウンタによって管理されます。

List3-1-7　決められた回数のループ処理

```
    LDX     #0          ;Xレジスタに0をセット
loop:
    STZ     table,X     ;table+Xのメモリに0をストア
    INX                 ;Xレジスタの値を1増加
    CPX     #20         ;Xレジスタと20を比べる
    BCC     loop        ;Xレジスタが20未満ならloopへ分岐
      :
      :
```

●回数の決まっていないループ

ループ処理の基本は条件分岐です。分岐の条件を繰り返した回数ではなく特定の条件に置き換えると、繰り返す回数の決まっていないループ処理になります。

List3-1-8　回数が不定のループ処理

```
loop:
    JSR     ex_joysns   ;パッドの状態を調べるBIOS
    TST     #8,joytrg   ;RUNボタンが押されたかの判定
    BEQ     loop        ;押されていなければloopへ分岐
      :
      :
```

List3-1-7の動作

処理の始めにXレジスタは0にセットされます。ループ回数はCPXによって判定され、BCCで繰り返すかどうかの分岐がおこなわれます。

Xレジスタの値が0～19の間の計20回ループが繰り返されます。

List3-1-8の動作

EX_JOYSNSはCD-ROM² BIOSに用意されているパッドの状態を調べるBIOSです。ワークエリアのjoytrgには各ボタンが押されたかどうかの情報が記録されていて、bit3はRUNボタンに対応します。

TST命令は、メモリと定数とのAND演算をおこない、結果をフラグのみに反映させる命令で、これによってRUNボタンが押されたかを判定します。押されていた場合はZフラグがクリアされ、押されていなければZフラグは1になります。

●256回を超えるループ

回数の決まっているループ処理で、256回以上繰り返したい場合には、ループを2重にすると実現できます。

例えば1000回ループさせるのであれば、5回のループの中に200回のループを置けば5×200で合計1000回のループになります。ただしこの場合、カウンタもそれぞれに用意する必要があります。

List3-1-9　2重のループ処理:5×200回のループ

```
    LDY     #5      ;Yレジスタに5をセット
loop1:
    LDX     #200    ;Xレジスタに200をセット
loop2:
      :             ;繰り返して実行する処理
      :
    DEX             ;Xレジスタの値を1減らす
    BNE     loop2   ;Xレジスタが0でなければloop2へ分岐
    DEY             ;Yレジスタの値を1減らす
    BNE     loop1   ;Yレジスタが0でなければloop1へ分岐
      :
      :
```

List3-1-9の動作

Yレジスタをカウンタにしたループが1回実行されると、Xレジスタをカウンタにしたループが200繰り返されます。Yレジスタのループは5回、Xレジスタのループは200回なので、合計1000回のループになります。

また、ループ回数を数えるカウンタは、回数が計算できればよいので、繰り返す回数をセットしてから1ずつ減らしても、0に初期化してから1ずつ増加させて目的の回数になるまで繰り返しても、どちらもループ処理として成立します。繰り返される処理の内容によって使い分けるとよいでしょう。

パッドの入力判定

ゲームなどのプログラムではパッドの入力判定は不可欠です。ここではパッドを扱う場合の例を示します。

List3-1-10　パッドの入力判定

```
PADsub:
        JSR     ex_vsync        ;vsync割込みを待つ
        TST     #1,joytrg       ;Ⅰボタンの入力判定
        BEQ     skip1           ;入力がなければskip1へ分岐
        JSR     btn1sub         ;Ⅰボタン入力時の処理を呼び出す
skip1:
        TST     #2,joytrg       ;Ⅱボタンの入力判定
        BEQ     skip2           ;入力がなければskip2へ分岐
        JSR     btn2sub         ;Ⅱボタン入力時の処理を呼び出す
skip2:
                :
                :
```

パッドはvsyncの割込みが発生するたびに、BIOS内部にあるIRQルーチンによって状態が調べられています。それを利用して、vsync割込みが発生するのを待つのがEX_VSYNCというBIOSです。

パッドの各ボタンの状態によってワークエリアの値が書き換わります。TST命令を使ってこの値を調べることで、各ボタンの入力判定がおこなえます。

joyは、押されているボタンに対応するbitが1になります。

joytrgは、たった今押されたボタンに対応するbitだけが1になります。

TST命令に指定する数値で調べるボタンが決まります。以下は例です。

```
TST   #1,joytrg       ;Ⅰボタンの入力判定
TST   #2,joytrg       ;Ⅱボタンの入力判定
TST   #4,joytrg       ;SELECTボタンの入力判定
TST   #8,joytrg       ;RUNボタンの入力判定
TST   #16,joy         ;方向キーの上が押されているかどうかの判定
TST   #32,joy         ;方向キーの右が押されているかどうかの判定
TST   #64,joy         ;方向キーの下が押されているかどうかの判定
TST   #128,joy        ;方向キーの左が押されているかどうかの判定
TST   #1+2,joytrg     ;ⅠボタンとⅡボタンの同時入力判定
```

なお、指定する数値を足し合わせれば、ボタンの組み合わせを判定することも可能です。ただし、判定する順序に注意しないと思ったとおりの結果が得られないことがあります。例えばⅠボタンの入力判定後にⅠ＋Ⅱボタンの入力判定をおこなうようなプログラムだと、前者の判定が優先されてしまいます。

List3-1-10の動作

パッドのⅠボタンが押されたらbtn1subというサブルーチンを呼び出します。同様にⅡボタンが押されたらbtn2subのサブルーチンを呼び出します。

BIOS内IRQルーチン

BIOSの内部に用意されているIRQ1の割込み処理です。ユーザーがIRQ1の割込みを書き換えた場合には、このBIOS内IRQルーチンが呼び出されなくなることもありますので注意が必要です。またこの場合、パッドの状態をBIOSで調べることができなくなります。

vsync割込み

垂直帰線割込みのこと。60分の1秒ごとに発生する割込みで、タイマ割込みと並んで使用頻度の高い割込みです。

ワークエリア

joytrgおよびjoyは、それぞれ5バイトずつの領域があり、マルチパッドの接続に対応できるようになっています。1番パッドから順に1バイトずつが割り当てられます。

ボタンの組み合わせ

通常の操作ではあり得ない組み合わせ(例えば上キーと下キーの同時入力など)を対象とすると、不具合のもとになるので注意してください。
また、例えば「下キーを押しながらⅠボタン」のような手順を含む入力を判定する場合には、判定を複数にわける必要があります。

割込み処理

割込みとは、本来のプログラムの流れとは別な要因から、プログラムを寄り道させることを意味します。例えばタイマ割込みであれば「時間」によって割込みが発生し、現在の処理を中断して指定された割込み処理を実行します。

List3-1-11　タイマ割込みの設定

```
    LDA     #02                 ;セットするベクターの番号
    LDX     #LOW( TimerSub )    ;割込みで呼び出されるアドレス
    LDY     #HIGH( TimerSub )   ;
    JSR     ex_setvec           ;ベクターアドレスをセット
    SMB2    irq_m               ;割込みを有効にする
        :
        :

TimerSub:
        :                       ;割込みでおこなう処理が入る
        :                       ;
    RTI                         ;割込み処理を終了する命令
```

割込みを使用する場合には、List3-1-11のようにCD-ROM² BIOSのEX_SETVECを呼び出してベクターアドレスを設定します。つぎにirq_mの適切なビットを1に設定すれば、EX_SETVECで設定したアドレスが割込みによって呼び出されるようになります。

●ポイント

タイマ割込みは、この他にも内蔵タイマの設定が必要になります。また割込みが発生するたびにインタラプト・リクエスト・レジスタに書き込みをおこなってTIQビットをリセットしなければなりません。

処理を中断

割込みが発生すると現在のプログラムが一時的に中断され、指定したアドレスへジャンプします。このとき、フラグを保持するステータスレジスタの値はスタックにセーブされますが、Aレジスタなどの値はセーブされないので注意が必要です。
割込み処理の始めでPHAなどをおこなってレジスタの値を保存し、割込み処理を終える前にPLAなどでレジスタの値を元にもどすようにしましょう。

List3-1-11の動作

EX_SETVECを呼び出すために必要なデータをレジスタに設定します。
ベクターアドレスを設定した後で、irq_mの適切なビットをセットすれば、以後は割込み処理が有効になります。
なお、ソフトウェアリセットでは、irq_mのビット操作は不要です。

内蔵タイマ

内蔵タイマについては、第1部「CPU（HuC6280）」を参照してください。

乱数について

乱数は、さまざまなプログラムで、予測不可能な要素として用いられますが、乱数をユーザープログラムで使いたい場合、ユーザープログラムに専用の乱数発生処理（または補助処理）を作成する必要があります。

CD-ROM² BIOSにはEX_RNDというBIOSファンクションがありますが、このBIOSによって乱数を得るためには、特定のワークエリアを自らのプログラムで変化させ続ける必要があるため、オリジナルの乱数発生処理を作成するほうが無難かもしれません。

●乱数作成の原理

乱数を発生させる処理の多くは「元の値をα倍してβを加える」という仕組みになっています。計算して出た結果を保存しておき、次回の計算で元の値として参照することで、乱数の計算を繰り返していきます。

乱数らしい値を得るための「α倍」には、5倍、9倍、17倍が適当です。これに加える「βの値」は、奇数値であればうまくいくでしょう。

List3-1-12にその例を示します。このサブルーチンを呼び出すたびに、Aレジスタに8ビットの乱数値が入ります。

List3-1-12　乱数発生サブルーチン（5倍して13を加える）

```
Random:
        LDA     Rnd         ;前回の乱数値をロード
        ASL     A           ;Aレジスタの値を4倍する
        ASL     A           ;
        CLC                 ;さらに前回の乱数値を加えて計5倍する
        ADC     Rnd         ;
        CLC                 ;5倍した値に13を加える
        ADC     #13         ;
        STA     Rnd         ;計算した値をメモリに保存
        RTS                 ;サブルーチンの終了
              :
              :
Rnd:
        DS      1           ;乱数用ワークエリア
```

乱数

乱数が変わればゲームのおもしろさまで変わってしまうことがあるほど乱数の役割は重要です。

一般にプログラムで用いられる乱数は擬似乱数と呼ばれるもので、単純な計算の繰り返しで生み出されているものがほとんどです。

List3-1-12の動作

Aレジスタに前回の乱数値を読み込んで、左シフト命令を2回おこなって4倍します。さらにAレジスタに前回の乱数値を加えることで、結果としてAレジスタには前回の乱数値の5倍の値が入ります。

Aレジスタ＝Rnd×2×2＋Rnd
　　＝Rnd×5

この5倍した値に13を加えて、Rndのワークエリアに保存したのち、乱数サブを終了します。

●乱数のくふう

乱数の計算では、プログラムで計算して作り出すため、どうしてもパターン化してしまうという欠点があります。例えばList3-1-12で作られる乱数を調べてみると、下位のビットがパターン化しているのがわかるでしょう。

そこで、乱数を16ビットで作成し、使用するときには上位のデータを使うなどのくふうをおこなうと、より乱数らしい値が得られます。

また、乱数の計算式（αとβの値）が常におなじでは、プログラムを実行するたびに、おなじ乱数が発生してしまうという問題も生じます。このような問題を回避するには、βの値を任意の数値にするだけでも効果があります。具体的には、プレイヤーの名前からβの値を計算して用いたり、内蔵タイマから得られる値をβとして使うなどの方法があります。

乱数発生サブを、タイマ割込みなどの割込み処理にするのも1つの方法です。この場合、プログラムの流れとは関係なく乱数の計算が繰り返されるので、乱数のパターン化を防ぐ方法として有効になる場合もあります。

●必要な乱数を得る方法

6面体ダイスを実現したい場合、1～6の範囲の乱数が必要になります。このような場合にはList3-1-13のように、乱数を0～7の値に調整した後で6と7の場合はふたたび乱数を引き直すことで、目的の乱数を得るようにします。

List3-1-13　1～6の乱数を得る

```
DiceSub:
    JSR     Random      ;乱数発生サブを呼び出す
    LSR     A           ;Aレジスタの下位5ビットを捨てる
    LSR     A           ;（下位ビットはパターン化が顕著なため）
    LSR     A           ;
    LSR     A           ;
    LSR     A           ;
    CMP     #6          ;6以上なら乱数を引き直す
    BCC     DiceSub     ;
    INC     A           ;1を加えて1～6の値に調整
    RTS                 ;元の処理へもどる
```

●ポイント

BIOS内のIRQルーチンが有効なときは、rndseedというワークエリアがvsyncの割込みごとに+1されています。乱数の計算にこのワークエリアの値を利用するのも有効な方法です。

下位のビットがパターン化

List3-1-13で発生する乱数の下位3ビットを調べると、５６３４１２７０という並びを繰り返します。8回でループしてしまうわけです。

bit3～5の部分の3ビットを調べると64回でループしています。このように、データの下位のビットほど繰り返されるパターンのサイクルが短くなり、乱数には相応しくないことがわかります。

乱数を0～7の値に調整

乱数を加工する場合は、2の乗数にするのが簡単です。

乱数を引き直す場合、あまり引き直しが頻発すると、個々の乱数が得られるまでの処理時間に違いが生じてくるので注意が必要です。

List3-1-13の動作

乱数発生サブからAレジスタに乱数値を設定します。

LSR命令を5回実行して、bit4～0を捨てます。結果としてAレジスタには0～7の値（元のbit7～5）が残ります。

加工した乱数値が6以上であった場合には、また処理の始めに戻って乱数を引き直します。

Aレジスタに残った値に1を加えて、目的の1～6の値に調整します。

他にも、テーブルを使った方法などもあります。いろいろとくふうしてみてください。

バックグラウンドへの文字の表示

バックグラウンドに表示されるのは、すべて8×8ドットのパターンです。絵も文字もこのパターンから成り立ちます。

でべろシステムでは、PCエンジンに内蔵されている8×8ドットのフォントパターンをCG（キャラクタジェネレータ）に定義しています。これらの文字を表示するには、List3-1-14のようにでべろBIOSのdv_Screen1を利用すると便利です。

List3-1-14　文字の表示

```
    LDA     #02             ;コマンド番号の設定
    STA     _dh             ;
    LDA     #10             ;X座標の設定
    STA     _al             ;
    LDA     #12             ;Y座標の設定
    STA     _ah             ;
    JSR     dv_Screen1      ;BIOSを呼び出して座標のセット
    LDA     #04             ;コマンド番号の設定
    STA     _dh             ;
    stwi    _bx,TextData    ;文字列データのアドレスを設定
    JSR     dv_Screen1      ;BIOSを呼び出して文字列の表示
          :
          :
TextData:
    DB      "Hello!",0      ;表示する文字列データ
```

●パターンの変更

表示される文字に色を付けたい場合には、以下の3つの方法があります。

1. CGに定義されているデータを変更する
2. あらたにパターンをCGに登録する
3. 表示されるパレットの色を変更する

CGに登録されたパターンデータを変更することなく実現できるのは、パレットを変更する方法ですが、この場合dv_Screen1によって色付きの文字を表示することはできません。

表示時の手間を考慮した場合、すでに定義されているパターンを変更するのが実用的でしょう。

List3-1-14の動作

Aレジスタを使って、BIOSを呼び出すのに必要なデータを設定します。設定するデータについては、第4部「BIOS一覧」を参照してください。なお、dv_Screen1を使って文字列を表示する場合、文字列データの末尾にはかならず0のデータを置く必要があります。

このプログラムが正常に実行された場合、画面の中央付近に「Hello!」の文字が表示されます。

・stwiについて

stwiは、DEVELO.INCに定義されたマクロ命令です。16ビットのデータを2バイトのメモリに設定します。

●代用文字

文字は、その文字に形が固定されているわけではありません。例えばAという文字に「A」のような形が与えられているために、そう表示されるだけです。

でべろシステムで用意されるフォントのうち、プログラムで必要としない文字に関しては、都合がいいように形を変えてしまうのもよいでしょう。例えばa～zの文字をプログラムで使用しないときに、色の違うA～Zの文字を必要とするなら、a～zの文字の形を色付きの「A」～「Z」に変えてしまえばいいわけです。

List3-1-15を実行すると、aの文字が緑色の「A」に変わります。

```
List3-1-15  aの文字を緑色の「A」に変える
    stwiv   _ax,Temp        ;作業用のメモリのアドレスを設定
    stwi    _bx,$2410       ;Aの文字のCGアドレスを設定
    stwi    _cx,$0010       ;読み込むデータ数を設定（ワード単位）
    JSR     dv_VRam2Ram     ;CGからAの文字のパターンを読み出す
    CLX                     ;ループカウンタの初期化
Loop:
    STZ     Temp,X          ;データの先頭から16バイトをクリア
    INX                     ;（これにより色が緑になる）
    CMX     #16             ;
    BCC     Loop            ;
    stwi    _ax,Temp        ;加工したデータのあるアドレスを設定
    stwi    _bx,$2610       ;aの文字のCGアドレスを設定
    stwi    _cx,$0010       ;書き込むデータ数を設定（ワード単位）
    JSR     dv_Ram2VRam     ;aの文字のパターンを転送する
        :
        :
Temp:
    DS      32              ;作業用の領域を確保
```

List3-1-15の動作

でべろBIOSのdv_VRam2Ramを使ってAのパターンデータをTempからの領域に読み出します。
次に読み出したデータを加工して色を変えます。
加工したデータをdv_Ram2VRamによってaのパターンデータの部分に転送します。

スプライトの表示

スプライトは文字などのバックグラウンドとは違い、1ドット単位で表示位置を指定できるので、キャラクタの動きをスムーズに表現できる便利なものです。ここではスプライトを扱うための手順をかんたんに解説します。

●スプライトのための初期化

スプライトを扱うには、VRAMにSG（スプライトジェネレータ）を設置して、スプライトのパターンを定義しておく必要があります。さらにスプライトの座標などの情報を管理するSAT（スプライトアトリビュートテーブル）もVRAMに設置しなければなりません。

プログラムの始めに、まずSGを作成します。あらかじめ、でべろのツールコマンドによってVRAMに読み込んでおいても構いません。

次に、SATを作成します。まずCD-ROM² BIOSのEX_SATCLRを呼び出してVRAMのSATを初期化した後、EX_SPRPUTで必要な情報を設定するとよいでしょう。VRAMのSATに直接データを設定しても構いません。

最後に、EX_SPRDMAとEX_IRQONを呼び出せば、設定したデータにもとづいてスプライトが画面に表示されます。

●スプライトを動かすとき

スプライトの座標を変更して動かしたり、色や表示するパターンを変えたりする場合には、EX_SPRPUTを使ってVRAMに配置したSATの情報を書き換えます。

この場合、スプライトを1つ動かすたびにBIOSを呼び出すのは手間になるので、メモリ上に仮想SATを用意してデータを書き換えてからVRAMにまとめて転送するのも有効な方法です。

●スプライトの座標

スプライトの表示座標の場合、バックグラウンドの表示とは違い、パターンの左上が画面の左上隅になる位置を（32,64）として右下にいくほど座標が大きくなっていきます。

これは1枚のスプライトが、最大時で2×4パターン（32×64ドット）の大きさになるためで、画面の外から徐々に現れるときなどに大きな意味を持ちます。

また、スプライトを一時的に消す場合には、座標を（0,0）にすることで実現できます。

SG
スプライトのパターンは16×16ドットで構成され、VRAMのSGテーブルの64ワードで1つのスプライトパターンを定義します。

SAT
VRAMに配置されるSATは、4ワードで1枚のスプライトの情報を管理している領域で、64枚ぶん計256ワードの大きさがあります。先頭から順に、Y座標、X座標、パターン番号、色コード等の情報が並びます。

SATの初期化
VRAMのSATとして用意した領域を初期化します。第4部「BIOS一覧」のEX_SATCLRを参照してください。

必要な情報の設定
詳しくは第4部「BIOS一覧」のEX_SPRPUTを参照してください。

EX_SPRDMAとEX_IRQON
EX_SPRDMAはVRAMに配置したSATの情報を、HuC6270のSATB（スプライトアトリビュートテーブルバッファ）にDMA転送させるためのBIOSです。DMA転送は垂直帰線期間ごとにおこなわれるデータ転送で、BIOS内IRQルーチンを有効にしておく必要があります。
EX_IRQONはBIOS内IRQルーチンを有効にするためのBIOSです。

スプライトの動き
SATの座標を変えれば、スプライトが表示される位置が変わるほかに、向きの違うパターンを切り換えて表示したり、色（パレットセット）を変えたりすることもできます。
またアトリビュートの操作次第で、例えば、ジェット機とノズル噴射のパターンを別々に作成しておいて、"噴射"の入力によって1×2のパターンにして両方を1つのスプライトとして表示するなども動きのテクニックの1つです。

スプライトの画面左上の座標
画面の表示サイズなどに関係なく、スプライトの画面左上の座標は固定されています。

●表示できるスプライトの数

一度に表示できるスプライトの数は64枚です。この場合、1パターンのみのスプライトも、最大時の2×4パターンのスプライトも1枚として数えます。

水平方向には16枚のスプライトまでが表示できます。17枚目以降のスプライトは画面に表示されないので注意が必要です。ただしこれは、水平方向1ドットラインごとに対する制約なので、17枚目とならない部分は表示されます。

●優先順位

スプライトが重なりあったとき、SATでのアドレスが若いスプライトが優先されて表示されます。

また、スプライトとバックグラウンドで、どちらを優先して表示するかを、個々のスプライトに対して設定することができます。

立体感のある画面を演出するには、このスプライトの定義番号や、スプライトとバックグラウンドの関係も考慮しましょう。

●サンプルプログラム

156～157ページに、付属CD-ROMに収録したスプライトを表示するサンプルプログラムのソースリストを掲載しています。サンプルの解説を参考に、スプライトを扱う方法を会得してください。

バックグラウンドとの優先順位

スプライトが優先される場合、スプライトのカラーコード0の部分は透明として扱われ、その部分はバックグラウンドが透けて見えます。

同様にバックグラウンドが優先される場合、バックグラウンドのカラーコード0の部分は透明として扱われ、その部分はスプライトが透けて見えます。

場合によっては見た目の不具合にもなりますので、この性質に注意してください。

PSGドライバーを利用
作成法などについては、第3部「サウンド」や第4部「PSGドライバー」を参照してください。

ADPCM
PCMデータの再生については第4部「BIOS一覧」の"CD_"または"AD_"で始まるBIOSを参照してください。

PSGのレジスタ
第1部「PSG（HuC6280）」にレジスタの一覧があります。

音量の決定
音量は、3つの音量用のレジスタの総合的な値によって決まります。
R4で設定した値の半分にR1とR5の値を加え、その結果が30以下になると音は出力されません。これは左右の音量が個々に計算されます。

音色
音色はR6およびR8とR9によって作成されますが、レジスタの直接操作で音色を作成する方法は、ここでは解説しません。
特定の音色で効果音を鳴らしたい場合には、PSGドライバーを使用すると便利です。

効果音

ミサイルの発射音、エンジン音、ボタンの入力音などの効果音は、PSGのレジスタを直接アクセスして鳴らす方法と、PSGドライバーを利用して鳴らす方法、ADPCMのデータを再生して鳴らす方法などがありますが、ここではPSGを使って音を鳴らす方法を解説します。

●PSGのレジスタ

PSGには全部で6つのチャンネルがあり、それぞれから個別に音を鳴らすことが可能です。

PSGには10種類のレジスタがあり、レジスタR2～レジスタR7は各チャンネルごとに用意されています。

レジスタR7は、チャンネル5とチャンネル6にだけ用意されていて、これによりノイズ音を出力することができます。ノイズを効果音として使用する場合には、チャンネル5かチャンネル6を使います。

●音データの設定

PSGレジスタに設定するデータのうち、メイン音量のR1、各チャンネルの音量のR4とL・R音量のR5の3つで、出力される音量が決定します。

音階は、周波数微調整のR2と周波数粗調整のR3で決定します。ノイズの場合はノイズ周波数のR7です。

音量と音階、またはノイズ周波数の設定で、ある程度の音は出せるようになります。しかしPSGのレジスタを直接操作して音を鳴らすには、タイミングの調整や音色の作成などといった作業もあるため、なるべくPSGドライバーを利用することをおすすめします。

エンジン音など、変化のある音はPSGのレジスタを直接操作して鳴らすようにして、BGMや固定の効果音は、PSGドライバーを使って鳴らすとよいでしょう。

●サンプルプログラム

158～159ページと160～161ページに、付属CD-ROMに収録した効果音のサンプルプログラムのソースリストを掲載しています。どちらのサンプルプログラムもPSGのレジスタを直接操作して効果音を鳴らしています。解説を参考に研究してみてください。

ソースプログラムの記述とアセンブル

ここでは、アセンブラのpeas.exe（MSXではpeas.com）を使用してプログラムを作成する際の注意などを解説します。

●ソースプログラムの記述

peasでは、大文字と小文字が区別されます。そのため、DEVELO.INCファイルをインクルードしていても、BIOSなどの名称が大文字で記されているとエラーになるので注意が必要です。定義済みのラベルを使用する場合には、かならずDEVELO.INCにある表記でおこなってください。

●ソースプログラムの文法

ソースプログラムを記述する際には、以下の文法を守ってください。

- 文字列は"でくくる
- 文字が'でくくられていた場合は、キャラクタコードの数値を表わす
- マクロ中でのラベルはすべてローカルラベル
- マクロ宣言の中ではマクロは使えない
- includeファイルの中でincludeはできない
- $6000～$DFFFのアドレスにしかコードを置けない。ただしdsなどは可能
- equは重複定義できる

●使用できる演算式

ソースプログラムの中で使用できる演算子は、以下のものがあります。

＋	加算　もしくは　単項演算子の＋
－	減算　もしくは　単項演算子の－
*	乗算
／	除算
%	あまり
&	ビットのAND
¦	ビットのOR
^	ビットのXOR
<<	ビット左シフト
>>	ビット右シフト
high	上位バイトを示す単項演算子
low	下位バイトを示す単項演算子
!	ビット反転の単項演算子

※優先順位は、単項演算子＞乗算、除算、あまり＞その他の順

DEVELO.INC内の表記

でべろシステムに関連するBIOSやワークエリアの名称には、大文字と小文字が混在しています。
CD-ROM²のBIOSおよびワークエリアなどは、すべて小文字で記述されています。
PSGドライバーおよびグラフィックドライバーのファンクション番号は、すべて大文字で記述されています。

マクロの宣言

マクロはユーザーが自由に宣言することもできます。マクロの記述については、第3部「サンプルプログラム解説」にあるDEVELO.INC内のマクロ解説を参考にしてください。

イミディエイトデータ

コード部でイミディエイトデータを使用する場合、#または#$を付けて表します。$のみも可能です。
なお、dbなどのデータ部分では、#を付ける必要はありません。

$6000～$DFFFのアドレス

でべろのシステムによってユーザーに解放されているエリアにしかプログラムは作成できないようになっています。

演算式

演算式には（）も使用できます。ただし式の先頭に（）を使用すると、インダイレクトアドレッシングと扱われるので注意が必要です。

org
ソースプログラムには、かならず開始番地を設定してください。

peas.comのスイッチ
第4部「でべろ外部コマンド」を参照してください。

dv_RegReport
動作については第4部「BIOS一覧」を参照してください。

perun
動作については第4部「でべろ外部コマンド」を参照してください。

●使用できる擬似命令

ソースプログラム中に、以下の擬似命令が使用可能です。

include	他のファイルをインクルードする。ネストはできない。
	例： include "develo.inc"
org	アセンブル開始番地の指定
	例： org $8000
db	バイト定義
	例： db "test message",13,10,'$'
dw	バイト定義
	例： dw $8000,100
ds	空間確保
	例： ds 100
macro	マクロ定義
endm	マクロ定義終端
list	リスト出力再開
nolist	リスト出力抑制

●MSXでのアセンブル

MSXでは、peas.comを使用してアセンブルするとき、かなりの実行時間を要します。peas.comではアセンブルした結果を出力するためにオプションスイッチを指定する必要があるので、実行の際は注意してください。

●ソースファイルを置くディレクトリ

アセンブルするときは、関連するすべてのソースファイルが同一のディレクトリにあるように注意してください。includeによって参照されるファイルがなかった場合でも、アセンブル作業は実行されてしまいます。

●プログラムのデバッグ

デバッグには、でべろBIOSのdv_RegReportを利用すると便利です。dv_RegReportでは、その時点でのレジスタとフラグの内容がパソコンの画面に表示されますが、同時にどこまでが実行されているかを知る手段にもなります。

なお、dv_RegReportを利用してデバッグをおこなうときは、perun.exe（MSXではperun.com）によってプログラムを実行してください。

●peasで出力されるエラーメッセージ一覧

Syntax error	書式ミス
Unterminated string	文字列終端の " が無い
Need Colon	ラベル宣言に：が無い
Formula too complex	演算式が複雑過ぎる。式の一部を () でくくると回避されることが多い
Undefined Symbol	未定義シンボル
Overflow	数値が$0000～$FFFFの範囲を超えている
Illegal addressing mode	アドレッシングモードが違う
Too many macro	マクロ定義数が多すぎる
Missing 'endm'	マクロ定義時に endm が無かった
Not allowed in macro	マクロ内で使用できない命令。includeやorg、マクロのネスティングなどがある
Too many local label	マクロ内で使えるラベルは 8 個まで
Parameter error	マクロのパラメータが宣言と一致していない
Duplicated label	ラベル宣言が重複している
Address error	コードが置けるのは、$6000～$DFFF。それ以外のアドレスは、ds命令しか使えない
Include nesting	includeはネスティングできない
file not found	includeするファイルが見つからない
Rerative out of range	リラティブジャンプの飛び先が遠すぎる
Too many symbol or macro	シンボルやマクロの定義量が多すぎる

VCEのコントロールレジスタ
とくに操作する必要はありません。

パレット操作
パレットの設定やパレットからの読み込みには、できるだけdv_Ram2Pltもしくはdv_Plt2Ramを利用するようにしてください。

7MHz
CD-ROM² BIOSのEX_SCRMODで設定することができます。

メモリ幅レジスタの設定
CD-ROM² BIOSのEX_DOTMODで設定することができます。

VDCのコントロールレジスタ
必要があって操作する場合には、目的のビット以外は変更しないようにおこなってください。

大量のVRAM～VRAM間DMA転送
垂直帰線期間内に終了できないような大量の転送をおこなう場合には、一時的に画面を表示禁止にしておこなうとよいでしょう。
垂直帰線期間内に転送できるデータ量は、おおむね8Kワード程度です。ただしパレットの操作などをおこなった場合は、さらに少なくなります。

注意事項

プログラムを作成するにあたって、禁止されている事項や注意が必要なことについて、解説しておきます。

●VCE（HuC6260）関連

コントロールレジスタへの設定値は、$00、$01、$04、$05以外の値を設定することが禁止されています。これ以外の設定では、テレビ・モニタによっては悪影響を与えてしまいます。最悪の場合、故障を引き起こす可能性もあります。

表示期間中にカラーパレットをアクセスすると、画面にノイズが発生します。パレットの操作は、できる限り垂直帰線期間内におこなうようにしてください。また、一度に大量のデータ転送をおこなうのも避けてください。

●VDC（HuC6270）関連

7MHz（320ドットモード）を指定した場合、かならずメモリ幅レジスタのVRAMドット幅とスプライトドット幅の設定を、2進で"10"にしてください。これをおこなわないと、VRAMのアクセススピードのスペックを越えてしまうため、機器によっては画面が正常に表示されない事があります。

コントロールレジスタの外部同期（EX）、DISP出力選択（TE）、ダイナミックRAMリフレッシュ（DR）の設定はすべて"0"にしてください。それ以外の場合の動作は保証されません。

VRAM～VRAM間DMA転送をおこなう場合、帰線期間中に転送が終了しなかったときは、表示期間に入るところで転送は中断されます。また、この時VRAM～VRAM間転送終了割込みは発生しません。

スプライトを表示している場合、垂直帰線期間になった直後から一定時間は、VRAM～SATB間DMA転送がおこなわれます。この間、CPUからのVRAMアクセスは制限され、CPUはBUSYで待たされることになります。

●CPU（HuC6280）関連

TAIなどの5つのブロック転送命令では、その実行中はIRQ割込みが待機状態になります。このため、大量のデータ転送をおこなうと、割込みのタイミングがずれる可能性があることに注意してください。

将来、システムが更新されることを考慮して、バンクRAMの$82と$83は使用しないようにしてください。使用するメモリが不足する場合には、プログラムを複数にわけるなどのくふうをおこなってください。

●PSG（HuC6280）関連

初期設定でLFOコントロールレジスタはかならず設定してください。リセット時は不定ですので、予期しない音が再生される場合があります。他のレジスタも初期設定が必要です。

全チャンネルの出力レベルが最大になると、出力される音が割れたり、ひずんだりしてしまいます。これは、1チャンネルだけの音でもそれなりに聞けるように（最大でも6チャンネルの出力の1／6しか出力されない）出力を設定しているためです。レベルの設定（とくにダイレクトD／Aモード）では注意してください。

周波数設定レジスタは0に設定してはいけません。

●コントロールパッド

イラストブースター、マウス、6ボタンパッドなどが接続されている場合は、方向キーのデータ部分が、通常ありえないデータ（例：上下が同時に押されているなど）が読み込まれることがありますので、そのようなデータを読み込んだ場合でも、プログラムの不具合が起きないように注意してください。

マルチタップの接続確認をワークエリアの内容でチェックする場合、まれに各パッドのセンス中にパッドの状態が変わる可能性がありますので、数回確認を繰り返すような作業が必要です。また、すべてのパッドで同一のボタンを押している状態では判別できません。

●BIOS

BIOSは、かならず指定されたアクセス方法で使用してください。BIOS内部を直接アクセスしたり、BIOSの使用しているワークRAMエリアを直接参照するようなプログラムは、BIOSのバージョンアップがあった場合などに動作が保証されなくなります。

PSGのレジスタ

でべろではメイン音量のみを初期化しています。その他のレジスタに関しては、その値は不定です。PSGを使用する場合には、LFOコントロールレジスタの初期化をおこなうようにしてください。

マウスなどの入力に関して

マウスなどの入力には、直接ポートからデータを読み込む必要があります。これについては、当マニュアルでは記載していません。将来的に、マウスなどの入力に関しては、でべろのBIOSを用意する予定です。

サンプルプログラム解説

サンプルプログラム「SPRITE.ASM」の解説

```
        cla                     ; (1)
        jsr     ex_dotmod       ;
        cla                     ;
        jsr     ex_scrsiz       ;
        cla                     ;
        ldx     #32             ;
        ldy     #30             ;
        jsr     ex_scrmod       ;

        stwi    sat_adr,$2000   ; (2)
        jsr     ex_satclr       ;

        stwi    _ax,plt_dat     ; (3)
        stwi    _bx,$100        ;
        stwi    _cx,16          ;
        jsr     dv_Ram2Plt      ;

        stwi    _ax,sat_dat     ; (4)
        stwi    _bx,$2000       ;
        stwi    _cx,32          ;
        jsr     dv_Ram2VRam     ;

        stwi    _ax,spr_dat     ; (5)
        stwi    _bx,$4000       ;
        stwi    _cx,128         ;
        jsr     dv_Ram2VRam     ;

        stwi    sat_adr,$2000   ; (6)
        jsr     ex_sprdma       ;
                                ;
        lda     #$10            ;
        jsr     ex_dmamod       ;

        jsr     ex_dspon        ; (7)
        jsr     ex_irqon        ;

loop:                           ; (8)
        lda     #$01            ;
        jsr     ex_joysns       ;
                                ;
        ldy     #0              ;
        lda     joytrg          ;
        and     #$02            ;
        beq     loop            ;
```

(1) 画面モード初期化

EX_DOTMOD：VRAMとスプライトのアクセスドット幅を初期化します。
EX_SCRSIZ：仮想画面サイズ（BG領域の大きさ）を32×32に初期化します。
EX_SCRMOD：表示画面サイズを32×30に初期化します。

(2) SAT初期化

SATはVRAMの$2000からに設定します。
EX_SATCLR：SATを初期化します。

(3) パレット初期化

plt_datからの16ワードのパレットデータを、カラーテーブルの$100からのアドレスに転送して設定します。これによりスプライトの色をセットします。
dv_Ram2Plt：メモリからカラーテーブルへのブロック転送。

(4) SAT設定

sat_datからの32ワードのデータをVRAMのSATに転送して設定します。32ワードぶんなので、スプライト8枚のデータになります。
dv_Ram2VRam：メモリからVRAMへのブロック転送。

(5) SG設定

spr_datからの128ワードのデータをVRAMのSGに転送してパターンを定義します。128ワードぶんなので、2つのパターンになります。SGはVRAMの$4000からに設定します。

(6) SATBへのDMA転送設定

VRAMのSATからHuC6270のSATBへのDMA転送モードを設定します。ただしこの時点では、まだBIOS内IRQルーチンが有効ではないので、スプライトは表示されません。
EX_SPRDMA：VRAM～SATB間のDMA転送に設定します。
EX_DMAMOD：DMA転送モードの設定。

```
    ldy     #10                                     ; (9)
    cla                                             ;
    clx                                             ;
loop2:                                              ;
    dec     a                                       ;
    bne     loop2                                   ;
    dex                                             ;
    bne     loop2                                   ;
    dey                                             ;
    bne     loop2                                   ;

    jmp     cd_boot                                 ; (10)

plt_dat:                                            ; (11)
    dw  $0000,$0004,$0020,$0024
    dw  $0100,$0104,$0120,$01b6
    dw  $006d,$0007,$0038,$003f
    dw  $01c0,$01c7,$01f8,$01ff

sat_dat:                                            ; (12)
;                   y,          x,    pat,    attr
    dw      16*3+64,16*3+32, $0200, $0080
    dw      16*4+64,16*4+32, $0200, $0080
    dw      16*5+64,16*5+32, $0200, $0080
    dw      16*6+64,16*6+32, $0200, $0080
    dw      16*3+64,16*6+32, $0202, $0080
    dw      16*4+64,16*5+32, $0202, $0080
    dw      16*5+64,16*4+32, $0202, $0080
    dw      16*6+64,16*3+32, $0202, $0080

spr_dat:                                            ; (13) -a
;No.$2000 用パターン（$4000 に配置）
    dw      $03c0,$0c30,$1008,$2004,$4002,$4002,$8001,$8001
    dw      $8001,$8001,$4002,$4002,$2004,$1008,$0c30,$03c0

    dw      $03c0,$0c30,$1008,$2004,$4002,$4002,$8001,$8001
    dw      $8001,$8001,$4002,$4002,$2004,$1008,$0c30,$03c0

    dw      $03c0,$0ff0,$1ff8,$3ffc,$7ffe,$7ffe,$ffff,$ffff
    dw      $ffff,$ffff,$7ffe,$7ffe,$3ffc,$3ffc,$0ff0,$03c0

    dw      $03c0,$0c30,$1008,$2004,$4002,$4002,$8001,$8001
    dw      $8001,$8001,$4002,$4002,$2004,$1008,$0c30,$03c0

;No.$2002 用パターン（$4080 に配置）                ; (13) -b
    dw      $0180,$0180,$0240,$0240,$0420,$0420,$0810,$0810
    dw      $1008,$1008,$2004,$2004,$4002,$4002,$8001,$ffff

    dw      $0180,$0180,$03c0,$03c0,$07e0,$07e0,$0ff0,$0ff0
    dw      $1ff8,$1ff8,$3ffc,$3ffc,$3ffc,$3ffc,$ffff,$ffff

    dw      $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
    dw      $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

    dw      $0180,$0180,$0240,$0240,$0420,$0420,$0810,$0810
    dw      $1008,$1008,$2004,$2004,$4002,$4002,$8001,$ffff
```

(7) スプライト表示

EX_DSPON：バックグラウンドとスプライトの表示を許可。
EX_IRQON：BIOS内IRQルーチンを有効にします。結果としてSATBへのDMA転送がおこなわれます。

(8) パッドの入力判定

AND命令によってIIボタンが押されたかを判定し、押されていなければloopへ分岐して処理を繰り返します。
EX_JOYSNS：パッドの状態を調べます。

(9) 時間待ちループ

256×256×10回のループをおこないウエイトをかけます。空ループといいます。

(10) プログラム終了

CD_BOOT：PCエンジンをリセットします。

(11) パレットデータ

カラーテーブルに転送するパレットデータ。

(12) SATデータ

VRAMに設置するSAT用のデータ。
データは4ワードで1セットとなり、順にY座標、X座標、パターン番号、色とアトリビュート情報となります。

(13) スプライトパターンデータ

aが○のパターンでbが△のパターンになります。

サンプルプログラム「SOUND1.ASM」の解説

```
Sound_Ch        equ     $0800           ; (1)
Sound_Vol       equ     $0801           ;
Sound_FrqL      equ     $0802           ;
Sound_FrqH      equ     $0803           ;
Sound_ChVol     equ     $0804           ;
Sound_Pan       equ     $0805           ;
Sound_Wave      equ     $0806           ;
Sound_Noise     equ     $0807           ;
Sound_LFO       equ     $0808           ;
Sound_LF_Ctrl   equ     $0809           ;

        cla                             ; (2)
        sta     Sound_Vol               ;
        cla                             ;
        sta     Sound_LF_Ctrl           ;

        cla                             ; (3)
clear:                                  ;
        sta     Sound_Ch                ;
        stz     Sound_ChVol             ;
        inc     a                       ;
        cmp     #6                      ;
        bne     clear                   ;

        cla                             ; (4)
        sta     Sound_Ch                ;
        lda     #$ff                    ;
        sta     Sound_Vol               ;
        sta     Sound_Pan               ;

        jmp     loop4                   ; (5)

loop0:                                  ; (6)
        stwi    _ax,$e000               ;

loop1:                                  ; (7)
        cla                             ;
loop2:                                  ;
        dec     a                       ;
        bne     loop2                   ;

        cla                             ; (8)
        jsr     ex_joysns               ;
        tst     #1,joytrg               ;
        bne     loop0                   ;
        tst     #2,joytrg               ;
        bne     quit                    ;

        lda     (_ax)                   ; (9)
        sta     Sound_Wave              ;

        lda     #$ff                    ; (10)
        sec                             ;
        sbc     _ah                     ;
        ora     #$c0                    ;
        sta     Sound_ChVol             ;
```

(1) ラベル定義
PSGレジスタのアドレスの定義。

(2)メイン音量初期化
Sound_Volはメイン音量で0に初期化して音を消します。
Sound_LF_CtrlはLFOコントロールレジスタです。

(3)各チャンネルの音量初期化
ループカウンタのAレジスタでチャンネルを指定して、各チャンネルの音量を初期化します。なお、Sound_Chに指定するチャンネルの番号は、実際のチャンネル番号-1の値です。

(4) チャンネル1初期化
チャンネルを1に変更して、メイン音量とチャンネル1の左右音量を最大に設定します。

(5) メイン処理へ
loop4のジョイパッドの入力判定へジャンプします。

(6) データアドレス設定
効果音に使用するデータのアドレスを_axに設定します。CD-ROM² BIOSを効果音のデータに使っています。

(7) 時間待ちループ
256回の空ループをおこないます。

(8) ジョイパッドの入力判定
Iボタンの入力があればloop0へ分岐して効果音の準備を、IIボタンの入力があれば中断してquitへジャンプ。どちらも入力がなければ次の処理へ進みます。

(9) 効果音データの書き込み
R6の波形レジスタにデータを書き込みます。これにより効果音が鳴ります。

(10) チャンネル音量の設定
_ahの値は$E0から$FFまで変化するのを利用して、徐々に音量が少なくなるように設定しています。
また、ORA命令によって、同時にダイレクトD/Aへの出力モードに設定しています。

```
    incw    _ax             ; (11)
    cmpwi   _ax,0           ;
    bne     loop1           ;

loop4:                      ; (12)
    cla                     ;
    jsr     ex_joysns       ;
    tst     #1,joytrg       ;
    bne     loop0           ;
    tst     #2,joytrg       ;
    beq     loop4           ;

quit:                       ; (13)
    cla                     ;
    sta     Sound_Vol       ;

    jmp     cd_boot         ; (14)
```

(11) データアドレス更新

_axの値を1増加し、$FFFFを超えていなければloop1の処理へもどります。これにより、実質的な効果音の演奏部分はloop1からの部分となります。
incw：連続する2バイトのメモリをワードデータとみなしてインクリメントするマクロ命令。
cmpwi：連続する2バイトのメモリとイミディエイトデータを比較するマクロ命令。

(12) メイン処理

パッドの状態を調べて、Ⅰボタンが押されていればloop0へジャンプ。Ⅱボタンが押されていなければloop4からの処理を繰り返します。

(13) 効果音消去

PSGレジスタのメイン音量を初期化して効果音を消します。

(14)プログラム終了

CD_BOOT：PCエンジンをリセットします。

サンプルプログラム「SOUND2.ASM」の解説

```
Sound_Ch        equ     $0800           ; (1)
Sound_Vol       equ     $0801           ;
Sound_FrqL      equ     $0802           ;
Sound_FrqH      equ     $0803           ;
Sound_ChVol     equ     $0804           ;
Sound_Pan       equ     $0805           ;
Sound_Wave      equ     $0806           ;
Sound_Noise     equ     $0807           ;
Sound_LFO       equ     $0808           ;
Sound_LF_Ctrl   equ     $0809           ;

        cla                             ; (2)
        sta     Sound_Vol               ;
        cla                             ;
        sta     Sound_LF_Ctrl           ;

        cla                             ; (3)
loop0:                                  ;
        pha                             ;
        sta     Sound_Ch                ;
        lda     #$9f                    ;
        sta     Sound_ChVol             ;
        lda     #$ff                    ;
        sta     Sound_Pan               ;
        lda     #0                      ;
        sta     Sound_Noise             ;
        pla                             ;
        inc     a                       ;
        cmp     #6                      ;
        bne     loop0                   ;

        lda     #$ff                    ; (4)
        sta     Sound_Vol               ;

loop1:                                  ; (5)
        cly                             ;
loop2:                                  ; (6)
        lda     work0,y                 ;
        clc                             ;
        adc     #1                      ;
        sta     work0,y                 ;
        lda     work0+1,y               ;
        adc     #0                      ;
        sta     work0+1,y               ;

        lda     work0+1,y               ; (7)
        and     #$08                    ;
        bne     down                    ;
```

(1) ラベル定義
PSGレジスタのアドレスの定義。

(2) メイン音量初期化
Sound_Volはメイン音量で0に初期化して音を消します。
Sound_LF_CtrlはLFOコントロールレジスタ。

(3) 各チャンネルの初期化
ループカウンタのAレジスタでチャンネルを指定します。Aレジスタの内容はスタックに退避しておきます。
次に各チャンネルの音量、左右音量、ノイズ周波数を初期化します。Aレジスタの値をスタックから読みもどして、すべてのチャンネルを初期化するまでloop0からを繰り返します。

(4) メイン音量設定
メイン音量を最大にする。この時点で各音量用レジスタが最大に設定されるので、音が鳴り出します。

(5) Yレジスタ初期化
参照するワークエリアのインデックスに使用するYレジスタを初期化します。

(6) ワークエリアの値を更新
Yレジスタでインデックスされる2バイトのワークエリアの値を1増加します。ワークエリアの値は周波数に使用されます。

(7) 周波数データ判定
ワークエリアの周波数データの最上位ビットによってdownへ分岐します。

```
up:                                     ; (8)
        tya                             ;
        lsr     a                       ;
        sta     Sound_Ch                ;
        lda     work0+1,y               ; (9)
        and     #$0f                    ;
        sta     Sound_FrqH              ;
        lda     work0,y                 ;
        sta     Sound_FrqL              ;
        bra     next                    ; (10)

down:                                   ; (8)
        tya                             ;
        lsr     a                       ;
        sta     Sound_Ch                ;
        lda     work0+1,y               ; (11)
        and     #$0f                    ;
        eor     #$0f                    ;
        sta     Sound_FrqH              ;
        lda     work0,y                 ;
        eor     #$ff                    ;
        sta     Sound_FrqL              ;

next:                                   ; (12)
        iny                             ;
        iny                             ;
        tya                             ; (13)
        cmp     #12                     ;
        bne     loop2                   ;

        ldy     #3                      ; (14)
        cla                             ;
wait:                                   ;
        dec     a                       ;
        bne     wait                    ;
        dey                             ;
        bne     wait                    ;

        cla                             ; (15)
        jsr     ex_joysns               ;
        tst     #2,joytrg               ;
        beq     loop1                   ;

        lda     #$00                    ; (16)
        sta     Sound_Vol               ;

        jmp     cd_boot                 ; (17)

work0:  dw      $0400                   ; (18)
work1:  dw      $0600                   ;
work2:  dw      $1000                   ;
work3:  dw      $1300                   ;
work4:  dw      $1B00                   ;
work5:  dw      $0000                   ;
```

(8) チャンネルの指定
Yレジスタによりチャンネルを指定します。

(9) 周波数設定1
ワークエリアの周波数データをPSGレジスタに書き込みます。

(10) nextへジャンプ
(11) 周波数設定2
ワークエリアの周波数データを反転させてPSGレジスタに書き込みます。反転させることにより、徐々に周波数は低くなります。

(12) Yレジスタ更新
Yレジスタを2増加し、次のチャンネルの処理をおこないます。

(13) 繰り返し判定
すべてのチャンネルを処理し終えたかを判定し、まだならloop2へジャンプします。

(14) 時間待ちループ
256×3回の空ループ。

(15) パッドの入力判定
IIボタンの入力がなければloop1へ分岐して効果音の演奏を続けます。

(16) 効果音消去
PSGレジスタのメイン音量を初期化して効果音を消します。

(17) プログラム終了
CD_BOOT：PCエンジンをリセット。

(18) 周波数データ
work0から順にチャンネル1からの周波数データになります。

サンプルプログラム「FILEREAD.ASM」の解説

```
        stwi    _ax,SampleFileName      ; (1)
        stwi    _bx,FileName            ;
        stwi    _cx,11                  ;
        jsr     memcpy                  ;

        jsr     dv_FileOpen             ; (2)
        bcs     NotFound                ;

        ldy     #$10                    ; (3) a
        lda     (_bx),y                 ;
        sta     FileTag                 ;
        iny                             ;
        lda     (_bx),y                 ;
        sta     FileTag+1               ;
        iny                             ;
        lda     (_bx),y                 ;
        sta     FileTag+2               ;
        stz     FileLoc                 ; (3) b
        stz     FileLoc+1               ;
        stz     FileLoc+2               ;
        stz     FileLoc+3               ;
        iny                             ; (3) c
        lda     (_bx),y                 ;
        sta     FileLen                 ;
        iny                             ;
        lda     (_bx),y                 ;
        sta     FileLen+1               ;
        iny                             ;
        lda     (_bx),y                 ;
        sta     FileLen+2               ;
        iny                             ;
        lda     (_bx),y                 ;
        sta     FileLen+3               ;

        stwi    FileAdd,$A000           ; (4)
        jsr     dv_FileRead             ;
        bcs     ReadErr                 ;

        stwi    _ax,$A000               ; (5)
        stwi    _bx,$0100               ;
        stwi    _cx,128                 ;
        jsr     dv_Ram2VRam             ;

end_loop:                               ; (6)
        jsr     ex_joysns               ;
        tst     #02,joytrg              ;
        beq     end_loop                ;

        jmp     cd_boot                 ; (7)

SampleFileName:                         ; (8)
        db      "TESTTEXTBIN"           ;
```

(1) ファイル名の設定
ファイル名の文字列データを、でべろBIOSのワークエリアに転送します。memcpyは_axのアドレスから_bxのアドレスへ_cxのバイト数を転送するサブルーチン。

(2)ファイルのオープン
ファイルが見つからなかった場合には、NotFoundへ分岐します。
dv_FileOpen：FileNameのワークエリアで指定されたファイルを検索してディレクトリ情報を得ます。

(3)ワークエリアの設定
dv_FileReadのためのワークエリアを設定します。
a:ファイル識別コード
b:ファイルの読み出し開始位置
c:読み出すデータ数

(4) ファイルからデータを読む
$A000のメモリへオープンしたファイルからデータを読み込みます。エラーが発生した場合にはReadErrへ分岐します。
dv_FileRead：オープンしたファイルからデータを読み込みます。

(5) VRAMへ転送して表示
読み込んだデータの先頭から256バイト（128ワード）をVRAMのBGに転送します。
dv_Ram2VRam：メモリからVRAMへのブロック転送。

(6) IIボタン入力待ち
IIボタンが入力されるまでsnd_loopを繰り返します。

(7) プログラム終了
CD_BOOT：PCエンジンをリセット。

(8) ファイル名データ
内容を読み出すファイルのファイル名用文字列データ。

```
NotFound:                                   ; (9)
    stwi    _bx,NotFound_Text               ;
    lda     #SCRN1_PUTS                     ;
    sta     _dh                             ;
    jsr     dv_Screen1                      ;
    jmp     dv_Standby                      ;

NotFound_Text:                              ; (10)
    db      "File Not Found",13,10          ;
    db      "OK",13,10,0                    ;

ReadErr:                                    ; (11)
    stwi    _bx,ReadErr_Text                ;
    lda     #SCRN1_PUTS                     ;
    sta     _dh                             ;
    jsr     dv_Screen1                      ;
    jmp     dv_Standby                      ;

ReadErr_Text:                               ; (12)
    db      "File Read Error",13,10         ;
    db      "OK",13,10,0                    ;

memcpy:                                     ; (13)
    cmpwi   _cx,0                           ;
    beq     memcpy_lb                       ;
memcpy_lp:                                  ;
    lda     (_ax)                           ;
    sta     (_bx)                           ;
    incw    _ax                             ;
    incw    _bx                             ;
    decw    _cx                             ;
    cmpwi   _cx,0                           ;
    bne     memcpy_lp                       ;
memcpy_lb:                                  ;
    rts                                     ;
```

(9) メッセージ表示1

ファイルが見つからなかったときのメッセージを表示してプログラムを終了します。

dv_Screen1：コマンド番号（_dh）が4のときは_bxからの文字列を表示します。

dv_Standby：画面は初期化せずに、でべろのシステムへ。

(10)メッセージデータ1

ファイルが見つからなかったときに表示される文字列データ。

(11) メッセージ表示2

ファイルからの読み出しに失敗したときのメッセージを表示してプログラムを終了します。

(12) メッセージデータ2

ファイルからの読み出しに失敗したときに表示される文字列データ。

(13) メモリ間データ転送サブルーチン

_axのアドレスから_bxのアドレスへ_cxのバイト数を転送するサブルーチン。

3つのソース
実際には、DEVELO.INCファイルを include しているため、プログラムは4本のファイルからなります。

調整が容易に
例えば、SPEEDER.Hの中で定義されている _plt0～_pltF の定義値を変更すると、ゲーム中の色が変更できます。

使用している割込み
ゲームではノイズによる効果音を使用しているため、RUN+SELECT入力によるソフトウェアリセットの割込みで効果音を消去するようにしています。

タイマ処理
この部分では、カウンタによって3回に1度タイマが移動し、2回に1度メイン処理が実行される仕組みになっています。その割合を管理しているのがSPEEDER.DAT内の spd_Label_data のデータです。
なお、内蔵タイマおよびタイマによる割込みとは関係ありません。

『SPEEDER』のプログラム解説

『SPEEDER』のプログラムソースは次の3つに分かれています。

SPEEDER.ASM　プログラム本体
SPEEDER.H　ラベルとマクロ定義
SPEEDER.DAT　データおよびワークエリア

それぞれについて、以下におおまかな解説を記載します。

●SPEEDER.Hの解説

プログラム中で参照するPSGレジスタなどのアクセス窓口や、データ部で使用している定数などのラベルを定義しています。定数の定義されている内容を変更することで、データ部分などの調整が容易になる仕組みです。

●SPEEDER.ASMの解説

・初期設定
割込みの初期化
PSGレジスタおよびPSGドライバーの初期化
画面関係の初期化（画面消去／パレット設定／BATおよびSATの設定／VRAMへのパターン定義など）
・メイン処理
ゲーム初期化（スコアなどの初期化／ゲーム開始入力待ち／効果音演奏）
ゲームメイン（スコア表示／タイマ処理／ゲームオーバー判定／クラッシュ判定／自機移動／効果音／コース更新／ゴール判定）
ゴール処理（ステージ更新／効果音）
ゲームオーバー処理
・サブルーチン処理
spd_reset:　RUN+SELECT入力時の割込み処理（ソフトウェアリセット）
Sound_Run:　走行音サブルーチン
Sound_Crush:　クラッシュ時の衝突音サブルーチン
spd_Timer:　タイマ移動処理サブルーチン
spd_Game_init:　ゲーム開始入力待ちサブルーチン
spd_NextStage:　ステージ更新サブルーチン
spd_Ship_init:　自機の座標、コース状態初期化サブルーチン
spd_Speed:　自機のスピード更新サブルーチン
spd_Ship_Move:　自機の移動計算サブルーチン
spd_ShipPut:　自機および自機の影表示サブルーチン
spd_Crush:／spd_Crush_ed:　クラッシュ処理サブルーチン
PutSprite_Ship:　自機のスプライト表示サブルーチン
PutSprite_Sadow:　自機の影のスプライト表示サブルーチン
PutSprite_Timer:　タイマのスプライト表示サブルーチン
PutSprite_Meter:　メーターのスプライト表示サブルーチン
Sub_setBackColor:　背景の色設定サブルーチン
Sub_ChangeColor:　背景の色切り換えサブルーチン
spd_Course:　コース変更サブルーチン
spd_Course2:　コース表示サブルーチン
Sub_ScoreCalc:　スコア計算サブルーチン

Sub_HiScoreCheck:　ハイスコア更新サブルーチン
Sub_ScorePrint:　スコア表示サブルーチン
Sub_joysns:　ジョイパッド入力サブルーチン
Sub_pause:　ポーズ処理サブルーチン
Sub_RND:　乱数発生サブルーチン

●SPEEDER.DATの解説

前半がデータ部分になっており、パターンデータを除いたほとんどの部分がSPEEDER.Hで定義された定数で書かれています。

メッセージ関係のデータは、すべてVRAMへの転送によって表示する構造になっています。

●使用VRAM

$0000～$02FF　バックグラウンドアトリビュートテーブル
$1000～$100F　スプライトアトリビュートテーブル
$2000～$2FFF　キャラクタジェネレータ
$6000～$64FF　スプライトジェネレータ

●プログラム改造のポイント

コースの表示では、9種類の画面が転送によって切り換わり表示されていますが、この画面の種類を増やしたり、各画面のグラフィックをよりリアルなものに変更するだけでもかなりの効果が望めます。

また、自機のスプライトは原作に基づいて単色3パターンの切り換え表示になっていますが、このパターン数を増やしたり、細かな色使いのパターンに変更してもよいでしょう。

データ部分のspd_Label_dataに設定されている内容を変更すると、自機の処理割合を変えられます。spd_ShipXの増分値を変更すると、コース状態による遠心力の変更ができます。

コース更新

自機が走行中でない場合には、背景が変化しないように分岐します。
コースの更新は、SPEEDER.DATにあるspd_BAT_stk0～spd_BAT_stk8のデータをVRAMのBATに転送することでおこなわれます。
また、背景の色の切り換えは、spd_StageColor_dataのデータによって切り換わっています。

自機の表示切り換え

自機のスプライトは、表示時にパターン番号を変更することで切り換え表示しています。

パッドの入力

このプログラムではEX_JOYSNSのBIOSを使ってパッドの入力をおこなっていますが、vsyncの割込みを待つだけでもおこなえます。

乱数処理

乱数は毎回おなじパターンで発生しています。この乱数の発生パターンを変更すると、コースの形状を変えることができます。

スプライトパターン

パターンはすべて2×2ドット単位で作成されています。

BENDBENTの構成

プログラムは、処理のほとんどがサブルーチン化されており、メイン処理ではそれらのサブルーチンを呼び出すだけでおこなっています。

ボールの出現条件

ボールは、まず BallCntSub がダウンカウントされ、次に Ball Cnt をダウンカウントします。BallCntSub による判定は、値が0になったかどうかによります。BallCnt による判定は、ゲームレベルの2倍との比較によります。

ゲームオーバー処理

ゲームオーバー時に、スコアがハイスコア未満だと無限ループに入ります。

『BENDBENT』のプログラム解説

●BENDBENT.ASMの解説

・初期設定

Start:

初期設定（BIOS内IRQルーチンを有効にする／VRAMに配置したSATの初期化／ハイスコア初期化）

・メイン処置

Replay:

タイトル（画面の表示許可／ボールのワークを初期化／ボール消去／画面消去／パレット設定／CGへのパターン定義／スプライトのパターン定義／スプライト表示禁止／タイトル表示およびスタート入力待ち／スプライト表示許可）

ゲーム開始（VRAM～SATB間DMA転送開始とモード設定／スコアなど初期化／ゲーム画面の作成と表示／ゲーム開始時の効果音／MainLoop2へ）

MainLoop1:

ゲーム画面作成（ボール初期化／ゲーム画面作成）

MainLoop2:

ボール出現（ボール配置／ボール出現カウンタ設定）

MainLoop3:

時間待ち（vsync割込み待ち／ポーズ判定とポーズ処理／乱数更新）

ボール処理（ボールの移動／ボール表示／ゴール判定／ゴール処理／レベルアップならMainLevelUpへジャンプ／ロストボールならMainLostへ）

カーソル処理（カーソル移動／カーソル表示／スクロール処理）

ニューボール判定（まだならMainLoop3へ／出現するならMainLoop2へ）

MainLevelUp:

レベルアップ処理（メッセージ表示／ボーナス表示とスコア加算／ノルマ設定／効果音／ボールの移動速度調整／MainLoop1へジャンプ）

MainLost:

ロストボール処理（効果音／チャンス減少／ゲームオーバー判定→まだならMainLoop1へジャンプ。ゲームオーバーなら効果音）

GameOver:

ゲームオーバー処理（メッセージ表示／ハイスコア更新／リプレイ入力待ち／Replayへジャンプ）

・サブルーチン処理

Pause:　ポーズの判定と処理

ClrBall:　ボールのワークエリアを初期化する

NewBall:　未使用のワークエリアを検索してボールのデータを設定する

MoveBall:　ボールの移動判定と移動計算

HereVram:　ボールの移動先のパターン検出

DispBall:　ボールのワークエリアをもとにボールのスプライトを表示する

MoveCsr:　ジョイパッドの入力によりカーソルを移動する

DispCsr:　カーソルのスプライトを表示する

ScrlUp:　アミダのスクロール処理

GoalCount:　ゴールしたボール数をゴールの種類ごとに計算する

GoalProcess:　GoalManyのデータをもとにスコアの加算などを処理する（戻り値Aレジスタ=0:通常、1:レベルアップ、2:ロストボール）

ScoreUp:　スコアアップ
DecNorm:　ノルマ減少
InitParam:　スコア、ノルマ、チャンス数などのパラメータを初期化する
DispScore:　スコアなどのパラメータを表示する
GameScreen:　スコア欄の表示とアミダの作成表示
MakeGoal:　ゴール配置
DispGoal:　ゴール表示
PlaySound:　効果音を演奏する
Title:　タイトルの表示とゲーム開始の入力待ち
Cls:　バックグラウンドを消去する
Random:　乱数更新
memset:　指定された領域を単一のデータで埋めるサブ
PrintStr:　メッセージ表示
PrintNum:　数値表示
PrintNum_Str:　数値表示用データ
SetBG:　CGにパターンを定義する
SetPalette:　バックグラウンドとスプライトのパレットを設定する
SetSprite:　SGにパターンを定義する
・データ＆ワークエリア
TitleData1～TitleData6:　タイトル画面表示用データ
BonusData1～BonusData5:　レベルアップ時の表示用データ
OverData1～OverData3:　ゲームオーバー時の表示用データ
GoalData:　ゴールの表示用データ
GameData1:　アミダの縦線表示用データ
GameData2～GameData7:　パラメータの項目表示用データ
GameData8:　アミダの横線表示用データ
PauseData1:　ポーズ時の表示用データ
PauseData2:　ポーズ時の消去用データ
BGData:　バックグラウンドのパターンデータ
PaletteData:　ゲーム中のパレットデータ
PausePltData:　ポーズ時のパレットデータ
SpriteData:　スプライトのパターンデータ
MusicData1:　ゲーム開始時の効果音データ
MusicData2:　スコアアップ時の効果音データ
MusicData3:　ノルマ減少時、ポーズ時の効果音データ
MusicData4:　ロストボール時の効果音データ
MusicData5:　ゲームオーバー時の効果音データ
MusicData6:　レベルアップ時の効果音データ
CsrX:　カーソルの横位置
CsrWork:　カーソル表示時の仮想SATデータ
ScrlUpAdd1:　スクロール時の転送先VRAMアドレス
ScrlUpAdd2:　スクロール時の転送元VRAMアドレス
ScrlUpBuf1:　スクロール時の転送データー時保存用
ScrlUpBuf2:　スクロール時の転送データー時保存用（最下段）
Rnd:　乱数用
Speed:　ボールの移動速度
SpeedCnt:　ボールの移動時用ディレイカウンタ
FlashFlag:　ボールの強制移動用フラグ

ボールの移動判定

ボールは、まず移動方向によって処理が分岐し、それぞれの方向によって座標が更新されます。
ボールの座標がキャラクタ単位になった場合には、移動先の状態をVRAMからBGのパターンコードを検出することによって調べます。検出したコードから、次の移動処理のために、移動方向を設定します。

スクロール

スクロールは、下から上へ1キャラクタぶんおこなわれます。まず、いちばん上の部分のBGデータを保存しておき、ループ処理ですぐ下のパターンを現在の位置へ転送することでずらしていきます。最後に保存しておいたBGデータをいちばん下の部分に転送して完了です。

ゴール処理

ボールがゴールに到達した場合には、GoalMany にゴールの場所ごとに到達したボール数を計算しておき、ひととおり移動が終了した後で、GoalMany のデータに基づいて処理されます。

InitParamサブルーチン

この部分を書き換えると、チャンス数などの変更がおこなえます。

ゴール配置

ゴールの配置では、Goal のワークエリアにスコアアップ2つ、レベルアップ、ロストボールのゴール番号を書き込んでおき、その後で20回のループによってシャッフルして配置しています。

乱数

乱数更新部分では、rndseed の値を参照することで、乱数のパターン化を防いでいます。rndseed は、BIOS内IRQルーチンが有効なときは、vsync割込みのたびに+1されています。

表示用データ

表示用データは、dv_Screen1で表示できるデータ構成で、Y座標位置、X座標位置、文字列、データの終了コードの順に並んでいます。

効果音データ

効果音データは、すべてPSGドライバーで演奏されるデータです。

FlashFlag

レベルアップ時やロストボール時に、ボールを強制移動させるためのフラグです。

無限ループの解除

ゲームオーバー処理の以下の部分

```
GameOver:
      :
      :
   cmpw    Score,HiScore
   jcc     GameOver
```

を、次のように変更すれば、無限ループを解除できます。

```
GameOver:
      :
      :
   cmpw    Score,HiScore
   jcc     GameOver_Lp
```

Chance:　チャンス数
Level:　ゲームレベル
Norm:　ノルマ
Goal:　ゴールの種類（ゴール位置別）
GoalMany:　ゴールしたボール数（ゴール種類別）
BallCnt:　ボールの出現判定用ダウンカウンタ
BallCntSub:　ボールの出現判定用サブダウンカウンタ
Score:　スコア
HiScore:　ハイスコア
BallWork:　ボールの状態用ワーク
SpriteWork:　ボール表示時の仮想SATデータ

●使用VRAM

$0000～$02FF　バックグラウンドアトリビュートテーブル
$2000～$2FFF　キャラクタジェネレータ
$3000～$30FF　スプライトアトリビュートテーブル
$4000～$407F　スプライトジェネレータ

●プログラム改造のポイント

InitParam で設定されるパラメータの値を変更すれば、チャンス数やボールの移動速度などが変更できます。

ゲーム中、スコアがハイスコア未満だった場合には、GameOver の処理部分で無限ループに陥ります。このため、RUN+SELECT によるリセット入力以外は受け付けなくなります。どの部分をどのように変更すればよいか、挑戦してみてください。

「DEVELO.INC」のマクロ解説

```
rtseq       macro
            bne         @label
            rts
@label:
            endm

rtsne       macro
            beq         @label
            rts
@label:
            endm

rtscs       macro
            bcc         @label
            rts
@label:
            endm

rtscc       macro
            bcs         @label
            rts
@label:
            endm

jeq         macro       @adres
            bne         @label
            jmp         @adres
@label:
            endm

jne         macro       @adres
            beq         @label
            jmp         @adres
@label:
            endm

jcs         macro       @adres
            bcc         @label
            jmp         @adres
@label:
    endm

jcc         macro       @adres
            bcs         @label
            jmp         @adres
@label:
            endm

jsreq       macro       @adres
            bne         @label
            jsr         @adres
@label:
            endm
```

- **rtseq**
  Zフラグが1のとき、RTS命令を実行します。
- **rtsne**
  Zフラグが0のとき、RTS命令を実行します。
- **rtscs**
  キャリーフラグが1のとき、RTS命令を実行します。
- **rtscc**
  キャリーフラグが0のとき、RTS命令を実行します。
- **jeq**
  Zフラグが1のとき、指定されたアドレスへジャンプします。
- **jne**
  Zフラグが0のとき、指定されたアドレスへジャンプします。
- **jcs**
  キャリーフラグが1のとき、指定されたアドレスへジャンプします。
- **jcc**
  キャリーフラグが0のとき、指定されたアドレスへジャンプします。
- **jsreq**
  Zフラグが1のとき、指定されたアドレスを呼び出します。

• jsrne
Zフラグが0のとき、指定されたアドレスを呼び出します。

• jsrcs
キャリーフラグが1のとき、指定されたアドレスを呼び出します。

• jsrcc
キャリーフラグが0のとき、指定されたアドレスを呼び出します。

• clrw / _clrw
指定アドレスからの2バイトの領域を0にクリアします。

• stwi / _stwi
指定アドレスからの2バイトの領域にワードデータを設定します。
※_stwiはレジスタの内容が保持される。以下"_"付きは同じです。

• incw / _incw
指定アドレスからの2バイトの領域の内容をインクリメントします。

• decw / _decw
指定アドレスからの2バイトの領域の内容をデクリメントします。

• addwi / _addwi
指定アドレスからの2バイトの領域にワードデータを加算します。

```
jsrne   macro   @adres
        beq     @label
        jsr     @adres
@label:
        endm

jsrcs   macro   @adres
        bcc     @label
        jsr     @adres
@label:
        endm

jsrcc   macro   @adres
        bcs     @label
        jsr     @adres
@label:
        endm

clrw    macro   @mem
        stz     @mem
        stz     @mem+1
        endm

stwi    macro   @mem,@imm
        lda     #high (@imm)
        sta     @mem+1
        lda     #low (@imm)
        sta     @mem
        endm

incw    macro   @mem
        inc     @mem
        bne     @label
        inc     @mem+1
@label:
        endm

decw    macro   @mem
        sec
        lda     @mem
        sbc     #1
        sta     @mem
        lda     @mem+1
        sbc     #0
        sta     @mem+1
        endm

addwi   macro   @mem,@imm
        clc
        lda     @mem
        adc     #low (@imm)
        sta     @mem
        lda     @mem+1
        adc     #high (@imm)
        sta     @mem+1
        endm
```

```
addw    macro   @mem,@mm2
        clc
        lda     @mem
        adc     @mm2
        sta     @mem
        lda     @mem+1
        adc     @mm2+1
        sta     @mem+1
        endm

subwi   macro   @mem,@imm
        sec
        lda     @mem
        sbc     #low (@imm)
        sta     @mem
        lda     @mem+1
        sbc     #high (@imm)
        sta     @mem+1
        endm

subw    macro   @mem,@mm2
        sec
        lda     @mem
        sbc     @mm2
        sta     @mem
        lda     @mem+1
        sbc     @mm2+1
        sta     @mem+1
        endm

cmpwi   macro   @mem,@imm
        lda     @mem+1
        cmp     #high (@imm)
        bne     @label
        lda     @mem
        cmp     #low (@imm)
@label:
        endm

movw    macro   @dst,@src
        lda     @src
        sta     @dst
        lda     @src+1
        sta     @dst+1
        endm
```

- **addw / _addw**
  指定アドレスからの2バイトの領域どうしで16ビットの加算をします。
- **subwi / _subwi**
  指定アドレスからの2バイトの領域からワードデータを減算します。
- **subw / _subw**
  指定アドレスからの2バイトの領域どうしで16ビットの減算をします。
- **cmpwi / _cmpwi**
  指定アドレスからの2バイトの領域とワードデータとを比較します。
- **movw / _movw**
  第2オペランドのアドレスから第1オペランドのアドレスへ16ビットのデータを転送します。

サウンド応用

PSGドライバー
PSGドライバーは曲データをミュージックトラックデータとして与えてやれば、自動的にそれを演奏してくれます。あとは停止させたり、曲を変更したりするとき以外は、プログラムで演奏を気にする必要はありません。第4部PSGドライバーの項を参照してください。

ミュージックトラックデータ
トラックデータは、1バイトや2バイトの音程データやコマンドデータが連続したものです。ですから、音を鳴らす順番通りに音程データを並べていけばいいのです。これらのデータは、テキストエディタでは扱えないバイナリのコードですから、アセンブラの擬似命令"DB"や"DW"で順番に書いていき、アセンブルしてトラックデータを作ることになります。第4部PSGドライバーの項に、トラックデータの解説があります。

MML
ミュージック・マクロ・ランゲージの略です。音程データをアルファベットにあてはめて、とてもかんたんに音楽データを表現できるようにした、楽譜の書き方のことです。
楽譜と言っても、5線譜もオタマジャクシの形をした音符も使いません。「ド」の音には"C"、「レ」の音には"D"というように、アルファベットをあてはめて、そのあとに音の長さを表す数字を付けます。"E4"なら「4分音符のミ」という意味です。

テキストエディタ
文章を書いて保存するためのツールです。付録CD-ROMにEditEngine（PC-98シリーズ用）、v&z_small（MSXシリーズ用）を収録してあります。

MMLコンバータ
PC-98用のmml.exe、MSX用のmml.comがあります。

peas
でべろ付属のアセンブラです。

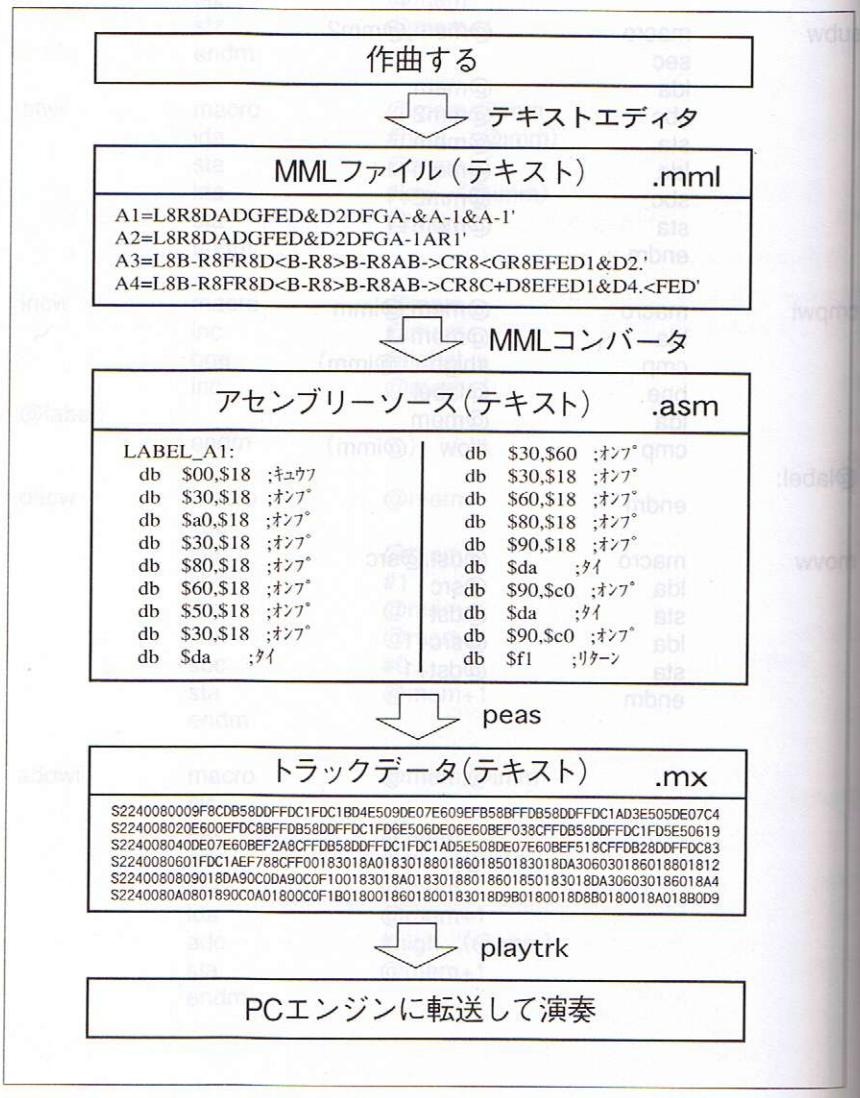
plytrk
MMLコンバータとアセンブラを使ってできた、トラックデータファイルを演奏するためのコマンドです。PCエンジンのメモリの中にトラックデータを読み込んで登録し演奏します。

MMLを使って作曲

PCエンジンで曲を演奏するには、PSGドライバーのミュージックトラックデータ（以下トラックデータ）を使うと便利です。トラックデータを作って演奏する場合は、図3-4-1のような手順でおこないます。

曲をMMLファイルの書式にしたがってテキストエディタで打ち込んでいき、拡張子「.mml」のファイル名で保存しておきます。これをMMLコンバータにかけると拡張子「.asm」のアセンブリソースファイルになります。peas でこれをアセンブルすれば、拡張子「.mx」の実行用トラックデータファイルができあがります。

MMLコンバータがサポートしていないコマンドを使用したいときは、アセンブリソースファイルをテキストエディタで修正してください。



MMLコンバータ

MMLで曲が書かれたMMLファイルから、peasでアセンブルが可能なアセンブリソースファイルを出力します。PC-98用はmml.exe、MSX用はmml.comです。使い方はどちらも同じで、DOSのコマンドラインから、

mml 〈ファイル名〉

で実行します。ファイル名にはMMLファイルを指定します。MMLファイルの拡張子は「.mml」です。実行してエラーがなければ、拡張子が「.asm」のアセンブリソースファイルが作られます。

サポートしているのはダイレクトレングスモードの音符のみです。タイムベースレングスモードはサポートしていません。

●MMLファイルの書式

テキストエディタを使ってMMLを記述していきます。スペースやタブは見やすくするために使ってもいいですが、コンバータで変換する際にすべて除去されます。当然ラベル名には使えません。

MMLファイルの書式は、

ラベル名=MML……

です。これが1つの演奏ブロックになり、いくつかのブロックを組み合わせて曲を作ります。

「=」までの文字列がラベル、つまりブロックの名前です。これがジャンプコマンドやコールコマンドで使用するラベル名になります。小文字は大文字に変換されます。ラベルの先頭に「.」（ピリオド）は使えません。

「=」の直後の部分からにMMLをずらずらと書き並べて、最後に改行を入れてください。これで1つの演奏ブロックになります。

1つの演奏ブロックが1行に入り切らない場合、次の行の先頭をスペースかタブのコードで字下げをしてMMLを書けば、前の行の演奏ブロックの続きと解釈されます。

演奏ブロックがコールで呼ばれるサブルーチンの場合、最後にかならず「'」（シングルクォーテーション）を入れてください。

「.」（ピリオド）で始まる特別な演奏ブロックがあります。これは

.START?=MML……

というブロックです。これが演奏チャンネルの指定も兼ねた、各チャンネルの先頭のブロックになります。「?」の部分には1～6の数字が入り、演奏するチャンネル番号を表します。同じチャンネルを二重に指定するとエラーになります。特別なラベルと考えてください。

「;」（セミコロン）はそれ以降の文字列がコメントだという記号です。コメントはコンバートのときに読み飛ばされます。何も書いてない空行も無視されます。

レングスモード

ダイレクトレングスモードやタイムベースレングスモードについては、第4部PSGドライバーの項を参照してください。

ジャンプコマンドとコールコマンド

演奏時に繰り返して演奏される部分などで、ジャンプコマンドやコールコマンドを使って実現することができます。詳しくは第4部PSGドライバーの項の、トラックデータの解説を参照してください。

サポートMML

MMLコンバータでサポートしているMMLを以下に示します。指定するパラメータはトラックデータのものと同じです。

「L」コマンドによる標準音長の設定と、「@M」コマンドによるパーカッションモードの設定が必要な場合は、音符や休符が出現する前に指定してください。新しい演奏ブロックになると、かならずLは4、@Mは0のノーマルモードに初期化されます。これは、ジャンプやコールコマンドのアドレス計算を、アセンブラに任せているという仕様からきているものです。

●音程： {C、D、E、F、G、A、B}《#、+、−》《音長》《.》

音程データです。

「CDEFGAB」がそれぞれ「ドレミファソラシ」にあたります。

「#+-」は半音処理、シャープ（#と+）とフラット（−）です。必要なら付けます。シャープは「#」と「+」のどちらでも構いません。

「音長」は音符の種類です。4分音符なら「4」、16分音符なら「16」と、そのままの数字を書きます。また、連符のような特別な記号はないので、この音長で対応します。4分音符を3つに分けるような連符の場合は、音長に「12」を指定します。音長は省略することができます。省略するとLコマンド（標準の音長）で設定した音長になります。Lコマンドが無指定の場合は「4」を指定したことになります。

「.」は付点音符を表します。音長の半分が付加されるので、音長は1.5倍になります。"C4."なら「付点4分音符のド」です。"A2.."は、音長の半分とそのまた半分が付加されるので、「ラの2分音符と4分音符と8分音符を合わせた音符」ということになります。

※注意　音長に「1.」は指定できません。音長1つまり全音符が、内部仕様で192に相当していて、「1.」だと288になってしまい、1バイトの範囲（255）をこえてしまうからです。長い音を出したいときは「C1&C2」などとして、タイ（&）を使ってください。

●休符：R《音長》《.》

休符は「R」で表します。音長は、音程の音長とほぼ同じですが、省略時はLコマンドに関らず「4」です。"R."は"R4."とおなじです。

●パーカッション音符： {B、M、S、C、H} 〈音長〉《.》

リズム音符です。音長は省略できません。「BMSCH」は順にバスドラム、ミッドタム、スネアドラム、シンバル、ハイハットです。CD-ROM内のpc.mxに合わせてあります。モードをパーカッションモード（@M1）にしたラベルでのみ有効です。

●オクターブ：O〈音階〉
オクターブを指定します。「音階」の部分には1～7の数値が入ります。1がもっとも低く、7がもっとも高い音階です。

●オクターブアップ：>
オクターブを1つ上げます。オクターブ7のときは使用できません。

●オクターブダウン：<
オクターブを1つ下げます。オクターブ1のときは使用できません。

●タイ：&
前後2つの音をつなげます。

●テンポ：T〈テンポ〉
曲の演奏速度の設定です。「テンポ」は35～255の範囲です。35はゆっくりで、255は速くなります。

●ボリューム：V〈音量〉
全体のボリュームです。「音量」は0～31で、31が最大です。

●パンポット：P〈右音量〉,〈左音量〉
左右の音量設定です。どちらも0～15の範囲で15が最大です。

●音長比：Q〈比率〉
1音中の発音の割合を指定します。8分の1単位で、1～8で指定します。

●リピートビギン：[〈回数〉
繰り返しの開始です。リピート回数を指定して使用します。ネストは、コールを使用しなければ、6段階まで可能です。

●リピートエンド：]
繰り返しの終了です。リピートビギンと組み合わせて使用します。

●音色（ウェーブ）：@〈ウェーブ番号〉
音色（ウェーブ）を設定します。ウェーブ番号には、内蔵が0～44、ユーザー定義が45～127まで指定できます。ウェーブデータを定義していないときは、ユーザー定義の番号を指定してはいけません。

●エンベロープ：@E〈エンベロープ番号〉
エンベロープを設定します。内蔵が0～15、ユーザー定義は16～127の範囲です。ユーザーが登録していない番号は指定してはいけません。

●デチューン：@D〈増分〉
そのチャンネルにデチューンをかけます。増分は－128～127の範囲です。

●ジャンプ：/ラベル名/
指定したラベルのブロックの先頭へジャンプします。

エンベロープ
エンベロープとは1つの音の鳴り方です。鳴りはじめは大きくすぐに消えてしまう音や、じわっと鳴り始めて長く続く音など、いろいろなパターンから1つを選びます。音色の一部と考えても間違いではありません。

ネスト
同じ動作を重ねておこなうこと。ここではリピートビギン・エンドの中にさらにリピートビギン・エンドを使うときなどを意味します。

音色を調べるために
内蔵のウェーブ45種類とエンベロープ15種類の組み合わせは、実際に音を出すまではどのような音色になるのかがよくわかりません。そこでPCエンジン側のプログラムで、ウェーブとエンベロープを自由に組み合わせて音を演奏するサンプルプログラムを用意しました。
でべろCD-ROMのプログラムコーナーから「WAVEとENVELOPEのサンプル」を実行してください。実行されるプログラムはCD-ROMの中のwave.mxです。
実際にスピーカーから流れる音を聞いて、どの音色を使うのかを決めてください。

サポートされていないコマンド

これらのコマンドのコードは、第4部PSGドライバーのトラックデータ（ページ228～264）を参照してください。

●コール：〈ラベル名〉

指定したラベルのブロックをコールします。リターンといっしょに使います。ネストはリピートを使用しなければ4段階まで可能です。

●リターン：'

呼び出し元のコールが置かれた次のデータに戻ります。コールと組み合わせて使用します。

●モード：@M〈モード番号〉

モード番号は0がノーマルモード、1がパーカッションモードです。モードのコマンドが出力されるほか、MMLコンバータ内で音程データの扱いが変わります。チャンネル5か6のみで、各ラベルごとに指定してください。

●データエンド：＊

演奏の終わりを表します。チャンネルごとに設定してください。

●標準の音長：L〈音長〉

音程で音長を省略したときの音長です。デフォルトでは4が設定されています。このMMLコンバータ内だけのコマンドで、アセンブリソースには出力されません。必要ならば各ラベルごとに設定してください。

●サポートされていないコマンド

楽譜のための最低限の機能は盛り込んだつもりですが、まだサポートしていない機能もあります。これらの機能を使用したい場合は、変換後のアセンブリソースから目的の場所を探し出して、コマンドを挿入してください。

・タイムベース（タイムベースレングスモードの音程も含む）
・相対ボリューム
・ダルセーニョ
・セーニョ
・周波数変調
・FMディレイ
・FM補正
・ピッチエンベロープ（PE）
・PEディレイ
・スイープ
・スイープタイム
・移調
・相対移調
・全体移調
・ボリュームチェンジ
・パンライトチェンジ
・パンレフトチェンジ
・フェードアウト

サポートMML一覧

MMLの意味	MML
音程（ノーマルモード）	{CDEFGAB}《{#＋－}》《音長》《. 》
休符	R《音長》《. 》
音符（パーカッションモード）	{BMSCH}〈音長〉《. 》
オクターブ	O〈音階〉
オクターブアップ	>
オクターブダウン	<
タイ	&
テンポ	T〈テンポ〉
ボリューム	V〈音量〉
パンポット	P〈右音量〉,〈左音量〉
音長比	Q〈比率〉
リピートビギン	[〈回数〉
リピートエンド	]
音色（ウェーブ）	@〈ウェーブ番号〉
エンベロープ	@E〈エンベロープ番号〉
デチューン	@D〈増分〉
ジャンプ	/ラベル名/
コール	(ラベル名)
リターン	;
モード	@M〈モード番号〉
データエンド	*
標準の音長	L〈音長〉

※ {} はその中の１つ、《》は省略可の意味

16色専用GP0形式について

PCエンジンの画像をかんたんに扱うために、16色（BGのアトリビュートのCG COLORがすべて0）の画像に限り、GP0という拡張子のファイル形式を作りました。ファイルの内容はファイルの先頭から、

〉 2バイト　X方向の幅（ドット数）
〉 2バイト　Y方向の幅（ドット数）
〉 32バイト　パレット
〉 28バイト　0
〉 それ以降　画像データ（1ドットが4ビットで、ドットの数だけ続く）

のように各データが格納されています。自作ツールなどの参考にしてください。また、将来の256色対応に関してGP0は一切の規定をしていません。

PC-98とMSX用にそれぞれGP0形式のファイルを作るツールを用意しました。入門編で描いた画像をPCエンジンで表示する際に使用してください。

MSX用GP0グラフィックコンバータ GRACON（gracon.com）
書式：　gracon　スクリーン5の画像ファイル
　　gracon　-r　GP0形式のファイル

解説　MSX-DOS1またはDOS2上で動作する、GP0形式のファイルとMSXのスクリーン5の画像ファイルとをコンバートするツールです。MSXの画像ファイルには、

・グラフサウルス、SII-ANIMEのSR5形式（PL5形式のパレットファイルが必要）
・F1ツールディスクのGE5形式（無圧縮に限る）
・BSAVE形式、SC5形式

のうちのどれかが使えます。GE5形式とSC5形式のファイルでは、パレットの部分までセーブされていないといけません。拡張子を省いた場合は、SR5、GE5、SC5の順番にファイルを探します。

-r オプションは逆変換です。GP0のファイルを、SR5の画像ファイルとPL5のパレットファイルに変換します。ただし、上端から212ラインまでしか変換しません。

GP0形式のドット数

GP0形式では、特にドット数の上限は規定していませんが、次のことに注意してください。
GP0を表示するdspgrpは、現在のバージョンでは、X方向の幅が256ドットのデータにしか対応していません。

GRACONについて

SR5形式のときにPL5形式のパレットファイルがない場合、エラーを出して終了します。GE5、SC5形式でパレットの範囲までセーブされていないファイルだと、GP0形式ファイル内のパレットがすべて黒になってしまいます。
コンバートできるGP0形式ファイルの画像の大きさは、MSXのスクリーン5の画像ファイルの大きさが256×212ドットなので、それ以外の大きさは無意味になります。

PC-98用GP0ローダ・セーバGP0(gp0.exe)

書式：gp0［オプション］GP0形式のファイル

・ロード系オプション
　-l：ロード(デフォルト)
　-k：表示後一時停止
　-r：画面をクリア後表示
・セーブ系オプション
　-s［xs,ys,xd,yd］：セーブ（範囲指定可。デフォルトは0,0,255,255）
　-fk：セーブファイル名をセーブ時にキーボードから入力
・常駐パレット系オプション
　-_：常駐パレットの作成
　-!：常駐パレットの解放

解説　PC-98用のGP0ローダ、セーバです。でべろシステムのGP0形式のファイルを扱うことができます。ロード、セーブには常駐パレットが必要です。作成してないときは、ロードもセーブもできませんので、常駐パレットを「-_」のオプションで作成してください。オプションは1つずつ指定してください。ロード系、セーブ系、パレット系のオプションは、同時には設定できません。ファイル名にワイルドカードを使用することはできません。

●ロード

　-Lオプションか、オプションなしのとき、ロードします。

　-kオプションは、表示後一時停止をします。なにかキーを押すとコマンドラインに戻ってきます。

　-rは、画面をクリアしてからロードするためのオプションです。ファイル名を指定しなければ、クリアしたままです。

●セーブ

　-Sオプションでセーブです。セーブ範囲やセーブ起点を変更できます。

　-S［xs,ys,xd,yd］は、(xs,yx)-(xd,yd)の範囲をセーブします。でべろ外部コマンドのdspgrpが、画像の横幅256以外をサポートしていないので、あまり変更しないほうが無難です。縦幅は336が上限です。

　-fkオプションで、ファイル名をコマンドラインからではなく、セーブ時に対話式に入力できます。マルチペイントなどのグラフィックツールの外部セーバとして使えば便利です。

●パレット

　常駐パレットに対応しています。というより、常駐パレットがないとセーブ、ロードともにできません。パソコンを起動したら、常駐パレットを作りましょう。よく使うのであまり解放はしないほうがいいです。

常駐パレット

PC-98では現在のパレット内容を取得することができずに、とても不便です。そのため、メモリの一部にパレットの内容を確保しておき、それぞれのアプリケーションからパレットを参照したり変更したりすることができるようにしたものが、常駐パレットです。マルチペイントをはじめ、数多くのツールが対応しています。

gp0.exeの開発に際して、ったく☆氏作のMagd.(PC-98用MAG画像ローダ）より常駐パレット関係のサービスを拝借し、コンパイル時に氏のUTILS.LIBを使用させていただきました。ありがとうございました。

オプションは1つずつ

-l -k -r：可
-lkr：不可
このような意味です。

ロード開始位置

PC-98の画面の左上から画像の大きさだけ展開されます。-rがないと、ロードする範囲外の画像はそのまま残ります。

セーブ範囲

PC-98の画面内ならどの位置でもセーブできますが、X方向は8ドット単位でしか設定できません。でべろ外部コマンドdspgrpの対象外の設定をしようとすると、そのままの設定でセーブをするかどうか確認を求めてきます。

マルチペイントの外部セーバ

マルチペイントのコンフィグファイルmps.cfgの中で「*CHILD MENU」以降に、
gp0セーブ=gp0 -s -fk
という一行を追加します。

パソコンを起動したら

autoexec.batに、
gp0 -_
というコマンドを入れておくといいでしょう。

サンプルプログラムの実行方法

このサンプルプログラムは、付録CD-ROMには収録されていませんので、まず間違いのないようにテキストエディタなどで入力して作成してください。

サンプルプログラムのソースプログラムが完成したら、peasコマンドを使ってアセンブルしてください。アセンブルが終了してMXファイルに変換できたら、execmxコマンドでPCエンジンに転送して実行してみましょう。

アセンブル方法などについては、30ページからを参照してください。

操作方法

サンプルプログラムを実行すると、PCエンジンの画面上にグラデーションが表示されます。ジョイパッドのIまたはIIボタンで終了しますが、画面は初期化せずにそのままで終わります。

256色表示のサンプルプログラム

```
        INCLUDE "DEVELO.INC"

dvmac   macro   @mem,@imm
        AND     @imm
        ORA     @mem
        STA     @mem
        endm

dvmvx   macro   @dst,@src
        LDA     @src+1
        STA     @dst+1,X
        LDA     @src
        STA     @dst,X
        endm

        ORG     $8000

        CLX
MakePLT:
        TXA
        ASL     A
        ASL     A
        ASL     A
        ASL     A
        ASL     A
        STA     _al
        STZ     _ah
        ROL     _ah
        TXA
        LSR     A
        LSR     A
        dvmac   _al,#$38
        TXA
        LSR     A
        LSR     A
        LSR     A
        dvmac   _al,#$02
        dvmvx   buf1,_ax
        SMB2    _al
        dvmvx   buf2,_ax
        INX
        INX
        BNE     MakePLT
        stwi    _ax,buf1
        stwi    _bx,0
        stwi    _cx,$100
        JSR     dv_Ram2Plt
        CLX
        stwi    data,$A0
MakeBAT:
        PHX
        stwi    _ax,$0100
        TXA
        dvmac   _al,#$0F
        TXA
        dvmac   _ah,#$F0
        movw    _bx,data
        stwi    _cx,2
        JSR     dv_FillVram
        incw    data
        incw    data
```

```
    PLX
    INX
    BNE    MakeBAT

    CLX
MakeCG:
    PHX
    stwi   data,0
    stwi   data+2,0
    STZ    _bl
    TXA
    LDX    #$FF
    LSR    A
    BCC    Skip1
    STX    data
Skip1:
    ROR    _bl
    LSR    A
    BCC    Skip2
    STX    data+1
Skip2:
    ROR    _bl
    LSR    A
    BCC    Skip3
    STX    data+2
Skip3:
    ROR    _bl
    LSR    A
    BCC    Skip4
    STX    data+3
Skip4:
    ROR    _bl
    CLX
Loop:
    dvmvx  buf1,data
    dvmvx  buf1+16,data+2
    INX
    INX
    CPX    #16
    BCC    Loop
    stwi   _ax,buf1
    LDA    #$10
    STA    _bh
    stwi   _cx,$10
    JSR    dv_Ram2VRam
    PLX
    INX
    CPX    #16
    BCC    MakeCG
Wait:
    JSR    ex_vsync
    TST    #3,joytrg
    BEQ    Wait
    JMP    dv_Standby
    JMP    dv_System

data:
    DS     4
buf1:
    DS     256
buf2:
    DS     256
```

パレットを16セット

16色のパレット16セットぶんのメモリは、1色につき2バイトなので、512バイト必要です。

便利な機能

グラフィックツールとしては、直線(連続直線)、ペイント、ボックス、ボックスフィル、サークル(楕円)などの機能があると便利ですが、CG COLORの違うキャラクタが途中にあった場合、CG COLORを変更しなければ機能を使って描いた線や領域が同じ色にはならないし、CG COLORを変更すればキャラクタ内の別のドットの色まで変更してしまうことになります。

ファイル形式

パレット256色ぶん、アトリビュートテーブル、キャラクタジェネレータの情報をすべて保存しておく必要があります。また、画像の大きさや画像の開始点(左上端の座標)などの情報があると応用が効きます。作者の名前などを保存するコメント領域なども画像ファイルに埋めこめると、いいでしょう。

256色への対応について

PCエンジンの256色に対応するグラフィックを描くには、PCエンジンのVRAMの構造、つまり8×8ドットのキャラクタジェネレータと、それが並んでいるアトリビュートテーブルという構造に適合したグラフィックツールが必要です。今のところそれを満たすツールは、メーカーが使っているHu7のキャラクターエディタ(ce.exe)しかありません。これはプロ用の開発機Hu7専用のツールなので、でべろには収録されていません。

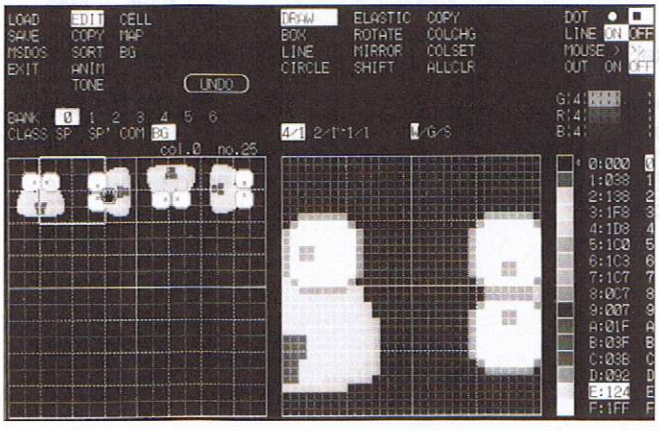
付録CD-ROMに収録されているグラフィックツールはパソコン専用なので16色までしか扱えず、256色のグラフィックを作成しようとするときには、PCエンジン(でべろ)用にグラフィックツールを自作する必要があります。

PCエンジン用のグラフィックツールに必要な機能は、512色中16色のパレットが16セット、これを常にメモリに所持していて、かつすべての8×8のキャラクタがどのパレット(CG COLOR)を使っているのかを把握していること。また、それらが自由に変更できることです。

ほかにもキャラクタをコピーするときに、CG COLORやキャラクタを別々に扱える機能や、キャラクタジェネレータのキャラクタをアトリビュートテーブルに自由に並べられる機能、現在編集している画像をワンタッチでPCエンジンに送り編集経過を確認できる機能など、他のグラフィックツールにはない機能が必要になってきます。

もちろん他のグラフィックツールが持っている便利な機能があればいいのはいうまでもありませんが、例えば直線を引くだけにしても、線の途中のキャラクタのCG COLORが違った場合、それを変更するのかどうかなど難しい問題もあります。データを保存しておくファイル形式も考案する必要があるでしょう。

これらの課題をクリアしたでべろ用のグラフィックツールを作ったならば、ぜひ編集部まで送ってください。



Hu7システム専用のグラフィックツールce.exe の実行画面。PCエンジンのグラフィックを描くのに使っている

第4部
データ&資料

C O N T E N T S

メモリマップ

メモリレイアウト

HuC62のメモリは、でべろシステム使用時と非でべろシステム使用時とでユーザーに解放されるエリアに違いがあります。また、PSGドライバーやグラフィックドライバーの起動時、演算パッケージの使用時、漢字ROMアクセス時には、一時的にメモリのレイアウトが下表のように変更されます。これらの場合にバンクが切り換わるときは、そのエリアに割込み処理を配置すると暴走する可能性があるので注意してください。

論理アドレス	通常時	PSGドライバー起動時	グラフィックドライバー起動時	演算パッケージ使用時	漢字ROMアクセス時(a)	漢字ROMアクセス時(b)
$0000 ～$1FFF	I/O (MPR0=$FF)	※固定				
$2000 ～$3FFF	WORK RAM (MPR1=$F8)	※固定				
$4000 ～$5FFF	でべろシステム (MPR2=$80)	※固定				
$6000 ～$7FFF	ユーザー解放 (MPR3=$84)	ユーザー解放 (MPR3=$84)	ユーザー解放 (MPR3=$84)	ユーザー解放 (MPR3=$84)	漢字 ROM 1 (MPR3=$??)	ユーザー解放 (MPR3=$84)
$8000 ～$9FFF	ユーザー解放 (MPR4=$85)	PSG DATA 0 (MPR4=$??)	ユーザー解放 (MPR4=$85)	ユーザー解放 (MPR4=$85)	漢字 ROM 2 (MPR4=$??)	ユーザー解放 (MPR4=$85)
$A000 ～$BFFF	ユーザー解放 (MPR5=$86)	PSG DATA 1 (MPR5=$??)	ユーザー解放 (MPR5=$86)	ユーザー解放 (MPR5=$86)	ユーザー解放 (MPR5=$86)	漢字 ROM 1 (MPR5=$??)
$C000 ～$DFFF	ユーザー解放 (MPR6=$87)	PSGドライバー (MPR6=$02)	ｸﾞﾗﾌｨｯｸﾄﾞﾗｲﾊﾞｰ (MPR6=$03)	演算パッケージ (MPR6=$??)	ユーザー解放 (MPR6=$87)	漢字 ROM 2 (MPR6=$??)
$E000 ～$FFFF	BIOS ROM (MPR7=$00)	※固定				

【解説】
でべろシステムで使用されるエリアは上表のものだけです。非使用時には、このエリアもユーザーに解放されます。
また、$80と$81のバンクのRAM はシステムが使用していますので、これらをユーザー解放エリアなどに切り換えて使用することは避けてください。$82と$83のバンクのRAMも、将来システムの拡張で使用される可能性があります。なお、これらのRAMも、非使用時には解放されます。
PSGドライバー起動時には、表のようにメモリが切り換わります。PSGドライバー用のデータを配置するメモリは、ユーザーに解放されているRAMを、そのまま指定することも可能です。
漢字ROMアクセス時には、フォントデータの転送先のアドレスの指定により、上表のいずれかに切り換わります。転送先のアドレスが$A000以上の場合は(a)の配置に、$A000未満の場合は(b)の配置になります。

ワークRAM

$2000～$3FFFの領域は、でべろシステムやCD-ROM² BIOS、PSGドライバー、グラフィックドライバーなどの動作に必要な情報が置かれています。これらで使用される領域を十分に考慮した上で、ユーザーの領域を確保してください。

ゼロページRAM

論理アドレス	
$2000～$209F	ユーザー解放
$20A0～$20DB	でべろシステム
$20DC～$20E5	グラフィックドライバー
$20E6～$20EB	PSGドライバー
$20EC～$20FF	BIOS

WORK RAM

論理アドレス	
$2200～$22CE	BIOS
$22D0～$2615	PSGドライバー
$2616～$2648	グラフィックドライバー
$2649～$2FFF	でべろシステム (リザーブを含む)
$3000～$3FFF	ユーザー解放

【解説】
でべろシステム用の領域は、将来の拡張用のリザーブ用領域などを含みます。なお、非でべろシステム使用時には、これらの領域を使用することも可能です。
PSGドライバーおよびグラフィックドライバー用の領域は、これらのドライバーを使用していない場合に限り、ユーザーが使用することが可能です。
$2100～$21FFは、スタックエリアとして使用されます。

I/Oマップ

I/Oへのアクセス

論理アドレスの$0000～$1FFFはI/Oになっています。I/Oを介して、HuC6270やHuC6260、PSG、内蔵タイマーなどへのアクセスが可能です。PSGのレジスタやタイマーの操作は、このI/Oを通してのみおこなえます。

HuC6270などへのアクセスは、なるべくBIOSを利用するようにして、直接I/Oからアクセスするのを避けてください。

$0000～$0003

HuC6270へのアクセス窓口です。

$0000にはHuC6270のアクセスするレジスタの番号を指定します。

$0002と$0003は、データの読み書きに利用します。$0002がデータの下位、$0003はデータの上位です。

内部動作に詳しくないうちは、I/Oからのアクセスは避けてください。

$0400～$0407

HuC6260へのアクセス窓口です。

$0400でHuC6260のコントロールレジスタにアクセスできます。

$0402でカラーテーブルのアドレスを指定します。

$0404からパレットテーブルのデータを読み書きできます。

内部動作に詳しくないうちは、I/Oからのアクセスは避けてください。

$0800～$080F

PSGへのアクセス窓口です。

アドレスの下位4ビットが、それぞれPSGのレジスタ番号になります。

チャンネルごとに用意されているレジスタにアクセスする場合は、$0800にチャンネル番号を設定してからアクセスします。

$0C00～$0C01

内蔵タイマーへのアクセス窓口です。

$0C00に7ビットのデータを書き込むと、タイマーのインターバルを設定できます。

$0C01にビット0が1のデータを書き込むとタイマーが作動します。

$1000

PCエンジンマウスなどで使用しますが、直接操作することは避けてください。

$1402～$1403

$1402は割込みの禁止・許可を設定しますが、とくに必要のない限り、これを操作するのは避けてください。

$1403はタイマー割込みを使用している場合に限り操作が必要です。タイマー割込みが発生した場合には、割込み処理でこのアドレスに書き込みを実行して、タイマー割込みのフラグを初期化します。

デバイス・レジスタアドレス一覧表

No.	物理アドレス (16進)	アドレス指定例(16進) MPR	アドレス指定例(16進) 論理アドレス	デバイス名 (レジスタ名)	ビット構成と機能(16進)
1	1FE000 〜 1FE3FF	MPR0= FF	0000 0002 0003	HuC6270 (A1〜A0)	論理ｱﾄﾞﾚｽ / HuC6270用命令 0000 ST0 0002 ST1 0003 ST2
2	1FE400 〜 1FE7FF	MPR0= FF	0400 〜 0407	HuC6260 (A2〜A0)	――
3	1FE800 〜 1FEBFF	MPR0= FF	0800 〜 080F	PSG (A3〜A0)	――
4	1FEC00 〜 1FEFFF (A0=1)	MPR0= FF	0C01	ﾀｲﾏｺﾝﾄﾛｰﾙ ﾚｼﾞｽﾀ (ﾗｲﾄのみ可)	7 6 5 4 3 2 1 0 未使用 1：ﾀｲﾏｽﾀｰﾄ 0：ﾀｲﾏｽﾄｯﾌﾟ
	1FEC00 〜 1FEFFF (A0=0)	MPR0= FF	0C00	ﾗｲﾄ時： ﾀｲﾏﾘﾛｰﾄﾞﾚｼﾞｽﾀ	7 6 5 4 3 2 1 0 ↑ ﾀｲﾏｲﾝﾀｰﾊﾞﾙ設定値(7bit) 未使用
				ﾘｰﾄﾞ時： ﾀｲﾏﾀﾞｳﾝｶｳﾝﾀ	7 6 5 4 3 2 1 0 ↑ ﾀｲﾏﾀﾞｳﾝｶｳﾝﾀ(7bit) 未使用
5	1FF000 〜 1FF3FF	MPR0= FF	1000	ﾗｲﾄ時： Oﾎﾟｰﾄ	O7〜O0へ出力
				ﾘｰﾄﾞ時： Kﾎﾟｰﾄ	K7〜K0より入力
6	1FF400 〜 1FF7FF (A1=1 A0=0)	MPR0= FF	1402	ｲﾝﾀﾗﾌﾟﾄ ﾃﾞｨｾｰﾌﾞﾙﾚｼﾞｽﾀ (ﾘｰﾄﾞ/ﾗｲﾄ可)	7 6 5 4 3 2 1 0 未使用 IRQ2D (bit0) 1：ﾃﾞｨｾｰﾌﾞﾙ 0：ｲﾈｰﾌﾞﾙ IRQ1D (bit1) 1：ﾃﾞｨｾｰﾌﾞﾙ 0：ｲﾈｰﾌﾞﾙ TIQD (bit2) 1：ﾃﾞｨｾｰﾌﾞﾙ 0：ｲﾈｰﾌﾞﾙ
	1FF400 〜 1FF7FF (A1=1 A0=1)	MPR0= FF	1403	ｲﾝﾀﾗﾌﾟﾄ ﾘｸｴｽﾄﾚｼﾞｽﾀ (ﾘｰﾄﾞ可 ﾗｲﾄで TIQをﾘｾｯﾄする)	7 6 5 4 3 2 1 0 未使用 IRQ2 (bit0) 1：割込み要求有 0：割込み要求無 IRQ1 (bit1) 1：割込み要求有 0：割込み要求無 TIQ (bit2) 1：割込み要求有 0：割込み要求無
7	1F0000 〜 1F7FFF	MPR1= F8	2000 〜 3FFF	ﾃﾞｰﾀﾒﾓﾘ (RAM)	

BIOS一覧

●解説の各項目について

[　] 内……BIOSの登録番号です。特に意識する必要はありません。また、でべろシステムのBIOSにはありません。

ラベル名……プログラムソースを記述する際には、この欄に記された名称で使用します。ただし、peas.exe（またはpeas.com）では、ラベル名の大文字と小文字を判別するので注意が必要です。詳しくは「プログラミング解説」の項をお読みください。

機能 …… そのBIOSの動作の紹介です。

IN …… そのBIOSを使用する際に必要な設定です。

OUT …… そのBIOSを使用したときに、結果として返ってくる情報です。

【解説】…… そのBIOSの動作や、使用する際の手順、注意などを解説します。使用する前に必ずお読みください。

【実行例】…… そのBIOSを使用する例です。

●解説中の表記について

AREG、XREG、YREG……それぞれ、CPUが持っている8ビットのレジスタ

CARRY……ステータスレジスタ中のキャリーフラグ

_AX、_BX、_CX、_DX……ゼロページに用意されている2バイトの汎用ワーク

_AL、_AH、_BL、_BH、_CL、_CH、_DL、_DH……それぞれ、上記汎用ワークを下位と上位に分けた1バイトのワーク

アドレスや16進数表記のデータは「$13FF」のように記述します。2進数表記のデータは「%0000_1010」のように記述します。これ以外の数値については、すべて10進数です。
全文を通して「bit0」が最下位ビットになります。
「LOW」または「L」と付加されている場合はデータの下位を、「HIGH」または「H」と付加されている場合はデータの上位を表わします。

●BIOS使用時の注意事項

CD-ROM²BIOS実行後のレジスタやワークの内容は、明記されていないものに関しては、その値は保証されません。BIOSを呼び出す際には、かならず適切な設定、退避および初期化をおこなってください。

また、ラベル名が「dv_」で始まるBIOSは、でべろシステムが有効な場合に限り使用できます。でべろシステムは、メインRAMの$4000～$5FFFにありますが、この部分を書き換えた場合には、動作は保証されません。

[$00] ラベル名：CD_BOOT アドレス：$E000

機能　CD-ROM² BIOSをブートする
IN　なし
OUT　なし

【解説】
CD-ROM²　BIOS起動時の状態に移行します。
起動時タイトル画面を表示して、RUNボタン入力待ちとなります。
自作プログラムでソフトウェアリセットに対応する場合には、このアドレスにジャンプして終了するようにしてください。

[$0F] ラベル名：CD_FADE アドレス：$E02D

機能　リニアPCM、ADPCMのフェードアウト開始または解除
IN　AREG: 動作モード
　　$00　フェードアウト解除
　　$08　PCM FADE OUT (6.0秒)
　　$0A　ADPCM FADE OUT (6.0秒)
　　$0C　PCM FADE OUT (2.5秒)
　　$0E　ADPCM FADE OUT (2.5秒)
OUT　なし

【解説】
フェードアウトをセットすると、オーディオ回路はミューティング状態となります。
フェードアウト後、再度再生を行う際には、あらかじめフェードアウトを解除しておく必要があります。
また、フェードアウトを解除せずにもう一度フェードアウトを実行したり、フェードアウト解除後200ミリ秒以内にフェードアウトを実行すると、フェードアウトせずにカットアウトされてしまうので、このような使用は避けてください。

[$10] ラベル名：AD_RESET アドレス：$E030

機能　ADPCMコントローラーをリセットする
IN　なし
OUT　なし

【解説】
ADPCMのコントローラーを初期化します。

[$12] ラベル名：AD_READ アドレス：$E036

機能　ADPCMバッファからメモリへデータを読み込む
IN　_CX:　ADPCMバッファのアドレス
_DH:　読み込み先のアドレスタイプ

	$00：LOCAL	$FF：VRAM	$02～$06：MPR No.
_BL:	ADR　L	ADR　L	BANK No.（$80～$87）
_BH:	ADR　H	ADR　H	No USE

_AX:　読み込みサイズ（バイト単位）
OUT　AREG:　実行結果
$00　正常終了
その他　エラー発生

【解説】
VRAMへの読み込み中は、$2272のVDTIN_FLGが1になります。
VRAMへの読み込み中に割込み処理などを行い、HuC6270のレジスタMAWR、MARRの内容を変更した場合、正常な動作は保証されません。
・重要注意
CD-ROM² BIOS ver.1.00では、このAD_READを使用すると不具合が生じる可能性があります。

[$13] ラベル名：AD_WRITE アドレス：$E039

機能　メモリからADPCMバッファへデータを書き込む
IN　_CX:　ADPCMバッファのアドレス
_DH:　書き込みデータアドレスタイプ

	$00：LOCAL	$FF:VRAM	$02～$06:MPR No.
_BL:	ADR　L	ADR　L	BANK No.（$80～$87）
_BH:	ADR　H	ADR　H	No USE

_AX:　書き込みサイズ（バイト単位）
OUT　AREG: 実行結果
$00　正常終了
その他　エラー発生

【解説】
VRAMからの読み込み中は、$2272のVDTIN_FLGが1になります。
VRAMからの読み込み中に割込み処理などを行い、HuC6270のレジスタMAWR、MARRの内容を変更した場合、正常な動作は保証されません。
・重要注意
CD-ROM² BIOS ver.1.00では、このAD_WRITEを使用すると不具合が生じる可能性があります。残念ながら、これについての不具合を解消する方法はありません。
なお、CD-ROM² BIOS ver.1.00以外では、正常に動作します。

[$14]　ラベル名：AD_PLAY　アドレス：$E03C

機能　ADPCMバッファのデータを部分再生する
IN　_BX:　ADPCMバッファ再生開始アドレス（ADR）
_AX:　再生サイズ（バイト単位。LENGTH）
_DH:　サンプリングレート（$00～$0Eの値。RATE）
fKHz＝32/(16－_DH)
_DL:　モード
bit0　COUNTER MODE
0:　ADR, LENGTH, RATE をセットする
1:　ADR, LENGTH, RATE を前回の値のままで再生する
bit7　PLAY MODE
0:　AUTO STOP
1:　REPEAT
OUT　AREG: 実行結果
$00　正常終了
その他　エラー発生

【解説】
ADPCM再生中にAD_PLAYを呼び出すとエラーが発生します。
実行前には、ファンクション[$16]のAD_STOPを呼び出して再生を停止させ、さらにファンクション[$17]のAD_STATを呼び出して停止を確認してください。
・重要注意
CD-ROM² BIOS ver.1.00では、このAD_PLAYを使用すると不具合が生じる可能性があります。

[$16] ラベル名：AD_STOP アドレス：$E042

機能　　ADPCMバッファのデータの再生を中止する
IN　なし
OUT　なし

【解説】
ADPCMの再生を中止します。
なお、再生は16KHz程度の速度でおこなわれているため、完全に再生が停止するまでは若干の時間を必要とします。
他の処理をおこなう前に、ファンクション[$17]のAD_STATによって、停止の確認をおこなってください。

[$17] ラベル名：AD_STAT アドレス：$E045

機能　　ADPCMコントローラの状態を読み出す
IN　なし
OUT　AREG:　ADPCMコントローラの状態
　　$00　　ADPCM NOT BUSY (END OR NOT PLAY)
　　その他　ADPCM BUSY
　XREG:　ADPCMバッファ、および再生の状態
　　$00　　再生中でバッファの残りデータが半分以上
　　$01　　再生停止
　　$04　　再生中でバッファの残りデータが半分以下

【解説】
ADPCMコントローラーのBUSYチェック、ADPCMバッファに残っているデータの量、再生状態をチェックします。

[$18] ラベル名：BM_FORMAT アドレス：$E048

機能　バックアップメモリを初期化する
IN　　_AX:　パスワードアドレス
　　パスワードは"!BM　FORMAT!"に固定
OUT　AREG:　実行結果
　　$00　　正常終了
　　$01　　エラー発生

【解説】
メインRAMの任意の場所にパスワードバッファを用意して、パスワードの文字列をセットしておきます。そ

のバッファの先頭アドレスを_AXに設定して呼び出します。
なお、パスワードは"!BM　FORMAT!"の11文字のデータに固定されています。
パスワードが一致しない場合には、フォーマットは実行されません。
・注意
バックアップメモリにアクセスする場合、アクセスウィンドウとして$8000～$DFFFが使用されます。CD-ROM² BIOS ver.1.00では、パスワードバッファが$8000～$DFFFの間にあると認識されません。$4000～$5FFFのRAMはでべろシステムが占有していますので、$6000～$7FFFのRAM上に設置するか、一時的に$3000からのワークエリアを使うことでおこなってください。

[$19]　ラベル名：BM_FREE　アドレス：$E04B

機能　バックアップメモリの空き容量を調べる
IN　なし
OUT　_CX:　空き容量（バイト数）

【解説】
バックアップメモリに書き込む前に、必ずこのBIOSを呼び出して、空き容量の確認をしてください。
結果として得られる_CXの値は、実際の空き容量よりも2バイト大きい数値になりますので、使用する際には十分な注意が必要です。
未使用領域の先頭アドレスは、バックアップメモリの$0006～$0007に格納されています。
・注意
CD-ROM² BIOS ver.1.00では、空き容量がまったくない場合に、_CXの値が負になることがあります。空き容量のチェックをおこなうときは、符号付きの比較をしてください。

[$1A]　ラベル名：BM_READ　アドレス：$E04E

機能　バックアップメモリからデータを読み込む
IN　_AX:　FCBアドレス
　　_BX:　読み込んだデータを格納する領域の先頭アドレス
　　_CX:　読み込むデータ数（バイト単位）
　　_DX:　先頭からのファイル内オフセット
OUT　_CX:　実際に読み込んだバイト数
　　AREG:　実行結果
　　$00　正常終了
　　$01　ファイルが見つからない
　　$02　データが壊れている（チェックサムエラー）
　　$FF　フォーマットエラー

【解説】

ファイルの先頭からデータを読み出す場合、_DXに設定する先頭からのファイル内オフセットは$00になります。ファイルの先頭16バイトには、ディレクトリ情報が存在しますが、BM_READでデータを読み込む場合、この16バイトを意識する必要はありません。

・注意

バックアップメモリにアクセスする場合、アクセスウィンドウとして$8000～$DFFFが使用されます。CD-ROM² BIOS ver.1.00では、FCBが$8000～$DFFFの間にあると認識されません。$4000～$5FFFのRAMはでべろシステムが占有していますので、$6000～$7FFFのRAM上に設置するか、一時的に$3000からのワークエリアを使うことでおこなってください。

・FCBの内容

FCBとして必要な情報は、以下のとおりです。

+$00～+$01 ユーザーID。ファイル識別用（2バイト）
+$02～+$0B ファイル名（10バイト）

ファイル名が10文字に満たない場合は、$20の空白で補ってください。また、ファイル名の途中に空白を置くことはできません。

ファイル名に使用できる文字種は、以下のものに限られています。

$2D	－
$2E	.
$2F	／
$30～$39	0～9
$41～$5A	A～Z

【使用例】

```
    stz   _al               ;FCB adr L
    lda   #$01
    sta   _ah               ;FCB adr H
    lda   #LOW sram_buf
    sta   _bx               ;destination buffer adr L
    lda   #HIGH sram_buf
    sta   _bx+1             ;destination buffer adr H
    stz   _ch
    lda   #$40
    sta   _cl               ;read length (64bytes)
    stz   _dh
    stz   _dl               ;file offset (data top)
    jsr   bm_read
      ⋮
      ⋮

sram_buf:
    ds    $40
```

[$1B] ラベル名：BM_WRITE アドレス：$E051

機能　バックアップメモリへデータを書き込む
IN　_AX:　FCBアドレス
　　_BX:　書き込むデータが格納された領域の先頭アドレス
　　_CX:　書き込むデータ数（バイト単位）
　　_DX:　先頭からのファイル内オフセット
OUT　AREG:　実行結果
　　$00　正常終了
　　$01　バックアップメモリが足りない
　　$FF　フォーマットエラー

【解説】
ファイルの先頭からデータを書き込む場合、_DXに設定する先頭からのファイル内オフセットは$00になります。ファイルの先頭16バイトには、ディレクトリ情報が存在しますが、BM_WRITEでデータを書き込む場合、この16バイトを意識する必要はありません。
・注意
バックアップメモリにアクセスする場合、アクセスウィンドウとして$8000～$DFFFが使用されます。CD-ROM² BIOS ver.1.00では、FCBが$8000～$DFFFの間にあると認識されません。$4000～$5FFFのRAMはでべろシステムが占有していますので、$6000～$7FFFのRAM上に設置するか、一時的に$3000からのワークエリアを使うことでおこなってください。
また、実行前に必ずファンクション[$19]のBM_FREEを実行して、空き容量の確認をおこなってください。実際の空き容量よりも大きいサイズの書き込みをおこなうと、バックアップメモリ全体への悪影響が発生します。
・FCBの内容
FCBとして必要な情報は、以下のとおりです。
　　+$00～+$01 ユーザーID。ファイル識別用（2byte）
　　+$02～+$0B ファイル名（10byte）
ファイル名が10文字に満たない場合は、$20の空白で補ってください。また、ファイル名の途中に空白を置くことはできません。
ファイル名に使用できる文字種は、以下のものに限られています。
　　$2D　－
　　$2E　.
　　$2F　／
　　$30～$39　0～9
　　$41～$5A　A～Z

【使用例】

```
stz   _al              ;FCB adr L
lda   #$01
sta   _ah              ;FCB adr H
lda   #LOW bm_data
sta   _bx              ;source data adr L
lda   #HIGH bm_data
sta   _bx+1            ;source data adr H
```

```
    stz     _ch
    lda     #$40
    sta     _cl             ;write length (64bytes)
    stz     _dh
    stz     _dl             ;file offset (data top)
    jsr     bm_write
        ⋮
bm_data:
    db      $ff,$00,$80
        ⋮
```

[$1C] ラベル名：BM_DELETE アドレス：$E054

機能　バックアップメモリのファイルを削除する
IN　_AX:　FCBアドレス
OUT　_AREG:　実行結果
　　$00　正常終了
　　$01　削除できない
　　$FF　フォーマットエラー

【解説】
ファイル削除によって生じたバックアップメモリの隙間は、削除ファイル以降を詰めることによって、自動的に整理されます。
・注意
バックアップメモリにアクセスする場合、アクセスウィンドウとして$8000～$DFFFが使用されます。CD-ROM² BIOS ver.1.00では、FCBが$8000～$DFFFの間にあると認識されません。$4000～$5FFFのRAMはでべろシステムが占有していますので、$6000～$7FFFのRAM上に設置するか、一時的に$3000からのワークエリアを使うことでおこなってください。

[$1D] ラベル名：BM_FILES アドレス：$E057

機能　バックアップメモリのファイル情報を得る
IN　_BX:　ファイル情報の転送先アドレス
　　_AL:　ファイルの先頭からの番号（先頭=1）
OUT　AREG:　実行結果
　　$00　正常終了

$01　ファイルが見つからない
_AL＝最大ファイル番号
$FF　フォーマットエラー

【解説】
ファイルの先頭からの番号に、ある程度大きな値を設定することにより、Aレジスタに$01が入り、_ALに最大ファイル番号を獲得することができます。
・得られる情報
BM_FILESによって得られる情報は、FCB情報にファイルサイズ情報の2バイトが付加された計14バイトの情報です。

+$00～+$01 ユーザーID（2バイト）
+$02～+$0B ファイル名（10バイト）
+$0C～+$0D ファイルサイズ（2バイト）

【使用例】

```
    lda    #LOW FCB_buf
    sta    _bx
    lda    #HIGH FCB_buf
    sta    _bx+1
    lda    #5                ; file number (5)
    sta    _al
    jsr    bm_files
      ⋮
      ⋮

FCB_buf:
    ds     14
```

[$1E] ラベル名：EX_GETVER アドレス：$E05A

機能　BIOSのバージョン番号を調べる
IN　なし
OUT　XREG:　バージョン番号の整数部
　　YREG:　バージョン番号の小数部

【解説】
CD-ROM²のBIOSには、ver.1.00、ver.2.00、ver.2.10、ver.3.00があります。
一部のBIOSファンクションには、バージョンによる動作の違いなど注意しなければならない場合がありますので、そのようなBIOSファンクションを使用する際には、バージョン番号の確認をおこなう必要があります。

[$1F] ラベル名：EX_SETVEC アドレス：$E05D

機能　ユーザー処理の割り込みベクターアドレスをセットする
IN　AREG: セットするベクター番号
　　$00　IRQ2
　　$01　IRQ
　　$02　TIMER
　　$03　NMI
　　$04　SYNC
　　$05　RCR
　　$06　SOFT RESET
　　XREG: ユーザー処理のベクターアドレス（LOW）
　　YREG: ユーザー処理のベクターアドレス（HIGH）
OUT　なし

【解説】
各割り込みベクターは、$20F5にあるIRQ_Mの対応するビットが1のとき有効になります。ただし、IRQ_Mを設定する前に、必ずこのファンクションを実行して、ベクターアドレスを設定しておく必要があります。
・IRQ_Mの各ビットの意味
　　bit0　IRQ2 jmp vector enb
　　bit1　IRQ jmp vector enb
　　bit2　TIMER jmp vector enb
　　bit3　NMI jmp vector enb
　　bit4　if IRQ vector off then SYNC jmp vector enb
　　bit5　nop sync irq
　　bit6　if IRQ vector off then RCR jmp vector enb
　　bit7　nop rcr irq
・割り込み処理の終了
ユーザー処理の終了は、SYNCとRCRはrtsによって、IRQ2、IRQ、TIMER、NMIはrtiによって終了してください。
・IRQを使用する場合
IRQによる割り込みのベクターアドレスを書き換える場合は、ユーザー処理で以下のワークを使用しないと不具合が生じる可能性があります。
　　$20F3　CRL_M　次のIRQ割り込み時に、HuC6270のCRレジスタに設定する値(LOW)
　　$20F4　CRH_M　次のIRQ割り込み時に、HuC6270のCRレジスタに設定する値(HIGH)
　　$20F7　REG_BOX 現在のHuC6270のARレジスタの内容を保存するワーク
CD-ROM² BIOSの内部や、でべろのシステムでも、IRQによる割り込みを使用しています。不用意に使用すると、動作は保証できません。
・TIMERを使用する場合
TIMERによる割り込みを使用する場合には、ベクターアドレスをセットしてIRQ_Mの対応するビットをセットした後、内蔵タイマーの設定と実行をおこなう必要があります。
また、ユーザーがTIMERによる割り込みを使用している場合、BIOS内PSGドライバーをTIMERによって呼び出すことは原則としてできませんが、ユーザーのTIMER処理からPSGドライバーを呼び出すことで共存は可能です。ただしこの場合、内蔵タイマーの動作はPSGドライバーに依存します。

BIOS内PSGドライバーは、VSYNCによって呼び出すこともできるので、TIMERを使用する場合には、できるだけVSYNCで呼び出すようにしてください。

・SOFT RESETのベクターアドレスについて

SOFT RESETのベクターアドレスには、BIOS内のIRQ処理、またはファンクション[$21]のEX_JOYSNSが実行されているときに、RUN+SELECTボタンが押された場合に分岐するアドレスを設定します。このベクターアドレスを設定する場合、IRQ_Mの操作は不要です。

ただし、BIOS内PSGドライバーを使用しているときにEX_JOYSNSによるSOFT RESETが発生すると、PSGドライバーが呼び出されなくなるので注意してください。この場合、音源の初期化はされません。

・重要注意

TIMER、SYNC、RCR、SOFT RESET以外の使用は、なるべく避けてください。でべろシステムやBIOSの動作に支障をきたす場合があります。

【実行例】

```
    lda   #02                ;TIMER
    ldx   #LOW timer_sub     ;割り込み処理のアドレス下位
    ldy   #HIGH timer_sub    ;割り込み処理のアドレス上位
    jsr   ex_setvec
    smb2  irq_m
      ⋮
      ⋮

timer_sub:                   ;ユーザーの割り込み処理


    rti
```

[$20] ラベル名：EX_GETFNT アドレス：$E060

機能　漢字ROM内のフォントデータ（32バイト）を転送する

IN　_AX:　漢字コード（シフトJISコード）

　　_BX:　転送先アドレス

　　_DH:　転送モード

　　$00　16×16ドットフォントを転送

　　$01　12×12ドットフォントを転送

OUT　AREG:　実行結果

　　$00　正常終了

　　$01　漢字コードが不正である

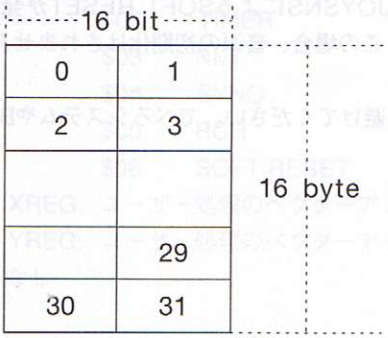
【解説】

用意されている漢字フォントは、JIS第1水準までです。

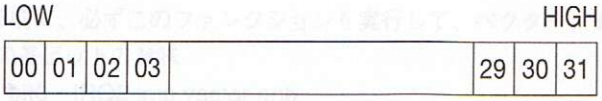
このBIOSが呼び出されると、転送先のアドレスによって、バンク5と6（$A000～$DFFF）またはバンク3と4（$6000～$9FFF）のいずれかが、漢字ROMと切り換わってデータを転送します。

・転送されるフォントパターンデータについて
データは、指定されたアドレスから32バイト連続して書き込まれます。データの1バイト目から順に、フォントの1ライン目左側8ドットのデータ、1ライン目右側8ドットのデータ、2ライン目左側8ドット……16ライン目右側8ドットのデータ、というように展開されます。

RAMへの転送形式



フォントの形状データ



・割り込み処理に関連する注意
漢字ROMに切り換わってしまう部分には、割り込み処理を置かないでください。
また、データ転送の際、わずかな時間ですが、割り込みが禁止状態になりますので、ラスター割り込みを多用している場合などでは、割り込みが発生しない状況（垂直帰線期間など）での使用をお薦めします。
・フォントに関する注意
CD-ROM² のBIOSのバージョンによって、フォントパターンに若干の違いがあります。また、ver.1.00では、「抜」のフォントに誤りがあるため、この文字を使用する場合には独自にフォントを用意する必要があります。

[$21] ラベル名：EX_JOYSNS アドレス：$E063

機能　ジョイパッドの状態を調べる
IN　$2227　JOYENA:　ソフトリセットの可能、不可能を指定する
　　bit0　1番のパッド（1でリセット可能。以下同）
　　bit1　2番のパッド
　　bit2　3番のパッド
　　bit3　4番のパッド
　　bit4　5番のパッド
OUT　$2228　JOY,X:　各ボタンの状態(X=0～4)
　　$222D　JOYTRG,X:　各ボタンで入力があったかの情報(X=0～4)
　　$2232　JOYOLD,X:　1ループ前のJOYの内容（作業用）(X=0～4)
　　実行後に得られる各ワークのデータの意味

bit0	ONE	0:off 1:on
bit1	TWO	0:off 1:on
bit2	SELECT	0:off 1:on
bit3	RUN	0:off 1:on
bit4	UP	0:off 1:on
bit5	RIGHT	0:off 1:on
bit6	DOWN	0:off 1:on
bit7	LEFT	0:off 1:on

【解説】
各5バイトのワークエリアに、順番に1番～5番のパッド情報が記録されます。
JOYTRGの内容は、前回OFFで今回ONになったボタンを示します。

[$23] ラベル名：EX_SCRSIZ アドレス：$E069

機能　仮想スクリーンのX、Y方向の文字数をセットする
IN　AREG: SCREEN SIZE

bit	210	x	y
	000	32	32
	001	64	32
	010	128	32
	011	128	32
	100	32	64
	101	64	64
	110	128	64
	111	128	64

OUT　なし

【解説】
設定に基づいてVRAMのデータが扱われます。
実際に表示される画面のサイズではありません。

[$24] ラベル名：EX_DOTMOD アドレス：$E06C

機能　アクセスドット幅をセットする
IN　AREG:　DOT SIZE
　　bit 7 6 5 4 3 2 1 0
　　x x x x x x ? ?　VRAMのアクセスドット幅
　　x x x x ? ? x x　スプライトのアクセスドット幅
OUT　なし

【解説】
VDCの内部動作に詳しくないうちは、不用意に使用しないでください。

[$25] ラベル名：EX_SCRMOD アドレス：$E06F

機能　スクリーンモードをセットする
IN　AREG:　CLOCK周波数
　　$00　5MHz
　　$01　7MHz
　　XREG:　横方向の表示文字数
　　10－34 (5MHz)
　　16－44 (7MHz)
　　YREG:　縦方向の表示文字数
　　10－30 (5MHz、7MHz共通)
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー終了

【解説】
エラー時には、クロック5MHz、横32文字、縦30行にセットされます。
横方向の表示文字数に、奇数値を設定することはできません。

[$26] ラベル名：EX_IMODE アドレス：$E072

機能　VRAMアクセス時のアドレスのインクリメント幅のセット
IN　AREG:　インクリメント幅
　　0　+1
　　1　+32
　　2　+64
　　3　+128

OUT　なし

【解説】
設定値にしたがって、VRAMのアドレスがインクリメントされます。

[$27]　ラベル名：EX_VMODE　アドレス：$E075

機能　VRAMアクセス時のアドレスのインクリメント幅のセット
IN　　なし
OUT　なし

【解説】
VRAMのアドレスをインクリメントする幅を、ファンクション[$23]のEX_SCRSIZによる設定に基づいて、32、64、128のいずれかにセットします。
このBIOSを使用する前に、必ずファンクション[$25]のEX_SCRMODを呼び出して、スクリーンモードを設定しておかなければいけません。
ファンクション[$2C]のEX_IRQONを実行したときに有効となります。
また、BIOS内のIRQルーチンを使用しなければいけません。

[$28]　ラベル名：EX_HMODE　アドレス：$E078

機能　VRAMアクセス時のアドレスのインクリメント幅を1にセットする
IN　　なし
OUT　なし

【解説】
このBIOSを使用する前に、必ずファンクション[$25]のEX_SCRMODを呼び出して、スクリーンモードを設定しておかなければいけません。
ファンクション[$2C]のEX_IRQONを実行したときに有効となります。
また、BIOS内のIRQルーチンを使用しなければいけません。

[$29]　ラベル名：EX_VSYNC　アドレス：$E07B

機能　　VSYNC割り込みが発生するまで待つ
IN　　なし
OUT　　なし
BREAK　AREG

【解説】
VSYNC割り込みが発生するとリターンします。
ただし、$20F5にあるIRQ_MのbIt1がON（ユーザーがIRQを使用中）の場合は、VSYNC割り込みの発生を待たずにリターンします。
また、VSYNCのIRQがOFFのときは、強制的にONになります。
BIOS内のIRQルーチンを使用しなければいけません。

[$2A] ラベル名：EX_RCRON アドレス：$E07E

機能　ラスター割り込みを発生させる
IN　なし
OUT　なし
BREAK　AREG

【解説】
BIOS内のIRQルーチンを使用しなければいけません。

[$2B] ラベル名：EX_RCROFF アドレス：$E081

機能　ラスター割り込みを停止する
IN　なし
OUT　なし
BREAK　AREG

【解説】
BIOS内のIRQルーチンを使用しなければいけません。

[$2C] ラベル名：EX_IRQON アドレス：$E084

機能　VSYNC割り込みを発生させる
IN　なし
OUT　なし
BREAK　AREG

【解説】
BIOS内のIRQルーチンを使用しなければいけません。

[$2D] ラベル名：EX_IRQOFF アドレス：$E087

機能　VSYNC割り込みを停止する
IN　なし
OUT　なし
BREAK　AREG

【解説】
BIOS内のIRQルーチンを使用しなければいけません。

[$2E] ラベル名：EX_BGON アドレス：$E08A

機能　バックグラウンドの表示を許可する
IN　なし
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
BIOS内のIRQルーチンを使用しなければいけません。

[$2F] ラベル名：EX_BGOFF アドレス：$E08D

機能　バックグラウンドの表示を停止する
IN　なし
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
BIOS内のIRQルーチンを使用しなければいけません。

[$30] ラベル名：EX_SPRON アドレス：$E090

機能　スプライトの表示を許可する
IN　なし
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
BIOS内のIRQルーチンを使用しなければいけません。

[$31] ラベル名：EX_SPROFF アドレス：$E093

機能　スプライトの表示を停止する
IN　なし
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
BIOS内のIRQルーチンを使用しなければいけません。

[$32] ラベル名：EX_DSPON アドレス：$E096

機能　バックグラウンド、スプライトの表示を許可する
IN　なし
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
BIOS内のIRQルーチンを使用しなければいけません。

[$33] ラベル名：EX_DSPOFF アドレス：$E099

機能　バックグラウンド、スプライトの表示を停止する
IN　なし
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
BIOS内のIRQルーチンを使用しなければいけません。

[$34] ラベル名：EX_DMAMOD アドレス：$E09C

機能　VRAM～SATB間、VRAM～VRAM間のDMA転送の動作モードのセット
IN　　AREG：DMAモード
　　bit0　VRAM～SATB間転送終了割り込み　(0:off 1:on)
　　bit1　VRAM～VRAM間転送終了割り込み　(0:off 1:on)
　　bit2　ソースアドレス自動INC/DEC　(0:INC 1:DEC)
　　bit3　ディスティネーションアドレス自動 INC/DEC
　　(0:INC 1:DEC)
　　bit4　VRAM～STAB間転送の自動繰り返し (0:off 1:on)
OUT　なし

【解説】
呼び出した後、最初のVSYNC割り込み時に有効になります。
また、BIOS内のIRQルーチンを使用しなければいけません。

[$35] ラベル名：EX_SPRDMA アドレス：$E09F

機能　VRAM～STAB間のDMA転送をおこなう
IN　　$2214　SAT_ADR:　VRAMのSATアドレスLOW
　　$2215　SAT_ADR+1:　VRAMのSATアドレスHIGH
OUT　なし

【解説】
VRAMの転送開始アドレスをセットして、VRAM～SATB間DMA転送をおこないます。
BIOS内のIRQルーチンを使用しなければいけません。

[$36] ラベル名：EX_SATCLR アドレス：$E0A2

機能　VRAMのSAT（スプライトアトリビュートテーブル）を初期化する
IN　　$2214　SAT_ADR:　VRAMのSATアドレスLOW
　　$2215　SAT_ADR+1:　VRAMのSATアドレスHIGH
OUT　$2216　SPRPTR:　0をセット

【解説】
あらかじめ、VRAMのアドレスインクリメント幅を、ファンクション[$28]のEX_HMODEによって、1にセットしておいてください。

[$37] ラベル名：EX_SPRPUT アドレス：$E0A5

機能　VRAMのSAT（スプライトアトリビュートテーブル）にデータを設定する

IN	$2216	SPRPTR:	定義番号
	$2217	SPRYL:	Y座標 LOW
	$2218	SPRYH:	Y座標 HIGH
	$2219	SPRXL:	X座標 LOW
	$221A	SPRXH:	X座標 HIGH
	$221B	SPRNL:	パターン番号 LOW
	$221C	SPRNH:	パターン番号 HIGH
	$2217	SPRAL:	アトリビュート、色 LOW
	$2218	SPRAH:	アトリビュート、色 HIGH
	$2214	SAT_ADR:	VRAMのSATアドレス LOW
	$2215	SAT_ADR+1:	VRAMのSATアドレス HIGH
OUT	$2216	SPRPTR:	呼び出し時の値に1加算される

【解説】

VRAMに設置されたSAT（スプライトアトリビュートテーブル）の指定の領域に、スプライトのアトリビュート情報を設定します。

あらかじめ、VRAMのアドレスインクリメント幅を、ファンクション[$28]のEX_HMODEによって、1にセットしておいてください。

[$38] ラベル名：EX_SETRCR アドレス：$E0A8

機能　ラスタ検出レジスクに値をセットする

IN	AREG:	走査線番号 LOW
	XREG:	走査線番号 HIGH
OUT	なし	

【解説】

VDCのラスタ検出レジスタに値がセットされます。

実際の画面表示と設定値との関係は、最上位の走査線番号を65として、順に1ずつ加算して設定します。RCRの割り込みが許可されていれば、走査線番号と設定値がおなじになったときに割り込みが発生します。

[$39] ラベル名：EX_SETRED アドレス：$E0AB

機能　VRAMからのデータ読み込みモードにセットする

IN	AREG:	VRAMの読み込み開始アドレス LOW
	XREG:	VRAMの読み込み開始アドレス HIGH

OUT なし

【解説】
VDCのメモリアドレスリードレジスタに値がセットされます。
読み出されたデータは、VDCのレジスタR02（VRR）にセットされます。
VDCの内部動作に詳しくない場合は、でべろシステムBIOSのdv_VRam2Ram（$4015:VRAMからRAMへのブロック転送）を使用してください。

[$3A] ラベル名：EX_SETWRT アドレス：$E0AE

機能 VRAMへのデータ書き込みモードにセットする
IN AREG: VRAMの書き込み開始アドレスLOW
XREG: VRAMの書き込み開始アドレスHIGH
OUT なし

【解説】
VDCのメモリアドレスライトレジスタに値がセットされます。
書き込むデータは、VDCのレジスタR02（VWR）にセットします。
VDCの内部動作に詳しくない場合は、でべろシステムBIOSのdv_Ram2VRam（$4012:RAMからVRAMへのブロック転送）などを使用してください。

[$3B] ラベル名：EX_SETDMA アドレス：$E0B1

機能 VRAM～VRAM間DMA転送に必要なデータを設定する
IN _BX: VRAMの転送元のアドレス
_DX: VRAMの転送先のアドレス
_CX: 転送する領域の大きさ
OUT なし

【解説】
各設定値は、VDCの適切なレジスタに設定されます。
転送する領域の大きさは、最大で64Kワードになります。

[$3C] ラベル名：EX_BINBCD アドレス：$E0B4

機能　バイナリー値をBCD値に変換する
IN　AREG:　バイナリー値
OUT　AREG:　BCD値
　　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生

【解説】
変換できるバイナリー値は、$00～$63の値です。

[$3D] ラベル名：EX_BCDBIN アドレス：$E0B7

機能　BCD値をバイナリー値に変換する
IN　AREG:　BCD値
OUT　AREG:　バイナリー値
　　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生

【解説】
変換できるBCD値は、0～99の値です。

[$3E] ラベル名：EX_RND アドレス：$E0BA

機能　乱数を得る
IN　なし
OUT　AREG:　乱数

【解説】
このBIOSを利用して乱数を得るためには、以下のワークの値を、非同期で変化させる必要があります。

$2249　RNDSEED:　BIOS内IRQルーチンを使用している場合にはVSYNC割り込みごとにインクリメントされます。1バイトのワークです。

$224B　RNDM:　必ずユーザーが変化させる必要があります。

[$4C] ラベル名：EX_COLORCMD アドレス：$E0E4

機能　カラーパレットのセット、読み込みをおこなう
IN　$221F　COLOR_CMD:　機能
　　$00 なにもしない
　　$01 カラーパレットを読み出す
　　$02 カラーパレットをセットする
　$2220　BGC_PTR:　バックグラウンドカラーデータバッファアドレス
　$2222　BGC_LEN:　バックグラウンドカラーパレット数
　$2223　SPRC_PTR:　スプライトカラーデータバッファアドレス
　$2225　SPRC_LEN:　スプライトカラーパレット数
OUT　$221F　COLOR_CMD:　$00

【解説】
このBIOSを実行後、最初のVSYNC割り込み時に有効になります。
BIOS内IRQルーチンを使用している場合で、このBIOSを使用せずにカラーパレットをアクセスする場合には、COLOR_CMD以外のワークに適切な値を設定した後、COLOR_CMDにコマンドを設定すれば動作します。
でべろシステムBIOSにも、dv_Ram2Plt（$401E:RAMからパレットへのブロック転送）や、dv_Plt2Ram（$401E:パレットからRAMへのブロック転送）があります。

[$4A] ラベル名：EX_MEMOPEN アドレス：$E0DE

機能　拡張RAMの使用を開始する
IN　なし
OUT　CARRY:　拡張RAMの有無
　　0　拡張RAMが存在する
　　1　拡張RAMは存在しない
　AREG:　拡張RAMのベースバンク番号
　XREG:　拡張RAMのサイズと種類
　　bit7　0：カード内蔵拡張RAM
　　1：セット内蔵拡張RAM
　　bit6～0　拡張RAMの大きさ（64Kバイト単位）

【解説】
拡張RAMを使用する場合、必ずこのBIOSによって得られたAレジスタの値をもとに、バンクをオフセット指定して使用してください。
なお、このBIOSは、ver.3.00のCD-ROM² BIOSに限り有効です。

[$3F] ラベル名：MA_MUL8U アドレス：$E0BD

機能　符号なし8ビット乗算をおこなう
IN　_AL:　被乗数（8ビット）
　　_BL:　乗数（8ビット）
OUT　_CX:　演算結果（16ビット）

【解説】
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$40] ラベル名：MA_MUL8S アドレス：$E0C0

機能　符号つき8ビット乗算をおこなう
IN　_AL:　被乗数（8ビット）
　　_BL:　乗数（8ビット）
OUT　_CX:　演算結果（16ビット）

【解説】
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$41] ラベル名：MA_MUL16U アドレス：$E0C3

機能　符号なし16ビット乗算をおこなう
IN　_AX:　被乗数（16ビット）
　　_BX:　乗数（16ビット）
OUT　_CX:　演算結果（下位16ビット）
　　_DX:　演算結果（上位16ビット）

【解説】
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$42] ラベル名：MA_DIV16U アドレス：$E0C9

機能　符号なし16ビット除算をおこなう
IN　_AX:　被除数（16ビット）
　　_BX:　除数（16ビット）
OUT　_CX:　商（16ビット）
　　_DX:　剰余（16ビット）

【解説】
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。
またver.1.00では、_AXに設定する値のbit15が1のとき、正常な演算結果が得られません。ver.2.00以降では、正常な演算結果が得られます。

[$43] ラベル名：MA_DIV16S アドレス：$E0C6

機能　符号つき16ビット除算をおこなう
IN　_AX:　被除数（16ビット）
　　_BX:　除数（16ビット）
OUT　_CX:　商（16ビット）
　　_DX:　剰余（16ビット）

【解説】
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$44] ラベル名：MA_SQRT アドレス：$E0CC

機能　平方根を求める
IN　_AX:　基数（16ビット）
OUT　_CL:　平方根（8ビット）

【解説】
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$45] ラベル名：MA_SIN アドレス：$E0CF

機能　三角関数のSIN値を求める
IN　　AREG:　角度(0度～90度)
OUT　CARRY:　実行結果
　　0　AREG
　　1　RESULT＝256

【解説】
結果として得られるAレジスタの値は、256を分母とした分子の値です。また、キャリーフラグがセットされている場合は、分子は256となります。
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$46] ラベル名：MA_COS アドレス：$E0D2

機能　三角関数のCOS値を求める
IN　　AREG:　角度(0度～90度)
OUT　CARRY:　実行結果
　　0　AREG
　　1　RESULT＝256

【解説】
結果として得られるAレジスタの値は、256を分母とした分子の値です。また、キャリーフラグがセットされている場合は、分子は256となります。
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$47] ラベル名：MA_ATNI アドレス：$E0D5

機能　三角関数のATN値を求める
IN　　AREG:　Y/X＊64
OUT　AREG:　角度（0～45度）

【解説】
このBIOSを使用する前に、あらかじめAレジスタに以下の値を設定しておきます。
　　Aレジスタ＝Y÷X×64
このBIOSを実行する際、一時的にメインRAMの$C000～$DFFFが演算パッケージと切り換わります。したがって、この領域に割り込み処理を置かないでください。

[$48] ラベル名：PSG_BIOS アドレス：$E0D8

機能　PSGドライバーを使用する
IN　　_DH:　ファンクション番号 ($00～$14)
OUT　なし

【解説】
_DHにPSGドライバーのファンクション番号を設定して、ファンクションごとに定められた適切な設定をおこなった後、このBIOSを呼び出すことで、PSGドライバーをコントロールします。
PSGドライバーの各ファンクションについては、別項「PSGドライバー」を参照してください。

[$49] ラベル名：GRP_BIOS アドレス：$E0DB

機能　グラフィックドライバーを使用する
IN　　_DH:　ファンクション番号 ($00～$14)
OUT　なし

【解説】
_DHにグラフィックドライバーのファンクション番号を設定して、ファンクションごとに定められた適切な設定をおこなった後、このBIOSを呼び出すことで、グラフィックドライバーをコントロールします。
グラフィックドライバーの各ファンクションについては、別項「グラフィックドライバー」を参照してください。

ラベル名：dv_System アドレス：$4000

機能　でべろシステムをリセットする（画面初期化を含む）
IN　なし
OUT　なし

【解説】
スタック、割り込み、サウンド、画面、パレット、バンクを初期化し、でべろの通信待ち状態に戻ります。
このBIOSを呼び出すと、もとの処理には戻らないので注意が必要です。

ラベル名：dv_Standby アドレス：$4003

機能　でべろシステムをリセットする（画面初期化を含まない）
IN　なし
OUT　なし

【解説】
スタック、割り込み、サウンド、バンクを初期化し、でべろの通信待ち状態に戻ります。
このBIOSを呼び出すと、もとの処理には戻らないので注意が必要です。

ラベル名：dv_GetVdp アドレス：$4006

解説　VDCのレジスタから値を読み出す
IN　AREG:　VDCレジスタ番号
OUT　XREG:　読み出した値（下位8ビット）
　　YREG:　読み出した値（上位8ビット）

【解説】
HuC6270のレジスタの値を読み出します。
VDCの内部動作に詳しくない場合は、不用意に使用しないでください。

ラベル名：dv_SetVdp アドレス：$4009

機能　VDCのレジスタに値をセットする
IN　AREG:　VDCレジスタ番号
　　XREG:　設定する値（下位8ビット）
　　YREG:　設定する値（上位8ビット）

OUT なし

【解説】
HuC6270のレジスタに値を書き込みます。
VDCの内部動作に詳しくない場合は、不用意に使用しないでください。

ラベル名：dv_Vpoke アドレス：$400C

機能 VRAMに1ワードのデータを書き込む
IN _BX: 書き込むVRAMのアドレス（$0000～$7FFF）
_AX: 書き込むデータ（ワードデータ）
OUT なし

【解説】
VRAMのアドレスに指定できるのは、$0000～$7FFFまでです。

ラベル名：dv_Vpeek アドレス：$400F

機能 VRAMから1ワードのデータを読み込む
IN _BX: 読み込むVRAMのアドレス（$0000～$7FFF）
OUT _AX: 読み込んだデータ（ワードデータ）

【解説】
VRAMのアドレスに指定できるのは、$0000～$7FFFまでです。

ラベル名：dv_Ram2VRam アドレス：$4012

機能 RAMからVRAMへのブロック転送
IN _AX: RAMのアドレス（転送元）
_BX: VRAMのアドレス（転送先）
_CX: 転送するデータ数（ワード単位）
OUT なし

【解説】
メインRAMの指定アドレスから、VRAMの指定アドレスへブロック転送をおこないます。
データの転送はワード単位となるため、転送元のRAMのデータは、下位1バイト、上位1バイトで1ワードとなることに注意してください。

ラベル名：dv_VRam2Ram アドレス：$4015

機能　VRAMからRAMへのブロック転送
IN　_AX:　RAMのアドレス（転送先）
　　_BX:　VRAMのアドレス（転送元）
　　_CX:　転送するデータ数（ワード単位）
OUT　なし

【解説】
VRAMの指定アドレスから、メインRAMの指定アドレスへブロック転送をおこないます。
データの転送はワード単位となるため、転送先のRAMには、下位1バイト、上位1バイトで1ワードのデータが格納されることに注意してください。

ラベル名：dv_FillVram アドレス：$4018

機能　VRAMを単一のワードデータで埋める
IN　_AX:　書き込むデータ（ワードデータ）
　　_BX:　VRAMの書き込み開始アドレス
　　_CX:　書き込む領域の大きさ（ワード単位）
OUT　なし

【解説】
VRAMの、_BXで指定されたアドレスから、_CXで指定されたワード数ぶんを、_AXのデータで埋めます。

ラベル名：dv_Ram2Plt アドレス：$401B

機能　RAMからカラーパレットへのブロック転送
IN　_AX:　RAMのアドレス（転送元）
　　_BX:　カラーパレットのアドレス（転送先）
　　_CX:　転送するデータ数（ワード単位）
OUT　なし

【解説】
メインRAMの指定アドレスから、カラーパレットの指定アドレスへブロック転送をおこないます。
データの転送はワード単位となるため、転送元のRAMのデータは、下位1バイト、上位1バイトで1ワードとなることに注意してください。
カラーパレットは512ワードの領域しかないため、アドレスの指定には下位9ビット（$0000～$01FF）が有効です。
・カラーパレットについて

カラーパレットのアドレスによって、以下の意味があります。

bit8　0:バックグラウンド用
　　1:スプライト用
bit4～7　パレットセットの番号
bit0～3　そのパレットセットのカラー番号

カラーパレットは、1アドレスが9ビットのメモリで構成されます。したがって、転送するデータは、下位9ビットが有効となります。

bit8～6　緑の輝度（0～7の8段階）
bit5～3　赤の輝度（0～7の8段階）
bit2～0　青の輝度（0～7の8段階）

ラベル名：dv_Plt2Ram アドレス：$401E

機能　カラーパレットからRAMへのブロック転送
IN　_AX:　RAMのアドレス（転送先）
　　_BX:　カラーパレットのアドレス（転送元）
　　_CX:　転送するデータ数（ワード単位）
OUT　なし

【解説】
カラーパレットの指定アドレスから、メインRAMの指定アドレスへブロック転送をおこないます。
データの転送はワード単位となるため、RAMに転送されるデータは、下位1バイト、上位1バイトで1ワードとなることに注意してください。
カラーパレットは512ワードの領域しかないため、アドレスの指定には下位9ビット（$0000～$01FF）が有効です。

ラベル名：dv_Screen1 アドレス：$4021

機能　テキストモード画面の操作
IN　_DH:　コマンド番号
　　0　初期化
　　1　画面消去
　　2　座標設定。さらに以下の設定が必要
　　_AL　X座標（キャラクタ単位）
　　_AH　Y座標（キャラクタ単位）
　　3　1文字表示。さらに以下の設定が必要
　　_AL　文字コード
　　4　文字列表示。さらに以下の設定が必要
　　_BX　文字列の先頭アドレス
OUT　なし

【解説】
でべろシステムの標準画面を扱うBIOSです。
標準画面は32文字×24行の画面構成で、8×8ドットの内蔵フォントで表示します。
このBIOSを使って表示される文字は、すべてモノクロ表示となります。
・文字データで使用できる制御コードについて
表示する文字データには以下の制御コードが使用できます。

$0D	CR（キャリッジリターン）コード
$0A	LF（ラインフィード）コード
$0C	CLS（画面消去）
$1C	座標を1文字ぶん右に移動
$1D	座標を1文字ぶん左に移動
$1E	座標を1行ぶん上に移動
$1F	座標を1行ぶん下に移動
$00	文字列データの終りを示す

ラベル名：dv_Send1Byte アドレス：$402D

機能　PCエンジンと通信状態にあるパソコンへ1バイト送信する
IN　AREG:　送信するデータ
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生

【解説】
でべろシステムによって通信状態にあるパソコンに、1バイトのデータを送信します。
受信するパソコン側では、専用の通信プログラムが起動されている必要があります。
・重要注意
スターターキットに含まれる外部コマンドのPERUNやSLAVEでは受信できません。添付のライブラリを利用して、専用の通信プログラムを自作する必要があります。

ラベル名：dv_SendBlock アドレス：$4030

機能　PCエンジンと通信状態にあるパソコンへブロック送信する
IN　$20C8　XferPtr:　送信するデータの先頭アドレス
　　$20CA　XferLen:　送信するデータの数（バイト単位）
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生

【解説】
でべろシステムによって通信状態にあるパソコンに、データをブロック送信します。
受信するパソコン側では、専用の通信プログラムが起動されている必要があります。
・重要注意
スターターキットに含まれる外部コマンドのPERUNやSLAVEでは受信できません。添付のライブラリを利用して、専用の通信プログラムを自作する必要があります。
また、送信側と受信側のどちらにも、おなじ値の転送長を指定する必要があります。

ラベル名：dv_Recv1Byte アドレス：$4033

機能　PCエンジンと通信状態にあるパソコンから1バイト受信する
IN　なし
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生
　　AREG:　受信したデータ

【解説】
でべろシステムによって通信状態にあるパソコンから、1バイトのデータを受信します。
送信するパソコン側では、専用の通信プログラムが起動されている必要があります。
・重要注意
スターターキットに含まれる外部コマンドのPERUNやSLAVEでは送信できません。添付のライブラリを利用して、専用の通信プログラムを自作する必要があります。
送信側のプログラムでは、このBIOSが呼び出されるまでに、1バイトのデータを送信する準備が整っている必要があります。

ラベル名：dv_Recv1ByteW アドレス：$4036

機能　PCエンジンと通信状態にあるパソコンから1バイト受信する（受信待ちあり）
IN　なし
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生
　　AREG:　受信したデータ

【解説】
でべろシステムによって通信状態にあるパソコンから、1バイトのデータを受信します。
送信するパソコン側では、専用の通信プログラムが起動されている必要があります。
・重要注意
スターターキットに含まれる外部コマンドのPERUNやSLAVEでは送信できません。添付のライブラリを利用して、専用の通信プログラムを自作する必要があります。

ラベル名：dv_RecvBlock アドレス：$4039

機能　PCエンジンと通信状態にあるパソコンからブロック受信する
IN　$20C8　XferPtr:　受信したデータを格納するアドレス
　　$20CA　XferLen:　受信するデータの数（バイト単位）
OUT　CARRY: 実行結果
　　0　正常終了
　　1　エラー発生

【解説】
でべろシステムによって通信状態にあるパソコンから、データをブロック受信します。
送信するパソコン側では、専用の通信プログラムが起動されている必要があります。
・重要注意
スターターキットに含まれる外部コマンドのPERUNやSLAVEでは送信できません。添付のライブラリを利用して、専用の通信プログラムを自作する必要があります。
また、送信側と受信側のどちらにも、おなじ値の転送長を指定する必要があります。

ラベル名：dv_FileOpen アドレス：$403C

機能　ファイルをオープンする
IN　$27E0　FileName:　オープンするファイル名
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生
　　_BX:　ファイル情報のある先頭アドレス

【解説】
でべろのシステムによって管理されているCD-ROMの中のファイルを検索して、そのファイルの識別コード、ファイル長などの情報を得ます。
外部コマンドのPERUNやSLAVEが起動されていて、PCエンジンとパソコンとが通信状態にある場合、パソコン側のファイルも検索できます。
・得られるファイルの情報
_BXで示されるアドレスには、以下の情報が置かれます。
+$00〜+$07　ファイル名。8文字未満の場合は、$20の空白で補われる
+$08〜+$0A　拡張子。3文字未満の場合は、$20の空白で補われる
+$0B　ファイル種別
　　$01：通常ファイル
　　$80：CDDAファイル
　　$00：ディレクトリ終端
+$0C〜+$0F ファイル作成日時
+$10〜+$12 ファイル識別コード。上位1バイト（+$12のデータ）が$FFであれば、通信経由のファイルであることを示す
+$13〜+$16 ファイルサイズ
+$17〜+$1F 未使用

ラベル名：dv_FileOpen2 アドレス：$403F

機能　ファイルをオープンする
IN　$27E0　FileName:　オープンするファイル名
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生
　　_BX:　ファイル情報のある先頭アドレス

【解説】
外部コマンドのPERUNやSLAVEが起動されていて、PCエンジンとパソコンとが通信状態にある場合に、パソコン側のファイルを検索して、そのファイルの識別コード、ファイル長などの情報を得ます。CD-ROM内は検索しません。
通信状態にない場合には、ファイルのオープンはできません。
得られるファイルの情報は、$403Cのdv_FileOpenとおなじです。

ラベル名：dv_FileRead アドレス：$4042

機能　オープンしたファイルからデータを読み出す
IN　$27EC　FileTag:　ファイル識別コード
　　$27EF　FileLoc:　読み出し開始位置
　　$27F3　FileLen:　読み出すデータ数（バイト単位）
　　$27F5　FileAdd:　データを格納する領域の先頭アドレス
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生

【解説】
dv_FileOpenまたはdv_FileOpen2でオープンされたファイルから、指定したアドレスへデータを読み込みます。
FileTagのワークに設定する値は、ファイルオープン時に得られるファイル識別コードの3バイトのデータです。
FileLocには、ファイルの先頭から数えて何バイト目からを読み込むかを設定します。4バイトのワークです。
FileLenには読み込むデータのバイト数を設定し、FileAddには読み込んだデータを格納するRAMの先頭アドレスを設定します。ともに2バイトのワークです。

ラベル名：dv_FileWrite アドレス：$4045

機能　オープンしたファイルにデータを書き込む
IN　$27EC　FileTag:　ファイル識別コード
　　$27EF　FileLoc:　書き込み開始位置
　　$27F3　FileLen:　書き込むデータ数（バイト単位）
　　$27F5　FileAdd:　データが格納されている領域の先頭アドレス
OUT　CARRY:　実行結果
　　0　正常終了
　　1　エラー発生

【解説】
dv_FileOpenまたはdv_FileOpen2でオープンしたファイルに、指定したアドレスのデータを書き込みます。
FileTagのワークに設定する値は、ファイルオープン時に得られるファイル識別コードの3バイトのデータです。
FileLocには、ファイルの先頭から数えて何バイト目からに書き込むかを設定します。4バイトのワークです。
FileLenには書き込むデータのバイト数を設定し、FileAddには書き込むデータが格納されているRAM領域の先頭アドレスを設定します。ともに2バイトのワークです。
なお、CD-ROMに対しては無効です。

ラベル名：dv_FileDelete アドレス：$4048

機能　ファイルを削除する
IN　$27E0　FileName:　　ファイル名
OUT　なし

【解説】
外部コマンドのPERUNやSLAVEが起動されていて、PCエンジンとパソコンとが通信状態にある場合に、パソコン側の該当するファイルを削除します。
なお、CD-ROMのファイルは削除できません。

ラベル名：dv_GetDirFirst アドレス：$404B

機能　ファイルの検索（1）
IN　なし
OUT　$2800～ Buffer:　ディレクトリエントリ情報

【解説】
このBIOSを実行すると、CD-ROMのディレクトリの先頭からディレクトリエントリの情報をBufferに格納します。また、PERUNやSLAVEによってPCエンジンとパソコンが通信状態にある場合は、CD-ROMの情報に続けてパソコン側の情報も格納されます。
獲得できる情報は、CD-ROM、パソコンを合わせて最初の64ファイルぶんです。情報の内容は、dv_FileOpenで示したものとおなじフォーマットになっています。
65ファイル目からの情報を得る場合には、$404Eのdv_GetDirNextを使用します。

ラベル名：dv_GetDirNext アドレス：$404E

機能　ファイルの検索（2）
IN　なし
OUT　$2800～ Buffer:　ディレクトリエントリ情報

【解説】
このBIOSを実行する前に、$404Bのdv_GetDirFirstを必ず実行する必要があります。
CD-ROMおよびPERUNやSLAVEによって通信状態にあるパソコンから、ディレクトリエントリの情報をBufferに格納します。
獲得できる情報は、前回得た情報の続きの64ファイルぶんです。
情報の内容は、dv_FileOpenで示したものとおなじフォーマットになっています。
さらに続きの情報を得る場合には、このBIOSを使用します。

ラベル名：dv_RegReport アドレス：$4051

機能　レジスタの内容をパソコンに出力する
IN　なし
OUT　なし

【解説】
プログラム作成中のデバッグなどで使用します。
PERUNやSLAVEで、PCエンジンパソコンが通信状態にあれば、パソコンの画面に以下の情報が表示されます。

Aレジスタ
Xレジスタ
Yレジスタ
スタックポインタ
ステータスレジスタ中の各フラグ
現在実行中のアドレス

ただし、このBIOSを呼び出した時点での情報に限ります。

ラベル名：dv_XferSync アドレス：$4054

機能　通信中のパソコンとの同期チェック
IN　なし
OUT　なし

【解説】
でべろでは、ジョイパッドの端子を使って通信をおこなっています。そのため、ゲームなどでパッドの読み込みをおこなうと、パソコンとの通信の同期が崩れることがあります。通信処理でエラーが頻発するときは、このBIOSを呼び出して通信の同期を正します。
おもに以下のBIOSでの通信エラーが対象となります。

$403C	dv_FileOpen	ファイルオープン
$403F	dv_FileOpen2	ファイルオープン
$4042	dv_FileRead	ファイル読出
$4045	dv_FileWrite	ファイル書込
$4048	dv_FileDelete	ファイル削除
$404B	dv_GetDirFirst	ファイルの検索（1）
$404E	dv_GetDirNext	ファイルの検索（2）
$4051	dv_RegReport	レジスタ内容報告

ラベル名：dv_SlaveQuit アドレス：$4057

機能　パソコン側の通信状態を解除する
IN　なし
OUT　なし

【解説】
PERUNやSLAVEによって、PCエンジンと通信状態にあるパソコンの、通信状態を解除します。

ラベル名：dv_AutoRun アドレス：$407D

機能　でべろシステムをリセットしてメニュープログラムを起動する
IN　なし
OUT　なし

【解説】
スタック、割り込み、サウンド、画面、パレット、バンクを初期化して、メニュープログラムを起動します。
このBIOSを呼び出すと、もとの処理には戻りません。

PSGドライバー

[$48] ラベル名：psg_bios アドレス：$E0D8

機能　PSGドライバーを使用します。
IN　_DH:　ファンクション番号

```
PSG_ON      EQU  $00  PSGドライバーON
PSG_OFF     EQU  $01  PSGドライバーOFF
PSG_INIT    EQU  $02  PSGドライバー初期化
PSG_BANK    EQU  $03  サウンドデータバンク番号登録
PSG_TRACK   EQU  $04  トラックデータインデックスアドレス登録
PSG_WAVE    EQU  $05  波形データアドレス登録
PSG_ENV     EQU  $06  エンベロープデータアドレス登録
PSG_FM      EQU  $07  周波数変調データアドレス登録
PSG_PE      EQU  $08  ピッチエンベロープアドレス登録
PSG_PC      EQU  $09  パーカッションデータアドレス登録
PSG_TEMPO   EQU  $0A  テンポセット
PSG_PLAY    EQU  $0B  トラックデータ演奏
PSG_MSTAT   EQU  $0C  メイントラックチェック
PSG_SSTAT   EQU  $0D  サブトラックチェック
PSG_MSTOP   EQU  $0E  メイントラック演奏停止
PSG_SSTOP   EQU  $0F  サブトラック演奏停止
PSG_ASTOP   EQU  $10  全トラック演奏停止
PSG_MVOFF   EQU  $11  メイントラックボリュームカット
PSG_CONT    EQU  $12  演奏再開
PSG_FDOUT   EQU  $13  フェードアウト
PSG_DCNT    EQU  $14  メイントラックディレイカウンターセット
```

OUT　なし

【解説】

_DHにファンクション番号を入れ、それぞれのファンクション番号に対応したパラメータをレジスタに入れて（レジスタを使用しないファンクションもあります）、psg_biosをコールすることによって、PSGドライバーをコントロールします。

概要

PCエンジンに搭載されているPSGドライバーは内部に仮想12チャンネルを持ち、ミュージックに6チャンネル（メイントラック）、エフェクトに6チャンネル（サブトラック）ずつ独立して管理しています。

それぞれのチャンネル（トラック）は、必要に応じて切り離すことが可能で、切り離したチャンネルのワークエリアは解放されます。

また、PSGドライバーのコールは、タイマーでおこなうか、IRQでおこなうかが自由に選択できます。

PSGドライバーがコールされた場合、サウンドデータの格納されたバンクが、論理アドレス空間（アドレスで$8000から2バンク分）に展開されるようになっています。そのバンク番号はユーザーが指定することができます。

●ファンクションコール

PSGドライバーには、21個のファンクションコールが用意されており、自由にコントロールすることが可能です。

ファンクションをコールすると、ACC（アキュムレータ）の内容は破壊されます。また、XREG（Xレジスタ）の内容も破壊されます。

●ミュージックトラックデータ

PSGドライバーでは、演奏するデータをミュージックトラックデータとして扱います。ミュージックトラックデータはトラックデータの登録部やチャンネルごとのデータ部からなり、データ部は音程データやデータコマンドでできています。データコマンドはエンベロープやオクターブ、ボリュームなど42個用意されています。

●ユーザー定義データについて

PSGドライバーには、波形データが45個内蔵されていますが、さらにユーザーが83個まで定義することが可能です（計128個）。

エンベロープデータとしては、内蔵16種類、ユーザー定義112種類（計128種類）が使用可能になっています。この他にユーザーが定義可能なものとして、パーカッションデータ12個、テーブルサイズ256バイトまでの周波数変調データ、ピッチエンベロープデータなどがあります。

ユーザー定義データを使用するには、データをメモリ内に配置して、その先頭アドレスを各ファンクションコールを使って登録します。

●ファンクションコール

PSG_ON

ファンクション番号 _DH:　$00

機能　PSGドライバーオン
IN　_AL:　PSGドライバーのコール
　　0　タイマーコール
　　1　IRQコール
OUT　なし

【解説】
PSGドライバーを使用可能にします。
_ALの値で、PSGドライバーをタイマーによってコールするか、IRQによってコールするかを指定します。タイマーコールを選択した場合、PSG_ONをコールすることによって、タイマーがONになります。

PSG_OFF

ファンクション番号 _DH:　$01

機能　PSGドライバーオフ
IN　なし
OUT　なし

【解説】
PSGドライバーの使用を停止します。
タイマーコールを選択していた場合、タイマーの動作を停止します。

PSG_INIT

ファンクション番号 _DH:　$02

機能　PSGドライバーの初期設定をおこないます
IN　_AL:　PSGドライバーシステム番号
　　0:　メイントラックのみ使用可能とします
　　サブトラック側のワークは解放されます
　　サブトラックに関する機能は停止されます
　　タイマーは1/60秒のインターバルで使用されます
　　1:　サブトラックのみ使用可能とします

メイントラック側のワークは解放されます
メイントラックに関する機能は停止されます
タイマーは1/60秒のインターバルで使用されます
2: メイントラックとサブトラックを使用します
タイマーは1/60秒のインターバルで使用されます
3: メイントラックとサブトラックを使用します
タイマーは1/120秒のインターバルで使用されます
4: メイントラックとサブトラックを使用します
タイマーは1/240秒のインターバルで使用されます
5: メイントラックとサブトラックを使用します
タイマーは1/300秒のインターバルで使用されます

OUT なし

【解説】
PSG内部レジスタの初期化、ワークの初期化、システム番号による各種設定をおこないます。
・注意
システム番号によるタイマーインターバルの影響を受けるのは、サブトラックのみです。
メイントラックは、ディレイカウンターによって1/60秒のタイマーインターバルが維持されます。IRQコールの場合、タイマーインターバルの設定は無視されます。

PSG_BANK

ファンクション番号 _DH: $03

機能 サウンドデータのバンク番号を登録します
IN _AL: バンクNo.0
_AH: バンクNo.1
OUT なし

【解説】
サウンドデータの格納されているバンクRAMの番号を指定します。
割り込みが発生すると、マッピングレジスタMPR4に_ALの値が、MPR5に_AHの値がセットされ、論理アドレス空間にサウンドデータが展開されます。

PSG_TRACK

ファンクション番号 _DH:　　$04

機能　トラックデータインデックステーブルのアドレスを登録します。
IN　　_AX:　トラックデータ先頭インデックステーブルの先頭アドレス
OUT　なし

【解説】
トラックデータインデックステーブルの先頭アドレスを登録します。

PSG_WAVE

ファンクション番号 _DH:　　$05

機能　波形データアドレスを登録します
IN　　_AX:　波形データ先頭アドレス
OUT　なし

【解説】
ユーザーが定義した波形データの先頭アドレスを設定します。

PSG_ENV

ファンクション番号 _DH:　　$06

機能　エンベロープデータインデックステーブルのアドレスを登録します
IN　　_AX:　エンベロープデータインデックステーブルの先頭アドレス
OUT　なし

【解説】
ユーザーが定義したエンベロープデータインデックステーブルの先頭アドレスを登録します。

PSG_FM

ファンクション番号 _DH:　　$07

機能　周波数変調データインデックステーブルのアドレスを登録します
IN　　_AX:　周波数変調データインデックステーブルの先頭アドレス
OUT　なし

【解説】
ユーザーが定義した周波数変調データインデックステーブルの先頭アドレスを登録します

PSG_PE

ファンクション番号 _DH:　　$08

機能　ピッチエンベロープデータインデックステーブルのアドレスを登録します
IN　　_AX:　ピッチエンベロープデータインデックステーブルの先頭アドレス
OUT　なし

【解説】
ユーザーが定義したピッチエンベロープデータインデックステーブルの先頭アドレスを登録します。

PSG_PC

ファンクション番号 _DH:　　$09

機能　パーカッションデータインデックステーブルのアドレスを登録します
IN　　_AX:　パーカッションデータインデックステーブルの先頭アドレス
OUT　なし

【解説】
ユーザーが定義したパーカッションデータインデックステーブルの先頭アドレスを登録します。

PSG_TEMPO

ファンクション番号 _DH:　　$0A

機能　テンポをセットします
IN　　_AL:　テンポ数 (35～255)
OUT　なし

【解説】
タイマのインターラプトを変化させて、演奏のテンポを変化させます。テンポ数に35以下をセットした場合、35をセットしたものとして処理されます。サブトラックを使用しているときは、できるだけテンポを変化させないようにしてください。

PSG_PLAY

ファンクション番号_DH:　　$0B

機能　トラックデータを演奏します
IN　　_AL:　サウンド番号（0〜127）
　　_AH:　波形番号（デバッグモード時のみ有効）
OUT　なし

【解説】
指定した番号に登録されているトラックデータを演奏します。デバッグモードでは、_AHの内容が波形番号として登録され、演奏されるトラックすべてが変更の対象となります。
ただし、トラックデータ内に波形番号指定があると、その番号に変更されます。トラックデータの設定されていない番号を指定しないでください。

PSG_MSTAT

ファンクション番号_DH:　　$0C

機能　メイントラックの情報をチェックします
IN　　なし
OUT　ACC:　チェック情報
　　メイントラックが使用されていない（つまりPSG_INITによって、メイントラックの機能がOFFとなっている）場合、ビットパターンは$80になります。使用されている場合は以下のようになります

```
bit76543210
xx??????
  |||||└ メイントラック0（1:演奏中 0:演奏停止）
  ||||└─ メイントラック1（1:演奏中 0:演奏停止）
  |||└── メイントラック2（1:演奏中 0:演奏停止）
  ||└─── メイントラック3（1:演奏中 0:演奏停止）
  |└──── メイントラック4（1:演奏中 0:演奏停止）
  └───── メイントラック5（1:演奏中 0:演奏停止）
```

【解説】
メイントラックを調べて、その演奏状態をビット情報としてリターンします。

PSG_SSTAT

ファンクション番号_DH:　　$0D

機能　サブトラックの情報をチェックします
IN　　なし

OUT　ACC:　チェック情報
サブトラックが使用されていない（つまりPSG_INITによって、サブトラックの機能がOFFとなっている）場合、ビットパターンは$80になります。使用されている場合は以下のようになります。

bit76543210
xx??????
- サブトラック0（1:演奏中 0:演奏停止）
- サブトラック1（1:演奏中 0:演奏停止）
- サブトラック2（1:演奏中 0:演奏停止）
- サブトラック3（1:演奏中 0:演奏停止）
- サブトラック4（1:演奏中 0:演奏停止）
- サブトラック5（1:演奏中 0:演奏停止）

【解説】
サブトラックを調べて、その演奏状態をビット情報としてリターンします。

PSG_MSTOP

ファンクション番号 _DH:　　$0E

機能　メイントラックの演奏を停止します
IN　　_AL:　停止トラックの指定

bit76543210
?x??????
- メイントラック0（1:演奏停止 0:演奏継続）
- メイントラック1（1:演奏停止 0:演奏継続）
- メイントラック2（1:演奏停止 0:演奏継続）
- メイントラック3（1:演奏停止 0:演奏継続）
- メイントラック4（1:演奏停止 0:演奏継続）
- メイントラック5（1:演奏停止 0:演奏継続）
- ポーズスイッチ（1:ポーズON 0:ポーズOFF）

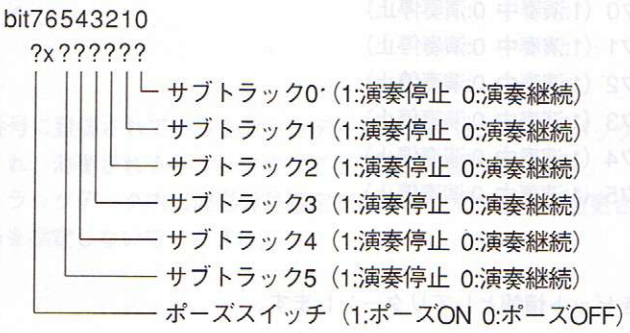
OUT　なし

【解説】
メイントラックの演奏を停止します。下位6ビットによって、演奏停止させるトラックを指定します。ビット7は、ポーズスイッチとなっており、1で全メイントラックの演奏が一時停止されます。
ポーズスイッチがONのときは、ビット0〜5までのトラック指定は無視されます。

PSG_SSTOP

ファンクション番号 _DH:　$0F

機能　サブトラックの演奏を停止します
IN　_AL:　停止トラックの指定



OUT　なし

【解説】
サブトラックの演奏を停止します。下位6ビットによって、演奏停止させるトラックを指定します。ビット7は、ポーズスイッチとなっており、1で全サブトラックの演奏が一時停止されます。
ポーズスイッチがONのときは、ビット0～5までのトラック指定は無視されます。

PSG_ASTOP

ファンクション番号 _DH:　$10

機能　全トラックの演奏を停止します
IN　なし
OUT　なし

【解説】
全トラックの演奏を停止します。

PSG_MVOFF

ファンクション番号 _DH:　$11

機能　メイントラックのボリュームをオフにします
IN　ACC:　ボリュームカットトラックの指定

bit 76543210
xx??????
- メイントラック0（1:ボリュームOFF 0:演奏継続）
- メイントラック1（1:ボリュームOFF 0:演奏継続）
- メイントラック2（1:ボリュームOFF 0:演奏継続）
- メイントラック3（1:ボリュームOFF 0:演奏継続）
- メイントラック4（1:ボリュームOFF 0:演奏継続）
- メイントラック5（1:ボリュームOFF 0:演奏継続）

OUT　なし

【解説】
演奏中のメイントラックのボリュームをカットします。
PSG_CONTによって、ボリュームをONにすることができます。

PSG_CONT

ファンクション番号 _DH:　$12

機能　演奏を再開します
IN　_AL:　再開データ
　0:　メイントラック再開
　1:　サブトラック再開
　2:　全トラック再開
OUT　なし

【解説】
PSG_MSTOP、PSG_SSTOP、PSG_ASTOPでポーズがかけられた演奏、あるいはPSG_MVOFFでボリュームカットされた演奏を再開します。

PSG_FDOUT

ファンクション番号 _DH:　$13

機能　メイントラックのフェードアウトを行います
IN　_AL:　フェードアウトスピード（1〜127）
OUT　なし

【解説】
メイントラックのフェードアウトを行います。
フェードアウトのスピードは、_ALに入れられる値が大きくなるほど速くなります。
指定値が負の場合は、正に補正されます。

PSG_DCNT

ファンクション番号 _DH:　　$14

機能　メイントラックディレイカウンターをセットします
IN　　_AL:　ディレイカウンター値
　　0:　1/1
　　1:　1/2
　　2:　1/3
　　3:　1/4
　　4:　1/5
　　5:　1/6
　　6:　1/7
　　7:　1/8
OUT　なし

【解説】

ディレイカウンター値が大きくなるほど、メイントラックの演奏速度が遅くなります。ディレイカウンター値は、PSG初期設定(PSG_INIT)によって、セットされますので、このファンクションコールは通常使用しないでください。

PSG_TEMPOによって、テンポを変化させる場合は、必ずディレイカウンター値は1/2(_AL=1)にセットします。

トラックデータ

CD-ROM²BIOSの扱えるミュージックデータは、音程と音長がセットになり1音を表現する音データや、オクターブ指定などのミュージックコマンドからなります。これが必要な数だけ連続して1つのトラックデータとなります。

個々のトラックデータは、トラックデータ先頭アドレス登録テーブルにチャンネルごと登録され1曲になります。このトラックデータ先頭アドレス登録テーブルはトラックデータインデックステーブルに登録されて、自由に曲を選べるようになります。

トラックデータの登録

各トラック別に作成したデータの先頭アドレスを以下の書式で登録し、アドレスを割り振ります。

・トラックデータ登録の例

```
TRK_INDEX:          ……トラックデータインデックステーブル
    DW MUSIC0
    DW MUSIC1
      ⋮
```

```
;
MUSIC0:                          ……トラックデータ先頭アドレス登録テーブル
    DB $3F      ;(%0011_1111) ……チャンネル設定スイッチ
    DW PART_1                    ……トラックデータ先頭アドレス
    DW PART_2
    DW PART_3
    DW PART_4
    DW PART_5
    DW PART_6
;
MUSIC1:
    DB $3F      ;(%0011_1111)
    DW PART_1
    DW PART_2
    DW PART_3
      ⋮
      ⋮
```

①トラックデータインデックステーブル
トラックデータ先頭アドレス登録テーブルの先頭アドレスを登録します。
インデックステーブルの先頭アドレス(TRK_INDEX)は、ファンクションコール(PSG_TRACK)によってPSGドライバーに登録します。
この登録によって、PSGドライバーは、ファンクションコール(PSG_PLAY)によって与えられたサウンド番号から、トラックデータ先頭アドレスを検索し、そのトラックデータ先頭アドレスを各チャンネルに割り当てて、演奏を開始します。

②トラックデータ先頭アドレス登録テーブル
各トラックデータをどのチャンネルに割り振るか、メイントラックとして登録するか、またはサブトラックとして登録するかなどを設定するためのテーブルです。
先頭1バイトの、チャンネル割り振りのためのチャンネル設定スイッチと、その後に続くトラックデータ先頭アドレスからなっています。

・チャンネル設定スイッチ

bit7　メイン/サブトラック切り替えスイッチ
　　0: メイントラック 1: サブトラック
bit6　デバッグモード切り替えスイッチ
　　0: ノーマルモード 1: デバッグモード
bit5～0　各チャンネルスイッチ
　　0: OFF　　1: ON
　　bit5:　チャンネル6
　　bit4:　チャンネル5
　　bit3:　チャンネル4
　　bit2:　チャンネル3
　　bit1:　チャンネル2
　　bit0:　チャンネル1

・トラックデータ先頭アドレス登録テーブル例

```
MUSIC1:
    DB   $33          ;$33(2進数で、%0011_0011)
                      ;メイントラック
                      ;ノーマルモード
                      ;チャンネル
                      ;1,2,5,6:ON
                      ;チャンネル
                      ;3,4:OFF
    DW   PART_A       ;チャンネル1
    DW   PART_B       ;チャンネル2
    DW   PART_C       ;チャンネル5
    DW   PART_D       ;チャンネル6
```

トラックデータフォーマット

ミュージックトラックデータのフォーマットは、一部をのぞき、メイントラック、サブトラック共通です。

2バイト(テンポモード／ダイレクトレングスモード) あるいは1バイト(タイムベースレングスモード)の音程データと、2バイトあるいは1バイトのコントロールデータの組合せによって、トラックデータは作成されます。

データ作成の方法には、タイマーインターバルの可変によってテンポを変更するテンポ制と、音長の調整によって演奏速度を設定するレングス制の2種類があります。

ただしテンポ制の場合、テンポを変化させるとサブトラック（エフェクト）の演奏速度も変化してしまいます。サブトラックのデータを作成するときは、テンポの指定機能はないので、レングス制を使用します。

音程

バイト数　2バイト ダイレクトレングスモード
　　1バイト タイムベースレングスモード

書式　音程コード、音長　ダイレクトレングスモード
　　音程コード+音長　タイムベースレングスモード

コード

$10:ド (C)	$70:ファ# (F#)
$20:ド#(C#)	$80:ソ (G)
$30:レ (D)	$90:ソ# (G#)
$40:レ#(D#)	$A0:ラ (A)
$50:ミ (E)	$B0:ラ# (A#)
$60:ファ(F)	$C0:シ (B)
$00:休符	

パラメータ　音長$01～$FF:ダイレクトレングスモード
　　音長$00～$0F:タイムベースレングスモード

機能　音程と音長を指定します。
　　ダイレクトレングスモード、タイムベースレングスモードそれぞれで音長の指定方法が異な

ります。レングスモードの切り換えは、タイムベース値(0:ダイレクトレングス 1～15:タイムベースレングス)によっておこないます。また、レングス制のデータを使用する場合には、テンポ指定をおこなわないでください。

・ダイレクトレングスモード
音長を$01～$FF(1～255)の値で指定します。音長を直接指定したい場合や、テンポ制のデータ作成をおこなう場合に便利なモードです。データフォーマットは2バイトになります。また、音程データの下位4ビットは無視されます。

・タイムベースレングスモード
音長を$01～$0F(0～15)の値で指定します。
指定された値は、ドライバー内で1～16に補正され、実際の音長はタイムベース値×音長で1から240となります。レングス制のデータを作成するときに便利です。データフォーマットは1バイトで上位4ビットが音程コード、下位4ビットが音長データとなります。

・テンポ制データ
テンポ制データでは、音長は192を音符長で割った長さが実際の長さとなります。
たとえば、音長符として4分音符を指定する場合、音長は48となります。
192／4=48
テンポ制のデータを使用する場合、ファンクションコール(PSG_DCNT)によって、ディレイカウンターの値を1/2にセットしておく必要があります。

タイムベース

バイト数	2バイト
書式	タイムベースコード、タイムベース値
コード	$D0
パラメータ	$00～$0F(0～15)：タイムベース値
機能	タイムベース値を設定します。 0：ダイレクトレングスモード 1～15：タイムベースレングスモード デフォルト値は0です。

オクターブ

バイト数	1バイト
書式	オクターブコード
コード	$D1～$D7
パラメータ	なし
機能	オクターブを指定します。 $D1～$D7がオクターブ1～7に対応します。 オクターブ指定を省略した場合、オクターブ4を指定したものとして扱われます。

オクターブアップ

バイト数	1バイト
書式	オクターブアップコード
コード	$D8
パラメータ	なし
機能	オクターブを現在指定されているものより1つ上げます。

オクターブダウン

バイト数	1バイト
書式	オクターブダウンコード
コード	$D9
パラメータ	なし
機能	オクターブを現在指定されているものより1つ下げます。

タイ

バイト数	1バイト
書式	タイコード
コード	$DA
パラメータ	なし
機能	前後の音をつなぎます。

テンポ

バイト数	2バイト
書式	テンポコード、テンポ値

コード　$DB
パラメータ　$23～$FF(35～255): テンポ値
機能　テンポを指定します。
複数のチャンネルに対してテンポ指定が行われている場合、番号の最も若いチャンネルに対する指定が優先されます。

・注意
サブトラックに対して、テンポ指定はできません。

ボリューム

バイト数　2バイト
書式　ボリュームコード、ボリューム値
コード　$DC
パラメータ　$00～$1F(0～31): ボリューム値
機能　ボリュームを指定します。ボリューム値が大きくなるほど、ボリュームも大きくなります。
デフォルト値は$1Fです。

パンポット

バイト数　2バイト
書式　パンポットコード、パンポット(RL)値
コード　$DD
パラメータ　$00～$FF: パンポット値
機能　ステレオの左右のボリューム値を指定します。パンポット値の上位4ビットがレフトチャンネル、下位4ビットがライトチャンネルの値となります。

・注意
デフォルト値は不定ですので、必ずこのコマンドを実行してください。

音長比

バイト数　2バイト
書式　音長比コード、音長比値
コード　$DE
パラメータ　$01～$08(1～8): 音長比値
$01: 1/8
$02: 2/8
⋮
$07: 7/8
$08: 8/8

機能　1音中の発音時間の割合（キーオン、オフの割合）を指定します。
1から8までを音長比として指定することにより、1/8～8/8までの音長比を指定できます。音長が100の音で音長比が4/8なら、100×(4/8)=50だけ発音し、あとの50は休むという意味です。
デフォルト値は8です。

相対ボリューム

バイト数　2バイト
書式　相対ボリュームコード、相対値
コード　$DF
パラメータ　$E1～$1F(-31～31): 相対値
機能　現在のボリューム値からの相対値でボリュームを変更します。
相対値1($01)でボリュームが1上がり、相対値−1($FF)でボリュームが1下がります。

ダルセーニョ

バイト数　1バイト
書式　ダルセーニョコード
コード　$E1
パラメータ　なし
機能　セーニョを指定している位置のデータに戻って演奏を繰り返します。セーニョが指定されていない場合、データの先頭に戻ります（ダ・カーポ）。

セーニョ

バイト数　1バイト
書式　セーニョコード
コード　$E2
パラメータ　なし
機能　ダルセーニョによって戻る位置を指定します。

リピートビギン

バイト数　2バイト
書式　リピートビギンコード、ループ値
コード　$E3
パラメータ　$01～$FF: ループ値
機能　繰り返し演奏の開始位置と繰り返しの回数を指定します。
ループ値に0を指定した場合は、2に補正されます。

リピートエンド

バイト数　1バイト
書式　リピートエンドコード
コード　$E4
パラメータ　なし
機能　繰り返し演奏の終点を指定します。

・注意
リピートビギンとリピートエンドはネストが可能です。
1段階のネストでユーザースタック領域を3バイト使用します。
ユーザースタックエリアは、1トラック12バイトですので、他のファンクションでユーザースタックエリアが使用されていない場合、4段階までのネストが可能になります。
サブトラックは、ユーザースタックエリアが9バイトであるため、最大3段階までのネストが可能です。
ユーザースタックエリアは、コール、リターンと共有しています。

ウェーブ

バイト数　2バイト
書式　ウェーブコード、波形番号
コード　$E5
パラメータ　$00～$7F(0～127):波形番号
機能　波形番号を指定して、音色を変更します。
0～44:　PSGドライバー内部定義波形番号
45～127:　ユーザー定義波形番号

・注意
デフォルト値は不定ですので、必ずこのコマンドを実行してください。
波形を定義していない番号は指定しないでください。

エンベロープ

バイト数　2バイト
書式　エンベロープコード、エンベロープ番号
コード　$E6
パラメータ　$00～$7F(0～127):エンベロープ値
機能　エンベロープ番号を指定します。
0～15:　PSGドライバー内部定義エンベロープ番号
16～127:　ユーザー定義エンベロープ番号
デフォルト値は0です。

・注意
エンベロープを定義していない番号は指定しないでください。
デフォルトで定義されているのは0～15までです。

周波数変調(FM)

バイト数　2バイト
書式　FMコード、FM番号
コード　$E7
パラメータ　$00～$7F(0～127): FM番号
機能　ユーザーが定義した周波数変調(FM)データを指定します。

FMディレイ

バイト数　2バイト
書式　FMディレイコード、ディレイ値
コード　$E8
パラメータ　$00～$FF(0～255): ディレイ値
機能　周波数変調のディレイタイムを指定します。
ディレイ値は、ダイレクトレングスモードの音長と同等です。デフォルト値は0です。

・注意
周波数変調データを定義していない場合、ディレイ値は必ず0とします。

FM補正

バイト数　2バイト
書式　FM補正コード、基準オクターブ
コード　$E9
パラメータ　$00～$07(0～7): 基準オクターブ
機能　周波数変調(FM)の補正をおこないます。
基準オクターブとは、変調データを補正なしでかけるオクターブです。補正は、オクターブの上下によっておこなわれます。基準オクターブに0を指定すると、補正がかからなくなります。デフォルト値は0です。補正は、FMの他にピッチエンベロープ、デチューン、スイープにも同時にかけられます。

・注意
補正をかけたままオクターブを上げると、補正がかからなくなることがあります。

ピッチエンベロープ(PE)

バイト数　2バイト
書式　ピッチエンベロープコード、ピッチエンベロープ番号
コード　$EA
パラメータ　$00～$7F(0～127): PE番号
機能　ユーザーが定義したピッチエンベロープ(PE)データを指定します。

ピッチエンベロープ(PE)ディレイ

バイト数　2バイト
書式　PEディレイコード、ディレイ値
コード　$EB
パラメータ　$00～$FF(0～255): ディレイ値
機能　ピッチエンベロープのディレイタイムを指定します。ディレイ値によるディレイタイムは、ダイレクトレングスモードの音長と同等です。デフォルト値は0です。

・注意
ピッチエンベロープデータを定義していない場合、ディレイ値は必ず0とします。

デチューン

バイト数　2バイト
書式　デチューンコード、デチューン値
コード　$EC
パラメータ　$80～$7F(−128～127): デチューン値
機能　音程の微調整をおこないます。デフォルト値は0です。

スイープ

バイト数　2バイト
書式　スイープコード、変化値
コード　$ED
パラメータ　$80～$7F(−128～127): 変化値
機能　スイープの変化量を指定します。
数値が大きいほど、変化が速くなります。
変化値＞0 音程下がる
変化値＜0 音程上がる
変化値＝0 スイープオフ
デフォルト値は0です。

スイープタイム

バイト数　2バイト
書式　スイープタイムコード、タイム値
コード　$EE
パラメータ　$00～$FF(0～255): タイム値
機能　スイープのかかるタイムを指定します。
タイム値は、ダイレクトレングスモードの音長と同等です。0を指定すると、タイムが無効になり、スイープはキーオフになるまでかかります。デフォルト値は0です。

ジャンプ

バイト数　3バイト
書式　ジャンプコード、アドレス
コード　$EF
パラメータ　アドレス:　アドレス下位、アドレス上位
機能　アドレスで指定されたトラックデータへジャンプします。

コール

バイト数　3バイト
書式　コールコード、アドレス
コード　$F0
パラメータ　アドレス:アドレス下位、アドレス上位
機能　アドレスで指定されたトラックデータをコールします。
リターンコマンドでコールした次のデータに制御が戻ります。

リターン

バイト数　1バイト
書式　リターンコード
コード　$F1
パラメータ　なし
機能　コールされた次のデータに戻ります。

・注意

コールとリターンはネストが可能です。1段階のネストでユーザースタック領域を2バイト使用します。ユーザースタックエリアは、1トラック12バイトですので、他のファンクションでユーザースタックエリアが使用されていない場合、6段階までのネストが可能になります。サブトラックは、ユーザースタックエリアが9バイトであるため、最大4段階までのネストが可能です。ユーザースタックエリアは、リピートビギン、リピートエンドと共有しています。

移調

バイト数　2バイト
書式　移調コード、移調値
コード　$F2
パラメータ　$80～$7F(−128～127): 移調値
機能　音程の移調をおこないます。
移調値は半音単位で、正で音程が上がり、負で下がります。デフォルト値は0です。

相対移調

バイト数　2バイト
書式　相対移調コード、相対移調値
コード　$F3
パラメータ　$80～$7F(−128～127): 相対移調値
機能　現在の移調値から相対的に移調をおこないます。移調値は半音単位で、正で音程が上がり、負で下がります。

全体移調

バイト数　2バイト
書式　全体移調コード、移調値
コード　$F4
パラメータ　$80～$7F(−128～127): 移調値
機能　全チャンネルの移調をおこないます。なお、このコマンドは有効なチャンネルの中で最も番号の大きいチャンネルに置いてください。

・注意
最も上あるいは下のオクターブで移調をおこなうと、音域からはずれる場合があります。

ボリュームチェンジ

バイト数　2バイト
書式　ボリュームチェンジコード、変化量
コード　$F5
パラメータ　$80～$7F(−128～127): 変化量
機能　ボリュームを徐々に変化させます。
変化量が、正の場合、ボリュームが上がり、負の場合、下がります。0を指定した場合、現在のボリューム値を維持したまま、変化が止まります。変化中にボリューム($DC)をセットすると、変化は解除されます。デフォルト値は0です。

パンライトチェンジ

バイト数　2バイト
書式　パンライトチェンジコード、変化量
コード　$F6
パラメータ　$80～$7F(−128～127): 変化量
機能　右チャンネルのボリュームを徐々に変化させます。
変化量が、正の場合、ボリュームが上がり、負の場合、下がります。0を指定た場合、現在のボリューム値を維持したまま、変化が止まります。変化中にパンポット($DD)をセットすると、変化は解除されます。デフォルト値は0です。

パンレフトチェンジ

バイト数　2バイト
書式　パンレフトチェンジコード、変化量
コード　$F7
パラメータ　$80～$7F(−128～127): 変化量

機能　左チャンネルのボリュームを徐々に変化させます。
変化量が、正の場合、ボリュームが上がり、負の場合、下がります。0を指定した場合、現在のボリューム値を維持したまま、変化が止まります。変化中にパンポット($DD)をセットすると、変化は解除されます。デフォルト値は0です。

モード

バイト数　2バイト
書式　モードコード、モード番号
コード　$F8
パラメータ　モード番号
0: ノーマルモード
1: パーカッションモード
2: ノイズモード

機能　ミュージックプレイモードを切り換えます。
0を指定すると、ノーマルモードとなります。
1を指定すると、パーカッションモードとなります。
パーカッションモードでノイズを使用する場合、チャンネル番号5,6以外は正常に発音されません。
2を指定するとノイズモードとなります。ノイズモードでは、チャンネル番号5,6以外は正常に発音されません。ノイズモードでの音程コードとノイズ番号の対応は以下の通りです。

ノイズコード	ノイズ番号	ノイズコード	ノイズ番号
$10:	0	$70:	6
$20:	1	$80:	7
$30:	2	$90:	8
$40:	3	$A0:	9
$50:	4	$B0:	10
$60:	5	$C0:	11

ノイズ番号は31までありますが、12以降のノイズ番号は移調によって指定します。
パーカッションモードでは、休符は以下の2通りの方法のいずれかで指定する必要があります。

1.先頭の休符をノーマルモードで指定する方法。

```
PC_PRT:
    DB    $D0,$0F  ; TIME BASE = 15
```

```
        DB    $DC,$1F ; VOL = 31
        DB    $DD,$EE ; PAN = $EE
        DB    $F8,$01  ; MODE = 1
        DB    $01      ; R
        DB    $01      ; R
        DB    $81      ; G
        DB    $01      ; R
        DB    $81      ; G
        DB    $FF      ; DATA END
```

このように演奏したいときは、先頭で休符が使用されているため、以下のように記述します。

```
PC_PRT:
        DB    $D0,$0F  ; TIME BASE = 15
        DB    $DC,$1F ; VOL = 31
        DB    $DD,$EE ; PAN = $EE
        DB    $F8,$00  ; MODE = 0
        DB    $01      ; R
        DB    $01      ; R
        DB    $F8,$01  ; MODE = 1
        DB    $81      ; G
        DB    $01      ; R
        DB    $81      ; G
        DB    $FF      ; DATA END
```

2.パーカッションデータテーブル中に休符のデータを作成する方法。

```
PC_INDEX_ADR:
        DW    PC0      ; C
        DW    PC1      ; C+
        DW    PC2      ; D
              ⋮
PC0: DB    $F0         ; DATA END .....PC0(C)
PC1:
              ⋮
```

パーカッションデータテーブル中のPC0(Cの音)を上記のように休符として設定しておきます。

```
PC_PRT:
        DB    $D0,$0F  ; TIME BASE = 15
        DB    $DC,$1F ; VOL = 31
        DB    $DD,$EE ; PAN = $EE
        DB    $F8,$01  ; MODE = 1
        DB    $01      ; R
```

```
        DB    $01       ;R
        DB    $81       ;G
        DB    $01       ;R
        DB    $81       ;G
        DB    $FF       ;DATA END
```

上記の例の休符を、休符として設定されているCに書き換えます。

```
PC_PRT:
        DB    $D0,$0F   ;TIME BASE = 15
        DB    $DC,$1F   ;VOL = 31
        DB    $DD,$EE   ;PAN = $EE
        DB    $F8,$01   ;MODE = 1
        DB    $11       ;C(R)
        DB    $11       ;C(R)
        DB    $81       ;G
        DB    $11       ;C(R)
        DB    $81       ;G
        DB    $FF       ;DATA END
```

フェードアウト

バイト数　2バイト
書式　フェードアウトコード、スピード値
コード　$FE
パラメータ　$01～$7F(1～127)：スピード値
機能　フェードアウトをおこないます。スピード値が大きいほど、フェードアウトスピードが速くなります。指定値が負の場合、正に補正されます。

データエンド

バイト数　1バイト
書式　データエンドコード
コード　$FF
パラメータ　なし
機能　トラックデータの終わりを示します。演奏停止位置には必ずこのデータを書き込みます。

波形データ

PSGドライバーには、波形データを内部で45種類定義しています。この他に83種類までの波形データをユーザーが外部に定義することができます。波形番号は、0～44($00～$2C)までが内部定義、45～127($2D～$7F)までがユーザー定義の波形データとなります。

内部定義波形データ

45種類の内部定義波形データは、実際にゲームで使用された波形から使用頻度の高いものなどを選択して、登録しています。実際の音を確認してご使用ください。

ユーザー定義波形データフォーマット

32バイトからなるユーザー定義波形データを波形番号45から順番にメモリに配置し、そのデータの先頭アドレスをファンクションコール(PSG_WAVE)によってPSGドライバーに登録します。

波形データレジスタは、各チャンネルともに5ビット/ワードであるため、波形データの各ワードは$00～$1Fまでとなり、1波形につき32ワード（1周期分）続きます。

・ユーザー定義波形データ例

```
WAVE_TOP:
   DB    $00, $00, $00, $00, $00, $00, $00, $00 ; NO.45
   DB    $1F, $1F, $1F, $1F, $1F, $1F, $1F, $1F
   DB    $00, $00, $00, $00, $00, $00, $00, $00
   DB    $1F, $1F, $1F, $1F, $1F, $1F, $1F, $1F
;
   DB    $00, $01, $02, $03, $04, $05, $06, $07 ; NO.46
   DB    $08, $09, $0A, $0B, $0C, $0D, $0E, $0F
   DB    $10, $11, $12, $13, $14, $15, $16, $17
   DB    $18, $19, $1A, $1B, $1C, $1D, $1E, $1F
;
   DB    $00, $08, $0F, $14, $19, $1B, $1D, $1E ; NO.47
   DB    $1E, $1D, $1B, $19, $14, $0F, $08, $00
   DB    $1F, $1E, $1D, $1C, $1B, $1A, $09, $08
   DB    $07, $06, $05, $04, $03, $02, $01, $00
;
```

エンベロープデータ

PSGドライバーはエンベロープデータを波形データと無関係に定義しています。

ユーザーは、0～127($00～$7F)のエンベロープ番号によって、選択することができます。エンベロープ番号0～15($00～$0F)が内部定義データ、16～127($10～$7F)が、ユーザー定義データとなります。

エンベロープデータは、インデックス部とデータ部によって構成され、インデックス部の先頭アドレスをファンクションコール(PSG_ENV)によってPSGドライバーに登録します。

●エンベロープデータフォーマット

リリースレートデータ

バイト数	3バイト
書式	コード、レベル変化量
コード	$FB
パラメータ	–$7C00～$7C00($8400～$7C00)：変化量
機能	リリースレート（キーオフしてからレベルが0になるまでの変化速度）を指定します。変化量が負の場合、レベルが0になるまで下がりますが、正の場合は、最大レベルになるまで上がります。

レベルデータ

バイト数	3バイト
書式	コード、レベル設定値
コード	$FC
パラメータ	0～$7C00：レベル設定値
機能	レベルが変化する際の初期レベルを設定します。

ディケイレートデータ

バイト数	3バイト
書式	タイム、レベル変化量
コード	なし
パラメータ	0～$250($00～$FA)：タイム値 –$7C00～$7C00($8400～$7C00)：レベル変化量
機能	ディケイレート（キーオンの間のレベル変化の速度）を設定します。変化量が正の場合、レベルは上がり、負の場合は下がります。タイム値は、キーオンの間、レベルが変化する時間を設定します。その長さはダイレクトレングスモードにおける音長と同じです。タイム値を0とした場合、レベルはキーオンの間、変化し続けます。

データエンド

バイト数　1バイト
書式　コード
コード　$FF
パラメータ　なし

機能　データの終わりを示します。
キーオン中にこのコードにあたると、現在のレベルをキーオフまで維持します。ディケイレートタイム値が0の場合は関係ありません。

・注意
レベル変化量、レベル設定値は、ミュージックトラックデータフォーマットのボリューム値に対応しています。$400がレベル1と同等であり、最大レベルである31が$7C00(31＊$400)となります。

データは必ずリリースレートデータ、レベルデータ、ディケイレートデータ、データエンドの順番で記述してください。特に、リリースレートデータは先頭以外では省略とみなされます。

リリースレートデータ、レベルデータ、ディケイレートデータは、省略が可能ですが、PSGドライバーの内部では0が設定されたものとして処理されます。

パーカッションデータで使用されるエンベロープデータは、リリースレートデータが設定されていても無視され、リリースレートが自動的に0に設定されます。

レベルデータ、ディケイレートデータは、1つのエンベロープデータのなかに複数設定することが可能です。この機能によって、複雑なエンベロープを定義することが可能です。ただし、1つのエンベロープを構成するデータ数は、85個までに制限されています。

ミュージックトラックデータで音長比が8に設定されている場合、リリースはかかりません。

・ユーザー定義エンベロープデータ例

```
ENV_INDEX:
   DW    ENV16
   DW    ENV17
    :      :
    :      :
;
ENV16:
   DB      $FB       ; RELEASE RATE DATA
   DW    −$100
   DB      $FC       ; LEVEL DATA
   DW      31＊$400
   DB      25        ; DECAY RATE DATA
   DW    −$180
   DB      $FF       ; DATA END
;
```

```
ENV17:
    DB      $FB
    DW    -$80
    DB      $FC
    DW      25*$400
    DB      3
    DW      $800
    DB      0
    DW    -$C0
    DB      $FF
;
  :       :
  :       :
```

・内部定義エンベロープデータ

```
;-----------------------------------------------
;    -- envelope data macro --
;-----------------------------------------------
;
;
evr   macro  r_rate
    db      $fb
    dw      r_rate
    endm
;
evl   macro  level
    if      level > 31
    fail    '[Music Macro Error ] -- Envelope Level Data --'
    else
    db      $fc
    dw      level*$400
    endif
    endm
;
evd   macro  d_time,d_rate
    if      d_time > 250
    fail    '[Music Macro Error ] -- Env.Decay Time Data --'
    else
    db      d_time
    dw      d_rate
    endif
    endm
;
eve
    macro
```

```
    db      $ff
    endm
;-----------------------------------------------
evt0:
    evr     -31*$400
    evl     31
    evd     0,0
    eve
;
evt1:
    evr     -$600
    evl     31
    evd     12,-$200
    eve
;
evt2:
    evr     -$300
    evl     31
    evd     12,-$200
    eve
;
evt3:
    evr     -$100
    evl     31
    evd     12,-$200
    eve
;
evt4:
    evr     -$600
    evl     31
    evd     0,-$80
    eve
;
evt5:
    evr     -$600
    evl     31
    evd     0,-$100
    eve
;
evt6:
    evr     -$600
    evl     31
    evd     0,-$200
    eve
```

```
;
evt7:
    evr     -$500
    evl     11
    evd     2,10*$400
    evd     0,-$80
    eve
;
evt8:
    evr     -$500
    evl     13
    evd     3,6*$400
    evd     0,-$80
    eve
;
evt9:
    evr     -$500
    evl     7
    evd     1,24*$400
    evd     2,-$200
    evd     8,-$400
    evd     50,$80
    eve
;
evt10:
    evr     -$300
    evl     1
    evd     3,10*$400
    evd     4,-$200
    eve
;
evt11:
    evr     -$300
    evl     0
    evl     0
    evl     31
    evd     0,-$40
    eve
;
evt12:
    evr     -$600
    evl     0
    evl     0
    evl     26
```

```
    evd     0,$60
    eve
;
evt13:
    evr     -$600
    evl     31
    evd     4,-$800
    evd     0,-$c0
    eve
;
evt14:
    evr     -$600
    evl     31
    evd     8,-$800
    evd     0,-$c0
    eve
;
evt15:
    evr     -$100
    evl     31
    evd     5,-$400
    evl     28
    evd     5,-$400
    evl     25
    evd     5,-$400
    evl     22
    evd     5,-$400
    evl     19
    evd     5,-$400
    evl     16
    evd     5,-$400
    evd     0,-$100
    eve
;-----------------------------------------------------
```

周波数変調データ

周波数変調データは、音程にかけるソフトLFOであり、ビブラート効果などを与えることができます。

変調データテーブル1つが、LFOの波形1周期分を、データ1バイト分がダイレクトレングスモードにおける音長1にあたる音程周波数の加算値を構成します。データの構成によって、各種の波形を構成することができます。

周波数変調データフォーマット

周波数変調(FM)データは、インデックス部とデータ部によって構成され、インデックス部の先頭アドレスをファンクションコール(PSG_FM)によってPSGドライバーに登録します。

なお、周波数変調データはすべてユーザーが定義し、最大128個まで定義できます。

周波数変調データ	−127～127($81～$7F)
データの終わり	−128($80)
変調データテーブル長	最大256バイト

・ユーザー定義周波数変調データ例

```
FM_INDEX:
    DW    FMD0
    DW    FMD1
    DW    FMD2
     :     :
;
FMD0:
    DB    00,01,02,03,02,01,00,−1,−2,−3,−2,−1,$80
FMD1:
    DB    00,−2,−4,−6,−4,−2,00,02,04,06,04,02,$80
FMD2:
    DB    03,03,02,02,01,01,00,00,−1,−1,−2,−2,−3,−3,$80
     :     :
```

ピッチエンベロープデータ

ピッチエンベロープデータは、ピッチ（音程）を部分的に変化させるためのものです。

データ1バイト分がダイレクトレングスモードにおける音長1にあたる音程（ピッチ）の変化値を示しています。データの構成によって、各種の効果をあげることができます。

ピッチエンベロープデータフォーマット

ピッチエンベロープデータは、インデックス部とデータ部によって構成され、インデックス部の先頭アドレスをファンクションコール(PSG_PE)によってPSGドライバーに登録します。

なお、ピッチエンベロープデータはすべてユーザーが定義し、最大128個まで定義できます。

ピッチエンベロープデータ　−127〜127($81〜$7F)
データの終わり　−128($80)
ピッチエンベロープデータテーブル長 最大256バイト

・ユーザー定義ピッチエンベロープデータ例

```
PE_INDEX:
    DW    PED0
    DW    PED1
    DW    PED2
     ⋮     ⋮
;
PED0:
    DB    −15,−12,−10,−8,−6,−5,−4,−3,−2,−1,00,$80
PED1:
    DB    0,1,2,4,6,8,12,15,17,00,$80
PED2:
    DB    −1,−3,−6,−10,−13,−14,−13,−10,−6,−3,−1,00,$80
     ⋮     ⋮
```

パーカッションデータ

PSGドライバーには、パーカッションデータを効果音的に作成し、そのデータを音程データに割り当てて発声する機能があります。データを発声するには、プレイモード1の場合、割り当てられた音程コードをトラックデータに定義することによっておこないます。

・注意
パーカッションデータ（ノイズ、音程）1個は、ダイレクトレングスモードにおける音長1の長さを持ちます。プレイモードは、トラックデータフォーマットでファンクション$F8によって設定します。

パーカッションデータフォーマット

パーカッションデータの構造は、インデックス部とデータ部に分かれています。
インデックス部には、12個のアドレスを定義することができ、その最初のアドレスから順 に音程コードの$10(C)から$C0(B)までに割り当てられます。定義されたパーカッションデータは、インデックス部の先頭アドレスをファンクションコール(PSG_PC)によって、PSGドライバーに登録します。
データ部は、以下の書式によってパーカッションデータを定義します。
データは、定義されている順に発声されます。

ノイズ番号

バイト数　1バイト
書式　ノイズ番号
コード　$00～$1F
パラメータ　なし
機能　ノイズ番号を定義します。
ノイズデータ1個は、ダイレクトレングスモードにおける音長1の長さを持ちます。

音程周波数

バイト数　2バイト
書式　コード＋周波数データ上位、周波数データ下位
コード　$B0
パラメータ　$00～$0F：周波数データ上位(4bits)
$00～$FF：周波数データ下位(8bits)
機能　音程周波数を定義します。
音程データ1個は、ダイレクトレングスモードにおける音長1の長さを持ちます。

・注意
12bits周波数データによって、以下の計算式に従い、出力周波数が決定されます。

fout=fmaster/(2＊32＊F)
fout　出力周波数
fmaster　PSG駆動クロック周波数(7.16MHz)
F　12bits周波数データ($001～$FFF)

エンベロープ番号

バイト数　2バイト
書式　コード、エンベロープ番号
コード　$C0
パラメータ　$0～127($00～$7F)：エンベロープ番号
機能　エンベロープ番号を指定します。
ここで指定されたエンベロープデータは、リリースレートが0に設定されます。

パンポット

バイト数　2バイト
書式　コード、レフトボリューム十ライトボリューム
コード　$D0
パラメータ　$00～$FF：上位4ビット　レフトボリューム値
　　下位4ビット　ライトボリューム値
機能　パンポットを指定します。

波形番号

バイト数　2バイト
書式　コード、波形番号
コード　$E0
パラメータ　$00～$7F(0～127)：波形番号
機能　波形番号を指定します。

データエンド

バイト数　1バイト
書式　コード
コード　$F0
パラメータ　なし
機能　データの終わりを示します。

・ユーザー定義パーカッションデータ例

```
PC_INDEX:
    DW    PCN0      ;C     ($10)
    DW    PCN1      ;C+    ($20)
    DW    PCN2      ;D     ($30)
    DW    PCN3      ;D+    ($40)
    DW    PCN4      ;E     ($50)
    DW    PCN5      ;F     ($60)
    DW    PCN7      ;F+    ($70)
    DW    PCN8      ;G     ($80)
    DW    PCN9      ;G+    ($90)
    DW    PCN10     ;A     ($A0)
    DW    PCN11     ;A+    ($B0)
    DW    PCN12     ;B     ($C0)
;
PCN0:
    DB    $C0,7     ;ENVELOPE NO.
    DB    $E0,6     ;WAVE NO.
```

```
        DB    $D0,$FF     ; PAN POT
        DB    $B0+3,$AC; TONE FRQ
        DB    $B0+4,$0C
        DB    $1F         ; NOISE FRQ
        DB    $1B
        DB    $1D
        DB    $1A
        DB    $F0         ; DATA END
;
PCN1:
        DB    $C0,21
        DB    $E0,6
        DB    $D0,$FF
        DB    $B0+4,$7C
        DB    $B0+4,$8C
        DB    $B0+4,$9C
        DB    $B0+4,$BC
        DB    $B0+4,$DC
        DB    $B0+4,$FC
        DB    $B0+5,$0C
        DB    $F0
PCN2:
        ⋮     ⋮
```

グラフィックドライバー

[$49] ラベル名：grp_bios　アドレス：$E0DB

機能　グラフィックドライバーを使用します。

IN　_DH:　ファンクション番号

VI_GINIT:	EQU $00	グラフィックBIOSを初期化します。
VI_CASHCLR:	EQU $01	グラフィックポインタキャッシュをクリアします。
VI_STRTADR:	EQU $02	スタートアドレス、スタックポイントを取得します。
VI_GETADRS:	EQU $03	座標アドレスから実アドレスを求めます。
VI_CLS:	EQU $04	BAT先頭アドレスから4画面分を消去します。
VI_PSET:	EQU $05	点を打ちます。
VI_POINT:	EQU $06	指定座標の色を取得します。
VI_LINE:	EQU $07	線を引きます。
VI_BOX:	EQU $08	ボックスを描きます。
VI_BOXF:	EQU $09	ボックスを描き、内部を塗りつぶします。
VI_FLOOD:	EQU $0A	色を塗ります。
VI_PAINT:	EQU $0B	境界色の内部に色を塗ります。
VI_GWINDOW:	EQU $0C	ウインドウを設定します。
VI_GFONT:	EQU $0D	フォントの種類を選択します。
VI_PUTFONT:	EQU $0E	漢字を表示します。
VI_SYMBOL:	EQU $0F	文字フォントを拡大表示します。

OUT　なし

【解説】
_DHにファンクション番号を入れ、それぞれのファンクション番号に対応したパラメータをレジスタに入れて、grp_biosをコールすることによって、グラフィックドライバーをコントロールします。

ファンクションコール

グラフィックドライバーには、16個のファンクションコールが用意されており、自由にコントロールすることが可能です。

・注意
ファンクションコールをおこなうと、ACC（アキュムレータ）の内容は破壊されます。また、XREG（Xレジスタ）の内容も破壊されます。

VI_GINIT

ファンクション番号 _DH:　$00
機能　グラフィックBIOSを初期化します。
IN　_AX:　グラフィックスタートアドレス
　　_BX:　ペイントスタック領域
OUT なし

【解説】
グラフィックスタートアドレス、ペイントスタック領域の設定、グラフィックポインタキャッシュのクリア、各種ワークの初期化などを行います。_ALの下位5ビットは無視されます。

VI_CASHCLR

ファンクション番号　_DH:　$01
機能　グラフィックポインタキャッシュをクリアします。
IN　なし
OUT なし

【解説】
VI_GINITによって初期化を行った場合、キャッシュもクリアされますので、このファンクションコールは通常使用しません。

VI_STRTADR

ファンクション番号　_DH:　$02
機能　グラフィックスタートアドレスおよびペイントスタック領域のスタックポインタを取得します。
IN　なし
OUT　_AX:　グラフィックスタートアドレス
　　_BX:　ペイントスタック領域のスタックポインタ

VI_GETADRS

ファンクション番号　_DH:　$03
機能　座標アドレスから実アドレスを算出します。
IN　_AH:　X座標
　　_AL:　Y座標
OUT　_AX:　実アドレス
　　_BH:　ビット位置

【解説】
ビット位置とは、その座標アドレスが、第7ビットから数えて、実アドレスの何ビット目に対応しているのかを求めた数値のことです。

VI_CLS

ファンクション番号　_DH:　　$04
機能　現在指定されているグラフィック用のBAT先頭アドレスから4画面分の領域を消去します。
IN　なし
OUT　なし

【解説】
BGはすべてクリアされます。

VI_PSET

ファンクション番号　_DH:　　$05
機能　指定された座標に点を打ちます。
IN　_AH:　X座標
　　_AL:　Y座標
　　_DL:　カラーコード(0～15)
OUT　なし

VI_POINT

ファンクション番号　_DH:　　$06
機能　指定座標のカラーコードを取得します。
IN　_AH:　X座標
　　_AL:　Y座標
OUT　_AREG:　指定座標のカラーコード(0～15)

VI_LINE

ファンクション番号　_DH:　　$07
機能　指定された座標の間に線を引きます。
IN　_AH:　始点のX座標
　　_AL:　始点のY座標
　　_BH:　終点のX座標

_BL: 終点のY座標
_DL: カラーコード(0～15)
OUT なし

VI_BOX

ファンクション番号 _DH: $08
機能 指定された座標を対角線とするボックスを描きます。
IN _AH: 始点のX座標
_AL: 始点のY座標
_BH: 終点のX座標
_BL: 終点のY座標
_DL: カラーコード(0～15)
OUT なし

VI_BOXF

ファンクション番号 _DH: $09
機能 指定された座標を対角線とするボックスを描き、内部を塗りつぶします。
IN _AH: 始点のX座標
_AL: 始点のY座標
_BH: 終点のX座標
_BL: 終点のY座標
_DL: カラーコード(0～15)
OUT なし

VI_FLOOD

ファンクション番号 _DH: $0A
機能 指定した座標から色を塗ります。
IN _AH: X座標
_AL: Y座標
_DL: ペイントカラーコード(0～15)
OUT なし

【解説】
塗る前に指定された座標にあった色以外の色すべてが境界色になります。

VI_PAINT

ファンクション番号　_DH:　　$0B

機能　境界色を指定し、その色で囲まれた部分を指定された座標から塗りつぶします。

IN　_AH:　X座標
　　_AL:　Y座標
　　_BH:　境界色ビット(H)
　　_BL:　境界色ビット(L)
　　_DL:　ペイントカラーコード(0～15)

OUT　なし

【解説】

漢字フォントバッファを破壊します。

VI_GWINDOW

ファンクション番号　_DH:　　$0C

機能　指定座標にウィンドウを設定します。

IN　_AH:　始点のX座標
　　_AL:　始点のY座標
　　_BH:　BATのスタートX座標
　　_BL:　BATのスタートY座標
　　_CH:　X方向のキャラクター数
　　_CL:　Y方向のキャラクター数
　　_DL:　カラーバンク

OUT　なし

【解説】

表示画面の一部にウィンドウを設定します。

オートインクリメント=+01、64×64キャラクタモードが前提となります。

VI_GFONT

ファンクション番号　_DH:　　$0D

機能　PUTFONT、SYMBOLによって出力されるフォントの種類を選択します。

IN　_AL:　文字フォント
　　0:　16ドットフォント
　　1:　12ドットフォント
　　4:　ユーザー定義文字

OUT　なし

VI_PUTFONT

ファンクション番号　_DH:　　$0E
機能　指定された座標に漢字を表示します。
IN　_AH:　X座標
　　_AL:　Y座標
　　_BX:　文字フォントナンバー
　　_CL:　バックカラーおよびモードの選択
　　bit0～3　XXXX????　バックカラーコード
　　bit6　X?XXXXXX　1:　バックトランスペアレントON
　　0:　バックトランスペアレントOFF
　　bit7　?XXXXXXX　1:　フォアトランスペアレントON
　　0:　フォアトランスペアレントOFF
　　_DL:　フォアカラーコード
OUT　なし

【解説】
文字フォントナンバーとはシフトJISコードです。

VI_SYMBOL

ファンクション番号　_DH:　　$0F
機能　指定された文字フォントを拡大して表示します。
IN　_AH:　X座標
　　_AL:　Y座標
　　_BX:　文字フォントナンバー
　　_CL:　バックカラーおよびモードの選択
　　bit0～3　XXXX????　バックカラーコード
　　bit6　X?XXXXXX　1:　バックトランスペアレントON
　　0:　バックトランスペアレントOFF
　　bit7　?XXXXXXX　1:　フォアトランスペアレントON
　　0:　フォアトランスペアレントOFF
　　_CH:　XY拡大率(X拡大率上位4ビット、Y拡大率下位4ビット)
　　bit7～4　????XXXX　X方向拡大率
　　bit3～0　XXXX????　Y方向拡大率
　　_DL:　フォアカラーコード
OUT　なし

【解説】
文字フォントナンバーとはシフトJISコードです。

でべろ外部コマンド

概要

でべろの通信環境を支援するMS-DOS(MSXではMSX-DOS)用の外部コマンドが用意されています。これらのコマンドは「転送」メニューの中の「98用(MSX用)でべろ基本コマンド」を選び、フロッピーディスクをインストールして作られるslave.exe(MSXではslave.com)を使って、でべろの「転送」メニューでCD-ROMから取り出すことができます。

・ヘルプについて

コマンド単体で入力したり、「/?」などのオプションをつけて起動すると、画面上にかんたんな説明を表示して、終了します。

・通信経過の表示について

一部のコマンドでは、プログラムやデータを転送するときに、デフォルトで通信経過を次のように表示します。

1024/ 8192 (512)

数字の意味は順に「転送済みバイト数／全バイト数（パケットの大きさ）」です。通信不良があると、転送するパケットの大きさを小さくしてリトライしていき、それでもうまくいかないときは、「communication error ...」を出して終了します。このときはケーブルやコネクタがしっかり繋がっているかどうか、調べてください。

コマンド解説

init

PC-98: init.exe　　MSX: init.com

機能　通信環境を終了して、CD-ROM² BIOS に戻ります。

書式　init

オプション　なし

clrscr

PC-98: clrscr.exe　　MSX: clrscr.com

機能　画面をクリアして、通信待ちの初期表示に戻します。

書式　clrscr

オプション　なし

syschg

PC-98: syschg.exe　　MSX: syschg.com

機能　通信システムを更新します。

書式　syschg <system_file> [/n]

<system_file>:　通信システムが書かれたMXファイル

オプション　/n:　通信経過を表示しません

【解説】

パソコンとPCエンジン間の通信システムを、更新するためのコマンドです。普段は使いません。

・注意

システムを更新するためのワークエリアとして、バンク$84を使用します。

peas

PC-98: peas.exe

機能　PCエンジン用アセンブラです。MXファイルを出力します。

書式　peas <source>

<source>:　アセンブルするソースファイル名
拡張子は付けません。

オプション　なし

・注意

ソースファイルの拡張子は「.asm」で保存しておいてください。

MSX:peas.com

機能　PCエンジン用アセンブラです。MXファイルを出力します。

書式　peas<source>[/e][/l][/o]

<source>:　アセンブルするソースファイル名
拡張子は付けません。

オプション　/e:　エラーメッセージファイルを作ります

/l:　リスティングファイルを作ります

/o:　オブジェクトファイルを作ります

・注意

ソースファイルの拡張子は「asm」で保存しておいてください。

disas

PC-98: disas.exe　　MSX: disas.com

機能　指定したアドレスからPCエンジンのメモリ内容をディスアセンブルして標準出力に出力します。

書式　disas <from> <to>

<from>:　16進数で表記した開始アドレス(0～FFFF)

<to>:　16進数で表記した終了アドレス(0～FFFF)

オプション　なし

【解説】

必要ならリダイレクトでファイルに保存してください。

・注意

disas.txtがdisas.exe（disas.com）と同じディレクトリに存在する必要があります。

mx2bin

PC-98: mx2bin.exe　　MSX: mx2bin.com

機能　指定されたMXファイルをバイナリファイルに変換します。

書式　mx2bin <mx_file> <bin_file>

<mx_file>:　変換するMXファイル

<bin_file>:　バイナリファイル名

オプション　なし

dvdir

PC-98: dvdir.exe　　MSX: dvdir.com

機能　でべろのCD-ROM内のディレクトリ情報を表示します。

書式　dvdir

dvdir <spec>

<spec>:　ファイルスペック

オプション　なし

【解説】

ファイルスペック（ファイル名）を指定すると、それに該当するファイルだけを表示します。ファイルスペックにはワイルドカードが使用できます。

getfil

PC-98: getfil.exe　　MSX: getfil.com

機能　CD-ROM内のファイルをパソコン側に転送します。

書式　getfil <file>

<file>:　　CD-ROM内のファイル名

オプション　なし

slave

PC-98: slave.exe　　MSX: slave.com

機能　PCエンジン側からのファイル転送要求等を処理します。

書式　slave

オプション　なし

【解説】

ESCキーでパソコン側はコマンドラインへ戻りますが、PCエンジン 側の処理は続行されます。PCエンジン側から終了命令を受け取ると、終了します。

slave2

PC-98: slave2.exe　　MSX: slave2.com

機能　PCエンジン側からのファイル転送要求等を処理します。

書式　slave2

オプション　なし

【解説】

ESCキーでパソコン側はコマンドラインへ戻りますが、PCエンジン側の処理は続行されます。PCエンジン側から終了命令を受け取っても、終了しません。

exec

PC-98: exec.exe　　MSX: exec.com

機能　アドレスを指定して、実行します。

書式　exec <address>

<address>:　　16進数で表記した実行開始アドレス(0～FFFF)

オプション　なし

execmx

PC-98: execmx.exe　　MSX: execmx.com

機能　指定されたMXファイルを転送し、その先頭アドレスから実行します。

書式　execmx <mx_file> [/n]

<mx_file>:　実行するプログラムの書かれたMXファイル

オプション　/n:　通信経過を表示しません

【解説】

転送に先立ち、バンクは 3:84, 4:85, 5:86, 6:87 にセットされます。

・注意

$0000～$5FFF, $E000～$FFFF はシステム領域であることに注意して MX ファイルを作成してください。

$4000 もしくは $4003 へジャンプすると再び通信環境へ戻ります。$4000 では画面の初期化もおこないます。

perun

PC-98: perun.exe　　MSX: perun.com

機能　指定された MX ファイルを転送し、その先頭アドレスから実行します。パソコン側はPCエンジンの通信待ち状態になります。

書式　perun <mx_file> [/n]

<mx_file>:　実行するプログラムの書かれたMXファイル

オプション　/n:　通信経過を表示しません

【解説】

パソコン上のファイルを利用しながら動作するプログラムを、実行するためのコマンドです。

プログラム転送に先立ち、バンクは 3:84, 4:85, 5:86, 6:87 にセットされます。

プログラムの転送が完了すると、PCエンジン側では実行が開始され、パソコン側ではPCエンジン側からのファイル転送要求等を処 理するために、通信待ち状態となります。パソコン側はESCキーでコマンドラインへ戻りますが、PCエンジン側の処理は続行されます。PCエンジン側のプログラムが実行終了した場合は、パソコン側も自動的にコマンドラインへ戻ります。

・注意

$0000～$5FFF, $E000～$FFFF はシステム領域であることに注意してMXファイルを作成してください。プログラムを終了するときには、$4003(dv_Standby)へジャンプすると、再び通信環境へ戻ります。また、PCエンジン側の画面を初期化して戻る場合は、$4000(dv_System) へジャンプしてください。

dspgrp

PC-98: dspgrp.exe　　MSX: dspgrp.com

機能　指定したグラフィックファイルを転送し、画像を表示します。

書式　dspgrp <graphic_file> [/r] [/n]

<graphic_file>:　gp0形式の画像ファイル

オプション　/r:　画像を表示したまま、通信環境へ戻ります

/n:　通信経過を表示しません

plytrk

PC-98: plytrk.exe　　MSX: plytrk.com

機能　トラックデータ、(指定されていれば) パーカッションデータを転送し、音楽を演奏します。

書式　plytrk <track_file>

plytrk <track_file> <percussion_file>

<track_file>:　トラックデータファイル
MXファイル形式ならアドレス自由 ($8000～$BFFF) です。
ベタファイル形式ならアドレス$8000からのデータとして扱います。

<percussion_file>:　パーカッションデータファイル
MXファイル形式ならアドレス自由 ($8000～$BFFF) です。ただしトラックデータと重複しない場所に配置してください。
ベタファイル形式ならアドレス$9800からのデータとして扱います。

オプション　なし

【解説】

演奏されるのは必ずトラックデータ中のサウンド番号0であって、他のサウンド番号の楽譜を演奏することはできません。また、現段階ではメイントラックのみの対応で、サブトラックのことは考慮されていません。パーカッションモードを使用する場合は、パーカッションデータファイルが必要です。Ⅱボタンを押すと、演奏を中断して通信環境に戻ります。演奏データが終了した場合も通信環境に戻ります。

getbnk

PC-98: getbnk.exe　　MSX: getbnk.com

機能　バンク設定(マッピングレジスタ)の情報を得て、表示します。

書式　getbnk

オプション　なし

getplt

PC-98: getplt.exe　　MSX: getplt.com

機能　パレットメモリ上のデータをファイルに転送します。

書式　getplt <address> <length> <binary_file> [/n]

<address>:　16進数で表記した転送開始アドレス (0～1FF)

<length>:　16進数で表記した転送データ長 (1～200)
<address>,<length> とも単位は word

<binary_file>:　転送データが格納されるファイル

オプション　/n: 通信経過を表示しません

getram

PC-98: getram.exe　　MSX: getram.com

機能　メインメモリ上のデータをファイルに転送します。

書式　getram <address> <length> <binary_file> [/n]

<address>:	16進数で表記した転送開始アドレス (0〜FFFF)
<length>:	16進数で表記した転送データ長 (1〜10000)
	<address>,<length> とも単位はバイト
<binary_file>:	転送データが格納されるファイル
オプション　/n:	通信経過を表示しません

getvram

PC-98: getvram.exe　　MSX: getvram.com

機能　VRAM上のデータをファイルに転送します。

書式　getvram <address> <length> <binary_file> [/n]

<address>:	16進数で表記した転送開始アドレス (0〜7FFF)
<length>:	16進数で表記した転送データ長 (1〜8000)
	<address>,<length> とも単位は word
<binary_file>:	転送データが格納されるファイル
オプション　/n:	通信経過を表示しません

setbnk

PC-98: setbnk.exe　　MSX: setbnk.com

機能　マッピングレジスタを変更して、指定した物理アドレス空間を、論理アドレス空間に展開します。

書式　setbnk <area> <bank>

<area>:	展開先の論理アドレス（0〜7）
	0:0000〜1FFF　4:8000〜9 FFF
	1:2000〜3FFF　5:A000〜BFFF
	2:4000〜5FFF　6:C000〜DFFF
	3:6000〜7FFF　7:E000〜FFFF
<bank>:	0〜FF:物理アドレスのブロック番号
オプション　なし	

【解説】

物理アドレス<bank>×$2000からの8Kバイトが展開されます。

・注意

0,1,2,7 のエリアを不用意に変更すると、PCエンジンが暴走するので注意してください。

setplt

PC-98: setplt.exe　　MSX: setplt.com

機能　パレットメモリ上にファイルからデータを転送してセットします。

書式　setplt <address> <binary_file> [/n]

<address>:　16進数で表記した転送開始アドレス(0～1FF)
<address> の単位は word

<binary_file>:　パレットデータがベタに書かれたファイル

オプション　/n:　通信経過を表示しません

setram

PC-98: setram.exe　　MSX: setram.com

機能　メインメモリ上にファイルからデータを転送してセットします。

書式　setram <address> <binary_file> [/n]

<address>:　16進数で表記した転送開始アドレス(0～FFFF)

<binary_file>:　機械語データがベタに書かれたファイル

オプション　/n:　通信経過を表示しません

・注意

$0000～$5FFF, $E000～$FFFF はシステム領域なので注意して転送してください。

setvram

PC-98: setvram.exe　　MSX: setvram.com

機能　VRAM上にファイルからデータを転送してセットします。

書式　setvram <address> <binary_file> [/n]

<address>:　16進数で表記した転送開始アドレス(0～7FFF)
<address> の単位は word

<binary_file>:　データがベタに書かれたファイル

オプション　/n:　通信経過を表示しません

edtplt

PC-98: edtplt.exe　　MSX: edtplt.com

機能　パレット上のデータを直接編集します。

書式　edtplt
edtplt <add>

<add>:　16進数で表記した編集開始アドレス(0～1FF)
省略した場合は、$0000

オプション　なし

【解説】
画面にパレットメモリの内容が表示されるので、カーソルキーで変更したいところを選んで(1～F)で直接メモリ内容を変更します。
F1キーで編集アドレスを変更します。ESCキーで終了です。

edtram

PC-98: edtram.exe　　MSX: edtram.com

機能　メインメモリ上のデータを直接編集します。
書式　edtram
　　edtram <add>
　　<add>:　　16進数で表記した編集開始アドレス(0～FFFF)
　　省略した場合は、$8000
オプション　なし

【解説】
画面にメインメモリの内容が表示されるので、カーソルキーで変 更したいところを選んで(1～F)で直接メモリ内容を変更します。
F1キーで編集アドレスを変更します。TABキーは編集方式を16進数とアスキー文字とで変更します。ESCキーで終了です。

edtvram

PC-98: edtvram.exe　　MSX: edtvram.com

機能　VRAM上のデータを直接編集します。
書式　edtram
　　edtram <add>
　　<add>:　　16進数で表記した編集開始アドレス(0～7FFF)
　　省略した場合は、$0000
オプション　なし

【解説】
画面にVRAMの内容が表示されるので、カーソルキーで変更したいところを選んで(1～F)で直接メモリ内容を変更します。
F1キーで編集アドレスを変更します。TABキーは編集方式を16進数とアスキー文字とで変更します。ESCキーで終了です。

dmpplt

PC-98: dmpplt.exe　　MSX: dmpplt.com

機能　パレットメモリのデータを表示します。

書式　dmpplt <from>

dmpplt <from> <to>

<from>:　16進数で表記した開始アドレス(0～1FF)

<to>:　16進数で表記した終了アドレス(0～1FF)

省略した場合は、128ワード表示

オプション　なし

dmprgb

PC-98: dmprgb.exe　　MSX: dmprgb.com

機能　パレットメモリのデータをRGBに分割した形で表示します。

書式　dmprgb <from>

dmprgb <from> <to>

<from>:　16進数で表記した開始アドレス(0～1FF)

<to>:　16進数で表記した終了アドレス(0～1FF)

省略した場合は、128ワード表示

オプション　なし

【解説】

表示される3ケタの数値の並びは、ＧＲＢの順です。

dmpram

PC-98: dmpram.exe　　MSX: dmpram.com

機能　メインメモリ上のデータを表示します。

書式　dmpram <from>

dmpram <from> <to>

<from>:　16進数で表記した開始アドレス(0～FFFF)

<to>:　16進数で表記した終了アドレス(0～FFFF)

省略した場合は、256バイト表示

オプション　なし

dmpvram

PC-98: dmpvram.exe　　MSX: dmpvram.com

機能　VRAM上のデータを表示します。

書式　dmpvram <from>

dmpvram <from> <to>

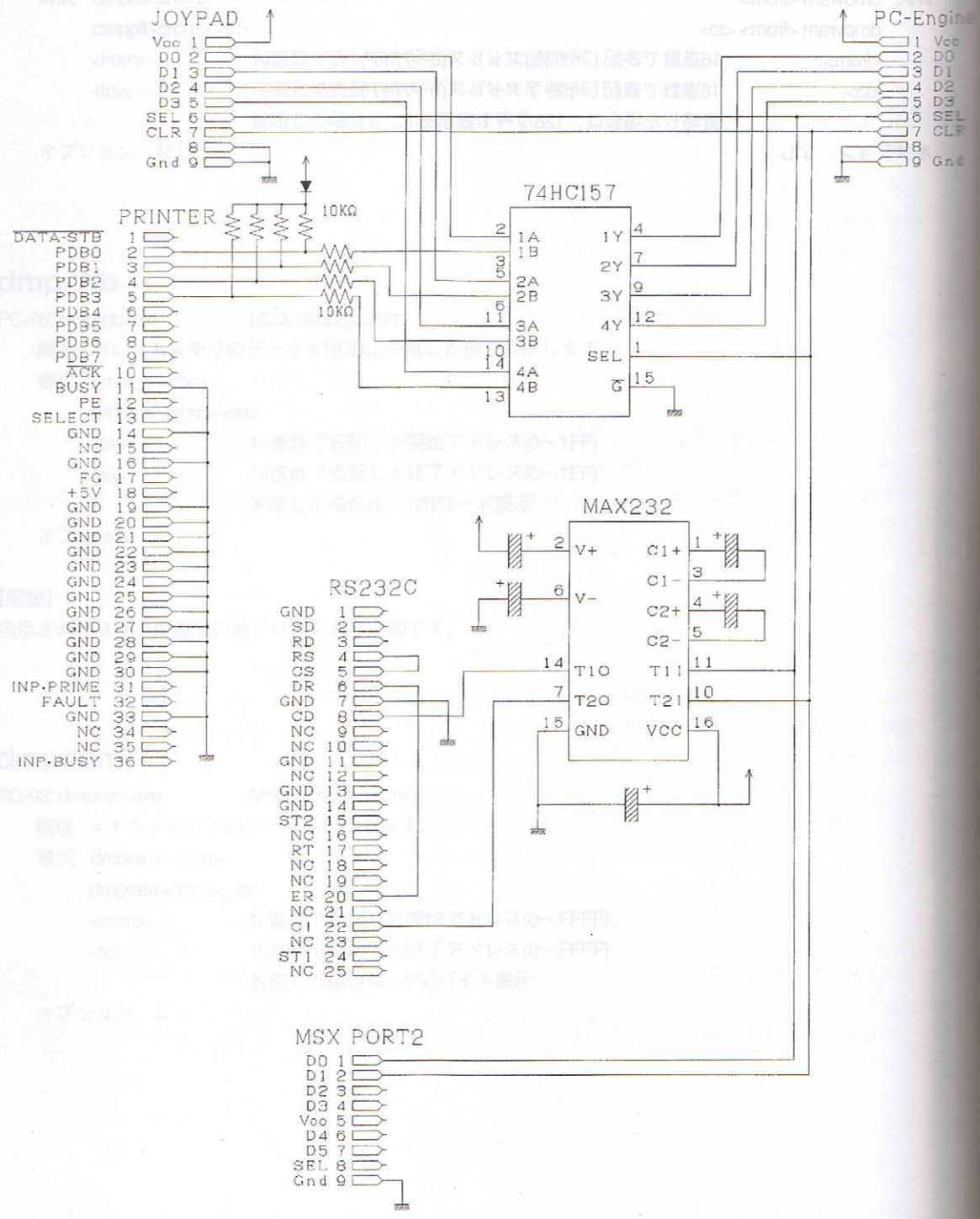
<from>:　　16進数で表記した開始アドレス(0～7FFF)

<to>:　　16進数で表記した終了アドレス(0～7FFF)

省略した場合は、128ワード表示

オプション　なし

でべろBOX基盤回路図



第5部
付録

C O N T E N T S

付録CD-ROM・FDの使い方

でべろのタイトル画面
タイトル画面でしばらくなにも入力がないとでべろの説明コーナーがはじまります。

PCエンジンだけで使用する

でべろの付録CD-ROMは、PCエンジン単体だけでも動作します。一部の機能しか使用することはできませんが、でべろで作ったプログラムやCG、サウンドを体験することができます。

まず、PCエンジンのCD-ROMドライブに付録CD-ROMをセットし、本体の電源を入れます。画面の表示にしたがってRUNボタンを押します。しばらくするとでべろのタイトル画面が表示されますので、Ⅰボタンを押してメニュー画面（写真）を表示します。

メニュー画面でのパッドの操作は、方向キーの↑と↓、Ⅰボタン、Ⅱボタンを使用します。画面上で点滅しているカーソル（写真の「プログラム」のところにある）を動かすには、方向キーの↑と↓を使用します。そして、見たいコーナーに入りたいときは、Ⅰボタンを押します。コーナーから出て、メニューに戻りたいときは、Ⅱボタンを押します。

付録CD-ROMのメニュー画面。パッドの↑・↓で操作してⅠボタンで決定。でべろで作ったプログラムやCG、サウンドなどが体験できます

プログラム

でべろで作ったサンプルプログラムを遊ぶコーナーです。ゲームプログラムが２本とサンプルプログラムが５本用意されています。

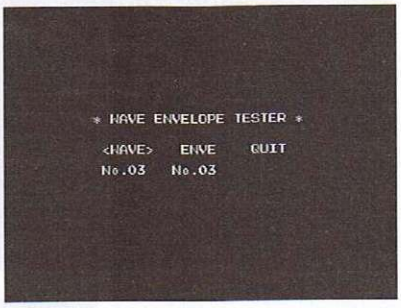
●SPEEDERの遊び方

SPEEDERは、MSX･FAN誌に投稿されたレースゲームです。原作をそのまま忠実に移植しました。使用するボタンは方向キーの←と→、Ⅰボタンです。Ⅰボタンでアクセルとなります。ブレーキはありません。画面下に表示されているのが、ライバルと自分の位置です。ライバルは上、自分は下にゲージが表示されています。このゲージが画面左へたどり着いた時点で勝負が決まります。ライバルよりも先にゴールすれば、コースクリアでつぎのコースに進みます。ライバルに負けるとゲームオーバーです。コースは右に左に曲がりくねっていますので、アクセルコントロールは慎重に。コースを飛び出してしまうと、クラッシュとなり貴重な時間を費やしてしまいます。リプレイはⅡボタンです。

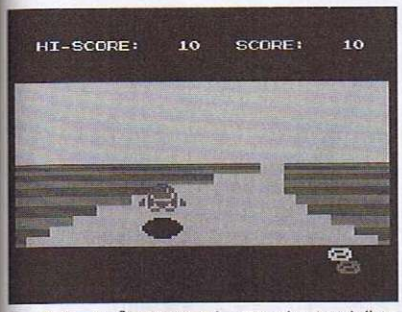
●BENDBENTの遊び方

BENDBENTは、MSX･FAN誌に投稿されたアクションパズルゲームです。プレイヤーは、あみだくじのような横木を動かして、ボールを目的のところへ導きます。使用するボタンは方向キーの←と→、Ⅰボタンです。Ⅰボタンであみだの横木が１段上に上がります。←と→であみだの上にある「▼」のカーソルを動かして選択します。クリアの条件は、画面右側に表示されているNORMの数ぶん、"LEVEL UP"のところにボールを落としてやること。"LOST BALL"にボールが落ちると、画面右側に表示されているCHANCEの数が減っていきます。0になったらゲームオーバーです。

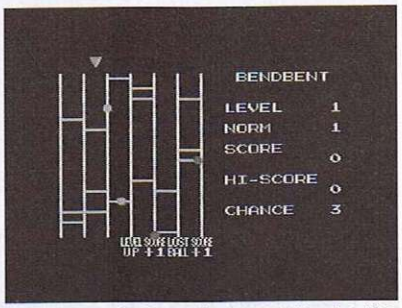
プログラムのコーナーでは、ゲームが２本とサンプルプログラムが使用できます



サンプルプログラム『WAVEとENVELOPEのサンプル』ではすべての音色を聴くことができます



レースゲーム『SPEEDER』。ライバルよりも先にゴールできなければゲームオーバー



アクションパズルゲーム『BENDBENT』。あみだの横木を操作してボールを導く

MSX・FAN誌

1995年８月号で休刊したMSX専門のパソコン誌。読者の投稿プログラムに支えられ、MSX自体の生産が打ち切られてからも雑誌を発刊しつづけました。MSXは、アスキーが提唱した統一規格パソコン。

WAVEとENVELOPEのサンプル

PCエンジンのPSGサウンドの音色を聴くことができるサンプルです。パッドの←・→でWAVEかENVEを選び、↑・↓でナンバーを変えます。Ⅰボタンを押すと演奏をはじめます。Ⅱボタンで演奏の中断です。終了は、QUITを選んでください。

スプライト表示のサンプル

画面左上にスプライトを表示します。Ⅱボタンで終了します。

効果音のサンプルⅠ

Ⅰボタンで効果音が鳴ります。Ⅱボタンで終了します。

効果音のサンプルⅡ

ランダムに効果音が鳴ります。Ⅱボタンで終了します。

ファイル読み込みのサンプル

CD-ROMに収録されたテキストファイルを読み込んで終了します。画面にメッセージが出ます。Ⅱボタンで終了します。

(CGの)コンバート
MSXのSCREEN5のＣＧをPCエンジンのBGのデータに変換したものを表示しています。すべて16色の画像です。

CG

MSX・FAN誌に投稿された読者のCG作品をPCエンジンのグラフィックにコンバートした作品を収録しています。

操作方法は、CGのメニュー画面で方向キーの↑と↓で選択カーソルを動かし、Ⅰボタンでグラフィックを表示します。

グラフィック表示中は、画面下に説明テロップが表示されます。テロップを消したいときは、Ⅰボタンを押してください。もう一度押すと再度テロップが表示されます。

また、グラフィック表示中に方向キーの↑と↓で表示されるグラフィックが変わります。Ⅱボタンを押すとメニューに戻ります。

ＣＧメニュー画面。パッドの↑、↓でカーソルを移動してⅠボタンで選択。次画面がある場合は、「▼」を表示

グラフィック表示中は、画面下にテロップが表示される。テロップのオン・オフは、Ⅰボタン

サウンド

MSX・FAN誌に投稿された読者の音楽作品をPCエンジンの音楽（PSG）にコンバートした作品を収録しています。

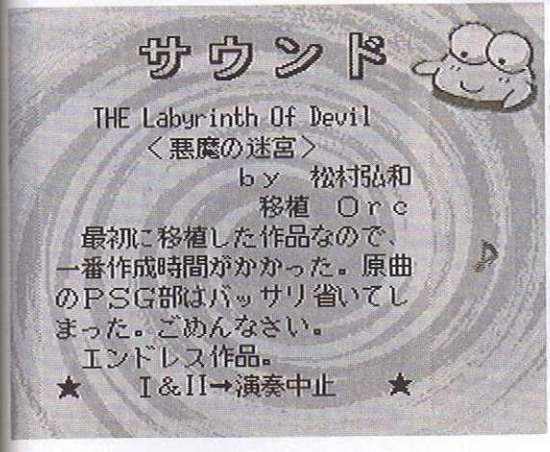
操作方法は、サウンドのメニュー画面で方向キーの↑と↓で選択カーソルを動かし、Ⅰボタンで曲を選びます。

説明文が表示されたあとⅠボタンを押すと演奏開始です。演奏を停止したいときは、ⅠボタンかⅡボタンを押してください。

「MSX版原曲」では、MSXでのサウンドを原音のままCD-DAで収録しました。元のサウンドと聴きくらべてください。

サウンドのメニュー画面でⅡボタンを押すとメニューに戻ります。

サウンドメニュー画面。操作方法は、各メニュー画面と同じ。PSGだけでなく、MSXの原音でも収録



サウンド演奏中。写真ではわかりませんが、演奏中は右の音符が踊っています

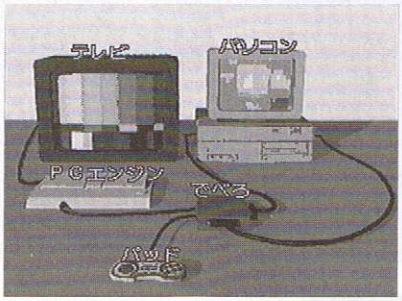
（サウンドの）コンバート

元はMSXのFM音源用のデータです。MSXのBASIC言語で作られているため、MML（ミュージック・マクロ・ランゲージ）の部分だけ抜き出して変換しました。

でべろ説明

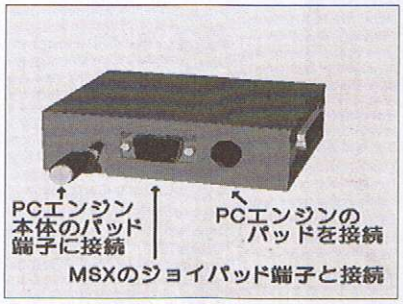
でべろＢＯＸとの接続方法について説明するコーナーです。メニューの説明コーナーを選択すると音声とスライドで、でべろについて説明をはじめます。タイトル画面やメニュー画面でなにも入力せずにしばらく待つと自動的に説明がはじまります。説明が終わるとメニューに戻ります。

説明中のパッド操作は、Ⅰボタンでつぎの説明にスキップします。Ⅱボタンで説明をキャンセルしてメニュー画面に戻ります。

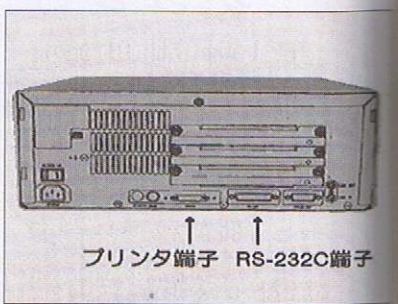


でべろ接続図。PC-98を例に解説します。各機器同士の接続は各機器の説明書をお読みください

PCエンジンのパッド端子がパソコンとのやりとりのすべての窓口です



でべろＢＯＸには、５つの端子が出ています。間違わないように接続してください



パソコン側は、プリンター端子とRS-232C端子とを使用します。ケーブルは別に用意してください

パソコンとつないで使用する

ここからは、パソコンと接続した場合の使用方法について説明します。

PCエンジンとパソコン、でべろＢＯＸを接続した状態にしてください。付録のフロッピーディスクを用意し、ディスクドライブに差し込んでください。

PC-98シリーズの場合は、付録ディスクのドライブに移動して、

install

と入力してください。すると写真左上のようにメッセージが表示されますので、ドライブ、ディレクトリを順番に指定してください。

これで、付録ディスクからslaveコマンドがインストールされます。

MSXシリーズの場合は、付録ディスクで起動してください。MSX-DOSかMSX-DOS2が立ち上がります。まず、インストールしたいフォーマット済みのディスクを用意してください。すべてのファイルをCD-ROMから転送するには、2DDディスクが３枚必要です。

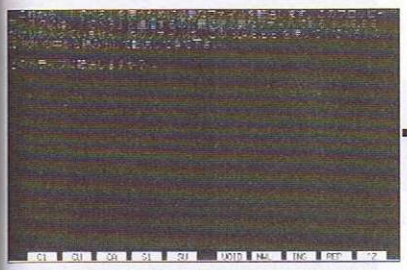
付録ディスクのドライブ上で

install

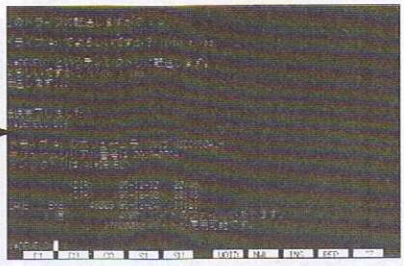
と入力してください。すると写真左上のようにメッセージが表示されますので、インストールしたいディスクを差し込んでリターンキーを押してください。

これで、付録ディスクからslaveコマンドがインストールされました。

PC-98シリーズの場合

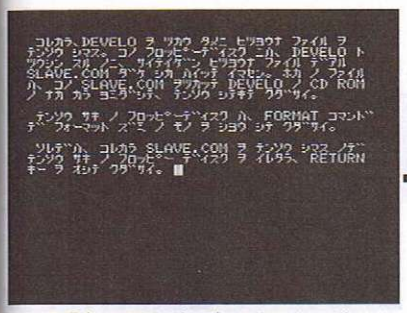


付録ディスクを入れて、installと入力するとメッセージが表示されます。ドライブを指定してください

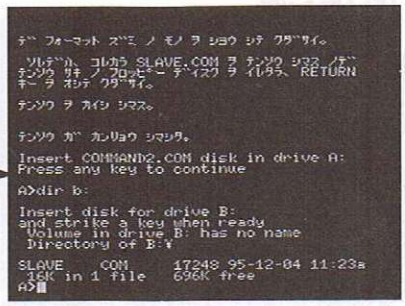


つぎにディレクトリを指定します。指定したディレクトリにslave.exeをインストールします

MSXシリーズの場合



MSXの場合は、フロッピーディスクにインストール。フォーマット済みのディスクを用意してください



転送が終了するとslave.comというファイルができています

付録ディスク

付録ディスクには、MSX-DOSのシステムファイルが入っています。フォーマットはMSX TurboRでおこなっています。

slaveコマンド

slaveコマンドには、PC-98シリーズ用のslave.exeとMSXシリーズ用のslave.comの２つがあります。PC-98シリーズでslave.comを実行すると暴走します。PC-98シリーズでは、拡張子が「com」のファイルが優先されますので注意してください。

転送
転送には、パソコン側のCPUスピードがかなり影響します。速いパソコンほど転送も早く終了します。また、フロッピーディスクよりもハードディスクに転送するほうが早く転送できます。

転送

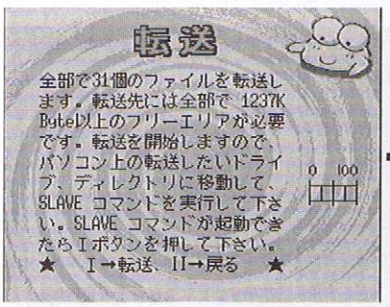
パソコン側にslaveコマンドがインストールできたら、付録CD-ROMの転送のコーナーを選択してください。まず、「ＰＣ98用でべろ基本ツール」（MSXの場合は「ＭＳＸ用でべろ基本ツール」）を選択します。でべろの外部コマンドがこのツールの中に収録されています。

あとは、画面の指示にしたがってインストールしたいディレクトリに移動してslaveコマンドを実行し、転送を開始します。でべろ基本ツールは、全部で31個のファイルがあり、転送には（機種によって異なりますが）数分～数十分かかります。

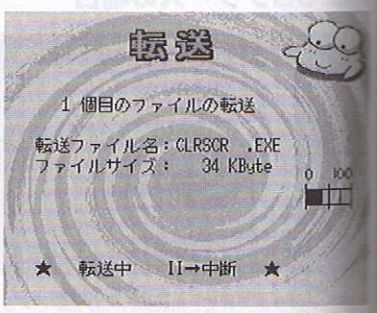
PCエンジン側

転送メニュー画面。方向キーの↑、↓でカーソルを移動してＩボタンで選択

まずは「ＰＣ98用でべろ基本ツール」を選択。でべろの外部コマンドなどが入っています

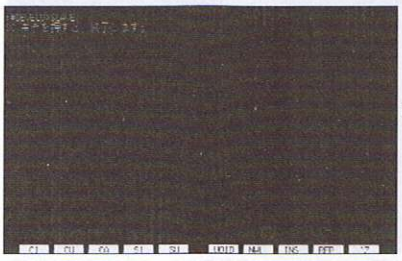


基本ツールのファイルは、全部で31個。1237Kバイトの容量が必要です

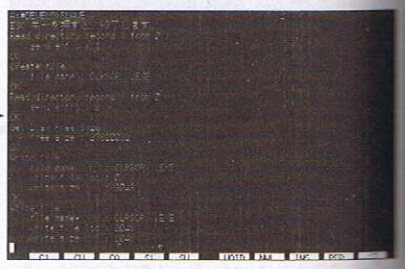


パソコンでslaveコマンドを実行していると転送を開始します。右のバーが100になったら終了

パソコン側



PCエンジンの表示が左中段の写真のようになったら、パソコン側で、slaveコマンドを実行します



PCエンジンで転送を開始するとパソコンのディスプレイに転送のもようが表示されていきます

転送中にユーザーが中断したり、問題があった場合にエラーを表示して処理を中止することがあります。以下の写真に示すのは、転送中止の例です。

ユーザーが中断する場合。転送中にユーザーがIIボタンを押すことによって転送を中断することができます。このファイルをスキップするか、すべての転送を中止して転送メニューに戻ります。

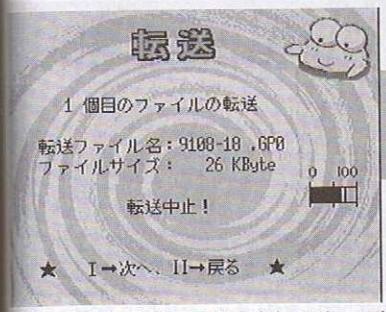
転送先の容量が不足している場合。転送先のパソコンのディスク容量が不足している場合に中断します。パソコン側のディスク容量を確保して、再実行してください。ただし、中断したファイルは、再実行しても転送されませんので、注意してください。

転送先に同名ファイルがある場合。転送先に同じ名前のファイルがある場合に転送を中断します。書き換えるか、スキップするか、選択できます。

転送中止

転送を中断、中止したファイルは再度転送しないとインストールされません。転送終了後にファイルがすべてあるかどうか確認してください（slaveコマンドを入れて32ファイル）。

PCエンジン

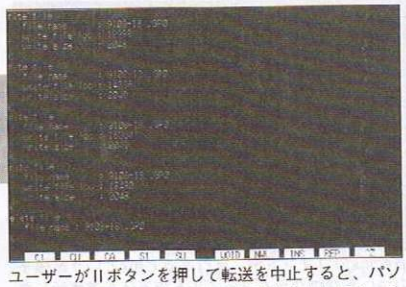


転送中にIIボタンを押すと転送を中止します。Iボタンでつぎのファイルへ、IIボタンで戻ります

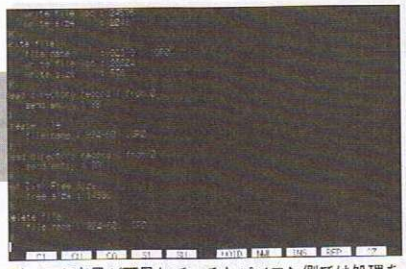
転送先のパソコンのディスク容量が不足していると写真のようなメッセージが表示されます

同名ファイルがあると転送を中断します。上書きしてよければIボタン、スキップするときはIIボタン

パソコン



ユーザーがIIボタンを押して転送を中止すると、パソコン側では現在おこなっていた処理を中止します。そのときに途中まで転送していたファイルも消去しますので注意してください



ディスク容量が不足しているとパソコン側では処理を中断します。このとき、ESCキーを押してslaveを抜けて空き容量を作り、立ち上げなおすことができます。ただし、現在のファイルはインストールされません

同名ファイルがあると処理を中断します。上書きを選択すると現在のファイルは消えてしまいますので注意してください。同名ファイルは、スキップすることをお奨めします

でべろ外部コマンド

271ページ以降を参照してください。

プログラムの転送・実行には、execmx というコマンドを使用します。実行したいプログラムファイル（拡張子がmxのファイル）が同じディレクトリに必要です。『SPEEDER』を実行する場合は、コマンドラインから以下のように入力します。

```
execmx speeder.mx
```

CGを表示する場合には、dspgrp というコマンドを使用します。表示するグラフィックファイル（拡張子がgp0のファイル）が同じディレクトリに必要です。9110-14.gp0 のグラフィックを表示する場合は、コマンドラインから以下のように入力します。

```
dspgrp 9110-14.gp0
```

サウンドデータを演奏する場合には、plytrk というコマンドを使用します。演奏するデータファイルが同じディレクトリに必要です。beast.mx のサウンドデータを演奏する場合は、コマンドラインから以下のように入力します。

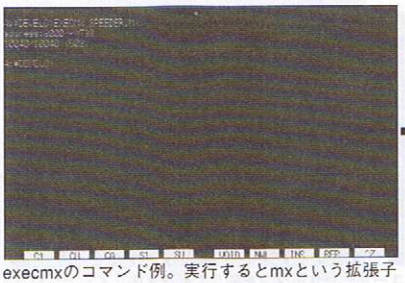
```
plytrk beast.mx pc.mx
```

でべろ

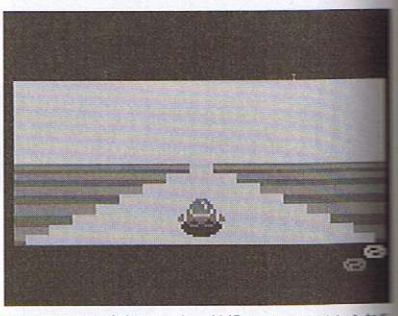
付録CD-ROMのメニューからでべろのコーナーを選択すると、PCエンジンはでべろの初期画面になります。でべろの初期画面では、パソコンからの通信処理待ちになっているので、でべろ外部コマンドなどの実行が可能になります。

でべろ外部コマンドのそれぞれの動作については、第4部でべろ外部コマンドの項に譲りますが、ここではプログラムの転送・実行、CGの表示、サウンドデータの演奏のそれぞれについて、下の写真に例を示します。

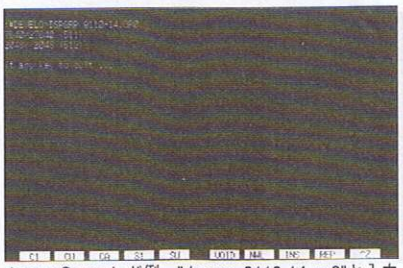
パソコン　　**PCエンジン**



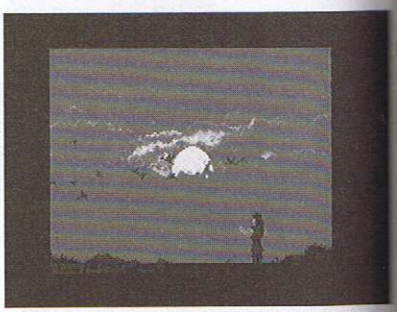
execmxのコマンド例。実行するとmxという拡張子のついたプログラムファイルを読み込んで実行します。終了はプログラムによるため、SPEEDERではリセットされます



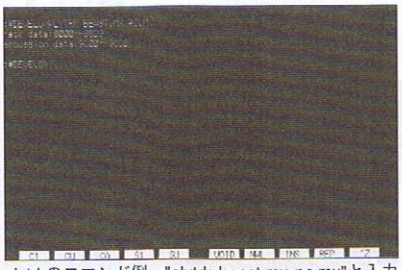
SPEEDERを実行します。付録のCD-ROMから起動するプログラムと変わりません



dspgrpのコマンド例。"dspgrp 9110-14.gp0"と入力すると右の写真のようなグラフィックを表示します。gp0という拡張子のグラフィックファイルを読み込んで表示します



9110-14.gp0を表示中。すべてを読み込んだあと画面いっぱいに表示します



plytrkのコマンド例。"plytrk beast.mx pc.mx"と入力するとサウンドを演奏します。beast.mxというファイルが演奏データでpc.mxというファイルはパーカッションデータです

サウンドデータ演奏中です（PCエンジンの画面ではわかりませんが）。ボタンを押すと終了

付録CD-ROM収録ファイル一覧

ファイル名	容量	ファイルの種別
SPEEDER .MX	24482	サンプルプログラム
BENDBENT.MX	12176	サンプルプログラム
WAVE .MX	1866	サンプルプログラム
SPRITE .MX	1284	サンプルプログラム
SOUND1 .MX	308	サンプルプログラム
SOUND2 .MX	392	サンプルプログラム
FILEREAD.MX	710	サンプルプログラム
PC .MX	512	音楽プログラム
BEAST .MX	5532	音楽プログラム
LABYRINT.MX	7930	音楽プログラム
SAPHIRE .MX	8416	音楽プログラム
STEP .MX	4910	音楽プログラム
TAKOIKA .MX	6716	音楽プログラム
9108-18 .GP0	27200	グラフィックデータ
9110-14 .GP0	27200	グラフィックデータ
9112-6 .GP0	27200	グラフィックデータ
921-30 .GP0	27200	グラフィックデータ
922-74 .GP0	27200	グラフィックデータ
923-9 .GP0	27200	グラフィックデータ
924-60 .GP0	27200	グラフィックデータ
924-77 .GP0	27200	グラフィックデータ
926-15 .GP0	27200	グラフィックデータ
926-44 .GP0	27200	グラフィックデータ
926-56 .GP0	27200	グラフィックデータ
927-32 .GP0	27200	グラフィックデータ
928-27 .GP0	27200	グラフィックデータ
9301-28B.GP0	27200	グラフィックデータ
9302-12 .GP0	27200	グラフィックデータ
9302-39 .GP0	27200	グラフィックデータ
9302-55 .GP0	27200	グラフィックデータ
9303-56 .GP0	27200	グラフィックデータ
9304-15D.GP0	27200	グラフィックデータ
9306-5 .GP0	27200	グラフィックデータ
PC .ASM	2105	音楽プログラムソース
BEAST .ASC	3788	音楽プログラムソース
BEAST .ASM	27913	音楽プログラムソース
BEAST .MML	4709	音楽プログラムソース
LABYRINT.ASC	5205	音楽プログラムソース
LABYRINT.ASM	40465	音楽プログラムソース
LABYRINT.MML	6420	音楽プログラムソース
SAPHIRE .ASC	5993	音楽プログラムソース
SAPHIRE .ASM	43699	音楽プログラムソース
SAPHIRE .MML	6394	音楽プログラムソース
STEP .ASC	3371	音楽プログラムソース
STEP .ASM	25512	音楽プログラムソース
STEP .MML	4980	音楽プログラムソース
TAKOIKA .ASC	4237	音楽プログラムソース
TAKOIKA .ASM	34441	音楽プログラムソース
TAKOIKA .MML	4982	音楽プログラムソース
SPEEDER .ASM	12852	サンプルプログラムソース
SPEEDER .H	1973	サンプルプログラムソース
SPEEDER .DAT	22106	サンプルプログラムソース
BENDBENT.ASM	28337	サンプルプログラムソース
BENDBENT.ASC	4285	サンプルプログラムソース
WAVE .ASM	4202	サンプルプログラムソース
SPRITE .ASM	2222	サンプルプログラムソース
SOUND1 .ASM	958	サンプルプログラムソース
SOUND2 .ASM	1303	サンプルプログラムソース
FILEREAD.ASM	1286	サンプルプログラムソース
LHA213 .EXE	45061	PC-98用解凍ツール（フリーソフトウェア）
EE070 .LZH	93410	PC-98用エディタ（フリーソフトウェア）
MPS .LZH	89610	PC-98用グラフィックツール（製品のお試し版）
WP941SP .LZH	178991	PC-98用通信ソフト（フリーソフトウェア）
WTDOC708.LZH	296849	PC-98用通信ソフトマニュアル（フリーソフトウェア）
DV050 .LZH	45983	PC-98用グラフィックコンバータ（読者制作のツール）
GP0_100 .LZH	11622	PC-98用グラフィックコンバータ（編集部制作のツール）
MML .LZH	26854	PC-98用MMLコンバータ（編集部制作のツール）

ファイル名	容量	ファイルの種別
PMEXT222.COM	15872	MSX用解凍ツール（フリーソフトウェア）
V&ZSMALL.LZH	47875	MSX用エディタ（製品のお試し版）
SIIANIME.LZH	18308	MSX用グラフィックツール（読者制作のツール）
GRACON .LZH	11361	MSX用グラフィックコンバータ（編集部制作のツール）
MMLMSX .LZH	17189	MSX用MMLコンバータ（編集部制作のツール）
CLRSCR .EXE	35315	PC-98用でべろ外部コマンド
DISAS .EXE	47411	PC-98用でべろ外部コマンド
DMPPLT .EXE	37619	PC-98用でべろ外部コマンド
DMPRAM .EXE	37491	PC-98用でべろ外部コマンド
DMPRGB .EXE	37251	PC-98用でべろ外部コマンド
DMPVRAM .EXE	37619	PC-98用でべろ外部コマンド
DSPGRP .EXE	42211	PC-98用でべろ外部コマンド
DVDIR .EXE	39939	PC-98用でべろ外部コマンド
EDTPLT .EXE	54051	PC-98用でべろ外部コマンド
EDTRAM .EXE	54643	PC-98用でべろ外部コマンド
EDTVRAM .EXE	54355	PC-98用でべろ外部コマンド
EXEC .EXE	35955	PC-98用でべろ外部コマンド
EXECMX .EXE	41987	PC-98用でべろ外部コマンド
GETBNK .EXE	35475	PC-98用でべろ外部コマンド
GETFIL .EXE	49635	PC-98用でべろ外部コマンド
GETPLT .EXE	40147	PC-98用でべろ外部コマンド
GETRAM .EXE	40067	PC-98用でべろ外部コマンド
GETVRAM .EXE	40131	PC-98用でべろ外部コマンド
INIT .EXE	35331	PC-98用でべろ外部コマンド
MX2BIN .EXE	37632	PC-98用でべろ外部コマンド
PEAS .EXE	63712	PC-98用でべろ外部コマンド
PERUN .EXE	51363	PC-98用でべろ外部コマンド
PLYTRK .EXE	48051	PC-98用でべろ外部コマンド
SETBNK .EXE	36371	PC-98用でべろ外部コマンド
SETPLT .EXE	41347	PC-98用でべろ外部コマンド
SETRAM .EXE	41331	PC-98用でべろ外部コマンド
SETVRAM .EXE	41315	PC-98用でべろ外部コマンド
SLAVE2 .EXE	48003	PC-98用でべろ外部コマンド
SYSCHG .EXE	49475	PC-98用でべろ外部コマンド
DISAS .TXT	3160	ラベル・マクロ
DEVELO .INC	9229	定義済みファイル
CLRSCR .COM	11728	MSX用でべろ外部コマンド
DISAS .COM	19738	MSX用でべろ外部コマンド
DMPPLT .COM	13742	MSX用でべろ外部コマンド
DMPRAM .COM	13592	MSX用でべろ外部コマンド
DMPRGB .COM	13537	MSX用でべろ外部コマンド
DMPVRAM .COM	13737	MSX用でべろ外部コマンド
DSPGRP .COM	16220	MSX用でべろ外部コマンド
DVDIR .COM	13788	MSX用でべろ外部コマンド
EDTPLT .COM	16271	MSX用でべろ外部コマンド
EDTRAM .COM	17243	MSX用でべろ外部コマンド
EDTVRAM .COM	17008	MSX用でべろ外部コマンド
EXEC .COM	12311	MSX用でべろ外部コマンド
EXECMX .COM	16257	MSX用でべろ外部コマンド
GETBNK .COM	11882	MSX用でべろ外部コマンド
GETFIL .COM	18694	MSX用でべろ外部コマンド
GETPLT .COM	14573	MSX用でべろ外部コマンド
GETRAM .COM	14541	MSX用でべろ外部コマンド
GETVRAM .COM	14567	MSX用でべろ外部コマンド
INIT .COM	11696	MSX用でべろ外部コマンド
MX2BIN .COM	12979	MSX用でべろ外部コマンド
PEAS .COM	29576	MSX用でべろ外部コマンド
PERUN .COM	20564	MSX用でべろ外部コマンド
PLYTRK .COM	17542	MSX用でべろ外部コマンド
SETBNK .COM	12388	MSX用でべろ外部コマンド
SETPLT .COM	14510	MSX用でべろ外部コマンド
SETRAM .COM	14504	MSX用でべろ外部コマンド
SETVRAM .COM	14506	MSX用でべろ外部コマンド
SLAVE2 .COM	17249	MSX用でべろ外部コマンド
SYSCHG .COM	15029	MSX用でべろ外部コマンド
P98102SR.LZH	63706	PC-98用でべろ外部コマンドソース
MSX102SR.LZH	60148	MSX用でべろ外部コマンドソース
DEVEL102.LZH	10122	システムのソース
DV102SRC.LZH	18274	システムのCのソース
AUTORUN .LZH	44960	メニュープログラムのソース

お知らせ

でべろBOX販売中！

でべろでソフト開発をおこなうためには、本書のほかにパソコンとPCエンジンをつなぐハード「でべろBOX」が必要です。このでべろBOXは通信販売でのみ扱っております。下の申し込み用紙をご使用になり、お申し込みください。ただいま、でべろBOXは発売記念キャンペーンにつき、特価1万円で販売中です。ぜひ、この機会をご利用ください。

このでべろBOXのセットは、でべろBOX本体に結線されたパッドケーブル以外ケーブル類は付いていません。プリンターケーブル、RS-232Cケーブル等は市販のものを用意してください。その際、プリンターケーブルには、でべろBOXとの接続に36ピンのセントロニクス仕様の端子を持つものを用意してください。RS-232Cは、でべろBOXとの接続に25ピンのストレートケーブルを用意してください。また、MSXとの接続にはジョイポートケーブルを使用します。一端がオス、もう一端がメスになっているケーブルでなかなか手に入りにくいものです。このケーブルは、2000円で販売していますので、お申し込みください。

でべろBOX、MSX用ジョイポートケーブルともに通信販売でのみ販売しています。かならず現金書留で1万円と申し込み用紙を同封のうえ、お申し込みください。

〈あて先〉

〒105　東京都港区東新橋1－1－16　徳間書店インターメディア株式会社
販売部でべろBOX係

までお送りください。発送は、迅速におこなっているつもりですが、でべろBOXの部品の入庫遅れ等で2週間ほどお待ちいただくことがあります。ご了承ください。とくにMSX用のケーブルは、在庫が少なく入手に時間がかかることがあります。

キリトリ線

でべろBOX注文書

お名前	フリガナ		
おところ	フリガナ 〒 電話（　　）　　－		
お持ちの機種に丸をつけてください PC-98・MSX	MSX専用ケーブル 要　・　不要 2000円×　　本 小計　　円	でべろBOX注文台数 10000円×　　台 小計　　円	合計金額 円

月刊PCエンジンファンで作品を発表できます

でべろで作った作品は、月刊誌『PCエンジンファン』編集部あてにお送りください。PCエンジンファン誌にPCエンジン用CD-ROMを付けて、プログラム等、作品発表の場を提供していきます。採用作品には、当社規定の投稿謝礼、優秀な作品は、市販ゲーム化への道もあります。

●投稿ジャンルについて

- ゲームプログラム／投稿プログラムの王道。パソコンのゲームとは一味違うゲーム機の投稿ゲームをお待ちしています。
- ツールプログラム／でべろでのプログラム開発をもっと便利にする支援ツールをとくに期待しています。PCエンジンの256色や512色が扱えるグラフィックツールやWindows用のでべろシステムなどをよろしく。
- グラフィック／16色～512色まで、オリジナリティの高いCG作品を待ってます。
- サウンド／オリジナル楽曲のほうが採用される確率が高いです。音楽著作権に触れないものをお願いします。

投稿上の注意

封筒のあて先の「○○」のところにはそれぞれのコーナー名を入れて応募してください。盗作や二重投稿（他の出版社などに同じ作品を送ること）はぜったいにやめてください。なお、応募作品は一切返却できません。また、採用作品の版権は徳間書店インターメディアに帰属するものとします。

締め切りは、とくにありませんが、5月～6月に応募作品をまとめたCD-ROM付きムックを発刊する予定です。この本の締め切りは、3月～4月ごろです。

キリトリ線

投稿応募用紙

作品名		使用機種	□PC-98シリーズ □MSXシリーズ □その他（　　）
氏名	フリガナ	ペンネーム	フリガナ
住所	□□□-□□　フリガナ 都道府県		
電話	―	ID	□ニフティサーブ □PC-VAN

●投稿ジャンル
□ゲームプログラム部門
□ツールプログラム部門
□グラフィック部門
□サウンド部門
●フロッピーディスクのフォーマット形式
□640Kbyte
□720Kbyte
□1.2Mbyte
□1.44Mbyte
□その他
●作品の説明
※操作方法などを解説した文章も添付してください
●作品のコメント

※この用紙は自由にコピーしてお使いください

サポートについて

でべろに関するお問い合わせは、平日の午後4時から6時までの間、下記電話番号で受け付けております。ただし、ゲームの作り方やCGの描き方といったご質問にはお答えできません。テクニカルなご質問は、なるべく文書でお送りください。月刊PCエンジンファン誌上でお答えしていく予定です。

本書をご購入の皆様には、付属のユーザー登録はがきにてご登録くださいますようお願いいたします。バージョンアップやでべろ関連の新刊本のご案内をお送りいたします。でべろのバージョンアップは、月刊PCエンジンファン誌の付録CD-ROMを通じておこないます。月刊PCエンジンファンのでべろ情報にご注目ください。

でべろ質問電話
03-3573-8644

※平日の午後4時～6時の時間帯に受け付けます

パソコン通信で最新のでべろ情報を

でべろは、パソコン通信での投稿の受け付けやサポート等をおこなう予定です。大手ネットのPC-VANとニフティサーブに会議室を設けて、でべろに関するさまざまな情報交換をリアルタイムでおこなえるようにします。ユーザー同士の情報交換はもちろん、でべろの最新情報やアップデートプログラムなども登録し、自由にダウンロードしていただけるようになります。

例えばニフティサーブではSFULABO（フューチャー・ラボ・ステーション）に会議室が設置される予定ですが、このような会議室開設のお知らせは、月刊PCエンジンファンの誌上などで報告していきます。

設置された会議室では、でべろのシステムのバージョンアップに伴うアップデートサービスや、ちょっとしたツール類をフリーウェアとして配布したりする予定です。また、会議室にお寄せいただいた、でべろのシステムに対するご意見・ご要望などは、今後のシステム向上の参考にさせていただきます。

でべろの今後の展開は、パソコン通信と月刊PCエンジンファンという2つの媒体をメインにしていきます。今までパソコン通信の経験がなかったかたも、これを機に、新しい情報がかんたんに引き出せるパソコン通信に、ぜひとも加入してみてください。

バージョンアップ

システムやツールコマンド類は、予告なくバージョンアップされる可能性があります。このようなとき、パソコン通信によって、より早く新しいバージョンを入手することが可能になります。

質問の受け付け

でべろに関するご質問は、パソコン通信上に設置された会議室にお寄せいただいても構いません。ただしすべての質問にはお答えできませんので、その際はご容赦ください。

読者のみなさんへのお願い

「でべろ」のCD-ROM、フロッピーディスクのパッケージを開ける前に必ずお読みください。

このたびは「でべろスターターキット・アセンブラ編」のご購入ありがとうございます。
今回みなさんに「でべろスターターキット・アセンブラ編」（本と付録のCD-ROMとフロッピーディスク）を通じて提供している情報（PCエンジンソフトの開発環境等）は、みなさんの個人的なPCエンジンのソフト制作のために、行われています。したがって、個人的な使用範囲にのみ使用を限ります。また、パッケージを開けたときに、読者のみなさんには以上のような使用条件のなかでのみの使用許諾を行うものとします。
なお、「でべろスターターキット・アセンブラ編」に収録している内容は、日本の著作権法上、その著作権のすべてが制作者である、日本電気ホームエレクトロニクス株式会社、株式会社ハドソン、徳間書店インターメディア株式会社等に帰属しています。また、前述の使用条件により、このソフトで制作した作品（ゲーム、ツール類、CG、音楽演奏、その他全プログラム等）での商業行為は禁じます。

トクマインターメディアムック **でべろスターターキット・アセンブラ編**

平成8年2月20日発行
定価5000円（本体価格4854円）
発行人●山森尚
編集人●北根紀子
編著●PCエンジンファン編集部（福成雅英、伊藤明照、三浦利夫、高嶋融）
発行●徳間書店インターメディア株式会社
〒105 東京都港区東新橋1-1-16
TEL.03-3573-8606（編集部）　TEL.03-3573-8613（販売部）
発売●株式会社徳間書店
〒105 東京都港区東新橋1-1-16
TEL.03-3573-0111（大代表）
印刷●大日本印刷株式会社

■Staff

AD＆表紙デザイン●TERRACE.Inc
本文デザイン●TERRACE.Inc
Geoge Eagle International Inc.
イラスト●しまづ★どんき
校閲●上野利幸
DEVELO開発チーム●飯田崇之、奥義之、諸橋康一
中村文雄、大野尚志
CD-ROMプログラム●飯田崇之
CD-ROMテーマミュージック制作●上野利幸

■Special Thanks

日本電気ホームエレクトロニクス株式会社
株式会社ハドソン
ハドソンコンピュータデザインスクール株式会社
未来計画株式会社

■付録CD-ROM・フロッピーディスク

●フリーソフトウェア作者
LHA／吉崎栄泰
PMext／ぺぱあ☆みんと
WTERM／H.INOUE
EditEngine／T.Gaito

●投稿作品より
（徳間書店刊／雑誌「MSX・FAN」）

CG作者
走りだしたらとめられない／猪口亮
CHINA／猪口亮
地図にない街／ぴなそめZ
あわわっ／まいきー
AKIKARA FUYU NI／実石哲也
時の人／南雲太郎
反逆の人形／ぴなそめZ
戦場の傷跡／三浦一誠
〜銘菓「ひ●こ」〜／ちづかの弟
にゃんぴ／くにはる
ねずみとねこ／南雲太郎
いってらっしゃいにゃ〜〜ん！／亜ず
メジロ／五藤雅士
つまんない！／佐々木達也
時給750円／SABUROWTAA
メカニック・ラボ／Rock Cut
きらくにしたまえ／M.PROJECT
HURRY!／HURRY
はいほー!／野沢智美
組織の男／瓦屋竹左衛門

音楽演奏プログラム作者（原曲）
THE Labyrinth Of Devil／松村弘和
Step！／なると
メカナイズド・ビースト／大家のケンちゃん
砂漠のサファイア／大家のケンちゃん
タコ＆イカ SUPER／みそねこ

オリジナルゲーム作者
SPEEDER／OZO
BENDBENT／Toxsoft

ツール作者
v&z small（MSX用テキストエディタ）／VIPER
SII-ANIME（MSX用CGツール）／SII
DV050（98用CG転送ツール）／South Section

●特別収録
マルチペイント試供版（98用CGツール）／
株式会社シー・ラボ

DEVELO

発行／徳間書店インターメディア株式会社　発売／株式会社徳間書店　**定価5000円**（本体4854円）
T1066450935001　雑誌66450-93