
ビジュアルメモリ チュートリアル

Visual Memory Tutorial Revision 1.00



Dreamcast™

はじめに

弊社のハードウェア用アプリケーション開発にご協力いただき誠にありがとうございます。
本書は、ソースコード書き方からツールの組み合わせかた、ビジュアルメモリへのプログラムの転送、Dreamcast の起動画面との連携についてをまとめたものです。
Dreamcast との連携は、インフォメーションフォークと呼ばれるデータ構造を用いて図ります。また転送ユーティリティの起動や操作方法についてもまとめてあります。

本書の内容

アプリケーションの開発手順

ソースコードを書き始める際の注意事項や、最終的にビジュアルメモリに転送するバイナリファイルを生成するまでに使うさまざまなツールの組み合わせ方を解説しています。
開発の大きな流れをこの章でつかんでください。

ビジュアルメモリと Dreamcast の連携

Dreamcast の起動画面と連携を取るためにしなければならないこと、起動画面の説明、実現する為の手段やファイル構造などについてを説明しています。

Memory Card Utility

PC から Dev.Box を経由してビジュアルメモリへプログラムを転送する方法や、ツールの起動方法、そして操作方法を説明しています。

付録

付録として、Dreamcast の BOOT ROM に内蔵しているラベルアイコン一覧と、ビジュアルメモリ SDK に付属しているサンプルプログラムを掲載しています。

ご注意

本書に記載されている事項は、将来予告なしに変更することがあります。
ビジュアルメモリおよびマニュアルを運用した結果の影響については、いっさい責任を負いかねますのでご了承ください。

商標

SEGA、ドリームキャスト、Dreamcast、ビジュアルメモリは、株式会社セガ・エンタープライゼスの商標です。

Microsoft、Windows、WindowsCE ロゴは、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

その他、本文中に記載する製品名は、一般に開発メーカーの商標または登録商標です。なお、本文中では TM よび (R) マークは明記しておりません。

改版履歴

1998 年 11 月 30 日 初版発行

Copyright (C) 1998 株式会社セガ・エンタープライゼス

Copyright (C) 1998 ASCII Corporation

編集・製作 株式会社アスキー AAP 書籍編集部

他のマニュアルとの関係

『ビジュアルメモリ ハードウェアマニュアル』

ビジュアルメモリのハードウェアとシステム BIOS の仕様をまとめた技術資料です。巻頭の概要編には、ビジュアルメモリのスペックを簡単にまとめたものを掲載しています。

ゲームデザイン担当の方は概要編を、プログラム担当の方は全般をご覧ください。

『ビジュアルメモリ プログラマーズガイド』

ビジュアルメモリに搭載されている三洋電機社製 LC86700 用のアセンブラ、リンカ、ライブラリマネージャ、ビジュアルメモリ用の実行ファイルを作成する“ E2H86K.EXE ”のインストールから使い方までをまとめたマニュアルです。また、LC86700 の命令セットやアセンブラの文法についても解説しています。

ビジュアルメモリシミュレータをはじめとしたビジュアルメモリ SDK のインストール手順は、こちらをご覧ください。

『ビジュアルメモリシミュレータガイド』

ビジュアルメモリの動作をソフトウェアにてエミュレーションする「ビジュアルメモリシミュレータ」を解説しています。

『SEGA ライブラリリファレンス Vol.2』

Dreamcast からビジュアルメモリへのデータやプログラムを転送したり、Dreamcast コントローラに接続したビジュアルメモリの LCD に画像を表示させる場合などは、このマニュアルを参照してください。

Dreamcast のゲームデータをビジュアルメモリに読み書きする場合は、Shinobi ライブラリのバックアップ関数をご覧ください。

LCD への描画については、Shinobi ライブラリの LCD 関数をご覧ください。

制限事項

別途ご用意いただくもの

Memory Card Utility では、RS-232C リバースケーブル、Windows 用の通信ソフトを必要とします。これらは、弊社の SDK には付属しておりませんので、別途ご用意ください。

なお、Memory Card Utility は、Dev.Box の SET 5.2X 以上でのみ動作します。SET 5.1X 以前の Dev.Box では動作しませんので注意してください。

また、実行にあたって CodeScape と GD Workshop が必要になります。こちらは、Dreamcast SDK に付属しておりますので、あらかじめインストールを行ってください。

テクニカルサポートご案内

開発にあたって技術的なご質問や装置の不具合、またマニュアルの不備や装置の故障などがございましたら、下記連絡先までご一報ください。

株式会社セガ・エンタープライゼス
テクニカルサポートセンター
〒144-8531 東京都大田区羽田 1-2-12
電 話：03-5736-7355
F A X：03-5736-5357
E-mail：katana@sft.sega.co.jp

contents

はじめに	2
他のマニュアルとの関係	4
制限事項	5
テクニカルサポートご案内	6

第1章

アプリケーションの開発手順 11

1.1	ソースコードを書く	11
1.2	“ GHEAD.ASM ”の修正	12
1.3	MAKEを使わないアセンブル	12
1.3.1	アセンブル	13
1.3.2	リンク	14
1.3.3	EVAファイルをHEXファイルに変換	15
1.3.4	HEXファイルをバイナリファイルに変換	16
1.3.5	MAKEファイルを作る	16
1.4	インフォメーションフォークの作成	18
1.5	ビジュアルメモリへの転送	18

ビジュアルメモリとDreamcast の連携 19

2.1	起動画面各部の名称	20
2.1.1	メモリ選択画面	20
2.1.2	ファイル管理画面	22
2.2	ボリュームアイコンの作成	25
2.3	アニメーションアイコンの作成	27
2.3.1	3つのファイル構造	27
2.3.2	インフォメーションフォーク	28
2.3.3	ビジュアルコメントデータの構造	33
2.3.4	ゲーム名のソート基準	34

Memory Card Utility 39

3.1	Memory Card Utility の準備と起動	39
3.1.1	転送に必要なもの	39
3.1.2	ソフトウェアの準備	42
3.1.3	Memory Card Utilityの起動	45
3.2	Memory Card Utility の操作方法	50
3.2.1	メインメニュー	50
3.2.2	メモリ選択メニュー	51
3.2.3	コマンド選択メニュー	52
3.2.4	ファイル操作メニュー	59
3.3	ビジュアルメモリの初期化	64
3.4	PCからビジュアルメモリへのファイル転送	67

付録 A

リトルエンディアン形式

73

付録 B

ラベルアイコン一覧

75

付録 C

サンプルプログラムリスト

77

C.1	LCD パターン表示	77
C.2	LCD キャラクタパターン表示	80
C.3	ベースタイマー割り込みを使ったカウンタ	84
C.4	ボタン押下の検出	90
C.5	PWM 音源の利用	95
C.6	タイマー 0 を使った割り込み	97
C.7	シリアル通信 (送信側)	101
C.8	シリアル通信 (受信側)	107
C.9	汎用シリアルドライバ	113
C.10	フラッシュメモリの読み書き	123
C.11	ローバッテリー検出とデータ退避	129

第 1 章

アプリケーションの開発手順

ここでは、すでにアプリケーションの仕様があるものとして、プログラムのコーディングから実機チェックまでを説明します。

1.1 ソースコードを書く

プログラムの先頭には、必ず次の宣言文を記述します。

```
chip LC868700
world external
public main
extern _game_end
```

ビジュアルメモリ用のアプリケーションは、すべてフラッシュメモリに配置されるので、world では必ず external を宣言してください。

また、システム BIOS からアプリケーションが呼び出されると、フラッシュメモリの 0000H 番地が呼び出されます。0000H 番地は、GHEAD.ASM にて jmp main と記述されていますので、アプリケーションはゲームモードの突入用の main というラベルを用意します。main は GHEAD.ASM から参照されるので、public 宣言をしておきます。

逆にアプリケーション終了時には、GHEAD.ASM の _game_end にジャンプしますので、このラベルが外部にあることを extern 宣言します。

アプリケーションでフラッシュメモリ関連や時計関連の BIOS を呼び出す場合は、fm_wrt_ex, fm_vrf_ex, fm_prd_ex などが、外部プログラムである旨を extern 宣言します。

次にデータセグメント (DSEG) の間接アドレスレジスタの構造を定義します。データセグメントは、すべて RAM に展開されます。なお、RAM の 0000 ~ 000FH までは間接アドレスレジスタなので、アプリケーションで間接アドレスレジスタを使う・使わないに関わらず、16 バイト分の RAM を確保してください。アプリケーションで利用できる RAM 領域は、0010H 以降です。

参照

間接アドレスレジスタの詳細については『ビジュアルメモリハードウェアマニュアル』を参照してください。

コードセグメント (CSEG) は、0280H より上位のアドレスからスタート (org 280H) させてください。0000H ~ 01FFH までは GHEAD.ASM が、0200H ~ 027FH (最小) まではインフォメーションフォークが利用します。

“ GHEAD.ASM ”には、割り込みベクタの定義と BIOS の呼び出し・戻り先などが定義されています。一方、インフォメーションフォークには、アプリケーション名やアイコンなどの情報を定義します。インフォメーションフォークの大きさが不定なのは、アイコンをアニメーションさせたり、大きめのアイコンを挿入したりといったことが選択できるためです。

あらかじめ、インフォメーションフォークのイメージが決まっている場合は、0200H にそのサイズを加えたアドレスからプログラムをスタートさせてください。



インフォメーションフォークの詳細については「第2章 ビジュアルメモリと Dreamcast の連携」以降を参照してください。

1.2 “ GHEAD.ASM ”の修正

アプリケーションのソースを書き終えたら“ GHEAD.ASM ”を修正します。

もしアプリケーションで割り込みを利用する場合は、利用する割り込みのベクタテーブルを“ GHEAD.ASM ”に記述します。

割り込みを利用しない場合でも、割り込みベクタテーブルの定義が必要です。また、割り込みハンドラには、なにもしないで RETI するように記述してください。

なお、ユーザーがゲームモードを選ぶと“ GHEAD.ASM ”の先頭 (0000H) にジャンプしてきますので、“ GHEAD.ASM ”の先頭にはゲームのメインルーチンへのジャンプ命令を記述します。

100H 以降は、BIOS 呼び出しを行う場合の処理が記述されています。この処理は、変更しないでください。ROM 内 BIOS が直接アドレスを指定して、フラッシュメモリへ制御を戻してくるため、アドレスが1バイトでも変更されると、正しく BIOS を呼び出せません。

特にフラッシュメモリの読み書き時には、システムクロックを RC 発振 (1/6 分周) にする必要がありますが、変更はアプリケーションプログラム内で行って、fm_wrt_ex, fm_vrf_ex, fm_prd_ex を呼び出し、戻ってきたら水晶発振に切り替わるようにしてください。

また、各 BIOS のスタートアドレスを指定する org 命令も変更しないように注意してください。

1.3 MAKE を使わないアセンブル

ここでは、ソースコードをアセンブル・リンクし、実際にビジュアルメモリで実行できる形式のファイルにビルドします。

注意

アセンブラやリンクは、EMS を利用します。あらかじめ MS-DOS プロンプトのプロパティを表示させ、[メモリ] タブの [EMS メモリ] グループで、EMS メモリを使えるようにしてください。

CONFIG.SYS の EMM386.EXE の組み込みで、NOEMS オプションを指定している場合、EMS メモリが利用できないので、NOEMS オプションを削除する必要があります。

1.3.1 アセンブル

MS-DOS のコマンドプロンプトから、M86K コマンドを実行しソースコードをアセンブルします。

たとえば、ソースコードのファイル名が“ TEST.ASM ”の場合は、カレントドライブ、カレントディレクトリを“ TEST.ASM ”のあるディレクトリにし、次のようにしてアセンブルします。

```
C>M86K TEST.ASM
```

```
SANYO (R) LC86K series Macro Assembler Version 4.0K
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights reserved.
```

```
Pass 1 .....
Source file:      TEST
Chip name:        LC868700
ROM size:         60K bytes
RAM size:         512 bytes
XRAM size:        196 bytes
Pass 2 .....
```

アセンブルが終わると、拡張子が“ .OBJ ”というオブジェクトファイルが作成されます。

また“ GHEAD.ASM ”も同様にしてアセンブルし、“ GHEAD.OBJ ”を作ります。

アセンブルの途中で次のようなエラーメッセージが表示されたら、メッセージに表示される番号の行に何らかの問題があることを示しています。

```
Pass 1 .....
TEST.ASM(93):                move                #080h,b
** Error, syntax error near #
0 warning(s) and 1 error(s) were detected. Further execution aborted.
```

ソースコードを修正して、警告やエラーが発生しないように修正してください。

参照

アセンブラ M86K の警告メッセージやエラーメッセージについては『ビジュアルメモリプログラマーズマニュアル』を参照してください。

注意

“ GHEAD.ASM ”相当のソースを自分のソースに埋め込む場合を除き、“ GHEAD.OBJ ”とユーザープログラムのオブジェクトファイルが必要です。なお“ GHEAD.ASM ”には、割り込みベクタや割り込みサービスルーチン、BIOS 呼び出しプログラムなどが記述されており、次に実行するリンクによってユーザープログラムの下位アドレスに配置されます。

1.3.2 リンク

アセンブラによって作成したオブジェクトファイルと、BIOS が書き込まれている内部 ROM のアドレスを示すための“ GDUMMY.OBJ ”ファイルを用意し、リンクを使って EVA 形式のファイルを作成します。

EVA 形式のファイルは、専用のデバッグ用ハードウェアを利用するためのファイルです。ビジュアルメモリ用アプリケーションの開発においては、ビジュアルメモリシミュレータを利用しますので、一時的なテンポラリファイルと考えてください。

リンクを実行する前に“ GDUMMY.OBJ ”のパスをメモに控えておきます。コマンドラインから次のように入力し、それぞれのオブジェクトファイルをリンクします。

```
D>L86K GHEAD.OBJ IFORK.OBJ TEST.OBJ -C=200 C:%VM_SDK%\LC86K\%OBJ%GDUMMY.OBJ,
TEST.EVA,,,
```

```
SANYO (R) LC86K series Linkage Loader Version 6.00c
Copyright (c) SANYO Electric Co., Ltd. 1989-1997. All right reserved.
```

```
Pass 1 ...
Pass 2 ...
Pass 3 ...
```

```
Link process complete !!
TEST.EVA created
```

-C=200 というオプションは、その直後に指定するユーザープログラムをフラッシュメモリの何番地に配置するかを指定するオプションです。

エラーメッセージが表示された場合は、“ GHEAD.ASM ”やユーザープログラムのソースを見直します。BIOS を呼び出す際のラベルを中心にチェックしてみてください。

1.3.3 EVA ファイルを HEX ファイルに変換

リンカによって、拡張子“ .EVA ”という 1 つのファイルにまとめられました。このファイルをビジュアルメモリやビジュアルメモリシミュレータで読み込めるファイルに変換します。コマンドラインから、次のように入力します。

```
D>E2H86K TEST.EVA
SANYO LC86000 Series EVA-file to HEX-file generator V1.21A
Copyright (C) SANYO Electric Co.,Ltd. 1992-1997

EVA file name: TEST.EVA
ROM data packed: FF(hex)
Chip name: LC868716

All ROM(64KB) block records: 03875
All ROM(64KB) block records: 04096
Module name: GHEAD External CSEG(In) 0000 - 0002 records: 00001
Module name: External CSEG(In) 0003 - 0004 records: 00001
Module name: External CSEG(In) 000B - 000C records: 00001
Module name: External CSEG(In) 0013 - 0014 records: 00001
Module name: External CSEG(In) 001B - 001C records: 00001
Module name: External CSEG(In) 0023 - 0024 records: 00001
Module name: External CSEG(In) 002B - 002C records: 00001
Module name: External CSEG(In) 0033 - 0034 records: 00001
Module name: External CSEG(In) 003B - 003C records: 00001
Module name: External CSEG(In) 0043 - 0044 records: 00001
Module name: External CSEG(In) 004B - 0057 records: 00002
Module name: External CSEG(In) 0100 - 0105 records: 00001
Module name: External CSEG(In) 0110 - 0115 records: 00001
Module name: External CSEG(In) 0120 - 0125 records: 00001
Module name: External CSEG(In) 0130 - 013B records: 00001
Module name: External CSEG(In) 01F0 - 01F4 records: 00001
Module name: IFORK External CSEG(In) 01F5 - 0474 records: 00041
Module name: TEST External CSEG(In) 0475 - 051C records: 00011
```

オプションスイッチなどはありません。

このコマンドを実行すると、拡張子が“ .H00 ”と“ .HEX ”というファイルが作成されます。

拡張子	内容
H00	このファイルは、ビジュアルメモリシミュレータで読み込むファイルです。コード本体だけが保存されていますので、読み込み時間を短縮できます。
HEX	フラッシュメモリのバンク 0 のイメージ 64K バイトがそのまま保存されます。 どんなに小さなプログラムでも 64K バイト分のファイルが作成されます。プログラム本体以外の領域は、すべて 00H で埋められます。 このファイルは、ビジュアルメモリシミュレータに読み込めますが、正常に動作しないので注意してください。

注意

通常は、H00 ファイルのみを使います。

H00 ファイルができあがったところで、ビジュアルメモリシミュレータにて動作チェックを行います。ただし、ビジュアルメモリシミュレータは、実機と同じクロックを持ち合わせていないため、タイミングのチェックには不向きです。

プログラムのロジカルチェックはシミュレータ、タイミングや速度をチェックする場合は実機というように、デバッグのフェーズごとに分けて利用してください。



ビジュアルメモリシミュレータについては『ビジュアルメモリシミュレータガイド』を参照してください。

1.3.4 HEX ファイルをバイナリファイルに変換

E2H86K で変換した H00 ファイルを、H2BIN.EXE でバイナリファイル(拡張子“ .BIN ”)に変換します。

コマンドラインから、次のように入力します。

```
H2BIN TEST.H00 TEST.BIN
```

オプションスイッチなどはありません。第 2 パラメータの“ TEST.BIN ”は省略してもかまいません。省略した場合は、拡張子が自動的に“ .BIN ”になります。

これでビジュアルメモリの実機に読み込ませるファイルができました。

1.3.5 MAKE ファイルを作る

MAKE コマンド用に、MAKE ファイルを作成すると、アセンブル・リンク・ファイル形式の変換という一連の手順を、バッチ処理化することができます。

どのコマンドにどんなファイルを入れると、どんなファイルが出力されるかという、依存情報を MAKE ファイルとして記述し、MAKE コマンドを実行すると、入れるファイルと出力されるファイルのタイムスタンプを比較し、更新のあったファイルのみをアセンブル・リンクします。

MAKE コマンドの詳細は『ビジュアルメモリ プログラマーズマニュアル』を参照してください。

注意

MAKE コマンドは、さまざまな開発環境に添付されているので、お使いのコンピュータに複数の開発環境がインストールされている場合は、コマンド検索パス(環境変数 PATH)の順序を変更するか、MAKE コマンドのファイル名を変更するなどしてください。

たとえば、先の一連の手順を MAKE するには、次のようなメイクファイルを作成します。ここでは、ファイル名を“ TEST.MAK ”とすることにします。

```
TARGET = test
OBJECTS = ifork.obj test.obj
HEADOBJ = ghead.obj
SYSOBJ = $(TOOL86)¥obj¥gdummy.obj

.asm.obj:
    m86k $*

$(TARGET).eva: $(HEADOBJ) $(OBJECTS)
    186k $(HEADOBJ) $(SYSOBJ) $(OBJECTS),$(TARGET).eva,,,

$(TARGET).h00: $(TARGET).eva
    e2h86k $(TARGET)

$(TARGET).hex: $(TARGET).h00
    h2bin $(TARGET).h00
```

このメイクファイルを次のように MAKE に指定してビルドを行います。こうすることで、更新のあったソースファイルがアセンブルされビルドが行われます。

注意

作成した MAKE ファイルは、/F オプションを付けて MAKE を実行してください。コマンドラインから MAKE /F < MAKE ファイル名> というように入力します。

```
D>MAKE /F TEST.MAK
```

```
SANYO LC86000 Series MAKE Utility Version 1.00A
Copyright (C) SANYO Electric Co.,Ltd. 1993-1994 All rights reserved.
```

```
m86k GHEAD
```

```
SANYO (R) LC86K series Macro Assembler Version 4.0K
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights reserved.
```

```
Pass 1 .....
Source file:      GHEAD
Chip name:        LC868700
ROM size:         60K bytes
RAM size:         512 bytes
XRAM size:        196 bytes
Pass 2 .....
```

```
m86k IFORK
```

```
SANYO (R) LC86K series Macro Assembler Version 4.0K
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights reserved.
```

《以降、リンカ E2H86K H2BINが実行され、バイナリファイルが生成されます》

1.4 インフォメーションフォークの作成

ビジュアルメモリ SDK のサンプルでは、IFORK.ASM を作成しバイナリを作成しています。バイナリエディタなどを用いて、できあがったバイナリファイルのインフォメーションフォークを埋め込み方法でもかまいません。

インフォメーションフォークには、Dreamcast のファイル管理画面で表示されるアイコンやアプリケーション名称、ビジュアルメモリのファイルモードで表示されるコメント、ゲーム名（ソートキー）、大きめのアイコンを使ったコメントなどを埋め込みます。

そのうち必須項目は、VM コメントデータ、GUI コメントデータ、ゲーム名、アイコン数、ビジュアルタイプ、アイコン情報（最低 1 つ）です。

「第 2 章 ビジュアルメモリと Dreamcast の連携」を読みながら、インフォメーションフォークを編集してください。

1.5 ビジュアルメモリへの転送

インフォメーションフォークの編集が終わったら、ファイルをビジュアルメモリに転送します。転送にはビジュアルメモリ SDK に添付されている「Memory Card Utility」を使います。

注意

Memory Card Utility は、Dev.Box 用のアプリケーションで ELF ファイル形式で提供されています。一般のパソコン用のプログラムではないので注意してください。

転送にあたって、次のものが必要になります。

- ① RS-232C クロスケーブル
- ② Windows 用通信プログラム
- ③ CodeScape などのデバッガ
- ④ GD Workshop
- ⑤ Dev.Box（SET 5.2X 以上）

①②については、弊社提供の SDK には添付されていないので、別途用意してください。

参照

転送方法の詳細については「第 3 章 Memory Card Utility」を参照してください。

ビジュアルメモリと Dreamcast の連携

ここでは、次のような Dreamcast の起動画面とビジュアルメモリのインターフェイスについて説明します。このインターフェイスは BOOT ROM に実装されており、Dreamcast 起動時に表示されるメニュー画面に深くかかわってきます。

そこでメニュー画面の説明に続いて、それを実現するためのファイルのデータ構造を説明します。

**注意**

このマニュアルでは、BOOT ROM Version 1.001 を元に説明します。今後のバージョンアップにより、一部機能や名称などが変更される場合があります。
なお、実機では画面右上のバージョン番号は表示されません。

2.1 起動画面各部の名称

Dreamcast の電源を入れるとオープニングアニメーションの後に、先に掲載した画面が表示されます。これを「メインメニュー」と呼びます。メインメニューは、Dreamcast に内蔵の BOOT ROM によって制御・管理されています。

2.1.1 メモリ選択画面

メインメニューからビジュアルメモリが表示されているファイルを選ぶと、次の画面が表示されます。



この画面を「メモリ選択画面」と呼びます。1998 年 11 月 30 日現在は「メモリ＝ビジュアルメモリ」ですが、今後新しい記憶媒体が発売される場合もあります。

複数のコントローラやメモリが Dreamcast およびコントローラに接続されている場合は、その一覧が表示されます。

注意

音声認識装置など記憶媒体以外の機器は表示されません。

一覧を見ると怪獣の画像が入ったもの、動物の画像が入ったもの、画像が入っていないものがあります。メモリにユニーク（メモリ 1 つに対して 1 つのアイコン）なアイコンを付けることができます。

このアイコンには 2 種類あり、エンドユーザーが独自に付けられるものと、特殊なファイルをビジュアルメモリに書き込むことで表示されるものがあります。中央下のアイコンは、初期化されていないメモリを意味しています。

ラベルアイコン

エンドユーザーが独自に付けられるアイコンは「ラベルアイコン」と呼びます。エンドユーザーはメモリを初期化した際に、BOOT ROM に内蔵されている 124 種類のアイコンの中から、自由にアイコンを付けることができます。ビジュアルメモリの場合は、画面と同じアイコンがビジュアルメモリの LCD にも表示されます。

なお、1 つのメモリに対してラベルアイコンと、後述のポリウムアイコンが付けられている場合は、ポリウムアイコンが優先して表示されます。

参照

BOOT ROM に内蔵のアイコンは、BOOT ROM フォント関数を利用することでパターンデータを読み出すことが可能です。BOOT ROM フォント関数については『SEGA ライブラリマニュアル Vol.2』を参照してください。ラベルの一覧は「付録 ラベルアイコン一覧」を参照してください。

ポリウムアイコン

怪獣のアイコンが「ポリウムアイコン」です。ポリウムアイコンは、ビジュアルメモリ内に“ICONDATA_VMS”というファイルを保存することで実現できます。したがって、エンドユーザーはポリウムアイコンを付けることはできません。

このアイコンは、32 × 32 ドットで 65,536 色 (ARGB4444) 中 16 色を使ったグラフィックを表示できます。ビジュアルメモリの場合、Dreamcast コントローラに接続すると、画面に表示されているポリウムアイコンと同じグラフィックがビジュアルメモリの LCD に白黒で表示されます。

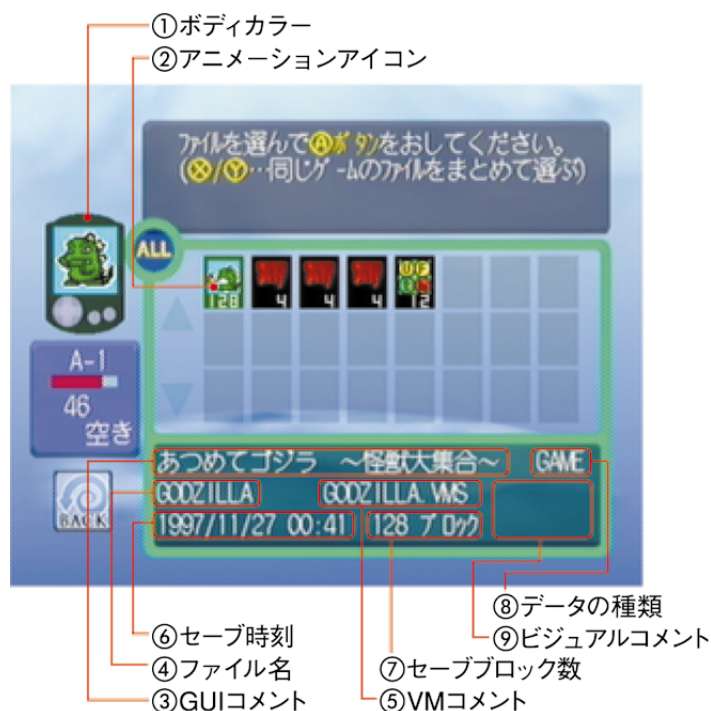
ポリウムアイコンを表示させるためには、“ICONDATA_VMS”というファイルを用意し「Memory Card Utility」を使ってその情報をメモリに転送する必要があります。

参照

ポリウムアイコンの転送方法については「第 3 章 Memory Card Utility」を参照してください。

2.1.2 ファイル管理画面

メモリ選択画面からメモリを選ぶと、次の画面が表示されます。この画面を「ファイル管理画面」と呼びます。



この画面には、選択したメモリに保存されているアプリケーションや、Dreamcast のゲームのセーブデータの一覧が表示されます。

ファイルを選ぶと、そのファイルについての詳しい情報が画面下に表示されます。

① ボディカラー

メモリ 1 つに 1 つの色を付けることができます。この色の指定は、メモリを初期化する際に指定できます。エンドユーザーにも変更が可能です。なお、色情報はビジュアルメモリ内に持っていますので、Dreamcast からリアルタイムに変更することはできません。

② アニメーションアイコン

32 × 32 ドットのアイコンで、65,536 色中 16 色を使ったグラフィックが表示できます。また、最大 3 パターンのデータを用いたアニメーションを表示させることが可能です。

アイコン 1 つが 1 つのファイルを表しています。選択すると枠の部分が黄色に点滅し、画面下に選択したファイルの詳細な情報が表示されます。アプリケーションファイルの場合は緑、データファイルの場合は黒の枠が表示されます。

× ボタンや Y ボタンで複数のファイルを選択した場合は、選択したファイルの GUI コメントとブロック数の合計が表示されます。

③ GUI コメント

半角文字のアルファベット、数字、記号、カナと、全角文字を表示できます。文字列の長さは 32 バイトです。利用できる半角文字はアスキーコードで 20H ~ 7EH, 0A1H ~ 0DFH までです。全角文字はシフト JIS コードです。

JIS 第2水準漢字もサポートしています。また、JIS X 0208-1983 対応なので音符などの記号類も表示できます。

④ ファイル名

メモリにどのようなファイル名で保存されているかが表示されます。ファイル名として利用できる文字は、次の表のアミがかかっていない部分です。

		下位4ビット															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
上位4ビット	0			空白	0	@	P	'	p				ー	タ	ミ		
	1			!	1	A	Q	a	q			。	ア	チ	ム		
	2			"	2	B	R	b	r			「	イ	ツ	メ		
	3			#	3	C	S	c	s			」	ウ	テ	モ		
	4			\$	4	D	T	d	t			、	エ	ト	ヤ		
	5			%	5	E	U	e	u			・	オ	ナ	ユ		
	6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7			'	7	G	W	g	w			ア	キ	ヌ	ラ		
	8			(8	H	X	h	x			イ	ク	ネ	リ		
	9)	9	I	Y	i	y			ウ	ケ	ノ	ル		
	A			*	:	J	Z	j	z			エ	コ	ハ	レ		
	B			+	;	K	[k	{			オ	サ	ヒ	ロ		
	C			,	<	L	¥	l				ヤ	シ	フ	ワ		
	D			-	=	M]	m	}			ユ	ス	ヘ	ン		
	E			.	>	N	^	n	~			ヨ	セ	ホ	°		
	F			/	?	O	_	o				ッ	ソ	マ	°		

ファイル名に利用できる文字

注意

小文字のアルファベットは利用できません。
Memory Card Utility では“ - ”が入力できますが、ファイル名には利用しないでください。
ソフトウェア作成基準に反したアプリケーションとなります。

⑤ VM コメント

ビジュアルメモリのファイルモードで表示するコメントです。ファイル名と同じ文字に加えて、小文字のアルファベットが VM コメントとして利用できます。

⑥ セーブ時刻

ファイルが保存された日付と時刻が表示されます。このデータは、アプリケーションから変更することはできません。

⑦ 使用ブロック数

ファイルの大きさをブロック単位で表しています。1 ブロックは 512 バイトで、ビジュアルメモリ (HKT-7000) の場合、データ保存用に最大 200 ブロック、ゲーム用に最大 128 ブロックの記憶容量を持っています。このデータは、アプリケーションから変更することはできません。

⑧ データの種類

そのファイルがアプリケーションなのか、Dreamcast のセーブデータなのかが表示されます。このデータは、アプリケーションから変更することはできません。

注意

Memory Card Utility を用いた場合にのみ変更が可能です。

⑨ ビジュアルコメント

72 × 56 ドットで最高 65,536 (ARGB4444) 色を使ったグラフィックを表示できます。ビジュアルコメントを表示させないことも可能です。

2.2 ボリュームアイコンの作成

メモリ選択画面で表示されるボリュームアイコンを表示させるには、次のファイル仕様にしたがった“ ICONDATA_VMS ”というファイルを作成します。

参照

ビジュアルメモリ SDK のサンプルには、Volumeicon フォルダにサンプルが収められています。アセンブラのソースも添付されていますので、あわせて参照してください。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	VMコメントデータ															
0010	モノクロアイコンデータの開始アドレス				カラーアイコンデータの開始アドレス				予約領域							
0020	モノクロアイコンのパターンデータ（32×32ドット）															
0030																
0040																
0050																
0060																
0070																
0080																
0090																
00A0	カラーアイコンのパレットデータ（16色分）															
00B0	カラーアイコンのパターンデータ（32×32ドット）															
00C0																
00D0																
00E0																
00F0																
0100																
0110																
0120																
0130																
0140																
0150																
0160																
0170																
0180																
0190																
01A0																
01B0																
01C0																
01D0																
01E0																
01F0																
0200																
0210																
0220																
0230																
0240																
0250																
0260																
0270																
0280																
0290																
02A0																
02B0																

VM コメントデータ

16 バイト分のコメントを埋め込みます。ビジュアルメモリのファイル管理や、Dreamcast のファイル管理画面にて“ ICONDATA_VMS ”を選んだ場合に表示されます。

ビジュアルメモリのファイルモードで表示するコメントです。VM コメントとして利用できる文字は「[2.1.2 ファイル管理画面](#)」を参照してください。あまった部分は、スペース (20H) で埋めてください。

モノクロアイコンデータの開始アドレス

モノクロアイコンのパターンデータの先頭番地をファイルの先頭からのオフセットアドレスで指定します。

通常は、00000020H (メモリダンプ上では 20 00 00 00) を埋め込みます。

注意

データはリトルエンディアンで指定してください。リトルエンディアンについては、「付録 リトルエンディアン形式」を参照してください。

カラーアイコンデータの開始アドレス

カラーアイコンのパレットデータの先頭番地を、ファイルの先頭からのオフセットアドレスで指定します。

通常は、00 00 00 A0H (メモリダンプ上では 0A 00 00 00) を埋め込みます。

注意

データはリトルエンディアンで指定してください。リトルエンディアンについては、「付録 リトルエンディアン形式」を参照してください。
カラーアイコンのパターンデータの先頭ではなく、パレットデータの先頭アドレスであることに注意してください。

予約領域

将来の拡張用の予約領域です。8 バイト分 00 で埋めてください。

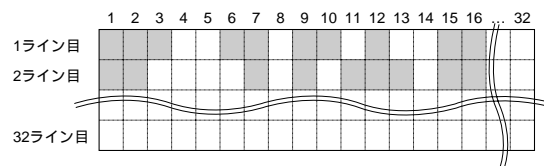
モノクロアイコンのパターンデータ

メモリ選択画面やファイル管理画面の表示中に、ビジュアルメモリの LCD に表示するボリウムアイコン (32 × 32 ドット白黒) を指定します。

データは、LCD の右上から左下に向ったパターンデータです。1 バイトには、8 ドット分のデータが入ります。データの MSB が左側のビット、LSB が右側のビットです。LCD を黒 (青) にするドットに対応するビットを 1 にセットします。

1 ライン (32 ドット) 分のデータは 4 バイトで、32 ドット × 32 ラインで、128 バイトのパターンデータを用意します。

LCD表示



11100110B	11010011B	2バイト分のデータ続く
E5H	D3H	
11000010B	10111011B	2バイト分のデータ続く
C2H	BBH	

全部で128バイトのパターンデータ

カラーアイコンのパレットデータ

カラーアイコンは、16 色を使ったグラフィックが表示できます。この 16 色分のパレットをこの領域に書き込みます。ARGB4444 のパレットデータで、2 バイトで 1 パレットデータです。

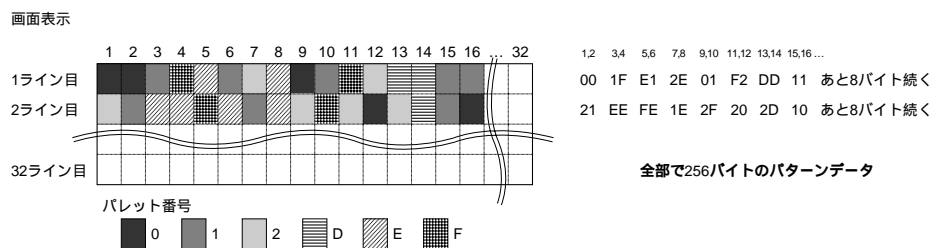
A, R, G, B それぞれに指定できる値は、0 ~ 0FH までです。なお、A は 0 で透明、0FH で不透明です。

カラーアイコンのパターンデータ

メモリ選択画面、ファイル管理画面で表示されるアイコンのパターンデータです。

データは、左上から左下に向ってパレット番号で指定します。1 ドットで 4 ビットのデータとなります。上位 4 ビットが左側のドット、下位 4 ビットが右側のドットのパレット番号です。

横 32 ドットの 1 ラインは、16 バイトのデータです。32 ドット × 32 ラインで、512 バイトのパターンデータを用意します。



2.3 アニメーションアイコンの作成

ファイル管理画面で表示されるアニメーションアイコンや GUI コメントは、これらの情報をファイル自身に埋め込む必要があります。3 つのファイル構造がありますので、ファイルの種類とファイル構造についてを説明します。

2.3.1 3 つのファイル構造

メモリに保存されるファイルには 3 種類あり、それぞれのファイル構造を持ちます。

ICONDATA_VMS 形式

先に説明したとおりの構造です。このファイルは、インフォメーションフォークを持ちません。

詳しくは「[2.2 ボリュームアイコンの作成](#)」を参照してください。

ビジュアルメモリ用アプリケーション形式

ビジュアルメモリのゲームモードで実行できるファイルは、次のようなファイル構造を満たす必要があります。

アドレス	内 容
0000	ビジュアルメモリのヘッダ (GHEAD.ASM に相当する部分)
0200	インフォメーションフォーク
xxxx	アプリケーションのコード

データファイル形式

Dreamcast 用アプリケーションのゲームデータを保存するファイルは、次のようなファイル構造を満たす必要があります。

アドレス	内 容
0000	インフォメーションフォーク
0200	ゲームデータ

2.3.2 インフォメーションフォーク

“ ICONDATA_VMS ”以外のファイルは、インフォメーションフォークと呼ばれるセクションがあります。この情報は、Memory Card Utility でファイルダンプを見たときに、色付きで表示されている部分です。ファイルの詳細な情報は、すべてこのインフォメーションフォークに埋め込みます。

注意

アニメーションアイコンのデータ領域は、色付きで表示されません。

ビジュアルメモリ SDK のサンプルの total では、アニメーションアイコンとビジュアルコメントを利用したインフォメーションフォークを使っていますので、参考にしてください。

インフォメーションフォークは、次のような構造を持ちます。なお、表のアドレスは、インフォメーションフォークの先頭からのオフセットアドレスを示します。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	VMコメントデータ															
0010	GUIコメントデータ															
0020																
0030	ゲーム名（ソートキー）															
0040	アイコン数	アニメスビード	ビジュアルタイプ	CRC				セーブデータサイズ				予約領域				
0050	予約領域															
0060	アイコンのパレットデータ（16色分）															
0070																
0080	アイコン#1のパターンデータ（32×32ドット）															
0090																
0260																
0270																
0280	アイコン#2のパターンデータ（32×32ドット）															
0290																
0460																
0470																
0480	アイコン#3のパターンデータ（32×32ドット）															
0490																
0660																
0670																
0680	ビジュアルコメント（パレット&パターンデータ）															
0690																
XXXX																

VM コメントデータ

16 バイト分のコメントを埋め込みます。ビジュアルメモリのファイル管理や、Dreamcast のファイル管理画面にて、そのファイルを選んだ場合に表示されます。

ビジュアルメモリのファイルモードで表示するコメントです。VM コメントとして利用できる文字は、次の表のアミがかかっていない部分です。なお、あまった部分はスペース (20H) で埋めてください。

		下位4ビット															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
上 位 4 ビ ッ ト	0			空白	0	@	P	'	p				ー	タ	ミ		
	1			!	1	A	Q	a	q			。	ア	チ	ム		
	2			"	2	B	R	b	r			「	イ	ツ	メ		
	3			#	3	C	S	c	s			」	ウ	テ	モ		
	4			\$	4	D	T	d	t			、	エ	ト	ヤ		
	5			%	5	E	U	e	u			・	オ	ナ	ユ		
	6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7			'	7	G	W	g	w			ァ	キ	ヌ	ラ		
	8			(8	H	X	h	x			ィ	ク	ネ	リ		
	9)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
	A			*	:	J	Z	j	z			ェ	コ	ハ	レ		
	B			+	;	K	[k	{			ォ	サ	ヒ	ロ		
	C			,	<	L	¥	l				ャ	シ	フ	ワ		
	D			-	=	M]	m	}			ュ	ス	ヘ	ン		
	E			.	>	N	^	n	~			ョ	セ	ホ	°		
	F			/	?	O	_	o				ッ	ソ	マ	°		

VM コメントに利用できる文字

GUI コメントデータ

ファイル管理画面で表示される 32 バイト分のコメントを埋め込みます。Dreamcast のファイル管理画面にて、そのファイルを選んだ場合に表示されます。

利用できる半角文字は、次の表でアミがかかっていない部分です。全角文字はシフト JIS コードで、第 2 水準をサポートしています。また、JIS X 0208-1983 対応なので音符などの記号類も表示できます。

あまった部分は、スペースで (20H) 埋めてください。

		下位4ビット															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
上位4ビット	0			空白	0	@	P	'	p				ー	タ	ミ		
	1			!	1	A	Q	a	q			。	ア	チ	ム		
	2			"	2	B	R	b	r			「	イ	ツ	メ		
	3			#	3	C	S	c	s			」	ウ	テ	モ		
	4			\$	4	D	T	d	t			、	エ	ト	ヤ		
	5			%	5	E	U	e	u			・	オ	ナ	ユ		
	6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7			'	7	G	W	g	w			ァ	キ	ヌ	ラ		
	8			(8	H	X	h	x			ィ	ク	ネ	リ		
	9)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
	A			*	:	J	Z	j	z			ェ	コ	ハ	レ		
	B			+	;	K	[k	{			ォ	サ	ヒ	ロ		
	C			,	<	L	¥	l				ャ	シ	フ	ワ		
	D			-	=	M]	m	}			ュ	ス	ヘ	ン		
	E			.	>	N	^	n	~			ョ	セ	ホ	°		
	F			/	?	O	_	o				ッ	ソ	マ	°		

GUI コメントに利用できる半角文字

ゲーム名 (ソートキー)

Dreamcast のファイル管理画面では、ファイルがソートされて表示されます。そのソートキーとして、この領域が使われます。16 バイトのデータで指定し、独自の文字コード表示にしたがったデータを埋め込みます。

参照

ゲーム名の付け方や独自のコード表は「[2.3.4 ゲーム名のソート基準](#)」を参照してください。

アイコン数

アニメーションアイコンにする場合は、2 か 3 を指定します。1 と指定すると静止画のアイコンになります。

指定できる値は 1～3 までで、アニメーションパターンは最大 3 パターンまで作れます。

注意

1～3 以外の値を指定しないでください。0 または 4 以上を指定した場合の動作は保証できません。

データはリトルエンディアンで指定してください。リトルエンディアンについては、「付録 リトルエンディアン形式」を参照してください。

アニメーションスピード

アニメーションアイコンを利用する場合に、アイコンを切り換える速度を指定します。指定できる値は 1～65,535 までです。指定値を n とすると、 $n / 30$ 秒でアニメーションパターンが切り替わります。

たとえば、1 を指定すると $1 / 30$ (約 0.03) 秒、30 を指定すると 1 秒、65,535 を指定すると約 36 分ごとにパターンが切り替わります。

アニメーションパターンが 3 枚ある場合は、1 2 3 1 ... の順に表示されます。2 枚の場合は、1 と 2 が交互に表示されます。アイコン数に 1 を指定した場合、この値は意味を持ちません。

注意

0 は指定しないでください。0 を指定した場合の動作は保証できません。

データはリトルエンディアンで指定してください。リトルエンディアンについては、「付録 リトルエンディアン形式」を参照してください。

ビジュアルタイプ

Dreamcast のファイル管理画面の右下に表示されるビジュアルコメントの種類を指定します。ビジュアルコメントに表示できるグラフィックは、 72×56 ドットです。

注意

ビジュアルコメントはアニメーションできません。また、カラーを多く使うグラフィックは、その分だけメモリを消費しますので注意してください。

データはリトルエンディアンで指定してください。リトルエンディアンについては、「付録 リトルエンディアン形式」を参照してください。

指定値とビジュアルコメントの種類、必要なバイト数、使用するブロック数は次のとおりです。

指定値	ビジュアルコメントの種類	必要バイト数	データ用	パレット用	使用ブロック数
0	なし	0	0	0	0
1	ダイレクトカラー (タイプ A)	8064	8064	0	16
2	256 色グラフィック (タイプ B)	4544	4032	512	9
3	16 色グラフィック (タイプ C)	2048	2016	32	4

CRC

CRC (エラーチェック / 訂正用の符号) を書き込みます。

SEGA ライブラリのバックアップユーティリティ関数の `buMakeBackupFileImage()` を利用して、ファイルを保存した場合は自動的に CRC が計算され、その値が書き込まれます。なお、CRC はインフォメーションフォークを除いたデータ部分のみが対象です。

注意

ビジュアルメモリ用のアプリケーションを Memory Card Utility を使って転送する場合、CRC を特に書き込む必要はありません。また CRC チェックを行う必要もありません。この場合は、00 00 で埋めてください。
データはリトルエンディアンで指定してください。リトルエンディアンについては、「[付録 リトルエンディアン形式](#)」を参照してください。

セーブデータサイズ

インフォメーションフォークを除いたデータ領域のサイズを、バイト単位で指定します。

注意

ビジュアルメモリ用のアプリケーションを Memory Card Utility を使って転送する場合、セーブデータサイズを特に書き込む必要はありません。この場合は、00 00 で埋めてください。
データはリトルエンディアンで指定してください。リトルエンディアンについては、「[付録 リトルエンディアン形式](#)」を参照してください。

予約領域

将来の拡張用の予約領域です。20 バイト分 00 で埋めてください。

アイコンのパレットデータ

これ以降のアイコンパターンで利用するパレットを 16 色分指定します。

パレットデータは、ARGB4444 で指定します。2 バイトで 1 パレットデータです。

A, R, G, B それぞれに指定できる値は、0 ~ 0FH までです。なお、A は 0 で透明、0FH で不透明です。

このパレットは、アイコン #2, #3 にも適応されます。アイコンごとにパレットは変えられませんので注意してください。

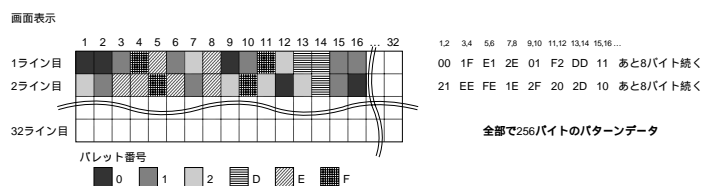
アイコン#n のパターンデータ

32 × 32 ドット 16 色アイコンのパターンデータです。

データは、左上から左下に向ってパレット番号で指定します。1 ドットで 4 ビットのデータとなります。上位 4 ビットが左側のドット、下位 4 ビットが右側のドットのパレット番号です。

横 32 ドットの 1 ラインは、16 バイトのデータです。32 ドット × 32 ラインで、512 バイトのパターンデータを用意します。

アニメーションさせない場合は、アイコンのパターンデータを 1 つ作ります。アニメーションさせる場合は、最大 3 つのパターンを用意します。



ビジュアルコメントデータ

ビジュアルコメントを利用する (ビジュアルタイプに 0 以外を指定した) 場合、そのデータを埋め込みます。ビジュアルコメントのグラフィックは、72 × 56 ドットのデータです。

2.3.3 ビジュアルコメントデータの構造

ビジュアルタイプとビジュアルコメントのデータ構造は、次のとおりです。

ダイレクトカラー (タイプ A)

パレットデータは持ちません。パターンデータそのものに ARGB4444 の値を指定します。1 ドットあたり 2 バイトのデータとなります。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	[0,0]のARGB4444	[1,0]のARGB4444	[2,0]のARGB4444	[3,0]のARGB4444	[4,0]のARGB4444	[5,0]のARGB4444	[6,0]のARGB4444	[7,0]のARGB4444	[8,0]のARGB4444	[9,0]のARGB4444	[A,0]のARGB4444	[B,0]のARGB4444	[C,0]のARGB4444	[D,0]のARGB4444	[E,0]のARGB4444	[F,0]のARGB4444
0010	[0,1]のARGB4444	[1,1]のARGB4444	[2,1]のARGB4444	[3,1]のARGB4444	[4,1]のARGB4444	[5,1]のARGB4444	[6,1]のARGB4444	[7,1]のARGB4444	[8,1]のARGB4444	[9,1]のARGB4444	[A,1]のARGB4444	[B,1]のARGB4444	[C,1]のARGB4444	[D,1]のARGB4444	[E,1]のARGB4444	[F,1]のARGB4444
...
1F70	[0,15]のARGB4444	[1,15]のARGB4444	[2,15]のARGB4444	[3,15]のARGB4444	[4,15]のARGB4444	[5,15]のARGB4444	[6,15]のARGB4444	[7,15]のARGB4444	[8,15]のARGB4444	[9,15]のARGB4444	[A,15]のARGB4444	[B,15]のARGB4444	[C,15]のARGB4444	[D,15]のARGB4444	[E,15]のARGB4444	[F,15]のARGB4444

256 色グラフィック (タイプ B)

256 色分のパレットデータを持ちます。パレットの指定は、ARGB4444 にて指定します。

パターンデータは、1 ドットあたり 1 バイトでパレット番号で指定します。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	パレット0	パレット1	パレット7
01F0	パレット255
0200	[0,0]の パレット 番号	[1,0]の パレット 番号	[2,0]の パレット 番号	[3,0]の パレット 番号	[4,0]の パレット 番号	[5,0]の パレット 番号	[6,0]の パレット 番号	[7,0]の パレット 番号	[8,0]の パレット 番号	[9,0]の パレット 番号	[A,0]の パレット 番号	[B,0]の パレット 番号	[C,0]の パレット 番号	[D,0]の パレット 番号	[E,0]の パレット 番号	[F,0]の パレット 番号
0010	[0,1]の パレット 番号	[1,1]の パレット 番号	[2,1]の パレット 番号	[3,1]の パレット 番号	[4,1]の パレット 番号	[5,1]の パレット 番号	[6,1]の パレット 番号	[7,1]の パレット 番号	[8,1]の パレット 番号	[9,1]の パレット 番号	[A,1]の パレット 番号	[B,1]の パレット 番号	[C,1]の パレット 番号	[D,1]の パレット 番号	[E,1]の パレット 番号	[F,1]の パレット 番号
...
11B0	[71,55]の パレット 番号

16 色グラフィック (タイプ C)

16 色分のパレットデータを持ちます。パレットの指定は、ARGB4444 にて指定します。

パターンデータは、1 ドットあたり 4 ビットです。1 バイトに 2 ドット分のデータを埋め込み、上位 4 ビットは左のドット、下位 4 ビットは右のドットに対応します。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	パレット0	パレット1	パレット7
0010	パレット15
0020	(10,0) パレット 番号	(2,0) (3,0)の パレット 番号	(4,0) (5,0)の パレット 番号	(30,0) (31,0)の パレット 番号
0010	(32,0) (33,0)の パレット 番号
07F0	(70,55) (71,55)の パレット 番号

2.3.4 ゲーム名のソート基準

インフォメーションフォーク内のゲーム名 (ソートキー) を使い、Dreamcast のファイル管理画面でアイコンを表示する順番が決まります。このため、ゲーム名を決め続編タイトルなどがある場合は、その順番に表示されるような仕掛けを作る必要があります。

ゲーム名のキャラクタコード表を用いて、16 バイトのゲーム名を設定します。ここでは、さまざまな例を見ながらゲーム名の付け方やソート基準を説明します。

アプリケーションの名称を 16 文字の読みにします。

「あつめてゴジラ 怪獣大図鑑」

「アツメテゴジラカイジュウダイズカ」

音引き記号は小さい母音に置き換えます。

「ネーム」

「ネェム」

「は」を「わ」と読む場合でも「ハ」を使います。

「王様の耳はロバの耳」

「オオサマノミミハロバノミミ」

「1ヶ月」「1ヶ年」などは「カ」を使います。

「無人島 10ヶ月」

「ムジントウジュツカゲツ」

英語のタイトルは読みを使います。

「Star」

「スタア」

「・」「-」スペースなどの記号類は無視します。

「ソニック・ザ・ヘッジホッグ」

「ソニックザヘッジホッグ」

日本国外向けのアプリケーションは 16 バイト 00 で埋めます。

正式名称が 16 文字を越え GUI コメントが略称の場合は、略称を用います。

正式名称が「弥次さん喜多さん道中記～東海道その1～」の場合で、GUI コメントに「弥次喜多道中記 その1」と表示しているとき。

「ヤジキタドウチュウキソノ1」

略称にアルファベットを用いている場合は、一般的な読み方を使用します。

「電腦戦機 V.R.」

「デンノウセンキバァチャロン」

数字は最後に用いてください。

「バーチャファイター 3」

「バァチャファイター3」

次のような使い方は禁止です。

「3びきの子ブタ」

×「3ビキノコブタ」

「サンビキノコブタ」

シリーズ番号が頭にある場合は、番号を最後にします。

「第1部 電腦戦機 V.R.」

「第2部 電腦戦機 V.R. オラトリオダンGRAM」

「デンノウセンキバァチャロン1」

「デンノウセンキバァチャロン2」

前編タイトルが略称の場合、後編もこれに合わせてください。

前編が「Ｊリーグサカつく」(正式名称はＪリーグプロサッカークラブを作ろう！) としていた場合、後編も「Ｊリーグサカつく」を用います。

「ジェイリーグサカつく」と「ジェイリーグサカつく 2」とします。

名称の最後に年号がある場合は 2000 年を念頭においてください。

誤った例：2000、98、99 の順にソートされます。

「グレイテストナイン 98」

「グレイテストナイン 99」

「グレイテストナイン 2000」

正しい例：正しくソートされます。

「グレイテストナイン 1」

「グレイテストナイン 2」

「グレイテストナイン 3」

シリーズが 10 を越えることが予測される場合は 0 サプレスしてください。

「日本文学全集 3」

「日本文学全集 12」

「ニホンブンガクゼンシュウ 03」

「ニホンブンガクゼンシュウ 12」

続編でも「 」編という分け方をする場合は、ゲーム名の最後を数字にしてきれいにソートされるようにすることを“推奨”します。

「武田武勇伝 風の巻」

「武田武勇伝 林の巻」

「タケダブユウデン 1」

「タケダブユウデン 2」

このような規則に基づいてゲーム名を決めたら、次のゲーム名キャラクタコード表を用いて、16 バイトのデータを作ります。

文字	コード	
	16進	10進
NULL	00	0
ア	01	1
ァ	02	2
イ	03	3
ィ	04	4
ウ	05	5
ヴ	06	6
ゥ	07	7
エ	08	8
ェ	09	9
オ	0A	10
ォ	0B	11
カ	0C	12
ガ	0D	13
キ	0E	14
ギ	0F	15
ク	10	16
グ	11	17
ケ	12	18
ゲ	13	19
コ	14	20
ゴ	15	21
サ	16	22
ザ	17	23

文字	コード	
	16進	10進
シ	18	24
ジ	19	25
ス	1A	26
ズ	1B	27
セ	1C	28
ゼ	1D	29
ソ	1E	30
ゾ	1F	31
タ	20	32
ダ	21	33
チ	22	34
ヂ	23	35
ツ	24	36
ヅ	25	37
ッ	26	38
テ	27	39
デ	28	40
ト	29	41
ド	2A	42
ナ	2B	43
ニ	2C	44
ヌ	2D	45
ネ	2E	46
ノ	2F	47

文字	コード	
	16進	10進
ハ	30	48
バ	31	49
パ	32	50
ヒ	33	51
ピ	34	52
ビ	35	53
フ	36	54
ブ	37	55
プ	38	56
ヘ	39	57
ベ	3A	58
ペ	3B	59
ホ	3C	60
ボ	3D	61
ポ	3E	62
マ	3F	63
ミ	40	64
ム	41	65
メ	42	66
モ	43	67
ヤ	44	68
ャ	45	69
ユ	46	70
ュ	47	71

文字	コード	
	16進	10進
ヨ	48	72
ョ	49	73
ラ	4A	74
リ	4B	75
ル	4C	76
レ	4D	77
ロ	4E	78
ワ	4F	79
ヰ	50	80
ヱ	51	81
ヲ	52	82
ン	53	83
0	54	84
1	55	85
2	56	86
3	57	87
4	58	88
5	59	89
6	5A	90
7	5B	91
8	5C	92
9	5D	93

ゲーム名キャラクタコード表

Memory Card Utility

ここでは、PC Dev.Box ビジュアルメモリの転送プログラム「Memory Card Utility」の操作方法について説明します。

3.1 Memory Card Utility の準備と起動

Memory Card Utility は、Dreamcast 用のプログラムです。ビジュアルメモリ SDK をインストールしたフォルダの utility フォルダの中に“ mem_util.elf ”というファイルで提供されています。このプログラムを実行するには、CodeScape や GD Workshop など Dreamcast SDK に含まれるアプリケーションが必要になります。

3.1.1 転送に必要なもの

Memory Card Utility は、PC とのファイルのやりとりを RS-232C (シリアルインターフェイス) を介して行います。このため、弊社提供の SDK とは別に次のものを用意してください。

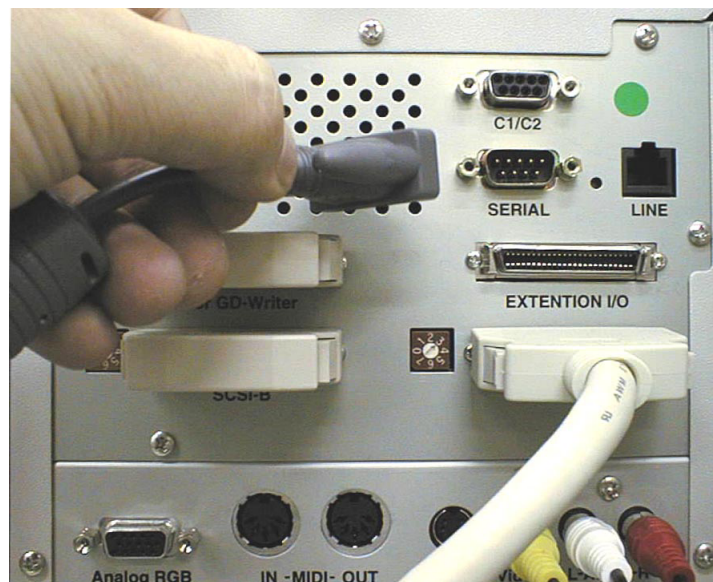
- ①RS-232C リバースケープル
- ②Windows 用通信プログラム

①は一般のパソコンショップなどでお買い求めください。PC の RS-232C インターフェイスと、Dev.Box 背面の「SERIAL」を接続します。

コンピュータ側 シリアルインターフェイス (RS-232C) コネクタ



Dev.Box 側 SERIAL コネクタ



注意

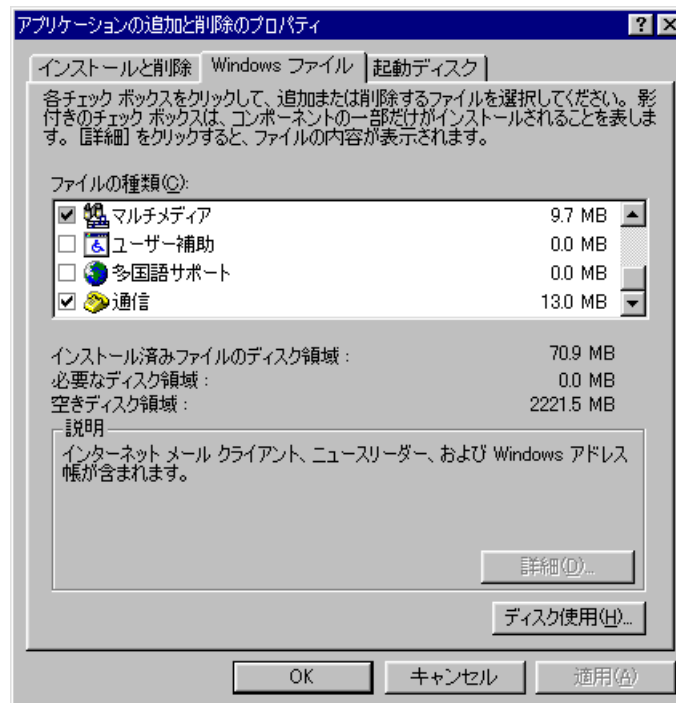
必ずリバースケーブルをお使いください。ストレートケーブルでは動作しません。

②は Windows などに添付されている「ハイパーターミナル」でかまいません。その他の通信ソフトをお使いになる場合は、必ず Xmodem でのファイルの送受信ができるものを利用してください。

お使いの Windows にハイパーターミナルが見あたらない場合は、次の手順にしたがってインストールしてください。

- ① [コントロールパネル] の [アプリケーションの追加と削除] アイコンをダブルクリックします。
- ② [Windows ファイル] タブをクリックします。

③ [通信] をダブルクリックします。



④ [ハイパーターミナル] をチェックします。

これでハイパーターミナルがインストールされます。

Windows98 に添付のハイパーターミナルは、漢字のファイル名を使うと正しくパスとファイル名を認識できなくなり、ひとつ上位のフォルダに文字化けしたファイルが保存される場合があります。この場合は、なるべく漢字のファイル名を使わないか、その他の通信ソフトをお使いになることをお勧めします。

その他に弊社提供の SDK に含まれる次のものを用意してください。

- ③ Dev.Box (SET 5.2X 以上)
- ④ CodeScape
- ⑤ GD Workshop
- ⑥ ビジュアルメモリ
- ⑦ Dreamcast コントローラ

これらの接続方法やセットアップについては「セットアップガイド」を参照してください。本書では、これらのセットアップが終わっているものとして説明します。

注意

Memory Card Utility は、SET 5.1X 以下の Dev.Box では動作しません。

3.1.2 ソフトウェアの準備

PC と Dev.Box を RS-232C で接続し、すべてのソフトウェアのセットアップが終わったところで、プロパティなどを設定します。

通信プロトコルの設定

PC と Dev.Box 間は、次のような通信プロトコルでデータ転送を行います。

お使いのアプリケーションをこの設定にしてください。設定方法については、それぞれのアプリケーションのマニュアルを参照してください。

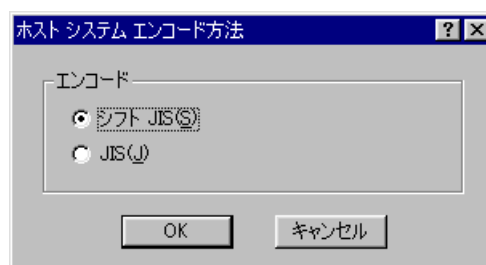
注意

Memory Card Utility を起動すると通信画面に起動メッセージが表示されます。このメッセージが正しく表示されない場合は、通信プロトコルを再確認し、クロスケーブルが接続されているインターフェイスを選んでいるかをチェックしてください。

設定項目	設定値
通信速度	38,400bps
データ長	8 ビット
ストップビット長	1 ビット
パリティチェック	なし
フロー制御	なし
漢字コード	シフト JIS コード

ハイパーターミナルの場合、各ダイアログボックスを次のように設定します。





注意

フロー制御は、必ず“ なし ”に設定してください。Xon/off やハードウェア制御を指定しないように注意してください。

RS-232C インターフェイスが複数ある場合は、クロスケーブルを接続したインターフェイスを選んでください。



GD Workshop の設定

Memory Card Utility は、起動時に GD-ROM ドライブのドアクローズ (GD-ROM のマウント) を検出します。したがって、ドアクローズをエミュレーションするためにダミーの GD-ROM を作成します。

参照

GD Workshop の詳しい操作手順は『GD Workshop マニュアル』を参照してください。

次の手順にしたがって、ダミーの GD-ROM を作成してください。

- ① GD Workshop を起動します。
- ② 新しいプロジェクトを作成します。ここでは仮に、DoorClose というプロジェクトを作成します。
- ③ 3つのトラックに適当なファイルをドラッグします。
ファイルが小さいとエミュレーションできませんから、大きめのアプリケーションなどの実行ファイルをコピーするとよいでしょう。
オーディオトラックには、Dreamcast SDK に添付のワーニング音を入れておくといよいでしょう。
- ④ プロジェクトを保存します。

正しくダミー GD-ROM が作成できたことを確認します。

- ① DA チェッカを起動し Dev.Box を OS モードで再起動します。
- ② GD Workshop を起動し、DoorClose プロジェクトを読み込みます。
- ③ Dreamcast メインメニューからサウンドを選びます。
- ④ GD Workshop で GD-ROM をエミュレーションモードに変えます。
- ⑤ DoorClose/Open ボタンで、GD-ROM をドアクローズしマウントさせます。

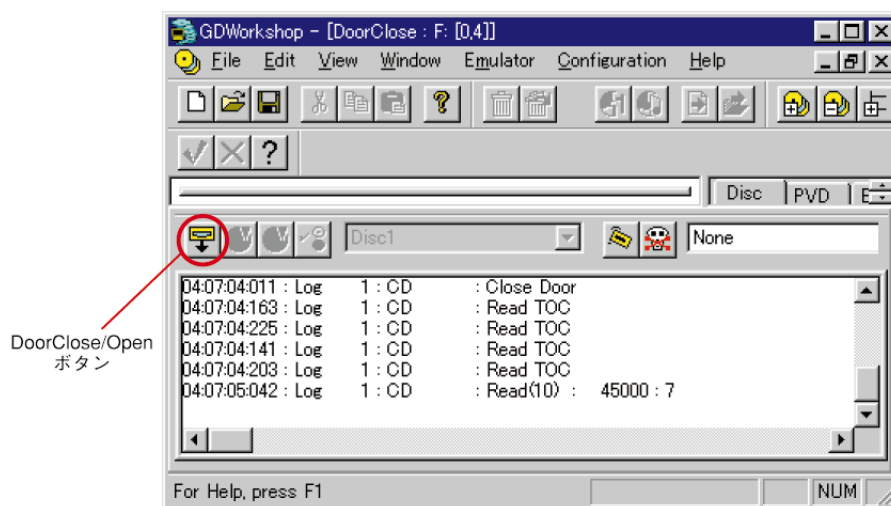
正しくダミー GD-ROM ができれば、サウンドメニューに CD-ROM のグラフィックが表示されます。再生すると、警告メッセージが聞こえるはずです。

これで次回からはプロジェクトを読み込むだけで、ダミー GD-ROM のエミュレーションができます。

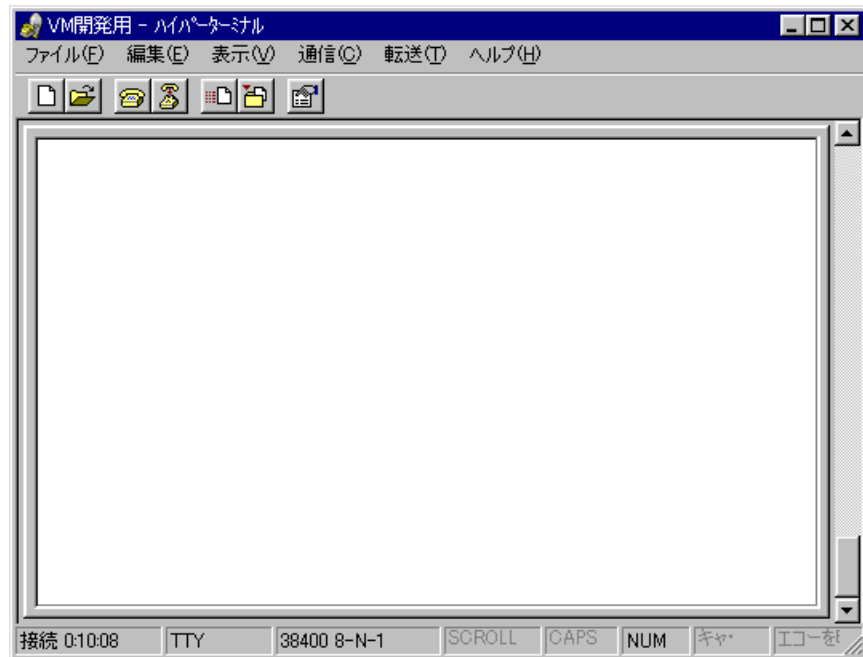
3.1.3 Memory Card Utility の起動

ここでは、CodeScope を使って Memory Card Utility を実行します。次の手順で起動してください。

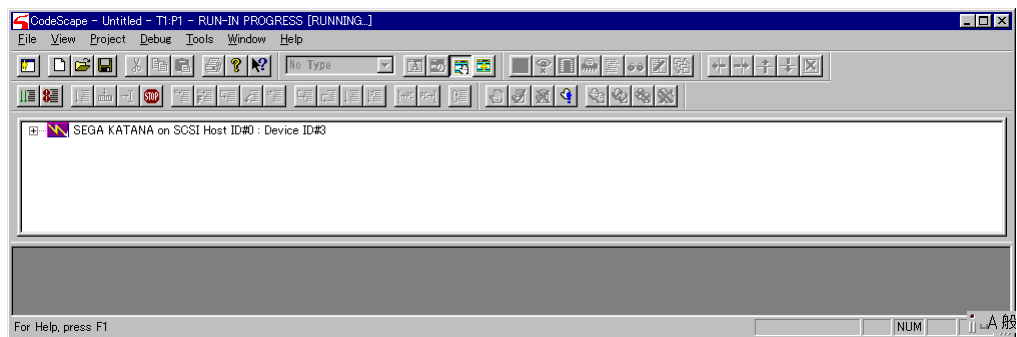
- ① Memory Card Utility を起動する前に GD Workshop を起動します。
起動したら DoorClose 用のプロジェクトを読み込み、エミュレーションを開始させます。
DoorClose/Open ボタンで GD-ROM をマウントさせます。



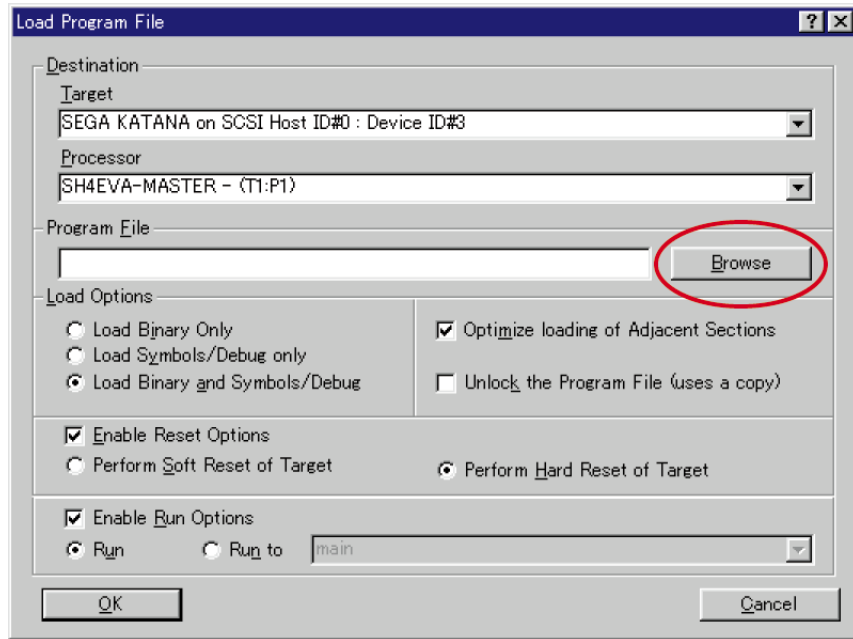
- ②通信ソフトを起動します。通信プロトコルの設定やインターフェイスの番号などを確認してください。



- ③CodeScape を起動します。

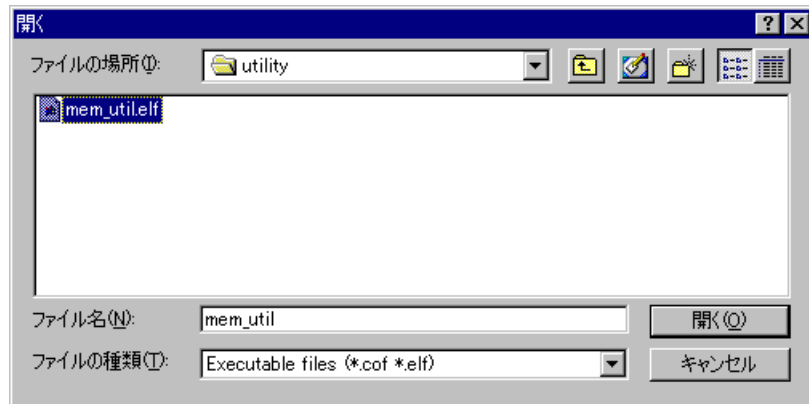


- ④ [File] メニューの [Load Program File...] を選びます。



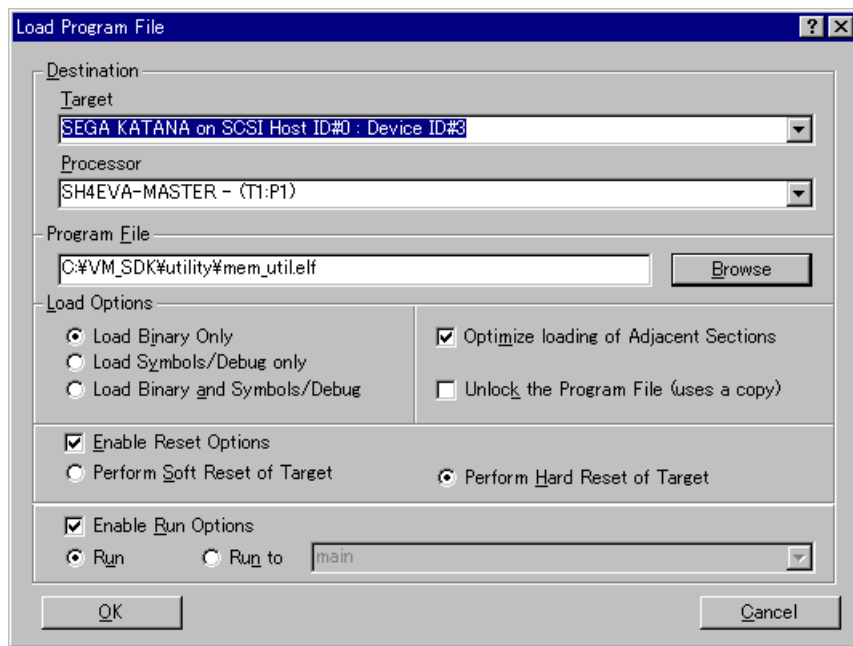
- ⑤ [Browse] ボタンをクリックし、“ mem_util.elf ”を選びます。

“ mem_util.elf ”は、ビジュアルメモリ SDK をインストールしたフォルダの utility フォルダの中にあります。

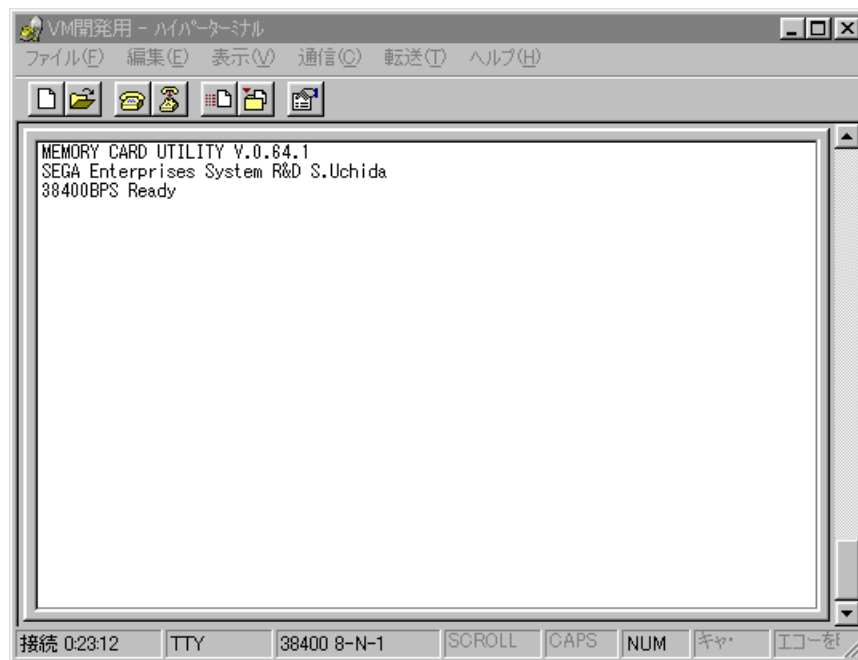


- ⑥ [Load Binary Only] オプションを選びます。
- ⑦ [Enable Reset Options] をチェックし [Perform Hard Reset of Target] オプションを選びます。
- ⑧ [Enable Run Options] をチェックし [Run] オプションを選びます。

- ⑨ ダイアログの設定を終えたら [OK] ボタンをクリックします。

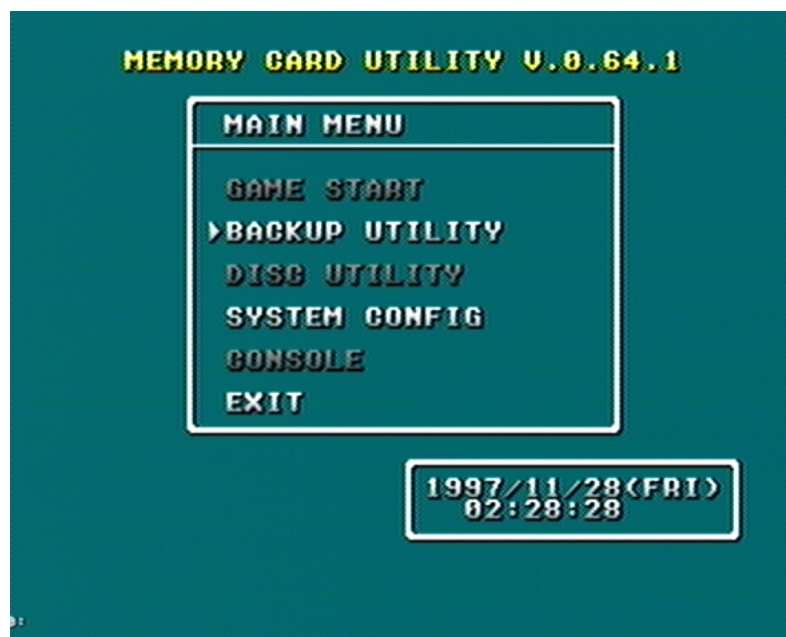


- ⑩ グラフが表示され 100 % まで達すると、Memory Card Utility が起動します。
このとき通信ソフトには、次のような起動メッセージが表示されます。



メッセージが表示されない場合や文字化けしている場合は、通信プロトコルの設定を再度確認してください。

一方、Dev.Box のディスプレイ(NTSC もしくは VGA モニタ)には、Memory Card Utility のメインメニューが表示されます。



ここで CodeScape の [File] メニューの [Session Save] を選ぶと、次回起動時にセッションを開くだけで Memory Card Utility を起動できます。

- ⑪以降、Memory Card Utility の操作は、PC と Dev.Box でファイルの転送を行う以外は、Dreamcast コントローラで行います。

3.2 Memory Card Utility の操作方法

ここでは Memory Card Utility の操作方法についてを説明します。なお、マニュアルは Version 0.64.1 を元に作成しました。

操作は、方向ボタンでメニューを選択し、A ボタンで決定します。B ボタンは操作をキャンセルし前のメニューに戻ります。メニューによっては、ボタンが異なる場合や他のボタンを使う場合があります。この場合は、そのメニューごとにボタンの機能を解説していきます。

3.2.1 メインメニュー

Memory Card Utility を起動すると次の画面が表示されます。これを「メインメニュー」と呼びます。



グレーで表示されているメニューは、機能が実装されていないため選べません。

BACKUP UTILITY

このメニューを選ぶと、メモリ選択メニューが表示されます。

ビジュアルメモリにファイルを転送する場合や、保存されているファイル进行操作する場合など、通常このメニューを選びます。

SYSTEM CONFIG

ビジュアルメモリの内蔵時計を、Dev.Box の内蔵時計に合わせます。

このメニューを選択すると、システムコンフィグレーションメニューが表示されます。MEMORY CARD TIME を選ぶと、確認メッセージが表示され、接続されているすべてのビジュアルメモリの時計を Dev.Box の時計に合わせます。

システムコンフィグレーションメニューでは、MEMORY CARD TIME のみ選べます。

EXIT

Memory Card Utility を終了して、BOOT ROM に制御を渡します。画面は、Dreamcast の起動メニューになります。

3.2.2 メモリ選択メニュー

メインメニューから BACKUP UTILITY を選ぶと、このメニューが表示されます。

注意

ビジュアルメモリの脱着は、この画面まで可能です。
なお、脱着の際はビジュアルメモリの動作モードをゲームモード以外にしてください。ゲームモードの状態で接続すると、接続状態が正しく認識されません。



ここでは、これから操作を行うメモリを選択します。

画面には、現在接続されているメモリがすべて表示されます。A, B, C, D は、コントローラのポートを表しています。上段と下段は、1 つのコントローラに複数のメモリが接続されている場合に表示されます。

なお、ビジュアルメモリの場合、LCD に画面と対応する番号が表示されます。

目的のメモリを選択して A ボタンを押すと、コマンド選択画面になります。

USED

データ保存領域のうち、すでに利用済みのブロック数が表示されます。

FREE

データ保存領域の空き領域がブロック数で表示されます。

GAME

ビジュアルメモリ用アプリケーション領域の使用状況を表しています。

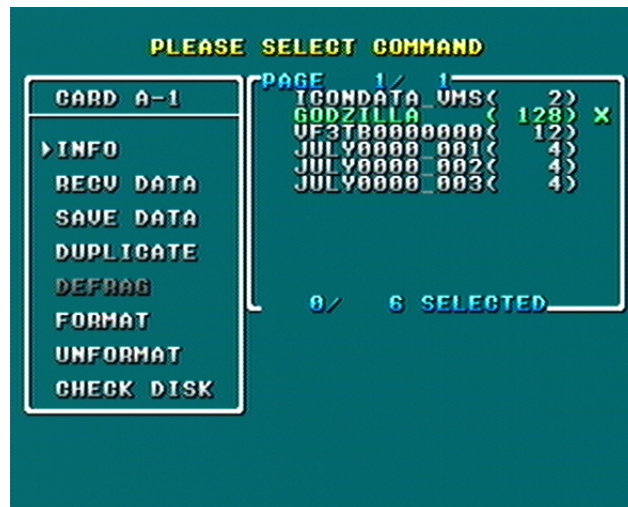
《使用中のブロック数》 / 《アプリケーションが利用可能な最大ブロック数》の形式で表示されています。

たとえば“ 128 / 128 ”と表示されている場合は、すでにアプリケーションが書き込まれていることを示します。“ 0 / 128 ”と表示されている場合は、アプリケーションが書き込まれていないビジュアルメモリであることが分ります。

3.2.3 コマンド選択メニュー

このメニューでは、ビジュアルメモリにデータを転送したり、書き込まれているファイルを操作します。

画面右側は、ビジュアルメモリに書き込まれているファイル一覧です。1 画面に表示しきれない場合は、ファイル一覧をアクティブに (マークをファイル一覧に移動) して、右方向ボタンを押すと続きが表示されます。



INFO

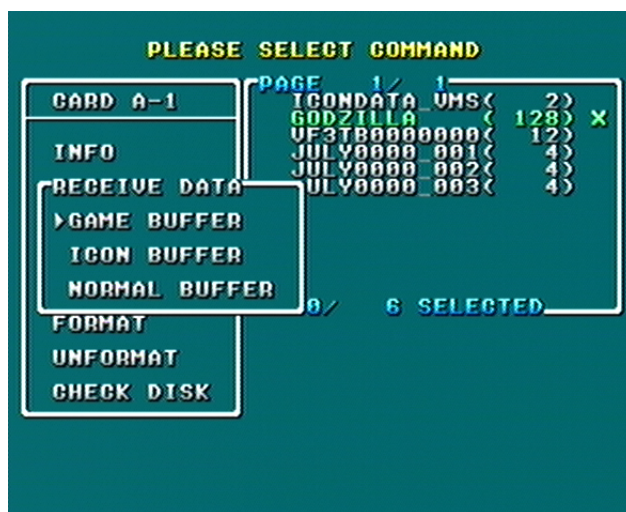
このメニューを選ぶと、次の画面が表示されます。



ここには、ビジュアルメモリの容量に関する情報、ボディーカラーの情報、ボリュームアイコンの情報、初期化された日時の情報が表示されます。

RECV DATA

このメニューでは、PC から Dev.Box のバッファにアプリケーションやボリュームアイコン用のファイル、データファイルを転送できます。RECV DATA を選ぶと次の画面が表示されます。



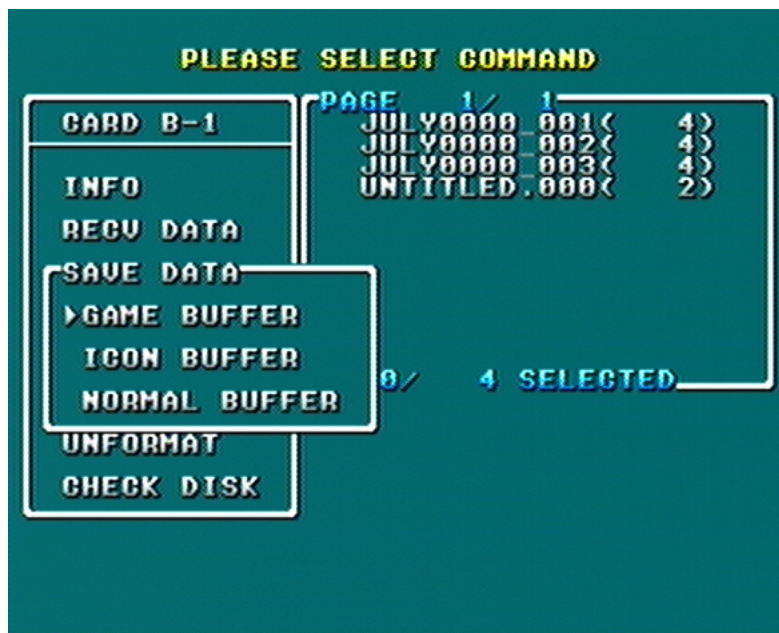
ここでは、PC から受信するファイルの種類を指定します。ビジュアルメモリ用のアプリケーションを受信するにはGAME BUFFERを、ボリュームアイコン用のデータを受信する場合はICON BUFFERを選びます。また、データファイルを受信する場合はNORMAL BUFFERを選びます。

参照

PC からビジュアルメモリへのデータ転送については「[3.4 PC からビジュアルメモリへのファイル転送](#)」を参照してください。

SAVE DATA

このメニューでは、あらかじめ Dev.Box のバッファ受信したデータを、ビジュアルメモリに書き込みます。SAVE DATA を選ぶと次の画面が表示されます。



ここでは、どのファイル(バッファのデータ) をビジュアルメモリに書き込むかを指定します。

注意

すでにアプリケーションが書き込まれているビジュアルメモリの場合、GAME BUFFER が選べません。あらかじめ、アプリケーションのファイルを削除してください。

ビジュアルメモリ用のアプリケーションを書き込む場合は GAME BUFFER を、ボリュームアイコン用のデータを書き込む場合は ICON BUFFER を選びます。また、データファイルを書き込む場合は NORMAL BUFFER を選びます。

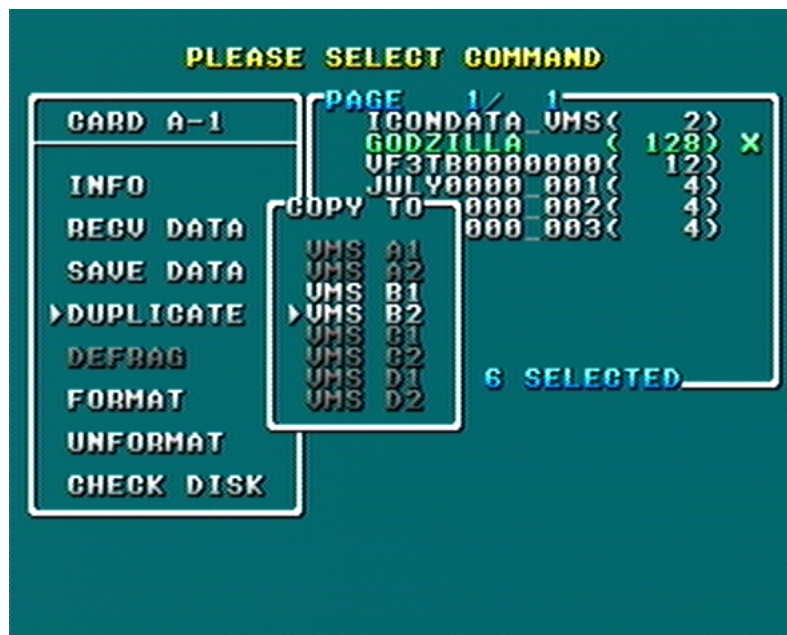
参照

PC からビジュアルメモリへのデータ転送については「[3.4 PC からビジュアルメモリへのファイル転送](#)」を参照してください。

DUPLICATE

このメニューでは、現在選択しているビジュアルメモリの内容を、別のビジュアルメモリにそっくりコピーします。Windows の [ディスクのコピー] や、MS-DOS の DISKCOPY 相当の機能です。

メニューを選ぶと次の画面が表示されます。



転送先のビジュアルメモリを選択すると、確認メッセージが表示されます。

注意

DUPLICATE では、フラッシュメモリの内容をそっくりそのままコピーします。転送先のビジュアルメモリに保存されていたデータは、すべて消去されますので注意してください。

OK を選ぶとメモリの内容がそのままコピーされます。

DEFRAG

このメニューを選ぶと、ファイルの保存や削除を繰り返し行った場合におこるフラグメント (断片化) を解消します。

注意

すでにアプリケーションが書き込まれているビジュアルメモリでは、このメニューは選べません。

ビジュアルメモリはファイルの管理に FAT 方式を用いているため、サイズの異なるファイルを書き込み / 消去しているうちに、データ保存領域が歯抜け状態に断片化されます。これを整理 (ガベージコレクション) し、データをひとつの塊にまとめます。

アプリケーションは、連続したメモリを確保する必要があるので、アプリケーションを転送する前には DEFRAG を行ってください。

容量は十分あるにもかかわらずアプリケーションが保存できない場合は、断片化していると考えられます。断片化を解消し、連続した空き領域を作成してください。

DEFRAG を選ぶと確認メッセージの後、断片化の解消が行われます。

FORMAT

このメニューを選ぶと、次の画面が表示され選択しているビジュアルメモリを初期化できます。



ここでボディーカラーやラベルアイコンの指定、初期化した日時の設定などを設定します。設定後の確認メッセージで OK を選ぶと初期化が行われます。初期化するビジュアルメモリに書き込まれていたファイルはすべて削除されます。

参照

初期化についての詳しい手順は「[3.3 ビジュアルメモリの初期化](#)」を参照してください。

UNFORMAT

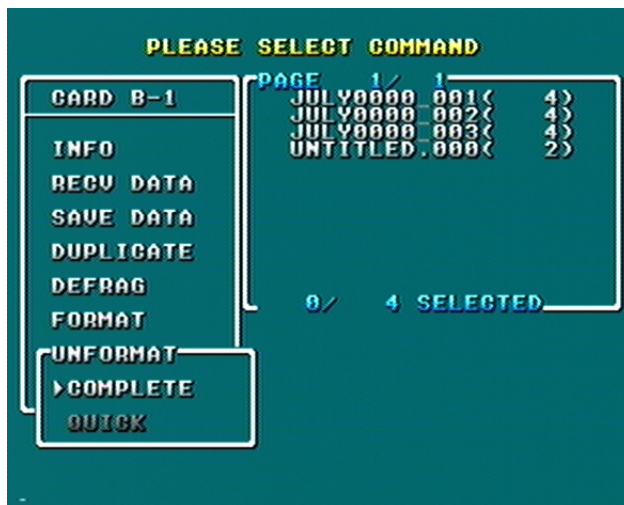
このメニューを選ぶと、初期化されていないビジュアルメモリを作ることができます。ビジュアルメモリは、システム BIOS が FAT などの管理領域（フラッシュメモリのバンク 1 上位アドレス）をチェックし、初期化されているか否かを判断します。

初期化されていないフロッピーディスクには、何もデータを書き込めないのと同じで、初期化されていないビジュアルメモリには、データもアプリケーションも書き込めません。

注意

販売されているビジュアルメモリは、すべて初期化済みの状態で出荷されています。ただし、出荷ロットによっては希に初期化されていないものがあります。

UNFORMAT を選ぶと、次の画面が表示されます。



COMPLETE を実行すると、フラッシュメモリ全体に 00H を書き込み未初期化状態を作ります。QUICK を選ぶと管理領域だけを破壊します。

注意

Memory Card Utility Version 0.64.1 では、QUICK 機能は実装されておりませんので、選択できません。

どちらの場合もビジュアルメモリに保存してあったデータは削除されます。QUICK を選んだ場合でも、データのサルベージ（復活）手段はありません。

CHECK DISK

このメニューを実行させると、次の画面が表示され FAT の整合性とデータのビット落ちなどをチェックします。Windows のスキャンディスクや MS-DOS の CHKDSK に相当する機能です。

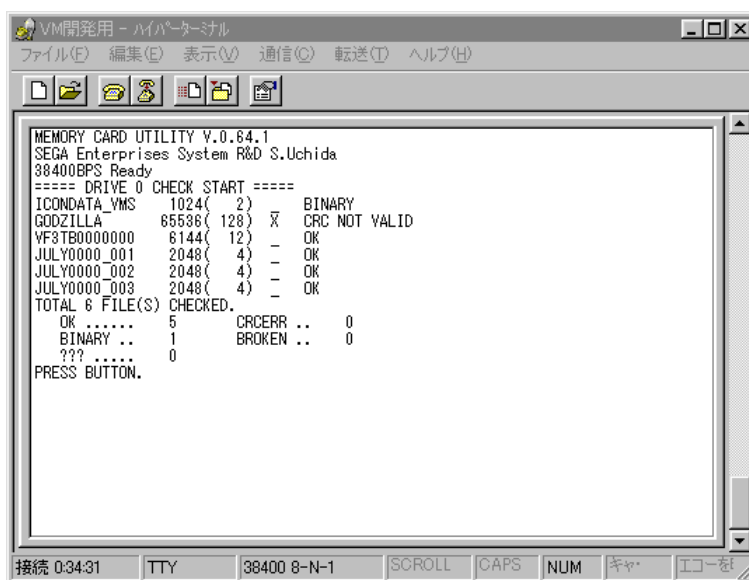
ただし、エラー修復機能はありません。

また、各ファイルのインフォメーションフォークの CRC をチェックし、ファイルにビット落ちがないかのチェックもあわせて行います。

インフォメーションフォークの CRC を省略した場合は、このチェックでエラーと判断されますが、無視してかまいません。



チェックの結果は、通信ソフト側にも次のように表示されます。



3.2.4 ファイル操作メニュー

ファイル一覧に マークを移動し、ファイルを選んで A ボタンを押すとファイル操作メニューが表示されます。

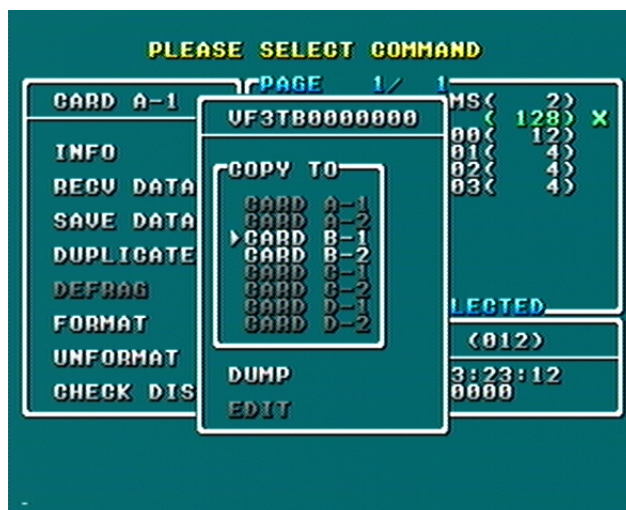


また、ファイルを選んで X ボタンを押すと複数のファイルを選択できます。複数ファイルを選んでいて A ボタンを押すと、選択されているファイルすべてに対して、メニューの操作が行われます。

Y ボタンを押すと、すべてのファイルについて、選択状態のものは非選択状態に、非選択状態のものは選択状態になります。

COPY

COPY を選ぶと次の画面が表示され、選択されたファイルを別のビジュアルメモリにコピーします。



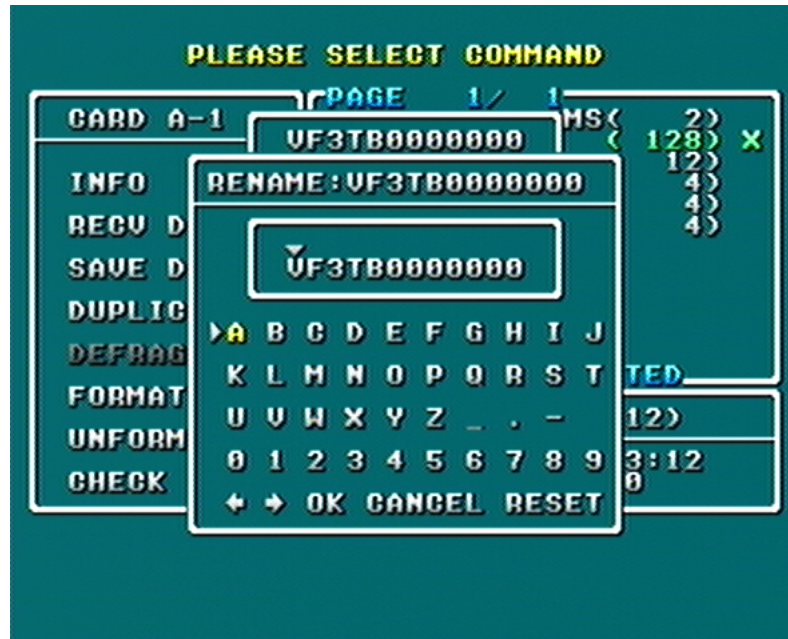
DELETE

選択されたファイルを消去します。DELETE を選んだあとに確認メッセージが表示されます。

RENAME

ファイル名を変更します。複数のファイルを選択している場合は選べません。

RENAME を選ぶと次の画面が表示され、ファイル名の変更ができます。



注意

Memory Card Utility はファイル名として“ - ”が利用できるように作成されていますが、利用しないようにしてください。

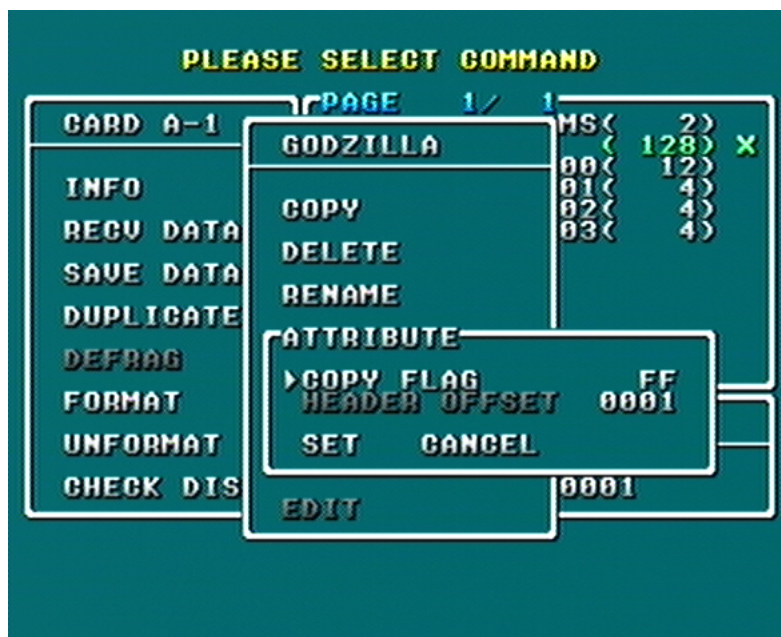
デバッグ用に“ - ”を利用するには問題ありませんが、製品には使用しないでください。ソフトウェア作成基準に反したアプリケーションになります。

はカーソルを移動します。OK を選ぶと入力したファイル名に変更されます。CANCEL は B ボタン相当です。RESET を選ぶと、入力中のファイル名を破棄して、元のファイル名を表示します。

このとき、R トリガーと L トリガーでカーソルを移動できます。またスタートボタンを押すと、OK を選んだことになります。

ATTRIBUTE

ATTRIBUTE を選ぶと次の画面が表示され、ファイルの属性（コピー禁止／許可）を変更します。



COPY FLAG を FF にすると、そのファイルはコピー禁止となります。コピー禁止ファイルは、Dreamcast のファイル管理画面でコピーができなくなります。これ以外の値（00～EF）は、コピー可能ファイルです。FF 以外の値は任意で利用できるもので、何世代目のコピーなのかを知るプログラムを作ることができます。

注意

COPY FLAG が 00～FE のファイルを Dreamcast のファイル管理画面でコピーした場合、コピー先の COPY FLAG は 00 になるので注意してください。

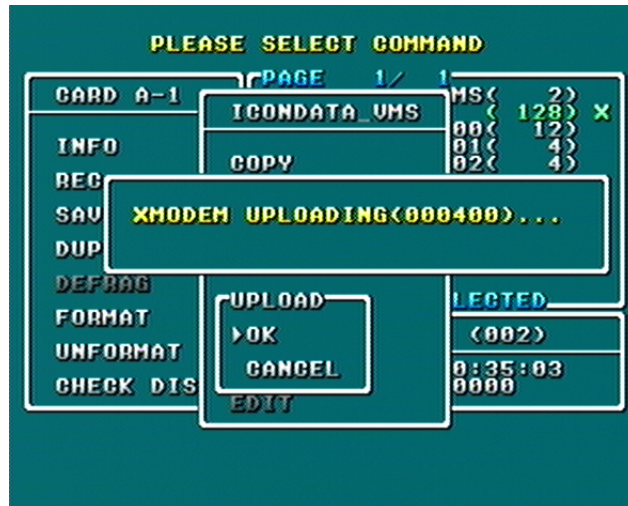
HEADER OFFSET は変更できません。

UPLOAD

選択したファイルを PC に転送することができます。

UPLOAD を選ぶ前に、通信ソフトのファイル転送で Xmodem でのダウンロードを実行してください。Xmodem であれば、CRC でも 1024 でもかまいません。また、ファイル名を入力する画面が表示されますので、適当なファイル名を付けてください。ここで付けるファイル名は、ビジュアルメモリとまったく同じである必要はありません。

UPLOAD を選ぶと“ NOW LOADING...” のメッセージが表示されたあと、PC へのファイル転送が始まります。



注意

UPLOAD の中断はできません。中断するには、転送中のファイルを転送しきってください。また複数のファイルが選択状態の場合は選べません。

INFO

INFO を選ぶと“ NOW LOADING...” の表示の後、次の画面が表示されます。

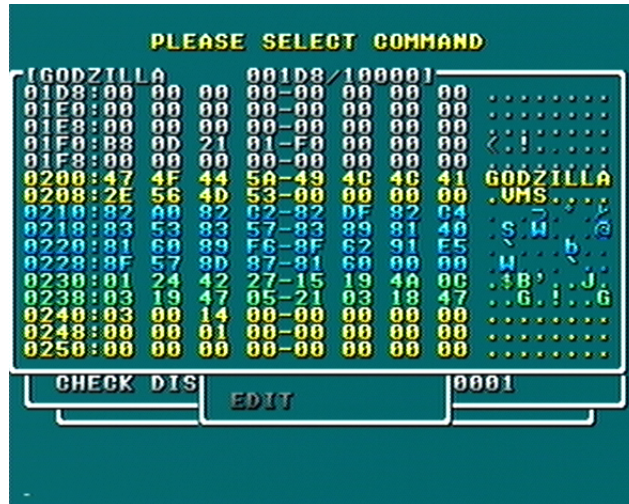


この画面には、インフォメーションフォークの内容が表示されます。

ゲーム名(ソートキー)も対応する文字で表示されますので、入力後の確認に利用できます。

DUMP

DUMP を選ぶと“ NOW LOADING...”の表示の後、ファイルダンプ画面が表示されます。



インフォメーションフォークの部分については、色付き文字で表示されます。また、右側にはキャラクタダンプが表示されます。

注意

複数ファイルが選択状態の場合は選べません。
キャラクタダンプでは、漢字の表示はできません。

ファイルダンプ画面では、次のボタンが利用できます。

上方向ボタン	ファイルの先頭に向かってスクロールします。
下方向ボタン	ファイルの末尾に向かってスクロールします。
左方向ボタン	ファイルの先頭に向かって高速にスクロールします。
右方向ボタン	ファイルの末尾に向かって高速にスクロールします。
L トリガー	ファイルの先頭を表示します。
R トリガー	ファイルの末尾を表示します。
X ボタン	ボタンを押すごとにバイト、WORD(2 バイト)、DWORD(4 バイト)区切りで表示されます。
スタートボタン	ファイルダンプを中止します。

EDIT

将来の拡張用です。Version 0.64.1 では編集機能は実装されていないため選べません。

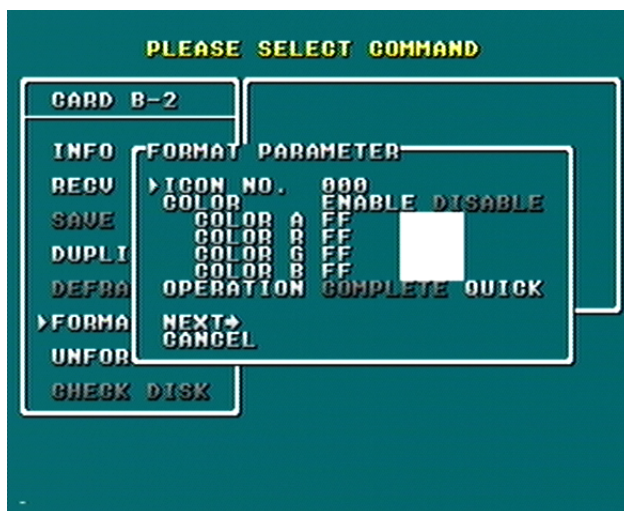
3.3 ビジュアルメモリの初期化

ここではビジュアルメモリの初期化の手順について説明します。

- ①初期化するビジュアルメモリを選び、コマンド選択メニューを表示させます。



- ②FORMAT を選んで次の画面を表示させます。



- ③ICON NO. には、ラベルアイコンの番号を指定します。00 を指定するとデフォルトのビジュアルメモリのアイコンになります。値は 000 ~ 123 までで指定します。124 以上は設定しないでください。

参照

ラベルアイコンのデザインと番号は「[付録 ラベルアイコン一覧](#)」を参照してください。

④次で指定するボディーカラー情報を有効にするか無効にするかを設定します。

有効にする場合は ENABLE を、無効にする場合は DISABLE を選びます。無効にするとボディーカラーは白になり、カラー情報の設定が選べなくなります。

⑤カラー情報を設定します。COLOR A は透明度です。FF で不透明、00 で透明になります。

COLOR R , G , B は、それぞれ赤、緑、青の成分をどれだけ発色させるかを指定します。FF で最大、00 で発色しなくなります。

注意

A を 00 にすると透明になってしまいますので注意してください。

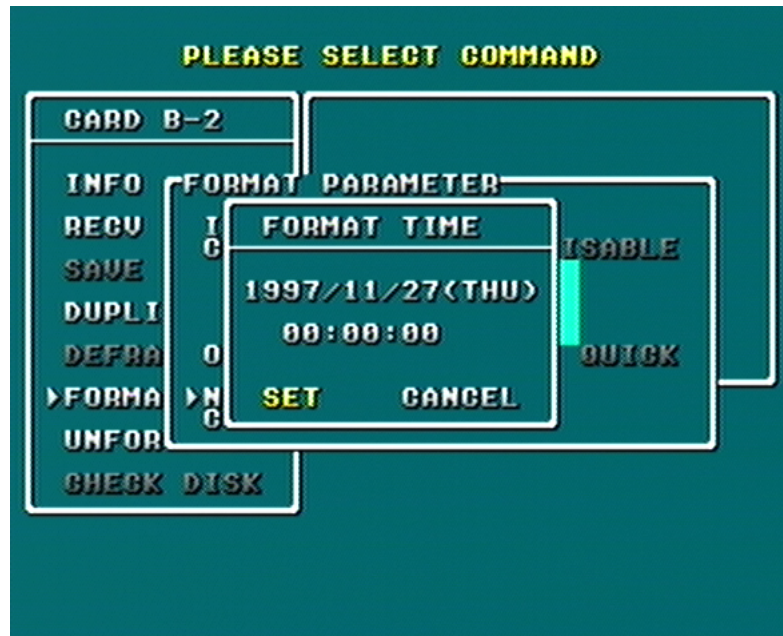
⑥OPERATION は、FAT のみを初期化するか、メモリ全体を初期化するかを指定します。

FAT のみを初期化する場合は QUICK を、メモリ全体を初期化する場合は COMPLETE を選びます。

⑦NEXT を選び、続きの画面を表示させます。



- ⑧ 初期化した日時を設定します。右左の方向ボタンで年月日と日時を選び、上下方向ボタンで値を変えます。



- ⑨ 最後に SET を選ぶと、確認メッセージが表示され OK を選ぶと初期化が始まります。

注意

初期化すると書き込んであったファイルはすべて消去されます。OPERATION で QUICK を選んだ場合でもサルベージする手段はありませんので注意してください。

3.4 PC からビジュアルメモリへのファイル転送

ここでは PC からビジュアルメモリへのファイル転送の手順を説明します。通信ソフトは Windows98 に添付のハイパーターミナルを利用しています。

①ファイルを転送するビジュアルメモリを選んで、コマンド選択メニューを表示させます。



②RECV DATA を選び、転送するファイル用のバッファを選択します。

アプリケーションを転送する場合は GAME BUFFER を選びます。

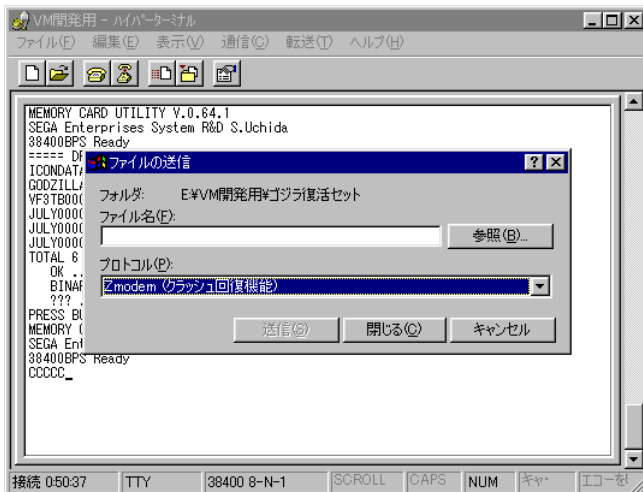
ボリュームアイコン (" ICONDATA_VMS ") を転送する場合は、ICON BUFFER を選択します。

データファイルを転送する場合は、NORMAL BUFFER を選択します。



③確認メッセージが表示されたら OK を選択します。

- ④通信ソフトのファイル転送を選びます。[転送] メニューの[ファイルの送信] を選びます。



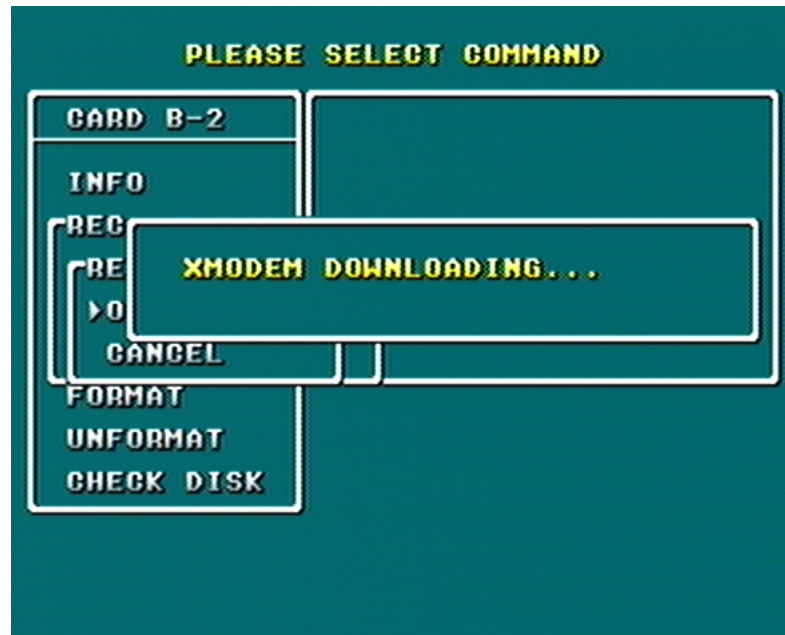
- ⑤プロトコルは[Xmodem]を選びます。また[参照] ボタンを使って、転送するファイルを指定します。最後に[送信] ボタンをクリックします。



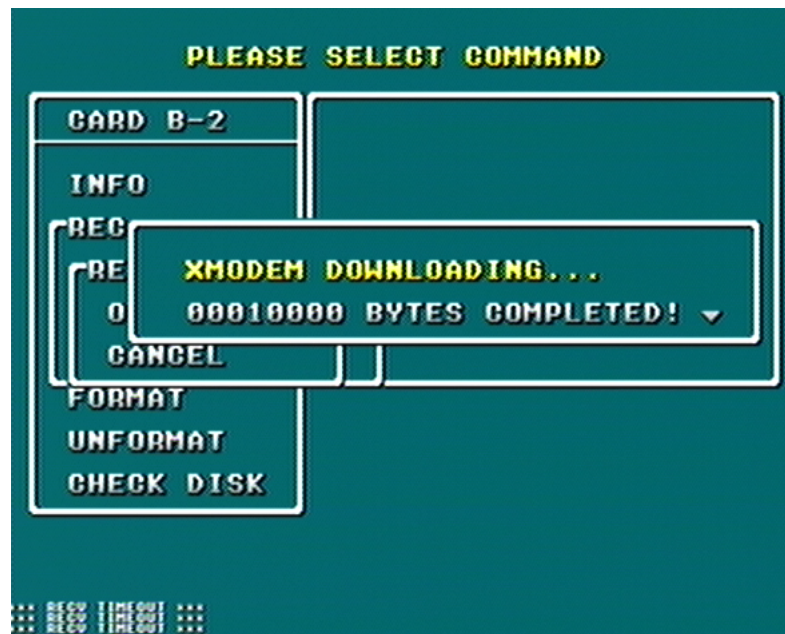
- ⑥ファイル転送中は、それぞれに次の画面が表示されます。

通信ソフト側





- ⑦ ファイルの転送が終わると Dev.Box 側に次の画面が表示されます。A ボタンを押してください。



- ⑧SEVE DATA を選び、どのバッファの内容をビジュアルメモリに書き込むかを指定します。
- アプリケーションを書き込む場合は、GAME BUFFER を選びます。
- ポリウムアイコン(" ICONDATA_VMS ")を書き込む場合は、ICON BUFFER を選びます。
- データファイルを書き込む場合は、NORMAL BUFFER を選びます。



注意

すでにビジュアルメモリにアプリケーションが書き込まれている場合は、GAME BUFFER を選べません。ファイルを削除してから、再度 SAVE DATA を選んでください。

- ⑨ICON BUFFER 以外を選んだ場合は、次の画面が表示されます。



ファイル名を入力して OK を選んでください。ファイル名はピリオドをはさんで 8.3 形式になっている必要はありません。

すでに同名のファイル名がある場合は、上書きするかどうかを確認されます。

これで PC のファイルをビジュアルメモリに転送できました。

アプリケーションを転送した場合は、メモリ選択画面まで戻ってからビジュアルメモリを切り離し、ゲームモードにしてアプリケーションを実行してください。

付録 A

リトルエンディアン形式

複数バイトのデータをメモリに格納する際に CPU によっては、上位のバイトからメモリの上位バイトに格納していく形式と、上位のバイトからメモリの下位に格納していく形式があります。

たとえば、unsigned long int (符号なし 32 ビット整数) で内容が 00 FE 2E EF というデータをメモリに保存する場合、次の 2 通りがあります。

ビッグエンディアン形式

	+00	+01	+02	+03
0000	00	FE	2E	EF

リトルエンディアン形式

	+00	+01	+02	+03
0000	EF	2E	FE	00

このように、データの上位バイトと下位バイトを反転させてメモリに格納する形式をリトルエンディアンと呼びます。逆に反転させないでそのまま格納することをビッグエンディアンと呼びます。

Dreamcast に内蔵の SH4 では、リトルエンディアン方式を採用しているため、バイトデータ以外はすべて、リトルエンディアンでメモリに格納する必要があります。

たとえば、“ ICONDATA_VMS ”の中にある“ モノクロアイコンデータの先頭アドレス ”は、00 00 00 20 を指定する必要があるので、メモリに格納する場合は、20 00 00 00 という並びで格納します。また、符号なし 16 ビットデータ (00 FF) をメモリに格納する場合は、FF 00 の順で格納します。

ラベルアイコン一覧

Dreamcast の BOOT ROM に内蔵のラベルアイコンは、次のとおりです。

アイコン	アイコン番号	
	16進	10進
	00	0
	01	1
	02	2
	03	3
	04	4
	05	5
	06	6
	07	7
	08	8
	09	9
	0A	10
	0B	11
	0C	12
	0D	13
	0E	14
	0F	15
	10	16
	11	17
	12	18
	13	19
	14	20
	15	21
	16	22
	17	23
	18	24
	19	25
	1A	26
	1B	27
	1C	28
	1D	29
	1E	30

アイコン	アイコン番号	
	16進	10進
	1F	31
	20	32
	21	33
	22	34
	23	35
	24	36
	25	37
	26	38
	27	39
	28	40
	29	41
	2A	42
	2B	43
	2C	44
	2D	45
	2E	46
	2F	47
	30	48
	31	49
	32	50
	33	51
	34	52
	35	53
	36	54
	37	55
	38	56
	39	57
	3A	58
	3B	59
	3C	60
	3D	61

アイコン	アイコン番号	
	16進	10進
	3E	62
	3F	63
	40	64
	41	65
	42	66
	43	67
	44	68
	45	69
	46	70
	47	71
	48	72
	49	73
	4A	74
	4B	75
	4C	76
	4D	77
	4E	78
	4F	79
	50	80
	51	81
	52	82
	53	83
	54	84
	55	85
	56	86
	57	87
	58	88
	59	89
	5A	90
	5B	91
	5C	92

アイコン	アイコン番号	
	16進	10進
	5D	93
	5E	94
	5F	95
	60	96
	61	97
	62	98
	63	99
	64	100
	65	101
	66	102
	67	103
	68	104
	69	105
	6A	106
	6B	107
	6C	108
	6D	109
	6E	110
	6F	111
	70	112
	71	113
	72	114
	73	115
	74	116
	75	117
	76	118
	77	119
	78	120
	79	121
	7A	122
	7B	123

ここでは、ビジュアルメモリ SDK に付属のサンプルプログラムを掲載します。
インフォメーションフォークのソースファイル“ IFORK.ASM ”の説明、“ GHEAD.ASM ”
の詳細は割愛します。

注意

すべてのサンプルプログラムは、タブ(09H)を4(バイト)にしてご覧ください。

C.1 LCD パターン表示

このサンプルでは、LCD (XRAM) に単純なパターンを表示させます。
51, 55, 59, 63, 67 行目でパターンを指定し、matrix ルーチンで LCD 全体にこのパターンを描画します。

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** LCD表示処理 サンプル1 **
005 ;
006 ; 表示用RAMにデータを転送してLCDに単純なパターンを表示する
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip    LC868700          ; チップの種類をアセンブラに指定
012 world   external         ; 外部メモリ用プログラム
013
014 public  main              ; ghead.asm から参照されるシンボル
015
016 extern  _game_end         ; アプリケーション終了
017
018
019 ; **** システム定数の定義 ****
020
021                                ; OCR(発振制御レジスタ)設定値
022 osc_rc    equ 081h         ; システムクロックに内蔵RC発振を指定
023 osc_xt    equ 082h         ; システムクロックに水晶発振を指定
024
025
026 ; *** データセグメント ****
027
028          dseg              ; データセグメント開始
029
030 r0:      ds      1         ; 間接アドレッシングレジスタ r0
031 r1:      ds      1         ; 間接アドレッシングレジスタ r1
032 r2:      ds      1         ; 間接アドレッシングレジスタ r2
033 r3:      ds      1         ; 間接アドレッシングレジスタ r3
034          ds      12        ; その他のシステム予約レジスタ
035

```

```

036
037 ; *** コードセグメント *****
038
039         cseg                ; コードセグメント開始
040
041 ; *-----*
042 ; * ユーザープログラム *
043 ; *-----*
044 main:
045     mov     #0f0h,c          ; 表示用データ
046     call    matrix           ; LCDにパターンを表示
047     set1    PCON,0           ; HALTモードに移行して割り込みを待つ。
048                                     ; ベースタイマの割り込みがかかることにより
049                                     ; HALTモードは解除され処理が先に進む。
050
051     mov     #00fh,c          ; 以下同様に何パターンが表示する
052     call    matrix
053     set1    PCON,0
054
055     mov     #0cch,c
056     call    matrix
057     set1    PCON,0
058
059     mov     #033h,c
060     call    matrix
061     set1    PCON,0
062
063     mov     #055h,c
064     call    matrix
065     set1    PCON,0
066
067     mov     #0aah,c
068     call    matrix
069     set1    PCON,0
070
071                                     ; ** [M](モード)ボタンチェック **
072     ld      P3
073     bn      acc,6,finish      ; [M] ボタンが押されていたらアプリ終了
074
075     jmp     main              ; 繰り返す
076
077 finish:
078     jmp     _game_end         ; ** アプリケーション終了処理 **
079                                     ; アプリケーション終了
080
081 ; *-----*
082 ; * LCD全体にパターンを表示する *
083 ; * 入力      c: 基本表示パターン *
084 ; *-----*
085 matrix:
086                                     ; **** LCD 一面を描く ****
087     push    acc                ; 各レジスタを退避する
088     push    b
089     push    c
090     push    XBNK
091
092 xb0_a:    mov     #000h,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK0)
093     mov     #080h,b
094
095 la1:      ld      c              ; c:表示データ
096     call    line2              ; 2行表示
097     ld      b              ; 2行先へアドレスを進める
098     add     #010h            ;
099     st      b              ;
100     bnz     la1              ; バンクの終端まで繰り返し
101
102 xb1_a:    mov     #001h,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK1)
103     mov     #080h,b
104
105 la2:      ld      c              ; c:表示データ
106     call    line2              ; 2行表示
107     ld      b              ; 2行先へアドレスを進める

```

```

108      add    #010h      ;
109      st     b          ;
110      bnz    la2        ; バンクの終端まで繰り返し
111
112      pop     XBNK       ; 退避レジスタの復帰
113      pop     c          ;
114      pop     b          ;
115      pop     acc        ;
116
117      ret              ; matrix終了
118
119
120 line2:              ; **** LCD 2行分表示 ****
121
122      push    acc        ; 各レジスタを退避する
123      push    b          ;
124      push    c          ;
125      push    PSW        ;
126      push    OCR        ;
127      mov     #osc_rc,OCR ; システムクロックを指定
128      set1    PSW,1      ; データRAMバンク1を選択
129      st     c          ; 表示データを c に退避
130      ld     b          ; 表示用RAMアドレスを、r2にセット
131      st     r2         ;
132
133 lp1:                ; **** 1行目表示処理 ****
134      ld     c          ; 表示データを表示用RAMに転送
135      st     @r2        ;
136      inc     r2        ; 次の表示位置にアドレスを移動
137      ld     r2        ;
138      and     #00fh      ; 表示位置が 1行目右端にくるまで..
139      xor     #006h      ;
140      bnz     lp1        ; 繰り返し
141
142      ld     c          ; cレジスタのビットパターンを反転する
143      xor     #0ffh      ;
144      st     c          ;
145
146 lp2:                ; **** 2行目表示処理 ****
147      ld     c          ; 表示データを表示用RAMに転送
148      st     @r2        ;
149      inc     r2        ; 次の表示位置にアドレスを移動
150      ld     r2        ;
151      and     #00fh      ; 表示位置が 2行目右端にくるまで..
152      xor     #00ch      ;
153      bnz     lp2        ; 繰り返し
154
155      pop     OCR        ; 退避レジスタを復帰
156      pop     PSW        ;
157      pop     c          ;
158      pop     b          ;
159      pop     acc        ;
160
161      ret              ; line2終了
162
163      end

```

C.2 LCD キャラクタパターン表示

このサンプルでは、LCD (XRAM) に “ SEGA 1998 ” という文字を表示させます。

内蔵のフォントはアプリケーションから利用できないので、表示させるフォントパターンをあらかじめ用意する必要があります。

メインルーチンの B , C レジスタで指定した座標に指定のフォントパターンを書き込む putch を呼び出します。フォント情報は、222 行目以降に 8 × 8 ドットのデータを持たせています。

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** LCD表示処理 サンプル2 **
005 ;
006 ; ・表示用RAMを0でフィルして表示イメージを消去する
007 ; ・キャラクタパターンを指定位置に表示する
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip      LC868700          ; チップの種類をアセンブラに指定
013 world     external         ; 外部メモリ用プログラム
014
015 public    main              ; ghead.asm から参照されるシンボル
016
017 extern    _game_end         ; アプリケーション終了
018
019
020 ; **** システム定数の定義 ****
021
022                                ; OCR(発振制御レジスタ)設定値
023 osc_rc     equ 081h         ; システムクロックに内蔵RC発振を指定
024 osc_xt     equ 082h         ; システムクロックに水晶発振を指定
025
026
027 ; *** データセグメント ****
028
029          dseg                ; データセグメント開始
030
031 r0:       ds      1          ; 間接アドレッシングレジスタ r0
032 r1:       ds      1          ; 間接アドレッシングレジスタ r1
033 r2:       ds      1          ; 間接アドレッシングレジスタ r2
034 r3:       ds      1          ; 間接アドレッシングレジスタ r3
035          ds      12          ; その他のシステム予約レジスタ
036
037
038 ; *** コードセグメント ****
039
040          cseg                ; コードセグメント開始
041
042 ; *-----*
043 ; * ユーザー プログラム *
044 ; *-----*
045 main:
046          call     cls         ; LCD表示イメージを消去する
047
048          mov      #1,c        ; 水平座標
049          mov      #1,b        ; 垂直座標
050          mov      #0ah,acc     ; キャラクタコード 'S'
051          call     putch       ; 一文字表示
052
053          mov      #2,c        ;
054          mov      #1,b        ;
055          mov      #0bh,acc     ; 'E'
056          call     putch
```

```

057
058     mov     #3,c
059     mov     #1,b
060     mov     #0ch,acc    ; 'G'
061     call    putch
062
063     mov     #4,c
064     mov     #1,b
065     mov     #0dh,acc    ; 'A'
066     call    putch
067
068     mov     #1,c
069     mov     #2,b
070     mov     #1,acc      ; '1'
071     call    putch
072
073     mov     #2,c
074     mov     #2,b
075     mov     #9,acc      ; '9'
076     call    putch
077
078     mov     #3,c
079     mov     #2,b
080     mov     #9,acc      ; '9'
081     call    putch
082
083     mov     #4,c
084     mov     #2,b
085     mov     #8,acc      ; '8'
086     call    putch
087
088 loop0:                                ; ** [M](モード)ボタンチェック **
089     ld      P3
090     bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
091
092     jmp     loop0          ; ループして待つ
093
094 finish:                                ; ** アプリケーション終了処理 **
095     jmp     _game_end      ; アプリケーション終了
096
097
098 ; *-----*
099 ; * LCD表示イメージを消去する *
100 ; *-----*
101 cls:
102     push    OCR            ; OCR値を退避
103     mov     #osc_rc,OCR    ; システムクロックを指定
104
105     mov     #0,XBNK        ; 表示用RAMのバンクアドレスを指定(BANK0)
106     call    cls_s          ; そのバンク内のデータをクリア
107
108     mov     #1,XBNK        ; 表示用RAMのバンクアドレスを指定(BANK1)
109     call    cls_s          ; そのバンク内のデータをクリア
110     pop     OCR            ; OCR値を復帰
111
112     ret                                ; cls終わり
113
114 cls_s:                                ; *** 表示用RAM 1BANK分の消去 ***
115     mov     #80h,r2        ; 間接アドレッシングレジスタを表示用RAMの先頭に
116     mov     #80h,b         ; ループカウンタbにループ数をセット
117 loop3:
118     mov     #0,@r2         ; アドレスをインクリメントしながら0を書き込む
119     inc     r2              ;
120     dbnz    b,loop3        ; bが0になるまで繰り返す
121
122     ret                                ; cls_s終わり
123
124
125 ; *-----*
126 ; * 指定位置に1キャラクタ表示する *
127 ; * 入力 acc : キャラクタコード *
128 ; *      c : 文字水平位置 *

```

```

129 ; *          b : 文字垂直位置          *
130 ; *-----*
131 putch:
132     push    XBNK
133     push    acc
134     call    locate      ; 座標から表示RAMのアドレスを計算
135     pop     acc
136     call    put_chara   ; 1キャラクタ表示する
137     pop     XBNK
138
139     ret              ; putch終わり
140
141
142 locate: ; **** 表示位置指定から表示用RAMのアドレスを計算 ****
143         ; ** 入力 c: 水平位置(0~5) b: 垂直位置(0~3)
144         ; ** 出力 r2: RAMアドレス XBNK: 表示用RAMバンク
145
146         ; *** 表示用RAMバンクアドレスの判断 ***
147         ld     b          ; b>=2 のとき next1 へ
148         sub    #2         ;
149         bn     PSW,7,next1 ;
150
151         mov    #00h,XBNK  ; 表示用RAMのバンクアドレスを指定(BANK0)
152         br     next2
153 next1:
154         st     b          ;
155         mov    #01h,XBNK  ; 表示用RAMのバンクアドレスを指定(BANK1)
156 next2:
157
158         ; *** 表示指定位置のRAMアドレス計算 ***
159         ld     b          ; b * 40h + c + 80h
160         rol
161         rol
162         rol
163         rol
164         rol
165         rol
166         add    c          ;
167         add    #80h       ;
168         st     r2        ; RAMアドレスをr2に格納
169
170         ret              ; locate終わり
171
172
173 put_chara:
174     push    PSW          ; PSW値を退避
175     set1    PSW,1        ; データRAMバンク1を選択
176
177         ; *** キャラクタデータアドレスの計算 ***
178         rol              ; (TRH,TRL) = acc*8 + fontdata
179         rol
180         rol
181         add    #low(fontdata) ;
182         st     TRL        ;
183         mov    #0,acc      ;
184         addc   #high(fontdata) ;
185         st     TRH        ;
186
187         push    OCR        ; OCR値を退避
188         mov     #osc_rc,OCR ; システムクロックを指定
189
190         mov     #0,b        ; キャラクタデータ読み出し用オフセット値
191         mov     #4,c        ; ループカウンタ
192 loop1:
193         ld     b          ; 1ライン目の表示データを読み出す
194         ldc
195         inc     b          ; 読み出しデータのオフセットを +1
196         st     @r2        ; 表示データを表示用RAMに転送
197         ld     r2         ; 表示用RAMアドレス +6
198         add    #6         ;
199         st     r2         ;
200

```

```

201      ld      b          ; 2ライン目の表示データを読み出す
202      ldc          ;
203      inc      b          ; 読み出しデータのオフセットを +1
204      st      @r2        ; 表示データを表示用RAMに転送
205      ld      r2         ; 表示用RAMアドレス +10
206      add      #10       ;
207      st      r2         ;
208
209      dec      c          ; ループカウンタのデクリメント
210      ld      c          ;
211      bnz      loop1      ; 8ライン分(4回)繰り返し
212
213      pop      OCR        ; OCR値を復帰
214      pop      PSW       ; PSW値を復帰
215
216      ret              ; put_chara終わり
217
218
219 ; -----*
220 ; * キャラクタのビットイメージデータ *
221 ; -----*
222 fontdata:
223      db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ; '0' 00
224      db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ; '1' 01
225      db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ; '2' 02
226      db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ; '3' 03
227      db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ; '4' 04
228      db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ; '5' 05
229      db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ; '6' 06
230      db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ; '7' 07
231      db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ; '8' 08
232      db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ; '9' 09
233
234      db 07ch, 0e6h, 076h, 038h, 0dch, 0ceh, 07ch, 000h ; 'S' 0a
235      db 0feh, 0c0h, 0c0h, 0f8h, 0c0h, 0c0h, 0feh, 000h ; 'E' 0b
236      db 07ch, 0e6h, 0c0h, 0dch, 0c6h, 0e6h, 07ch, 000h ; 'G' 0c
237      db 01eh, 036h, 066h, 0c6h, 0c6h, 0feh, 0c6h, 000h ; 'A' 0d

```

C.3 ベースタイマー割り込みを使ったカウンタ

ベースタイマーからかかる 0.5 秒ごとの割り込みを検知し、その数をカウントします。

割り込みが発生すると“ GHEAD.ASM ”の 35 行目にジャンプし、さらにその先のラベル INT_1B にジャンプします。108 行目からベースタイマー割り込み処理が始まりますが、ここでは内蔵の時計処理を行っています。112 行目までの ROM 内の時計処理をいったん行った後、“ B_TIMER1.ASM ”にあるラベル int_BaseTimer にジャンプさせます。

“ B_TIMER1.ASM ”では、外部プログラムから参照されるラベル int_BaseTimer を PUBLIC 宣言 (16 行目) しています。242 行目以降は、ユーザー側ベースタイマー割り込みハンドラです。割り込みハンドラ内では、カウンタをインクリメントしています。

再び“ GHEAD.ASM ”に戻り、IE レジスタの内容を int_1b が呼び出されたときの値に戻し、割り込みからの復帰 (IRET) を行います。

メインルーチンでは、カウンタの値を常に LCD に表示しています。

GHEAD.ASM

```
001 chip    LC868700
002 world   external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *                               05/20-'98    *
006 ; *-----*
007
008 public   fm_wrt_ex_exit,fm_vrf_ex_exit
009 public   fm_prd_ex_exit,timer_ex_exit,_game_start,_game_end
010 other_side_symbol   fm_wrt_in,fm_vrf_in
011 other_side_symbol   fm_prd_in,timer_in,game_end
012
013 extern   main          ;ユーザープログラム側にあるシンボル
014 extern   int_BaseTimer ;ユーザープログラム側にあるシンボル
015
016 ; *-----*
017 ; * Vector table(?)                      *
018 ; *-----*
019 cseg
020 org 0000h
021 _game_start:
022 ;reset:
023         jmpf      main          ; main program jump
024 org 0003h
025 ;int_03:
026         jmp int_03
027 org 000bh
028 ;int_0b:
029         jmp int_0b
030 org 0013h
031 ;int_13:
032         jmp int_13
033 org 001bh
034 ;int_1b:
035         jmp int_1b
036 org 0023h
037 ;int_23:
038         jmp int_23
039 org 002bh
040 ;int_2b:
041         jmp int_2b
042 org 0033h
043 ;int_33:
044         jmp int_33
045 org 003bh
046 ;int_3b:
```

```

047         jmp int_3b
048 org 0043h
049 ;int_43:
050         jmp int_43
051 org 004bh
052 ;int_4b:
053         jmp int_4b
054 ; *-----*
055 ; * interrupt programs                *
056 ; *-----*
057 int_03:
058         reti
059 int_0b:
060         reti
061 int_13:
062         reti
063 int_23:
064         reti
065 int_2b:
066         reti
067 int_33:
068         reti
069 int_3b:
070         reti
071 ; *-----*
072 int_43:
073         reti
074 int_4b:
075         clr1    p3int,1    ;interrupt flag clear
076         reti
077
078 org 0100h
079 ; *-----*
080 ; * flash memory write external program      *
081 ; *-----*
082 fm_wrt_ex:
083         change  fm_wrt_in
084         fm_wrt_ex_exit:
085         ret
086 org 0110h
087 ; *-----*
088 ; * flash memory verify external program      *
089 ; *-----*
090 fm_vrf_ex:
091         change  fm_vrf_in
092         fm_vrf_ex_exit:
093         ret
094
095 org 0120h
096 ; *-----*
097 ; * flash memory page read external program      *
098 ; *-----*
099 fm_prd_ex:
100         change  fm_prd_in
101         fm_prd_ex_exit:
102         ret
103
104 org 0130h
105 ; *-----*
106 ; * flash memory => timer call external program *
107 ; *-----*
108 int_1b:
109 timer_ex:
110         push    ie
111         clr1    ie,7    ;interrupt prohibition
112         change  timer_in
113         timer_ex_exit:
114         call    int_BaseTimer    ;(ユーザー側ベースタイマ割り込み処理)
115         pop ie
116         reti
117
118 org 01f0h

```

```

119 _game_end:
120     change game_end
121 end

```

B_TIMER1.ASM

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** ベースタイマ割り込み利用サンプル1 **
005 ;
006 ; ・ベースタイマ割り込み(0.5秒ごと)をカウントする
007 ; ・カウンタの値を2桁10進数でLCDに表示する
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip    LC868700          ; チップの種類をアセンブラに指定
013 world   external         ; 外部メモリ用プログラム
014
015 public  main              ; ghead.asm から参照されるシンボル
016 public  int_BaseTimer    ; ghead.asm から参照されるシンボル
017
018 extern  _game_end         ; ghead.asm への参照シンボル
019
020
021 ; **** システム定数の定義 ****
022
023                                ; OCR(発振制御レジスタ)設定値
024 osc_rc      equ 081h      ; システムクロックに内蔵RC発振を指定
025 osc_xt      equ 082h      ; システムクロックに水晶発振を指定
026
027
028 ; *** データセグメント ****
029
030          dseg              ; データセグメント開始
031
032 r0:      ds      1          ; 間接アドレッシングレジスタ r0
033 r1:      ds      1          ; 間接アドレッシングレジスタ r1
034 r2:      ds      1          ; 間接アドレッシングレジスタ r2
035 r3:      ds      1          ; 間接アドレッシングレジスタ r3
036          ds      12         ; その他のシステム予約レジスタ
037
038 counter: ds      1          ; ベースタイマ割り込みカウンタ
039 work1:   ds      1          ; 作業用 (put2digit)
040
041
042 ; *** コードセグメント ****
043
044          cseg              ; コードセグメント開始
045
046 ; *-----*
047 ; * ユーザープログラム *
048 ; *-----*
049 main:
050     mov    #0,counter      ; カウンタの値をリセット
051
052     call   cls             ; LCD表示イメージを消去する
053
054 loop0:
055     mov    #2,c            ; 表示位置(水平)
056     mov    #1,b            ; 表示位置(垂直)
057     ld     counter         ; カウンタの値をaccへ
058     call   put2digit       ; accの値(2桁)を数値表示する
059
060     set1   pcon,0          ; 次の割り込みまでHALTモードで待つ
061
062                                ; ** [M](モード)ボタンチェック **
063     ld     P3
064     bn     acc,6,finish    ; [M]ボタンが押されていたらアプリ終了

```

```

065
066     br      loop0      ; 繰り返し
067
068 finish:                ; ** アプリケーション終了処理 **
069     jmp     _game_end   ; アプリケーション終了
070
071
072 ; *-----*
073 ; * 2桁の数値を表示する
074 ; * 入力 acc : 数値
075 ; *      c : 文字水平位置
076 ; *      b : 文字垂直位置
077 ; *-----*
078 put2digit:
079     push    b           ; 座標データを退避
080     push    c           ;
081     st      c           ; 10の位と1の位の値をそれぞれ計算
082     xor     a           ; ( acc = acc/10, work1 = acc mod 10 )
083     mov     #10,b       ;
084     div     ;
085     ld      b           ;
086     st      work1       ; 1の位の計算結果をwork1に保存
087     ld      c           ;
088     pop     c           ; 座標値を(c,b)に復帰
089     pop     b           ;
090     push    b           ; また退避
091     push    c           ;
092     call    putch       ; 10の位を表示
093     ld      work1       ; 1の位の値を読み込み
094     pop     c           ; 座標値を(c,b)に復帰
095     pop     b           ;
096     inc     c           ; 表示座標を右へ
097     call    putch       ; 1の位を表示
098
099     ret            ; put2digit終わり
100
101
102 ; *-----*
103 ; * LCD表示イメージを消去する
104 ; *-----*
105 cls:
106     push    OCR         ; OCR値を退避
107     mov     #osc_rc,OCR ; システムクロックを指定
108
109     mov     #0,XBNK     ; 表示用RAMのバンクアドレスを指定(BANK0)
110     call    cls_s       ; そのバンク内のデータをクリア
111
112     mov     #1,XBNK     ; 表示用RAMのバンクアドレスを指定(BANK1)
113     call    cls_s       ; そのバンク内のデータをクリア
114     pop     OCR         ; OCR値を復帰
115
116     ret            ; cls終わり
117
118 cls_s:                ; *** 表示用RAM 1BANK分の消去 ***
119     mov     #80h,r2     ; 間接アドレッシングレジスタを表示用RAMの先頭に
120     mov     #80h,b      ; ループカウンタbにループ数をセット
121 loop3:
122     mov     #0,@r2      ; アドレスをインクリメントしながら0を書き込む
123     inc     r2          ;
124     dbnz    b,loop3     ; bが0になるまで繰り返す
125
126     ret            ; cls_s終わり
127
128
129 ; *-----*
130 ; * 指定位置に1キャラクタ表示する
131 ; * 入力 acc : キャラクタコード
132 ; *      c : 文字水平位置
133 ; *      b : 文字垂直位置
134 ; *-----*
135 putch:
136     push    XBNK

```

```

137     push    acc
138     call    locate    ; 座標から表示RAMのアドレスを計算
139     pop     acc
140     call    put_chara  ; 1キャラクタ表示する
141     pop     XBNK
142
143     ret                      ; putch終わり
144
145
146 locate: ; **** 表示位置指定から表示用RAMのアドレスを計算 ****
147     ; ** 入力 c: 水平位置(0~5) b: 垂直位置(0~3)
148     ; ** 出力 r2: RAMアドレス XBNK: 表示用RAMバンク
149
150     ; *** 表示用RAMバンクアドレスの判断 ***
151     ld      b          ; b>=2 のとき next1 へ
152     sub     #2         ;
153     bn      PSW,7,next1 ;
154
155     mov     #00h,XBNK  ; 表示用RAMのバンクアドレスを指定(BANK0)
156     br      next2
157 next1:
158     st      b
159     mov     #01h,XBNK  ; 表示用RAMのバンクアドレスを指定(BANK1)
160 next2:
161
162     ; *** 表示指定位置のRAMアドレス計算 ***
163     ld      b          ; b * 40h + c + 80h
164     rol
165     rol
166     rol
167     rol
168     rol
169     rol
170     add     c          ;
171     add     #80h       ;
172     st      r2         ; RAMアドレスをr2に格納
173
174     ret                      ; locate終わり
175
176
177 put_chara:
178     push    PSW        ; PSW値を退避
179     set1    PSW,1      ; データRAMバンク1を選択
180
181     ; *** キャラクタデータアドレスの計算 ***
182     rol
183     rol
184     rol
185     add     #low(fontdata) ;
186     st      TRL        ;
187     mov     #0,acc      ;
188     addc    #high(fontdata) ;
189     st      TRH        ;
190
191     push    OCR        ; OCR値を退避
192     mov     #osc_rc,OCR ; システムクロックを指定
193
194     mov     #0,b        ; キャラクタデータ読み出し用オフセット値
195     mov     #4,c        ; ループカウンタ
196 loop1:
197     ld      b          ; 1ライン目の表示データを読み出す
198     ldc
199     inc     b          ; 読み出しデータのオフセットを +1
200     st      @r2        ; 表示データを表示用RAMに転送
201     ld      r2         ; 表示用RAMアドレス +6
202     add     #6         ;
203     st      r2        ;
204
205     ld      b          ; 2ライン目の表示データを読み出す
206     ldc
207     inc     b          ; 読み出しデータのオフセットを +1
208     st      @r2        ; 表示データを表示用RAMに転送

```

```

209      ld      r2          ; 表示用RAMアドレス +10
210      add     #10         ;
211      st      r2          ;
212
213      dec     c            ; ループカウンタのデクリメント
214      ld      c            ;
215      bnz     loop1        ; 8ライン分(4回)繰り返し
216
217      pop     OCR          ; OCR値を復帰
218      pop     PSW          ; PSW値を復帰
219
220      ret              ; put_chara終わり
221
222
223 ; -----*
224 ; * キャラクタのビットイメージデータ *
225 ; -----*
226 fontdata:
227      db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ; '0' 00
228      db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ; '1' 01
229      db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ; '2' 02
230      db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ; '3' 03
231      db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ; '4' 04
232      db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ; '5' 05
233      db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ; '6' 06
234      db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ; '7' 07
235      db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ; '8' 08
236      db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ; '9' 09
237
238
239 ; -----*
240 ; * ベースタイマ割り込みハンドラ部 *
241 ; -----*
242 int_BaseTimer:
243      push    acc          ; 使用レジスタを退避
244      inc     counter      ; カウンタをインクリメント
245      ld      counter      ; カウンタが100になったら..
246      bne     #100,next3   ;
247      mov     #0,counter    ; カウンタのリセット
248 next3:      ;
249      pop     acc          ; 退避レジスタを復帰
250
251      ret              ; (ユーザー側)割り込み処理終了

```

C.4 ボタン押下の検出

ビジュアルメモリのボタン押下状態(リセットボタンとMODEボタンを除く)を調べ、押されているボタンをLCDに表示します。

46行目でポート3に0FFHを書き込み、全ビットをプルアップします。

ボタンの押下状態は、51行目でポート3の状態をACCに読み込みます。この時点で押されているボタンがあれば、対応するビットが0にリセットされています。

52行目では、そのビットがセット(ボタンが押されていない)されているかをチェックし、次のボタンチェック処理に移ります。もしボタンが押されている場合は、52行目の条件は偽となり、対応するボタンを表示する処理を行います。

システム BIOS から呼び出された直後は、ポート3 割り込みが有効になっています。また、ポート3 割り込みはレベル割り込みなので、ボタンが押されている間は割り込みが上がり続けます。

そこで、ポート3 割り込みを無効にすると、アプリケーション全体のパフォーマンスが向上します。

なお、ポート3 割り込みで HALT モードの解除を行うアプリケーションは、その直前でポート3 割り込みを許可する必要があります。

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** ボタン状態検出サンプル1 **
005 ;
006 ; ・ボタンの状態を読み込んで、押下されているボタンをLCDに表示する
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; チップの種類をアセンブラに指定
012 world     external         ; 外部メモリ用プログラム
013
014 public    main              ; ghead.asm から参照されるシンボル
015
016 extern    _game_end         ; ghead.asm への参照シンボル
017
018
019 ; **** システム定数の定義 ****
020
021                                ; OCR(発振制御レジスタ)設定値
022 osc_rc     equ 081h         ; システムクロックに内蔵RC発振を指定
023 osc_xt     equ 082h         ; システムクロックに水晶発振を指定
024
025
026 ; *** データセグメント ****
027
028          dseg                ; データセグメント開始
029
030 r0:       ds      1          ; 間接アドレッシングレジスタ r0
031 r1:       ds      1          ; 間接アドレッシングレジスタ r1
032 r2:       ds      1          ; 間接アドレッシングレジスタ r2
033 r3:       ds      1          ; 間接アドレッシングレジスタ r3
034          ds      12          ; その他のシステム予約レジスタ
035
036 ; *** コードセグメント ****
037
038          cseg                ; コードセグメント開始
039
040 ;-----*
```

```

041 ; * ユーザープログラム *
042 ; *-----*
043 main:
044     call    cls          ; LCD表示イメージを消去する
045
046     mov     #0ffh,P3     ; P3の初期化(プルアップ設定)
047
048 loop0:
049     ; ** [A] ボタンのチェック **
050     mov     #0,b
051     ld      P3           ; P3の状態を読み込む
052     bp      acc,4,next3  ; [A] ボタン押されてなければ next3
053     mov     #1,b        ; 表示文字コード 'A'
054 next3:
055     ld      b
056     mov     #4,c        ; 表示座標(水平)
057     mov     #3,b        ; 表示座標(垂直)
058     call    putchar     ; 一文字表示
059
060     ; ** [B] ボタンのチェック **
061     mov     #0,b
062     ld      P3
063     bp      acc,5,next4  ; [B] ボタン押されてなければ next4
064     mov     #2,b        ; 表示文字コード 'B'
065 next4:
066     ld      b
067     mov     #5,c
068     mov     #2,b
069     call    putchar
070
071     ; ** [ ] ボタンのチェック **
072     mov     #0,b
073     ld      P3
074     bp      acc,0,next5  ; [ ] ボタン押されてなければ next5
075     mov     #3,b        ; 表示文字コード ' '
076 next5:
077     ld      b
078     mov     #1,c
079     mov     #1,b
080     call    putchar
081
082     ; ** [ ] ボタンのチェック **
083     mov     #0,b
084     ld      P3
085     bp      acc,3,next6  ; [ ] ボタン押されてなければ next6
086     mov     #4,b        ; 表示文字コード ' '
087 next6:
088     ld      b
089     mov     #2,c
090     mov     #2,b
091     call    putchar
092
093     ; ** [ ] ボタンのチェック **
094     mov     #0,b
095     ld      P3
096     bp      acc,1,next7  ; [ ] ボタン押されてなければ next7
097     mov     #5,b        ; 表示文字コード ' '
098 next7:
099     ld      b
100     mov     #1,c
101     mov     #3,b
102     call    putchar
103
104     ; ** [ ] ボタンのチェック **
105     mov     #0,b
106     ld      P3
107     bp      acc,2,next8  ; [ ] ボタン押されてなければ next8
108     mov     #6,b        ; 表示文字コード ' '
109 next8:
110     ld      b
111     mov     #0,c
112     mov     #2,b

```

```

113      call    putch
114
115                      ; ** [S] ボタンのチェック **
116      mov     #0,b
117      ld      P3
118      bp      acc,7,next9 ; [S] ボタン押されてなければ next9
119      mov     #8,b      ; 表示文字コード 'S'
120 next9:
121      ld      b
122      mov     #4,c
123      mov     #1,b
124      call    putch
125
126                      ; ** [M] ボタンのチェック **
127      ld      P3
128      bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
129
130      brf     loop0      ; 繰り返し
131
132 finish:
133      jmp     _game_end  ; アプリケーション終了
134
135
136 ; -----*
137 ; * LCD表示イメージを消去する *
138 ; -----*
139 cls:
140      push    OCR        ; OCR値を退避
141      mov     #osc_rc,OCR ; システムクロックを指定
142
143      mov     #0,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK0)
144      call    cls_s      ; そのバンク内のデータをクリア
145
146      mov     #1,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK1)
147      call    cls_s      ; そのバンク内のデータをクリア
148      pop     OCR        ; OCR値を復帰
149
150      ret                     ; cls終わり
151
152 cls_s:
153                      ; *** 表示用RAM 1BANK分の消去 ***
154      mov     #80h,r2    ; 間接アドレッシングレジスタを表示用RAMの先頭に
155      mov     #80h,b      ; ループカウンタbにループ数をセット
156 loop3:
157      mov     #0,@r2      ; アドレスをインクリメントしながら0を書き込む
158      inc     r2          ;
159      dbnz    b,loop3     ; bが0になるまで繰り返す
160      ret                     ; cls_s終わり
161
162
163 ; -----*
164 ; * 指定位置に1キャラクタ表示する *
165 ; * 入力 acc : キャラクタコード *
166 ; *      c : 文字水平位置 *
167 ; *      b : 文字垂直位置 *
168 ; -----*
169 putch:
170      push    XBNK
171      push    acc
172      call    locate      ; 座標から表示RAMのアドレスを計算
173      pop     acc
174      call    put_chara   ; 1キャラクタ表示する
175      pop     XBNK
176
177      ret                     ; putch終わり
178
179
180 locate: ; **** 表示位置指定から表示用RAMのアドレスを計算 ****
181      ; ** 入力 c: 水平位置(0~5) b: 垂直位置(0~3)
182      ; ** 出力 r2: RAMアドレス XBNK: 表示用RAMバンク
183
184                      ; *** 表示用RAMバンクアドレスの判断 ***

```

```

185      ld      b          ; b>=2 のとき next1 へ
186      sub     #2         ;
187      bn      PSW,7,next1 ;
188
189      mov     #00h,XBNK   ; 表示用RAMのバンクアドレスを指定(BANK0)
190      br      next2
191 next1:
192      st      b
193      mov     #01h,XBNK   ; 表示用RAMのバンクアドレスを指定(BANK1)
194 next2:
195
196      ; *** 表示指定位置のRAMアドレス計算 ***
197      ld      b          ; b * 40h + c + 80h
198      rol
199      rol
200      rol
201      rol
202      rol
203      rol
204      add     c          ;
205      add     #80h       ;
206      st      r2         ; RAMアドレスをr2に格納
207
208      ret
209      ; locate終わり
210
211 put_chara:
212      push    PSW        ; PSW値を退避
213      set1    PSW,1      ; データRAMバンク1を選択
214
215      ; *** キャラクターデータアドレスの計算 ***
216      rol
217      rol
218      rol
219      add     #low(fontdata) ;
220      st      TRL        ;
221      mov     #0,acc      ;
222      addc    #high(fontdata) ;
223      st      TRH        ;
224
225      push    OCR        ; OCR値を退避
226      mov     #osc_rc,OCR ; システムクロックを指定
227
228      mov     #0,b        ; キャラクターデータ読み出し用オフセット値
229      mov     #4,c        ; ループカウンタ
230 loop1:
231      ld      b          ; 1ライン目の表示データを読み出す
232      ldc
233      inc     b          ; 読み出しデータのオフセットを +1
234      st      @r2        ; 表示データを表示用RAMに転送
235      ld      r2         ; 表示用RAMアドレス +6
236      add     #6
237      st      r2
238
239      ld      b          ; 2ライン目の表示データを読み出す
240      ldc
241      inc     b          ; 読み出しデータのオフセットを +1
242      st      @r2        ; 表示データを表示用RAMに転送
243      ld      r2         ; 表示用RAMアドレス +10
244      add     #10
245      st      r2
246
247      dec     c          ; ループカウンタのデクリメント
248      ld      c
249      bnz     loop1      ; 8ライン分(4回)繰り返し
250
251      pop     OCR        ; OCR値を復帰
252      pop     PSW        ; PSW値を復帰
253
254      ret
255      ; put_chara終わり
256

```

```

257 ; *-----*
258 ; * キャラクタのビットイメージデータ *
259 ; *-----*
260 fontdata:
261         db 000h, 000h, 038h, 038h, 038h, 000h, 000h, 000h ;' ' ' 00
262         db 01eh, 036h, 066h, 0c6h, 0c6h, 0feh, 0c6h, 000h ;'A' 01
263         db 0fch, 066h, 066h, 07ch, 066h, 066h, 0fch, 000h ;'B' 02
264
265         db 010h, 038h, 07ch, 0feh, 038h, 038h, 038h, 000h ;' ' ' 03
266         db 010h, 018h, 0fch, 0feh, 0fch, 018h, 010h, 000h ;' ' ' 04
267         db 038h, 038h, 038h, 0feh, 07ch, 038h, 010h, 000h ;' ' ' 05
268         db 010h, 030h, 07eh, 0feh, 07eh, 030h, 010h, 000h ;' ' ' 06
269
270         db 0c6h, 0eeh, 0feh, 0d6h, 0c6h, 0c6h, 0c6h, 000h ;'M' 07
271         db 07ch, 0e6h, 076h, 038h, 0dch, 0ceh, 07ch, 000h ;'S' 08

```

C.5 PWM 音源の利用

高い音(約 781Hz)と低い音(約 342Hz)を交互に発生させます。

このサンプルの本質は、72 行目以降のサブルーチンです。

SndInit では、PWM を利用できるようにしています。Snd1(2) Start では、タイマー 1 に発生する周波数を設定し、カウント動作を開始させます。SndStop では、カウント動作を停止し音声出力を停止します。

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** サウンド使用サンプル1 **
005 ;
006 ; ・2種類の音(高/低音)を断続的に出力する
007 ;   (低い音 0.5秒 - 無音 0.5秒 - 高い音 0.5秒 - 無音 0.5秒 ...)
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip    LC868700          ; チップの種類をアセンブラに指定
013 world   external         ; 外部メモリ用プログラム
014
015 public  main              ; ghead.asm から参照されるシンボル
016
017 extern  _game_end         ; アプリケーション終了
018
019
020 ; **** システム定数の定義 ****
021
022                                ; OCR(発振制御レジスタ)設定値
023 osc_rc    equ 081h        ; システムクロックに内蔵RC発振を指定
024 osc_xt    equ 082h        ; システムクロックに水晶発振を指定
025
026
027 ; *** データセグメント ****
028
029          dseg              ; データセグメント開始
030
031 r0:      ds      1          ; 間接アドレッシングレジスタ r0
032 r1:      ds      1          ; 間接アドレッシングレジスタ r1
033 r2:      ds      1          ; 間接アドレッシングレジスタ r2
034 r3:      ds      1          ; 間接アドレッシングレジスタ r3
035          ds      12         ; その他のシステム予約レジスタ
036
037
038 ; *** コードセグメント ****
039
040          cseg              ; コードセグメント開始
041
042 ; *-----*
043 ; * ユーザープログラム *
044 ; *-----*
045 main:
046          call     SndInit    ; サウンド初期化
047
048 loop0:
049          call     Snd1Start   ; 約342Hz の発音開始
050          set1     PCON,0      ; ベースタイマ割り込みまでHALT(0.5秒)
051
052          call     SndStop     ; ブザー音停止
053          set1     PCON,0      ; ベースタイマ割り込みまでHALT(0.5秒)
054
055          call     Snd2Start   ; 約781Hz の発音開始
056          set1     PCON,0      ; ベースタイマ割り込みまでHALT(0.5秒)
057
058          call     SndStop     ; ブザー音停止

```

```

059      set1    PCON,0      ; ベースタイマ割り込みまでHALT(0.5秒)
060
061
062                                ; ** [M](モード)ボタンチェック **
063      ld      P3
064      bn      acc,6,finish ; [M]ボタンが押されていたらアプリ終了
065
066      br      loop0       ; 繰り返し
067
068 finish:                                ; ** アプリケーション終了処理 **
069      jmp     _game_end   ; アプリケーション終了
070
071
072 ; *-----*
073 ; * 音声出力関係 *
074 ; *-----*
075 SndInit:                                ; *** 音声出力ハードの初期化 ***
076      clr1    P1,7        ; 音声出力ポートセット
077
078      ret
079
080 Snd1Start:                                ; *** 約342Hz の発音開始 ***
081      mov     #0f0h,T1LR ; 周期 = 100h-0f0h = 16
082      mov     #0f8h,T1LC ; Lレベル幅 = 100h-0f8h = 8
083      mov     #0D0h,T1CNT ; 音声出力開始
084
085      ret
086
087 Snd2Start:                                ; *** 約781Hz の発音開始 ***
088      mov     #0f9h,T1LR ; 周期 = 100h-0f9h = 7
089      mov     #0fch,T1LC ; Lレベル幅 = 100h-0fch = 4
090      mov     #0D0h,T1CNT ; 音声出力開始
091
092      ret
093
094 SndStop:                                ; *** 発音停止 ***
095      mov     #0,T1CNT    ; 音声出力停止
096
097      ret

```

C.6 タイマー 0 を使った割り込み

タイマー 0 を使って 1 秒ごとの割り込みを発生させます。割り込みが発生したことは、音で知らせます。

このプログラムの本質は、78～93 行目の T0Mode2Init ルーチンです。タイマー 0 のモード 2、プリスケアラ付き 16 ビットカウンタを利用します。タイマーには 32KHz が入るので、それをプリスケアラおよびカウンタで約 1 秒ごとにオーバーフローを起こすように設定します。

タイマー 0 はオーバーフローで割り込みを発生するので、その割り込みハンドラを用意します。割り込みハンドラ内では、カウントアップを行い、割り込み要因フラグを 0 にリセットして、割り込み処理を終了します。

メインルーチンでは、カウンタの値が偶数か奇数かを判定し、高い音と低い音を交互に出力します。61 行目では、CPU を HALT モードにし、割り込みが発生するまで動作を停止します。タイマー 0 割り込みやポート 3 割り込みが発生すると、次の行から動作を開始します。

GHEAD.ASM

```

001 chip      LC868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *                      05/20-'98          *
006 ; *-----*
007
008 public  fm_wrt_ex_exit, fm_vrf_ex_exit
009 public  fm_prd_ex_exit, timer_ex_exit, _game_start, _game_end
010 other_side_symbol  fm_wrt_in, fm_vrf_in
011 other_side_symbol  fm_prd_in, timer_in, game_end
012
013 extern  main                ; ユーザープログラム側にあるシンボル
014 extern  int_T0H             ; ユーザープログラム側にあるシンボル
015
016 ; *-----*
017 ; * Vector table(?)          *
018 ; *-----*
019 cseg
020 org 0000h
021 _game_start:
022 ;reset:
023         jmpf      main                ; main program jump
024 org 0003h
025 ;int_03:
026         jmp int_03
027 org 000bh
028 ;int_0b:
029         jmp int_0b
030 org 0013h
031 ;int_13:
032         jmp int_13
033 org 001bh
034 ;int_1b:
035         jmp int_1b
036 org 0023h
037 ;int_23:
038         jmp int_23
039 org 002bh
040 ;int_2b:
041         jmp int_2b
042 org 0033h
043 ;int_33:
044         jmp int_33
045 org 003bh
046 ;int_3b:

```

```

047         jmp int_3b
048 org 0043h
049 ;int_43:
050         jmp int_43
051 org 004bh
052 ;int_4b:
053         jmp int_4b
054 ; *-----*
055 ; * interrupt programs                *
056 ; *-----*
057 int_03:
058         reti
059 int_0b:
060         reti
061 int_13:
062         reti
063 int_23:
064         jmp     int_T0H ;(ユーザ側割り込み処理へ)
065 int_2b:
066         reti
067 int_33:
068         reti
069 int_3b:
070         reti
071 ; *-----*
072 int_43:
073         reti
074 int_4b:
075         clr1    p3int,1    ;interrupt flag clear
076         reti
077
078 org 0100h
079 ; *-----*
080 ; * flash memory write external program *
081 ; *-----*
082 fm_wrt_ex:
083         change  fm_wrt_in
084         fm_wrt_ex_exit:
085         ret
086 org 0110h
087 ; *-----*
088 ; * flash memory verify external program *
089 ; *-----*
090 fm_vrf_ex:
091         change  fm_vrf_in
092         fm_vrf_ex_exit:
093         ret
094
095 org 0120h
096 ; *-----*
097 ; * flash memory page read external program *
098 ; *-----*
099 fm_prd_ex:
100         change  fm_prd_in
101         fm_prd_ex_exit:
102         ret
103
104 org 0130h
105 ; *-----*
106 ; * flash memory => timer call external program *
107 ; *-----*
108 int_1b:
109 timer_ex:
110         push    ie
111         clr1    ie,7          ;interrupt prohibition
112         change  timer_in
113         timer_ex_exit:
114         pop ie
115         reti
116
117 org 01f0h
118 _game_end:

```

```

119         change game_end
120     end

```

TIMER1.ASM

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** タイマ/カウンタT0割り込み使用サンプル1 **
005 ;
006 ; ・ブザー音を断続的(2秒周期)に出力する
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; チップの種類をアセンブラに指定
012 world     external         ; 外部メモリ用プログラム
013
014 public    main              ; ghead.asm から参照されるシンボル
015 public    int_T0H           ; ghead.asm から参照されるシンボル
016
017 extern    _game_end         ; アプリケーション終了
018
019
020 ; **** システム定数の定義 ****
021
022                                ; OCR(発振制御レジスタ)設定値
023 osc_rc     equ 081h          ; システムクロックに内蔵RC発振を指定
024 osc_xt     equ 082h          ; システムクロックに水晶発振を指定
025
026
027 ; *** データセグメント ****
028
029         dseg                  ; データセグメント開始
030
031 r0:       ds      1          ; 間接アドレッシングレジスタ r0
032 r1:       ds      1          ; 間接アドレッシングレジスタ r1
033 r2:       ds      1          ; 間接アドレッシングレジスタ r2
034 r3:       ds      1          ; 間接アドレッシングレジスタ r3
035          ds      12          ; その他のシステム予約レジスタ
036
037 counter:  ds      1          ; タイマ割り込みカウンタ
038
039
040 ; *** コードセグメント ****
041
042         cseg                  ; コードセグメント開始
043
044 ; *-----*
045 ; * ユーザープログラム *
046 ; *-----*
047 main:
048         call    SndInit       ; 音声出力初期化
049         call    T0Mode2Init   ; タイマ(T0)の初期化
050         mov     #0,counter    ; カウンタのクリア
051
052 loop0:
053         ld      counter       ; カウンタの値を読み込む
054         bp      acc,1,next1    ; カウンタのbit0が1ならnext1へ
055
056         call    Snd2Start     ; 発音開始
057         br      next2
058 next1:
059         call    SndStop       ; 音停止
060 next2:
061         set1    PCON,0        ; 次の割り込みまでHALT
062
063
064                                ; ** [M](モード)ボタンチェック **
065         ld      P3

```

```

066      bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
067
068      br      loop0      ; 繰り返し
069
070 finish:      ; ** アプリケーション終了処理 **
071      jmp     _game_end  ; アプリケーション終了
072
073
074 ; -----*
075 ; * タイマ/カウンタT0 の初期化 *
076 ; *   モード2(16bitカウンタ)で、約1秒毎に割り込みをかける *
077 ; -----*
078 T0Mode2Init:
079      mov     #255,TOPRR ; プリスケアラ値セット
080                      ;   8bitプリスケアラなので
081                      ;   周期 = (256-255)*0.000183 = 0.000183 (sec)
082      mov     #high(65536-5464),TOHR ; プリセット値(H)セット
083      mov     #low(65536-5464),TOLR ; プリセット値(L)セット
084                      ;   プリスケアラとセットで
085                      ;   0.000183 * 5464 = 0.999912 ( 1sec)
086                      ;   1秒毎にオーバーフローが起こる
087      mov     #0ffh,TOL ; 最初すぐにオーバーフローするようにする
088      mov     #0ffh,TOH ;
089      mov     #0e4h,TOCNT ; モード2 (16bitカウンタ)
090                      ; TOHオーバーフローで割り込みを発生させる
091                      ; TO動作開始
092
093      ret      ; T0Mode2Init終わり
094
095
096 TOHStop:      ; *** TOHタイマ停止 ***
097
098      clr1    TOCNT,7 ; TOHカウント動作停止
099      ret
100
101
102 ; -----*
103 ; * タイマTOH割り込みハンドラ *
104 ; -----*
105 int_TOH:      ; *** TOH割り込みハンドラ ***
106      inc     counter
107
108      clr1    TOCNT,3 ; タイマTOH割り込み要因クリア
109      reti
110
111
112 ; -----*
113 ; * 音声出力関係 *
114 ; -----*
115 SndInit:      ; *** 音声出力ハードの初期化 ***
116      clr1    P1,7 ; 音声出力ポートセット
117
118      ret
119
120 Snd1Start:      ; *** 約342Hz の発音開始 ***
121      mov     #0f0h,T1LR ; 周期 = 100h-0f0h = 16
122      mov     #0f8h,T1LC ; Lレベル幅 = 100h-0f8h = 8
123      mov     #0D4h,T1CNT ; 音声出力開始
124
125      ret
126
127 Snd2Start:      ; *** 約781Hz の発音開始 ***
128      mov     #0f9h,T1LR ; 周期 = 100h-0f9h = 7
129      mov     #0fch,T1LC ; Lレベル幅 = 100h-0fch = 4
130      mov     #0D4h,T1CNT ; 音声出力開始
131
132      ret
133
134 SndStop:      ; *** 発音停止 ***
135      mov     #0,T1CNT ; 音声出力停止
136
137      ret

```

C.7 シリアル通信 (送信側)

シリアルインターフェイスを使って、0～99 のデータを送信します。

注意

シリアル通信は、水晶発振で動作させてください。

受信は、次に掲載されている「シリアル通信 (受信側)」で行ってください。

53 行目ではローバッテリー自動検出を禁止しています。実体は 158 行目以降の BattChkOn と BattChkOff です。

56 行目では、シリアルインターフェイスの初期化を行っています。初期化の実体は 95 行目以降のルーチンです。非常に手間がかかるので、このロジックをそのままコピーして利用することをお勧めします。

初期化を行った後は、0.5 秒ごとに発生するベースタイマー割り込みでカウンタをインクリメントします。このカウンタの数値をシリアルインターフェイスで送信します。

MODE ボタンによりプログラムが中断されると、シリアルインターフェイスを規定の値に書き戻し (126 行目からの SioEnd ルーチン)、ローバッテリー自動検出を有効にしてプログラムを終了します。

GHEAD.ASM

```
001 chip      LC868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00      *
005 ; *                                05/20-'98  *
006 ; *-----*
007
008 public  fm_wrt_ex_exit,fm_vrf_ex_exit
009 public  fm_prd_ex_exit,timer_ex_exit,_game_start,_game_end
010 other_side_symbol  fm_wrt_in,fm_vrf_in
011 other_side_symbol  fm_prd_in,timer_in,game_end
012
013 extern  main                ;ユーザープログラム側にあるシンボル
014 extern  int_BaseTimer       ;ユーザープログラム側にあるシンボル
015
016 ; *-----*
017 ; * Vector table(?)          *
018 ; *-----*
019 cseg
020 org 0000h
021 _game_start:
022 ;reset:
023         jmpf      main                ; main program jump
024 org 0003h
025 ;int_03:
026         jmp int_03
027 org 000bh
028 ;int_0b:
029         jmp int_0b
030 org 0013h
031 ;int_13:
032         jmp int_13
033 org 001bh
034 ;int_1b:
035         jmp int_1b
```

```

036 org 0023h
037 ;int_23:
038     jmp int_23
039 org 002bh
040 ;int_2b:
041     jmp int_2b
042 org 0033h
043 ;int_33:
044     jmp int_33
045 org 003bh
046 ;int_3b:
047     jmp int_3b
048 org 0043h
049 ;int_43:
050     jmp int_43
051 org 004bh
052 ;int_4b:
053     jmp int_4b
054 ; *-----*
055 ; * interrupt programs          *
056 ; *-----*
057 int_03:
058     reti
059 int_0b:
060     reti
061 int_13:
062     reti
063 int_23:
064     reti
065 int_2b:
066     reti
067 int_33:
068     reti
069 int_3b:
070     reti
071 ; *-----*
072 int_43:
073     reti
074 int_4b:
075     clr1    p3int,1    ;interrupt flag clear
076     reti
077
078 org 0100h
079 ; *-----*
080 ; * flash memory write external program      *
081 ; *-----*
082 fm_wrt_ex:
083     change  fm_wrt_in
084     fm_wrt_ex_exit:
085     ret
086 org 0110h
087 ; *-----*
088 ; * flash memory verify external program      *
089 ; *-----*
090 fm_vrf_ex:
091     change  fm_vrf_in
092     fm_vrf_ex_exit:
093     ret
094
095 org 0120h
096 ; *-----*
097 ; * flash memory page read external program      *
098 ; *-----*
099 fm_prd_ex:
100     change  fm_prd_in
101     fm_prd_ex_exit:
102     ret
103
104 org 0130h
105 ; *-----*
106 ; * flash memory => timer call external program *
107 ; *-----*

```

```

108 int_1b:
109 timer_ex:
110     push    ie
111     clr1    ie,7           ;interrupt prohibition
112     change  timer_in
113     timer_ex_exit:
114     call    int_BaseTimer ; ユーザー側割り込み処理
115     pop ie
116     reti
117
118 org 01f0h
119 _game_end:
120     change  game_end
121 end

```

TIMER1.ASM

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** シリアル通信サンプル1 (データ送信) **
005 ;
006 ; ・ 一定期間ごとにシリアル通信ポートから単純なデータを送信する
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip    LC868700           ; チップの種類をアセンブラに指定
012 world   external          ; 外部メモリ用プログラム
013
014 public  main               ; ghead.asm から参照されるシンボル
015 public  int_BaseTimer      ; ghead.asm から参照されるシンボル
016
017 extern  _game_end          ; アプリケーション終了
018
019
020 ; **** システム定数の定義 ****
021
022                                     ; OCR(発振制御レジスタ)設定値
023 osc_rc   equ 081h           ; システムクロックに内蔵RC発振を指定
024 osc_xt   equ 082h           ; システムクロックに水晶発振を指定
025
026 LowBattChk equ 06eh         ; ローバッテリー検出フラグ(RAMバンク0)
027
028
029 ; *** データセグメント ****
030
031         dseg                ; データセグメント開始
032
033 r0:     ds      1           ; 間接アドレッシングレジスタ r0
034 r1:     ds      1           ; 間接アドレッシングレジスタ r1
035 r2:     ds      1           ; 間接アドレッシングレジスタ r2
036 r3:     ds      1           ; 間接アドレッシングレジスタ r3
037         ds      12          ; その他のレジスタ
038
039 counter: ds      1           ; カウンタ
040
041
042 ; *** コードセグメント ****
043
044         cseg                ; コードセグメント開始
045
046 ; *-----*
047 ; * ユーザープログラム *
048 ; *-----*
049 main:
050     push    PSW             ; PSW値を退避
051     set1    PSW,1           ; データRAMバンク1を選択
052
053     call    BattChkOff      ; ローバッテリー自動検出機能 OFF

```

```

054
055 cwait:
056     call    SioInit      ; シリアル通信の初期化
057     bz      start       ; VM が接続されていればスタート
058
059     ld      P3           ; [M] ボタンチェック
060     bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
061
062     jmp     cwait        ; VM が接続されるまで待機
063 start:
064
065     set1    pcon,0       ; 次の割り込みまで(0.5秒)HALTで待つ
066
067     mov     #0,counter   ; カウンタの値を0にリセット
068 loop0:
069     ld      counter      ; カウンタの値を読み込む
070
071     call    SioSend1     ; 1byte送信する
072
073     set1    pcon,0       ; 次の割り込みまで(0.5秒)HALTで待つ
074
075                     ; ** [M](モード)ボタンチェック **
076     ld      P3           ;
077     bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
078
079     jmp     loop0        ; 繰り返し
080
081 finish:
082                     ; ** アプリケーション終了処理 **
083     call    SioEnd       ; シリアル通信終了処理
084     pop     PSW          ; PSW値を復帰
085     jmp     _game_end    ; アプリケーション終了
086
087 ; *-----*
088 ; * シリアル通信の初期化 *
089 ; * 出力: acc = 0 : 正常終了 *
090 ; *      acc = Offh: VM が接続されていない *
091 ; *-----*
092 ; シリアル通信の初期化
093 ; システムクロックが水晶モードであることを前提としています。
094
095 SioInit:
096                     ; **** VM同士の接続を確認 ****
097     ld      P7           ; 接続状態の確認
098     and     #00001101    ; P70, P72, P73 をチェック
099     sub     #00001000    ; P70=0, P72=0, P73=1 か判定
100     bz      next2        ; 接続されているときnext2へ
101
102     mov     #0ffh,acc     ; 接続されていないとき acc=0ffh として異常終了
103     ret
104 next2:
105
106                     ; **** シリアル通信の初期化 ****
107     mov     #0,SCON0      ; LSBを先頭に出力するよう指定
108     mov     #0,SCON1      ; LSBを先頭に入力するよう指定
109     mov     #0ddh,SBR     ; 転送レートを設定する
110     clr1    P1,0          ; P10ラッチをクリア(P10/S00)
111     clr1    P1,2          ; P12ラッチをクリア(P12/SCK0)
112     clr1    P1,3          ; P13ラッチをクリア(P13/S01)
113
114     mov     #00000101,P1FCR ; 端子機能設定
115     mov     #00000101,P1DDR ; 端子機能設定
116
117     mov     #0,SBUF0      ; 転送バッファクリア
118     mov     #0,SBUF1      ; 転送バッファクリア
119
120     ret
121
122
123 ; *-----*
124 ; * シリアル通信の終了 *
125 ; *-----*

```

```

126 SioEnd:                ; **** シリアル通信終了処理 ****
127
128     mov     #0,SCON0    ; SCON0 = 0
129     mov     #0, SCON1  ; SCON1 = 0
130     mov     #0bfh,P1FCR ; P1FCR = 0bfh
131     mov     #0a4h,P1DDR ; P1DDR = 0a4h
132
133     ret                ; SioEnd終わり
134
135
136 ; -----*
137 ; * シリアルポートから1byte送信する *
138 ; *   入力 acc: 送信データ *
139 ; -----*
140 SioSend1:                ; **** 1byte送信する ****
141
142     push    acc         ; 送信データの退避
143
144 sslp1: ld     SCON0     ; 前のデータの送信中なら待つ
145     bp      acc,3,sslp1 ;
146
147     pop     acc         ; 送信データの復帰
148
149     st      SBUF0       ; 転送するデータをセットする
150     set1    SCON0,3     ; 送信開始
151
152     ret                ; SioSend1終わり
153
154
155 ; -----*
156 ; * ローバッテリー自動検出機能 ON *
157 ; -----*
158 BattChkOn:
159     push    PSW         ; PSW値を退避
160
161     clr1    PSW,1       ; データRAMバンク0を選択
162     mov     #0,acc      ; ローバッテリー検出をする(0)
163     st      LowBattChk  ; ローバッテリー自動検出フラグ(RAMバンク0)
164
165     pop     PSW         ; PSW値を復帰
166     ret                ; BattChkOn終わり
167
168
169 ; -----*
170 ; * ローバッテリー自動検出機能 OFF *
171 ; -----*
172 BattChkOff:
173     push    PSW         ; PSW値を退避
174
175     clr1    PSW,1       ; データRAMバンク0を選択
176     mov     #Offh,acc   ; ローバッテリー検出をしない(Offh)
177     st      LowBattChk  ; ローバッテリー自動検出フラグ(RAMバンク0)
178
179     pop     PSW         ; PSW値を復帰
180     ret                ; BattChkOff終わり
181
182
183 ; -----*
184 ; * ベースタイマ割り込みハンドラ *
185 ; -----*
186 int_BaseTimer:
187     push    PSW         ; PSW値を退避
188     push    acc
189
190     set1    PSW,1       ; データRAMバンク1を選択
191
192     inc     counter     ; カウンタを進める
193
194     ld      counter     ; カウンタの値が..
195     bne     #100,next1  ; 100でなければnext1
196     mov     #0,counter  ; 100ならば0にリセット
197 next1:

```

```
198      pop      acc
199      pop      PSW          ; PSW値復帰
200
201      clr1     BTDR,1        ; ベースタイマ割り込み要因クリア
202      ret                                ; ユーザー側割り込み処理終了
```

C.8 シリアル通信 (受信側)

シリアルインターフェイスを使って、データを受信します。

このプログラムは、シリアル通信の本質を説明するため、SIO 割り込みを利用していません。実用的なシリアル通信は、次の「汎用シリアルドライバ」を参照してください。

ローバッテリー自動検出を停止し、シリアルインターフェイスを初期化します。

64 行目では、シリアルインターフェイスにデータが(1 バイト)溜っているかをチェックしています。もしデータがあれば、受信したデータを put2digit ルーチンで 10 進数に変換し LCD に表示します。

MODE ボタンによりプログラムが中断されると、シリアルインターフェイスを規定の値に書き戻し(121 行目からの SioEnd ルーチン)、ローバッテリー自動検出を有効にしてプログラムを終了します。

注意

「シリアル通信(送信側)」では、0.5 秒ごとにデータを送信しているので問題にはなりませんが、受信処理が完了していない時点でデータを送信すると、データを取りこぼします。連続したデータを受信する場合は、SIO 割り込みを利用してください。

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** シリアル通信サンプル2 (データ受信) **
005 ;
006 ; ・シリアル通信ポートから受信した数値をLCDに表示する
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; チップの種類をアセンブラに指定
012 world     external         ; 外部メモリ用プログラム
013
014 public    main              ; ghead.asm から参照されるシンボル
015
016 extern    _game_end         ; アプリケーション終了
017
018
019 ; **** システム定数の定義 ****
020
021                                ; OCR(発振制御レジスタ)設定値
022 osc_rc     equ 081h          ; システムクロックに内蔵RC発振を指定
023 osc_xt     equ 082h          ; システムクロックに水晶発振を指定
024
025 LowBattChk equ 06eh          ; ローバッテリー検出フラグ(RAMバンク0)
026
027
028 ; *** データセグメント ****
029
030          dseg                ; データセグメント開始
031
032 r0:      ds      1           ; 間接アドレッシングレジスタ r0
033 r1:      ds      1           ; 間接アドレッシングレジスタ r1
034 r2:      ds      1           ; 間接アドレッシングレジスタ r2
035 r3:      ds      1           ; 間接アドレッシングレジスタ r3
036          ds      12          ; その他のレジスタ
037
038 counter: ds      1           ; カウンタ
039 work1:   ds      1           ; 作業用(put2digitで使用)
040
041
042 ; *** コードセグメント ****

```

```

043
044         cseg                ; コードセグメント開始
045
046 ; *-----*
047 ; * ユーザープログラム *
048 ; *-----*
049 main:
050         call    cls          ; LCD表示の消去
051         call    BattChkOff   ; ローバッテリー自動検出機能 OFF
052
053 cwait:
054         call    SioInit      ; シリアル通信の初期化
055         bz      start        ; VM が接続されていればスタート
056
057         ld      P3           ; [M] ボタンチェック
058         bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
059
060         jmp     cwait        ; VM が接続されるまで待機
061 start:
062
063 loop0:
064         call    SioRecv1     ; 1byte受信する
065         bnz     next4        ; 受信データがない場合はnext4へ
066
067         ld      b            ; 受信したデータをaccへ
068         mov     #2,c          ; 表示座標(水平)
069         mov     #1,b          ; 表示座標(垂直)
070         call    put2digit     ; LCDに2桁数値表示
071
072 next4:                                ; ** [M](モード)ボタンチェック **
073         ld      P3
074         bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
075
076         jmp     loop0        ; 繰り返し
077
078 finish:                                ; ** アプリケーション終了処理 **
079         call    SioEnd        ; シリアル通信終了処理
080         call    BattChkOn     ; ローバッテリー自動検出機能 ON
081         jmp     _game_end     ; アプリケーション終了
082
083 ; *-----*
084 ; * シリアル通信の初期化 *
085 ; * 出力: acc = 0 : 正常終了 *
086 ; *      acc = 0ffh: VM が接続されていない *
087 ; *-----*
088 ; シリアル通信の初期化
089 ; システムクロックが水晶モードであることを前提としています。
090 SioInit:
091                                ; **** VM同士の接続を確認 ****
092         ld      P7            ; 接続状態の確認
093         and     #%00001101    ; P70, P72, P73 をチェック
094         sub     #%00001000    ; P70=0, P72=0, P73=1 か判定
095         bz      next3        ; 接続されているときnext3へ
096
097         mov     #0ffh,acc     ; 接続されていないとき acc=0ffh として異常終了
098         ret                    ; SioInit終わり
099 next3:
100
101                                ; **** シリアル通信の初期化 ****
102         mov     #0,SCON0      ; LSBを先頭に出力するよう指定
103         mov     #0,SCON1      ; LSBを先頭に入力するよう指定
104         mov     #0ddh,SBR     ; 転送レートを設定する
105         clr1    P1,0          ; P10ラッチをクリア(P10/S00)
106         clr1    P1,2          ; P12ラッチをクリア(P12/SCK0)
107         clr1    P1,3          ; P13ラッチをクリア(P13/S01)
108
109         mov     #%00000101,P1FCR ; 端子機能設定
110         mov     #%00000101,P1DDR ; 端子機能設定
111
112         mov     #0,SBUF0      ; 転送バッファクリア
113         mov     #0,SBUF1      ; 転送バッファクリア
114

```

```

115         ret                ; SioInit終わり
116
117
118 ; *-----*
119 ; * シリアル通信の終了 *
120 ; *-----*
121 SioEnd:                ; **** シリアル通信終了処理 ****
122
123         mov     #0,SCON0    ; SCON0 = 0
124         mov     #0,SCON1    ; SCON1 = 0
125         mov     #0bfh,P1FCR ; P1FCR = 0bfh
126         mov     #0a4h,P1DDR ; P1DDR = 0a4h
127
128         ret                ; SioEnd終わり
129
130
131 ; *-----*
132 ; * シリアルポートから1byte受信する *
133 ; *   出力 b: 受信データ *
134 ; *   acc=0      : 受信データあり *
135 ; *   acc=0ffh   : 受信データなし *
136 ; *-----*
137 SioRecv1:                ; **** 1byte受信する ****
138         ld      SCON1
139         bp      acc,1,next5 ; 受信データがあるときnext5へ
140         bp      acc,3,next6 ; 現在転送中ならnext6へ
141
142         set1    SCON1,3     ; 転送開始
143 next6:
144         mov     #0ffh,acc   ; acc=0ffh(受信データなし)として帰る
145         ret      ; SioRecv1終わり
146 next5:
147
148         ld      SBUF1       ; 受信データを読み込む
149         st      b           ; データをbへコピー
150
151         clr1    SCON1,1     ; 転送終了フラグをリセット
152
153         mov     #0,acc      ; acc=0(受信データあり)として帰る
154         ret      ; SioRecv1終わり
155
156
157 ; *-----*
158 ; * 2桁の数値を表示する *
159 ; *   入力 acc : 数値 *
160 ; *   c       : 文字水平位置 *
161 ; *   b       : 文字垂直位置 *
162 ; *-----*
163 put2digit:
164         push    b           ; 座標データを退避
165         push    c           ;
166         st      c           ; 10の位と1の位の値をそれぞれ計算
167         xor     a           ; ( acc = acc/10, work1 = acc mod 10 )
168         mov     #10,b       ;
169         div     ;
170         ld      b           ;
171         st      work1        ; 1の位の計算結果をwork1に保存
172         ld      c           ;
173         pop     c           ; 座標値を(c,b)に復帰
174         pop     b           ;
175         push    b           ; また退避
176         push    c           ;
177         call    putch       ; 10の位を表示
178         ld      work1        ; 1の位の値を読み込み
179         pop     c           ; 座標値を(c,b)に復帰
180         pop     b           ;
181         inc     c           ; 表示座標を右へ
182         call    putch       ; 1の位を表示
183
184         ret                ; put2digit終わり
185
186

```

```

187 ; *-----*
188 ; * LCD表示イメージを消去する *
189 ; *-----*
190 cls:
191     push    OCR        ; OCR値を退避
192     mov     #osc_rc,OCR ; システムクロックを指定
193
194     mov     #0,XBNK     ; 表示用RAMのバンクアドレスを指定(BANK0)
195     call    cls_s       ; そのバンク内のデータをクリア
196
197     mov     #1,XBNK     ; 表示用RAMのバンクアドレスを指定(BANK1)
198     call    cls_s       ; そのバンク内のデータをクリア
199     pop     OCR         ; OCR値を復帰
200
201     ret                     ; cls終わり
202
203 cls_s:
204     mov     #80h,r2      ; *** 表示用RAM 1BANK分の消去 ***
205     mov     #80h,b       ; 間接アドレッシングレジスタを表示用RAMの先頭に
206 loop3:
207     mov     #0,@r2       ; ループカウンタbにループ数をセット
208     inc     r2           ; アドレスをインクリメントしながら0を書き込む
209     dbnz    b,loop3      ; r2
210                             ; bが0になるまで繰り返す
211     ret                     ; cls_s終わり
212
213
214 ; *-----*
215 ; * 指定位置に1キャラクタ表示する *
216 ; * 入力 acc : キャラクタコード *
217 ; *      c : 文字水平位置 *
218 ; *      b : 文字垂直位置 *
219 ; *-----*
220 putch:
221     push    XBNK
222     push    acc
223     call    locate       ; 座標から表示RAMのアドレスを計算
224     pop     acc
225     call    put_chara    ; 1キャラクタ表示する
226     pop     XBNK
227
228     ret                     ; putch終わり
229
230
231 locate: ; **** 表示位置指定から表示用RAMのアドレスを計算 ****
232         ; ** 入力 c: 水平位置(0~5) b: 垂直位置(0~3)
233         ; ** 出力 r2: RAMアドレス XBNK: 表示用RAMバンク
234
235         ; *** 表示用RAMバンクアドレスの判断 ***
236         ld     b         ; b>=2 のとき next1 へ
237         sub     #2
238         bn     PSW,7,next1 ;
239
240         mov     #00h,XBNK ; 表示用RAMのバンクアドレスを指定(BANK0)
241         br     next2
242 next1:
243         st     b
244         mov     #01h,XBNK ; 表示用RAMのバンクアドレスを指定(BANK1)
245 next2:
246
247         ; *** 表示指定位置のRAMアドレス計算 ***
248         ld     b         ; b * 40h + c + 80h
249         rol
250         rol
251         rol
252         rol
253         rol
254         rol
255         add     c
256         add     #80h
257         st     r2        ; RAMアドレスをr2に格納
258

```

```

259         ret                ; locate終わり
260
261
262 put_chara:
263     push    PSW             ; PSW値を退避
264     set1    PSW,1          ; データRAMバンク1を選択
265
266                                     ; *** キャラクタデータアドレスの計算 ***
267     rol                    ; (TRH,TRL) = acc*8 + fontdata
268     rol                    ;
269     rol                    ;
270     add     #low(fontdata) ;
271     st      TRL             ;
272     mov     #0,acc          ;
273     addc    #high(fontdata);
274     st      TRH             ;
275
276     push    OCR             ; OCR値を退避
277     mov     #osc_rc,OCR     ; システムクロックを指定
278
279     mov     #0,b            ; キャラクタデータ読み出し用オフセット値
280     mov     #4,c            ; ループカウンタ
281 loop1:
282     ld      b               ; 1ライン目の表示データを読み出す
283     ldc                    ;
284     inc     b               ; 読み出しデータのオフセットを +1
285     st      @r2             ; 表示データを表示用RAMに転送
286     ld      r2              ; 表示用RAMアドレス +6
287     add     #6              ;
288     st      r2              ;
289
290     ld      b               ; 2ライン目の表示データを読み出す
291     ldc                    ;
292     inc     b               ; 読み出しデータのオフセットを +1
293     st      @r2             ; 表示データを表示用RAMに転送
294     ld      r2              ; 表示用RAMアドレス +10
295     add     #10             ;
296     st      r2              ;
297
298     dec     c               ; ループカウンタのデクリメント
299     ld      c               ;
300     bnz     loop1          ; 8ライン分(4回)繰り返し
301
302     pop     OCR             ; OCR値を復帰
303     pop     PSW             ; PSW値を復帰
304
305     ret                    ; put_chara終わり
306
307
308 ; -----*
309 ; * キャラクタのビットイメージデータ *
310 ; -----*
311 fontdata:
312     db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ; '0' 00
313     db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ; '1' 01
314     db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ; '2' 02
315     db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ; '3' 03
316     db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ; '4' 04
317     db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ; '5' 05
318     db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ; '6' 06
319     db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ; '7' 07
320     db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ; '8' 08
321     db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ; '9' 09
322
323
324 ; -----*
325 ; * ローバッテリー自動検出機能 ON *
326 ; -----*
327 BattChkOn:
328     push    PSW             ; PSW値を退避
329
330     clr1    PSW,1          ; データRAMバンク0を選択

```

```

331      mov    #0,acc      ; ローバッテリー検出をする(0)
332      st     LowBattChk  ; ローバッテリー自動検出フラグ(RAMバンク0)
333
334      pop     PSW         ; PSW値を復帰
335      ret                      ; BattChkOn終わり
336
337
338 ; *-----*
339 ; *   ローバッテリー自動検出機能 OFF   *
340 ; *-----*
341 BattChkOff:
342      push    PSW         ; PSW値を退避
343
344      clr1    PSW,1       ; データRAMバンク0を選択
345      mov     #0ffh,acc   ; ローバッテリー検出をしない(0ffh)
346      st     LowBattChk  ; ローバッテリー自動検出フラグ(RAMバンク0)
347
348      pop     PSW         ; PSW値を復帰
349      ret                      ; BattChkOff終わり

```

C.9 汎用シリアルドライバ

ポート 3 割り込みを利用したバッファ付き汎用シリアルドライバを使った、シリアル送受信プログラムです。

このプログラムを 2 つのビジュアルメモリで実行させると、お互いにデータを送信 / 受信し、その内容を LCD に表示します。

メインルーチンでは、受信バッファをチェックしデータがある限り最優先で、受信データを LCD に表示します。バッファが空になると、送信するデータ (0~99) を出力します。送信するデータは、ベースタイマー割り込みにより 0.5 秒に 1 回カウントアップされます。

128~161 行目の SioInit ルーチンは、ビジュアルメモリの接続確認、インターフェイスの初期化に加え、バッファ (RAM) を初期化し、SIO 割り込みを有効にしています。203~245 行目の SioGet1 ルーチンは、受信バッファに溜っているデータを 1 バイト取り出します。

SIO 割り込みを受けて動作する SIO 受信ハンドラは、278~317 行目の int_SioRx ルーチンです。受信したデータをバッファに溜めていきます。

注意

このサンプル同士で通信を行う場合は、データの取りこぼしはありませんが、連続したデータ転送を行う場合は、送信後一定時間のウェイトを置くなどしてください。

先のサンプル「シリアル通信 (受信側)」にデータを送信すると、データの取りこぼしが発生し、うまく通信できません。

GHEAD.ASM

```

001 chip      LC868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *                               05/20-'98    *
006 ; *-----*
007
008 public  fm_wrt_ex_exit, fm_vrf_ex_exit
009 public  fm_prd_ex_exit, timer_ex_exit, _game_start, _game_end
010 other_side_symbol  fm_wrt_in, fm_vrf_in
011 other_side_symbol  fm_prd_in, timer_in, game_end
012
013 extern  main                ; ユーザープログラム側にあるシンボル
014 extern  int_BaseTimer      ; ユーザープログラム側にあるシンボル
015 extern  int_SioRx          ; ユーザープログラム側にあるシンボル
016
017 ; *-----*
018 ; * Vector table(?)          *
019 ; *-----*
020 cseg
021 org 0000h
022 _game_start:
023 ;reset:
024         jmpf    main                ; main program jump
025 org 0003h
026 ;int_03:
027         jmp int_03
028 org 000bh
029 ;int_0b:
030         jmp int_0b
031 org 0013h
032 ;int_13:
033         jmp int_13
034 org 001bh
035 ;int_1b:
036         jmp int_1b

```

```

037 org 0023h
038 ;int_23:
039         jmp int_23
040 org 002bh
041 ;int_2b:
042         jmp int_2b
043 org 0033h
044 ;int_33:
045         jmp int_33
046 org 003bh
047 ;int_3b:
048         jmp int_3b
049 org 0043h
050 ;int_43:
051         jmp int_43
052 org 004bh
053 ;int_4b:
054         jmp int_4b
055 ; *-----*
056 ; * interrupt programs          *
057 ; *-----*
058 int_03:
059         reti
060 int_0b:
061         reti
062 int_13:
063         reti
064 int_23:
065         reti
066 int_2b:
067         reti
068 int_33:
069         reti
070 int_3b:
071         jmp     int_SioRx    ; SIO受信割り込みハンドラ
072
073 ; *-----*
074 int_43:
075         reti
076 int_4b:
077         clr1    p3int,1      ;interrupt flag clear
078         reti
079
080 org 0100h
081 ; *-----*
082 ; * flash memory write external program      *
083 ; *-----*
084 fm_wrt_ex:
085         change  fm_wrt_in
086         fm_wrt_ex_exit:
087         ret
088 org 0110h
089 ; *-----*
090 ; * flash memory verify external program      *
091 ; *-----*
092 fm_vrf_ex:
093         change  fm_vrf_in
094         fm_vrf_ex_exit:
095         ret
096
097 org 0120h
098 ; *-----*
099 ; * flash memory page read external program    *
100 ; *-----*
101 fm_prd_ex:
102         change  fm_prd_in
103         fm_prd_ex_exit:
104         ret
105
106 org 0130h
107 ; *-----*
108 ; * flash memory => timer call external program *

```

```

109 ; *-----*
110 int_1b:
111 timer_ex:
112     push    ie
113     clr1    ie,7           ;interrupt prohibition
114     change  timer_in
115     timer_ex_exit:
116     call    int_BaseTimer ;(ユーザー側割り込み処理)
117     pop ie
118     reti
119
120 org 01f0h
121 _game_end:
122     change  game_end
123 end

```

TIMER1.ASM

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** シリアル通信サンプル3 (割り込み駆動受信バッファ付きシリアルドライバ) **
005 ;
006 ; ・16byte受信バッファ付きシリアル送受信通信ドライバと利用サンプル
007 ; ・受信データを数値表示する
008 ; ・一定期間ごとに単純なデータを送信する
009 ;-----
010 ; 1.00 981208 SEGA Enterprises,LTD.
011 ;-----
012
013 chip    LC868700           ; チップの種類をアセンブラに指定
014 world   external         ; 外部メモリ用プログラム
015
016 public  main              ; ghead.asm から参照されるシンボル
017 public  int_BaseTimer     ; ghead.asm から参照されるシンボル
018 public  int_SioRx         ; ghead.asm から参照されるシンボル
019
020 extern  _game_end         ; アプリケーション終了
021
022
023 ; **** システム定数の定義 ****
024
025                                     ; OCR(発振制御レジスタ)設定値
026 osc_rc      equ 081h       ; システムクロックに内蔵RC発振を指定
027 osc_xt      equ 082h       ; システムクロックに水晶発振を指定
028
029 LowBattChk  equ 06eh       ; ローバッテリー検出フラグ(RAMバンク0)
030
031 SioRxCueSize equ 16        ; シリアル受信バッファサイズ
032
033
034 ; *** データセグメント ****
035
036     dseg                      ; データセグメント開始
037
038 r0:    ds    1              ; 間接アドレッシングレジスタ r0
039 r1:    ds    1              ; 間接アドレッシングレジスタ r1
040 r2:    ds    1              ; 間接アドレッシングレジスタ r2
041 r3:    ds    1              ; 間接アドレッシングレジスタ r3
042     ds    12              ; その他のレジスタ
043
044                                     ; ** シリアルドライバ用 **
045 SioRxCueBehind: ds 1        ; 受信データ溜り量
046 SioRxCueRPnt:  ds 1        ; 受信バッファ読み出しポイント
047 SioRxCueWPnt:  ds 1        ; 受信バッファ書き込みポイント
048 SioRxCue:      ds SioRxCueSize ; 受信バッファ
049 SioOverRun:    ds 1        ; 受信オーバーランフラグ
050
051                                     ; ** 利用サンプル用ワーク **
052 bcount: ds    1            ; ベースクロックカウンタ

```

```

053 work1: ds      1          ; ワーク1
054 work2: ds      1          ; ワーク2
055
056 work0: ds      1          ; ワーク(put2digit)
057
058
059 ; *** コードセグメント *****
060
061          cseg              ; コードセグメント開始
062
063 ; *-----*
064 ; * シリアル通信ドライバ使用サンプル *
065 ; * 一定期間後とに単純なデータを送信する。 *
066 ; * 受信したデータは順次LCDに数値表示する。 *
067 ; *-----*
068 main:
069          mov     #0,bcount
070          mov     #0,work1      ; 送信データの初期値
071
072          call    cls           ; LCDを消去する
073          call    BattChkOff    ; ローバッテリー自動検出機能 OFF
074          call    SioInit       ; シリアル通信の初期化
075          bnz     finish        ; VM が接続されていなければ終了
076
077 stlp1:
078          ; *** 受信データがあれば表示 ***
079          call    SioGet1       ; 1byte受信
080          be      #0ffh,stnx1   ; 受信データがなければスキップ
081          bz      stnx3         ; 正常受信データありのときstnx3へ
082 error:   br      finish        ; 各種エラーを検出したとき強制終了
083 stnx3:
084          ld      b             ; acc <- b 受信データ
085          mov     #0,c          ; 表示座標(水平)
086          mov     #0,b          ; 表示座標(垂直)
087          call    put2digit      ; LCDに数値表示
088          br      stlp1         ; 受信データがある限り優先して繰り返す
089 stnx1:
090          ;
091          set1     pcon,0        ; 次の割り込みまで待つ
092
093          ; *** 一定期間毎に単純なデータ送信する ***
094          ld      bcount        ; ベースタイマのカウント値
095          be      work2,stnx4   ; 前回と同じならば送信しない
096          st      work2         ; work2を更新
097
098          ld      work1         ; 送信用データを取り出す
099          call    SioPut1       ; 送信する
100
101          inc     work1         ; 送信用データの更新
102          ld      work1         ; (0~99 の値を次々に送信する)
103          bne     #100,stnx2    ;
104          mov     #0,work1      ;
105 stnx2:
106 stnx4:
107
108          ; ** [M](モード)ボタンチェック **
109          ld      P3
110          bn      acc,6,finish   ; [M]ボタンが押されていたらアプリ終了
111
112          jmp     stlp1         ; 繰り返す
113
114 finish:
115          ; ** アプリケーション終了処理 **
116          call    SioEnd        ; シリアル通信終了処理
117          call    BattChkOn     ; ローバッテリー自動検出機能 ON
118          jmp     _game_end     ; アプリケーション終了
119 ; *****
120 ; ***** 簡易シリアル通信ドライバ *****
121 ; *****
122
123 ; *-----*
124 ; * シリアル通信の初期化 *

```

```

125 ; *
126 ; * システムクロックが水晶モードであることを前提としている。 *
127 ; *-----*
128 SioInit:
129 ; **** VM同士の接続を確認 ****
130     ld     P7                ; 接続状態の確認
131     and     #00001101        ; P70, P72, P73 をチェック
132     be      #00001000,next3   ; P70=0, P72=0, P73=1 か判定し
133 ; 接続されているときnext3へ
134     mov     #0ffh,acc        ; 接続されていないとき acc=0ffh で異常終了
135     ret
136 next3:
137
138 ; **** シリアル通信の初期化 ****
139     mov     #0,SCON0        ; LSBを先頭に出力するよう指定
140     mov     #0,SCON1        ; LSBを先頭に入力するよう指定
141     mov     #0ddh,SBR        ; 転送レートを設定する
142     clr1    P1,0            ; P10ラッチをクリア(P10/S00)
143     clr1    P1,2            ; P12ラッチをクリア(P12/SCK0)
144     clr1    P1,3            ; P13ラッチをクリア(P13/S01)
145
146     mov     #00000101,P1FCR ; 端子機能設定
147     mov     #00000101,P1DDR ; 端子機能設定
148
149     mov     #0,SBUF0        ; 転送バッファクリア
150     mov     #0,SBUF1        ; 転送バッファクリア
151
152     mov     #0,acc
153     st      SioRxCueBehind   ; 受信データ溜り量リセット
154     st      SioRxCueRPnt     ; 受信バッファ読み出しポイント
155     st      SioRxCueWPnt     ; 受信バッファ書き込みポイント
156     st      SioOverRun       ; 受信オーバーランフラグのリセット
157
158     set1    SCON1,0          ; 受信側転送終了割り込み有効
159     set1    SCON1,3          ; 受信待機
160
161     ret                ; SioInit終わり
162
163
164 ; *-----*
165 ; * シリアル通信の終了 *
166 ; *-----*
167 SioEnd:
168 ; **** シリアル通信終了処理 ****
169     mov     #0,SCON0        ; SCON0 = 0
170     mov     #0,SCON1        ; SCON1 = 0
171     mov     #0bfh,P1FCR     ; P1FCR = 0bfh
172     mov     #0a4h,P1DDR     ; P1DDR = 0a4h
173
174     ret                ; SioEnd終わり
175
176
177 ; *-----*
178 ; * 1byte送信する *
179 ; *
180 ; * 入力 acc: 送信データ *
181 ; *-----*
182 SioPut1:
183     push    acc                ; 送信データを退避
184     splp1:  ld     SCON0        ; 転送中なら転送終了まで待つ
185     bp      acc,3,splp1        ;
186     pop     acc                ; 送信データ復帰
187
188     st      SBUF0              ; 転送するデータをセットする
189     set1    SCON0,3            ; 転送開始
190
191     ret                ; SioPut1終わり
192
193
194 ; *-----*
195 ; * 受信バッファから1byte読み込む(非同期受信) *
196 ; *

```

```

197 ; * 出力 acc: 0=正常終了
198 ; *      0ffh=受信データなし
199 ; *      0feh=バッファ・オーバーフロー
200 ; *      0fdh=オーバーラン・エラー
201 ; *      b: 受信したデータ(正常終了時のみ有効)
202 ; *-----*
203 SioGet1:
204 ; ** 溜まっているデータ量をチェック **
205     ld      SioRxCueBehind    ; 溜まり量
206     bnz     sgmx1             ; 溜まり量 != 0 のとき
207     mov     #0ffh,acc         ; 溜まり量 == 0 のとき..
208     ret                                ; acc = 0ffh で帰る (受信データなし)
209 sgmx1:
210 ; ** バッファオーバーフローの検出 **
211 ; SioRxCueBehind - SioRxCueSize
212     be      #SioRxCueSize,sgmx3 ; SioRxCueBehind == SioRxCueSize
213     bp      PSW,7,sgmx3        ; SioRxCueBehind < SioRxCueSize
214 ; SioRxCueBehind > SioRxCueSize
215     mov     #0feh,acc         ; バッファ量を越えてしまったとき
216     ret                                ; acc = 0feh で帰る (Buffer overflow)
217 sgmx3:
218 ; ** オーバーランエラーの検出 **
219     ld      SioOverRun        ; オーバーランフラグ
220     bz      sgmx4             ; 検出されていないとき
221     mov     #0fdh,acc         ; 検出されているとき
222     ret                                ; acc = 0fdh で帰る (Overrun error)
223 sgmx4:
224
225     dec     SioRxCueBehind    ; dec 溜まり量
226
227 ; ** 受信データ取り出しポイントの計算
228     ld      SioRxCueRPnt      ; r0 = SioRxCue + SioRxCueRPnt
229     add     #SioRxCue        ;
230     st      r0                ;
231
232     inc     SioRxCueRPnt      ; inc データ取り出しポイント
233
234 ; ** 取り出しポイント==バッファサイズ なら
235 ; ** 取り出しポイント=0にリセットする
236     ld      SioRxCueRPnt
237     bne     #SioRxCueSize,sgmx2 ; SioRxCueRPnt != SioRxCueSize のとき
238     mov     #0,SioRxCueRPnt    ; SioRxCueRPnt == SioRxCueSize のとき
239 sgmx2:
240
241     ld      @r0                ; 入力データをaccに取り込み..
242     st      b                  ; bへ
243     mov     #0,acc             ; acc = 0 (正常終了 - データあり)
244
245     ret                                ; SioGet1終わり
246
247
248 ; *-----*
249 ; * 受信バッファから1byte読み込む
250 ; * (受信データが無い場合はデータがくるまで待つ)
251 ; *
252 ; * 出力 acc: 0=正常終了
253 ; *      0feh=バッファ・オーバーフロー
254 ; *      0fdh=オーバーラン・エラー
255 ; *      b: 受信したデータ(正常終了時のみ有効)
256 ; *-----*
257 SioGet1W:
258     call    SioGet1           ; 非同期受信
259     be      #0ffh,SioGet1W    ; データがくるまで待つ
260
261     ret                                ; SioGet1W終わり
262
263
264 ; *-----*
265 ; * 受信バッファに溜まっているデータ量を取得する
266 ; *
267 ; * 出力 acc: データ量(bytes)
268 ; *-----*

```

```

269 SioGetRxLen:
270     ld      SioRxCueBehind    ; 溜まり量
271
272     ret                                ; SioGetRxLen終わり
273
274
275 ; *-----*
276 ; * SIO受信割り込みハンドラ *
277 ; *-----*
278 int_SioRx:
279     push    acc                ; 使用レジスタ退避
280     push    PSW               ;
281     set1    PSW,1             ; データRAMバンク1を選択
282     push    r0                ; レジスタ退避
283
284                                ; ** 書き込みポイントの計算 **
285     ld      SioRxCueWPnt      ; r0 = SioRxCue + SioRxCueWPnt
286     add     #SioRxCue        ;
287     st      r0                ;
288
289     ld      SBUF1             ; 受信データを読み込む
290     st      @r0               ; バッファに書き込む
291
292     inc     SioRxCueWPnt      ; 書き込みポイント++
293
294                                ; ** 書き込みポイントがバッファサイズに
295                                ; ** 達したらポイントをリセット
296     ld      SioRxCueWPnt      ;
297     bne     #SioRxCueSize,isnx1 ;
298     mov     #0,SioRxCueWPnt   ;
299 isnx1:
300
301     inc     SioRxCueBehind    ; データ溜まり量++
302
303     clr1    SCON1,1           ; 転送終了フラグをリセット
304
305                                ; ** オーバーランエラーのチェック **
306     bn      SCON1,6,isnx2      ; オーバーランしていないならisnx2
307     mov     #1,SioOverRun      ; オーバーランしている フラグセット
308     clr1    SCON1,6           ; オーバーランフラグをリセット
309 isnx2:
310
311     set1    SCON1,3           ; 次の転送を開始
312
313     pop     r0                ; 使用レジスタ復帰
314     pop     PSW               ;
315     pop     acc               ;
316
317     reti                                ; int_SioRx終わり
318
319
320 ; *-----*
321 ; * 2桁の数値を表示する *
322 ; * 入力 acc : 数値 *
323 ; *      c : 文字水平位置 *
324 ; *      b : 文字垂直位置 *
325 ; *-----*
326 put2digit:
327     push    b                ; 座標データを退避
328     push    c                ;
329     st      c                ; 10の位と1の位の値をそれぞれ計算
330     xor     a                ; ( acc = acc/10, work0 = acc mod 10 )
331     mov     #10,b            ;
332     div     ;
333     ld      b                ;
334     st      work0            ; 1の位の計算結果をwork0に保存
335     ld      c                ;
336     pop     c                ; 座標値を(c.b)に復帰
337     pop     b                ;
338     push    b                ; また退避
339     push    c                ;
340     call    putch            ; 10の位を表示

```

```

341      ld      work0      ; 1の位の値を読み込み
342      pop     c          ; 座標値を(c,b)に復帰
343      pop     b          ;
344      inc     c          ; 表示座標を右へ
345      call    putch      ; 1の位を表示
346
347      ret              ; put2digit終わり
348
349
350 ; *-----*
351 ; * LCD表示イメージを消去する *
352 ; *-----*
353 cls:
354      push    OCR        ; OCR値を退避
355      mov     #osc_rc,OCR ; システムクロックを指定
356
357      mov     #0,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK0)
358      call    cls_s      ; そのバンク内のデータをクリア
359
360      mov     #1,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK1)
361      call    cls_s      ; そのバンク内のデータをクリア
362      pop     OCR        ; OCR値を復帰
363
364      ret              ; cls終わり
365
366 cls_s:
367      mov     #80h,r2     ; *** 表示用RAM 1BANK分の消去 ***
368      mov     #80h,b      ; 間接アドレッシングレジスタを表示用RAMの先頭に
369 loop3:
370      mov     #0,@r2      ; アドレスをインクリメントしながら0を書き込む
371      inc     r2          ;
372      dbnz    b,loop3     ; bが0になるまで繰り返す
373
374      ret              ; cls_s終わり
375
376
377 ; *-----*
378 ; * 指定位置に1キャラクタ表示する *
379 ; * 入力 acc : キャラクタコード *
380 ; *      c : 文字水平位置 *
381 ; *      b : 文字垂直位置 *
382 ; *-----*
383 putch:
384      push    XBNK
385      push    acc
386      call    locate      ; 座標から表示RAMのアドレスを計算
387      pop     acc
388      call    put_chara   ; 1キャラクタ表示する
389      pop     XBNK
390
391      ret              ; putch終わり
392
393
394 locate: ; **** 表示位置指定から表示用RAMのアドレスを計算 ****
395      ; ** 入力 c: 水平位置(0~5) b: 垂直位置(0~3)
396      ; ** 出力 r2: RAMアドレス XBNK: 表示用RAMバンク
397
398      ; *** 表示用RAMバンクアドレスの判断 ***
399      ld      b          ; b>=2 のとき next1 へ
400      sub     #2          ;
401      bn      PSW,7,next1 ;
402
403      mov     #00h,XBNK   ; 表示用RAMのバンクアドレスを指定(BANK0)
404      br      next2
405 next1:
406      st      b
407      mov     #01h,XBNK   ; 表示用RAMのバンクアドレスを指定(BANK1)
408 next2:
409
410      ; *** 表示指定位置のRAMアドレス計算 ***
411      ld      b          ; b * 40h + c + 80h
412      rol

```

```

413         rol                ;
414         rol                ;
415         rol                ;
416         rol                ;
417         rol                ;
418         add     c           ;
419         add     #80h        ;
420         st      r2          ; RAMアドレスをr2に格納
421
422         ret                ; locate終わり
423
424
425 put_chara:
426         push    PSW         ; PSW値を退避
427         set1    PSW,1       ; データRAMバンク1を選択
428
429         ; *** キャラクタデータアドレスの計算 ***
430         ; (TRH,TRL) = acc*8 + fontdata
431         rol                ;
432         rol                ;
433         add     #low(fontdata) ;
434         st      TRL         ;
435         mov     #0,acc       ;
436         addc    #high(fontdata) ;
437         st      TRH         ;
438
439         push    OCR          ; OCR値を退避
440         mov     #osc_rc,OCR  ; システムクロックを指定
441
442         mov     #0,b         ; キャラクタデータ読み出し用オフセット値
443         mov     #4,c         ; ループカウンタ
444 loop1:
445         ld      b            ; 1ライン目の表示データを読み出す
446         ldc                ;
447         inc     b            ; 読み出しデータのオフセットを +1
448         st      @r2          ; 表示データを表示用RAMに転送
449         ld      r2           ; 表示用RAMアドレス +6
450         add     #6           ;
451         st      r2           ;
452
453         ld      b            ; 2ライン目の表示データを読み出す
454         ldc                ;
455         inc     b            ; 読み出しデータのオフセットを +1
456         st      @r2          ; 表示データを表示用RAMに転送
457         ld      r2           ; 表示用RAMアドレス +10
458         add     #10          ;
459         st      r2           ;
460
461         dec     c            ; ループカウンタのデクリメント
462         ld      c            ;
463         bnz     loop1        ; 8ライン分(4回)繰り返し
464
465         pop     OCR          ; OCR値を復帰
466         pop     PSW         ; PSW値を復帰
467
468         ret                ; put_chara終わり
469
470
471 ; -----*
472 ; * キャラクタのビットイメージデータ *
473 ; -----*
474 fontdata:
475         db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ;0
476         db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ;1
477         db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ;2
478         db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ;3
479         db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ;4
480         db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ;5
481         db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ;6
482         db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ;7
483         db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ;8
484         db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ;9

```

```

485
486
487 ; *-----*
488 ; * ローバッテリー自動検出機能 ON *
489 ; *-----*
490 BattChkOn:
491     push    PSW                ; PSW値を退避
492     clr1    PSW,1              ; データRAMバンク0を選択
493
494     mov     #0,LowBattChk      ; ローバッテリー検出をする(0)
495
496     pop     PSW                ; PSW値を復帰
497     ret                     ; BattChkOn終わり
498
499
500 ; *-----*
501 ; * ローバッテリー自動検出機能 OFF *
502 ; *-----*
503 BattChkOff:
504     push    PSW                ; PSW値を退避
505     clr1    PSW,1              ; データRAMバンク0を選択
506
507     mov     #0ffh,LowBattChk   ; ローバッテリー検出をしない(0ffh)
508
509     pop     PSW                ; PSW値を復帰
510     ret                     ; BattChkOff終わり
511
512
513 ; *-----*
514 ; * ベースタイマ割り込みハンドラ *
515 ; *-----*
516 int_BaseTimer:
517     clr1    bcr,1              ; ベースタイマ割り込み要因クリア
518     inc     bcount             ; カウンタ++
519     ret                     ; ユーザー側割り込み処理終了

```

C.10 フラッシュメモリの読み書き

フラッシュメモリへの書き込み、読み込み、ベリファイを行い、各フェーズが終了するごとに“SEGA”の文字を順次表示します。

61～79行目は、フラッシュメモリに書き込むデータをRAMに準備します。データは0～128の値で、間接アドレスレジスタを用いてRAMバンク1のアドレス10H～8FHにセットします。データの準備フェーズを終えると、LCDに“S”の文字を表示します。

92～104行目では、システムBIOSを呼び出すためにそのパラメータをセットし、ローバタリ自動検出を禁止します。

106～109行目は、システムBIOSを呼び出す前にシステムクロックをRC発振に切り換えています。システムBIOS呼び出し後は、元のクロック（水晶発振）に切り換えています。

クロック切り換え後は、ローバタリ自動検出を有効にして、LCDに“E”を表示します。

128～151行目は、フラッシュメモリに書き込んだデータと、RAMのデータをシステムBIOSのベリファイ機能を使って比較します。完全に同一であれば、LCDに“G”を表示します。データが一致せずシステムBIOSがエラーを返した場合は“G”の文字を表示せず次のフェーズを実行します。

157～169行目では、フラッシュメモリに書き込んだデータをRAMに読み出しています。読み出したデータは、172行目以降にある自前の比較ルーチンでデータを検証しています。データが完全に一致した場合は“ A ”の文字を表示しプログラムを終了します。もし、一致しなかった場合は“ A ”の文字を表示せずに終了します。

“G”や“ A ”の文字が表示されなかった場合は、それ以前に行ったフラッシュメモリへのアクセスでデータが化けていることになります。

357行目以降のデータは、データの書き込み先（フラッシュメモリ）です。

注意

フラッシュメモリに書き込みを行う場合は、必ずアプリケーション自身内に書き込む領域を用意します。アプリケーション外のフラッシュメモリへの書き込みは禁止されています。

フラッシュメモリのアクセスは、常に128バイト単位で行われますので、360行目のORGで128バイトのアライメントを行っています。

注意

フラッシュメモリの領域確保には、アセンブラ疑似命令のDSコマンドは利用できません。DSコマンドは、RAM領域に対してのみ有効です。

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** フラッシュメモリ使用サンプル1 **
005 ;
006 ; フラッシュメモリにデータを書き込み、ベリファイし、読み出し、検証する。
007 ; 全てが良好に動作するとLCD上に「SEGA」の文字が揃って表示される。
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip    LC868700                ; チップの種類をアセンブラに指定
013 world   external              ; 外部メモリ用プログラム
```

```

014
015 public main ; ghead.asm から参照されるシンボル
016
017 extern _game_end ; ghead.asm への参照シンボル
018 extern fm_wrt_ex, fm_vrf_ex, fm_prd_ex ; ghead.asm への参照シンボル
019
020
021 ; **** システム定数の定義 ****
022
023 ; OCR(発振制御レジスタ)設定値
024 osc_rc equ 081h ; システムクロックに内蔵RC発振を指定
025 osc_xt equ 082h ; システムクロックに水晶発振を指定
026
027 LowBattChk equ 06eh ; ローバッテリー検出フラグ(RAMバンク0)
028
029 fmflag equ 07ch ; フラッシュメモリ書き込み終了検出方式
030 fmbank equ 07dh ; フラッシュメモリバンク切替え
031 fmadd_h equ 07eh ; フラッシュメモリ上位アドレス
032 fmadd_l equ 07fh ; フラッシュメモリ下位アドレス
033
034 fmbuff equ 080h ; フラッシュメモリ読み書き用バッファ先頭
035
036
037 ; *** データセグメント ****
038
039 dseg ; データセグメント開始
040
041 r0: ds 1 ; 間接アドレッシングレジスタ r0
042 r1: ds 1 ; 間接アドレッシングレジスタ r1
043 r2: ds 1 ; 間接アドレッシングレジスタ r2
044 r3: ds 1 ; 間接アドレッシングレジスタ r3
045 ds 12 ; その他のシステム予約レジスタ
046
047 ; *** コードセグメント ****
048
049 cseg ; コードセグメント開始
050
051 ; *-----*
052 ; * ユーザープログラム *
053 ; *-----*
054 main:
055 call cls ; LCD表示イメージを消去する
056
057
058 ; **** テスト書き込み用のデータを準備する ****
059 ; fmbuff に 10h~8fh の 128byte のデータを用意
060
061 push PSW ; PSW値を退避
062 set1 PSW,1 ; データRAMバンク1を選択
063
064 mov #fmbuff,r0 ; 読み書きバッファのアドレスを r0 に
065 mov #128,c ; ループカウンタ(128回)
066 mov #010h,b ; 書き込むデータの初期値
067 loop4:
068 ld b ; データをバッファに置く
069 st @r0 ;
070
071 inc b ; 書き込みテスト用データを変える
072
073 inc r0 ; バッファのアドレスをインクリメント
074
075 dec c ; ループカウンタのデクリメント
076 ld c
077 bnz loop4 ; 128回の繰り返し
078
079 pop PSW ; PSW値を復帰
080
081
082 ; **** 「 S 」を表示 ****
083
084 mov #1,c ; 水平座標
085 mov #1,b ; 垂直座標

```

```

086      mov    #0ah,acc    ; キャラクタコード 'S'
087      call  putch       ; 一文字表示
088
089
090                      ; **** フラッシュメモリへの書き込み ****
091
092      push   PSW         ; PSW値を退避
093      set1   PSW,1       ; データRAMバンク1を選択
094
095      mov    #0,fmbank   ; フラッシュメモリのバンク指定 = 0
096      mov    #high(fmarea),fmadd_h ; 書き込み先アドレス(上位)
097      mov    #low(fmarea),fmadd_l  ; 書き込み先アドレス(下位)
098      mov    #0,fmflag   ; トクルビット方式により終了を検出
099
100      clr1   PSW,1       ; データRAMバンク0を選択
101      mov    #0ffh,acc    ; ローバッテリー検出をしない(0ffh)
102      st     LowBattChk   ; ローバッテリー自動検出フラグ(RAMバンク0)
103
104      pop     PSW        ; PSW値を復帰
105
106      push   OCR         ; OCR値を退避
107      mov    #osc_rc,OCR  ; システムクロックを指定(RC)
108      call   fm_wrt_ex    ; BIOS「フラッシュメモリへの書き込み」
109      pop     OCR        ; OCR値を復帰
110
111      push   PSW         ; PSW値を退避
112      clr1   PSW,1       ; データRAMバンク0を選択
113      mov    #0,acc       ; ローバッテリー検出をする(0)
114      st     LowBattChk   ; ローバッテリー自動検出フラグ(RAMバンク0)
115      pop     PSW        ; PSW値を復帰
116
117
118                      ; **** 「E」を表示 ****
119
120      mov    #2,c         ; 水平座標
121      mov    #1,b         ; 垂直座標
122      mov    #0bh,acc     ; キャラクタコード 'E'
123      call  putch       ; 一文字表示
124
125
126                      ; **** フラッシュメモリとのベリファイ ****
127
128      push   PSW         ; PSW値を退避
129      set1   PSW,1       ; データRAMバンク1を選択
130
131      mov    #0,fmbank   ; フラッシュメモリのバンク指定 = 0
132      mov    #high(fmarea),fmadd_h ; アドレス(上位)
133      mov    #low(fmarea),fmadd_l  ; アドレス(下位)
134
135
136      push   OCR         ; OCR値を退避
137      mov    #osc_rc,OCR  ; システムクロックを指定(RC)
138      call   fm_vrf_ex    ; BIOS「フラッシュメモリとのベリファイ」
139      pop     OCR        ; OCR値を復帰
140
141      pop     PSW        ; PSW値を復帰
142
143      bnz     vrt_bad     ; 書き込み失敗分岐
144                      ; 成功時のみ「G」を表示する
145
146                      ; **** 「G」を表示 ****
147
148      mov    #3,c         ; 水平座標
149      mov    #1,b         ; 垂直座標
150      mov    #0ch,acc     ; キャラクタコード 'G'
151      call  putch       ; 一文字表示
152 vrt_bad:
153
154
155                      ; **** フラッシュメモリのページデータ読み出し ****
156
157      push   PSW         ; PSW値を退避

```

```

158      set1    PSW,1      ; データRAMバンク1を選択
159
160      mov     #0,fmbank  ; フラッシュメモリのバンク指定 = 0
161      mov     #high(fmarea),fmadd_h  ; アドレス(上位)
162      mov     #low(fmarea),fmadd_l   ; アドレス(下位)
163
164      push    OCR        ; OCR値を退避
165      mov     #osc_rc,OCR ; システムクロックを指定(RC)
166      call    fm_prd_ex  ; BIOS「フラッシュメモリのページデータ読み出し」
167      pop     OCR        ; OCR値を復帰
168
169      pop     PSW        ; PSW値を復帰
170
171
172                                ; **** 読み込んだデータを検証する ****
173
174      push    PSW        ; PSW値を退避
175      set1    PSW,1      ; データRAMバンク1を選択
176
177      mov     #fmbuff,r0 ; 読み書きバッファのアドレスを r0 に
178      mov     #128,c     ; ループカウンタ(128回)
179      mov     #010h,b    ; 比較用データの初期値
180 loop5:
181      ld      b          ; データをバッファに置く
182      sub     @r0         ; データを比較
183      bnz     read_bad   ; 比較エラー.「A」を表示せず終了
184
185      inc     b          ; 書き込みテスト用データを変える
186
187      inc     r0         ; バッファのアドレスをインクリメント
188
189      dec     c          ; ループカウンタのデクリメント
190      ld      c
191      bnz     loop5      ; 128回の繰り返し
192
193      pop     PSW        ; PSW値を復帰
194
195
196                                ; **** 「A」を表示 ****
197
198      mov     #4,c       ; 水平座標
199      mov     #1,b       ; 垂直座標
200      mov     #0dh,acc    ; キャラクタコード 'A'
201      call    putch      ; 一文字表示
202
203
204 read_bad:
205 loop6:                                ; ** [M](モード)ボタンチェック **
206      ld      P3
207      bn      acc,6,finish ; [M]ボタンが押されていたらアプリ終了
208
209      br      loop6      ; [M]ボタンが押されるまで待つ
210
211 finish:                                ; ** アプリケーション終了処理 **
212      jmp     _game_end  ; アプリケーション終了
213
214
215 ; -----*
216 ; * LCD表示イメージを消去する *
217 ; -----*
218 cls:
219      push    OCR        ; OCR値を退避
220      mov     #osc_rc,OCR ; システムクロックを指定
221
222      mov     #0,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK0)
223      call    cls_s      ; そのバンク内のデータをクリア
224
225      mov     #1,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK1)
226      call    cls_s      ; そのバンク内のデータをクリア
227      pop     OCR        ; OCR値を復帰
228
229      ret                                ; cls終わり

```

```

230
231 cls_s: ; *** 表示用RAM 1BANK分の消去 ***
232     mov     #80h,r2 ; 間接アドレッシングレジスタを表示用RAMの先頭に
233     mov     #80h,b ; ループカウンタbにループ数をセット
234 loop3:
235     mov     #0,@r2 ; アドレスをインクリメントしながら0を書き込む
236     inc     r2 ;
237     dbnz    b,loop3 ; bが0になるまで繰り返す
238
239     ret     ; cls_s終わり
240
241
242 ; *-----*
243 ; * 指定位置に1キャラクタ表示する *
244 ; * 入力 acc : キャラクタコード *
245 ; *      c : 文字水平位置 *
246 ; *      b : 文字垂直位置 *
247 ; *-----*
248 putch:
249     push    XBNK
250     push    acc
251     call    locate ; 座標から表示RAMのアドレスを計算
252     pop     acc
253     call    put_chara ; 1キャラクタ表示する
254     pop     XBNK
255
256     ret     ; putch終わり
257
258
259 locate: ; **** 表示位置指定から表示用RAMのアドレスを計算 ****
260 ; ** 入力 c: 水平位置(0~5) b: 垂直位置(0~3)
261 ; ** 出力 r2: RAMアドレス XBNK: 表示用RAMバンク
262
263 ; *** 表示用RAMバンクアドレスの判断 ***
264     ld      b ; b>=2 のとき next1 へ
265     sub     #2 ;
266     bn      PSW,7,next1 ;
267
268     mov     #00h,XBNK ; 表示用RAMのバンクアドレスを指定(BANK0)
269     br      next2
270 next1:
271     st      b
272     mov     #01h,XBNK ; 表示用RAMのバンクアドレスを指定(BANK1)
273 next2:
274
275 ; *** 表示指定位置のRAMアドレス計算 ***
276     ld      b ; b * 40h + c + 80h
277     rol     ;
278     rol     ;
279     rol     ;
280     rol     ;
281     rol     ;
282     rol     ;
283     add     c ;
284     add     #80h ;
285     st      r2 ; RAMアドレスをr2に格納
286
287     ret     ; locate終わり
288
289
290 put_chara:
291     push    PSW ; PSW値を退避
292     set1    PSW,1 ; データRAMバンク1を選択
293
294 ; *** キャラクタデータアドレスの計算 ***
295     rol     ; (TRH,TRL) = acc*8 + fontdata
296     rol     ;
297     rol     ;
298     add     #low(fontdata) ;
299     st      TRL ;
300     mov     #0,acc ;
301     addc    #high(fontdata) ;

```

```

302      st      TRH      ;
303
304      push    OCR      ; OCR値を退避
305      mov     #osc_rc,OCR ; システムクロックを指定
306
307      mov     #0,b      ; キャラクタデータ読み出し用オフセット値
308      mov     #4,c      ; ループカウンタ
309 loop1:
310      ld      b          ; 1ライン目の表示データを読み出す
311      ldc          ;
312      inc     b          ; 読み出しデータのオフセットを +1
313      st      @r2       ; 表示データを表示用RAMに転送
314      ld      r2        ; 表示用RAMアドレス +6
315      add     #6        ;
316      st      r2        ;
317
318      ld      b          ; 2ライン目の表示データを読み出す
319      ldc          ;
320      inc     b          ; 読み出しデータのオフセットを +1
321      st      @r2       ; 表示データを表示用RAMに転送
322      ld      r2        ; 表示用RAMアドレス +10
323      add     #10       ;
324      st      r2        ;
325
326      dec     c          ; ループカウンタのデクリメント
327      ld      c          ;
328      bnz     loop1     ; 8ライン分(4回)繰り返し
329
330      pop     OCR        ; OCR値を復帰
331      pop     PSW        ; PSW値を復帰
332
333      ret            ; put_chara終わり
334
335
336 ; -----*
337 ; * キャラクタのビットイメージデータ *
338 ; -----*
339 fontdata:
340      db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ; '0' 00
341      db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ; '1' 01
342      db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ; '2' 02
343      db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ; '3' 03
344      db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ; '4' 04
345      db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ; '5' 05
346      db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ; '6' 06
347      db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ; '7' 07
348      db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ; '8' 08
349      db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ; '9' 09
350
351      db 07ch, 0e6h, 076h, 038h, 0dch, 0ceh, 07ch, 000h ; 'S' 0a
352      db 0feh, 0c0h, 0c0h, 0f8h, 0c0h, 0c0h, 0feh, 000h ; 'E' 0b
353      db 07ch, 0e6h, 0c0h, 0dch, 0c6h, 0e6h, 07ch, 000h ; 'G' 0c
354      db 01eh, 036h, 066h, 0c6h, 0c6h, 0feh, 0c6h, 000h ; 'A' 0d
355
356
357 ; -----*
358 ; * データ保存用フラッシュメモリ領域 *
359 ; -----*
360      org ((*-1) land 0ff80h) + 80h ; 128byte アライン
361 fmarea:
362      ; 128byte フラッシュメモリ領域を確保する
363      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
364      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
365      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
366      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
367      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
368      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
369      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
370      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
371
372 end

```

C.11 ローバッテリー検出とデータ退避

ビジュアルメモリは、ローバッテリーを検出するとその旨のメッセージを表示し、自動的にスリープ状態になる機能を内蔵しています。このサンプルは、アプリケーションが独自にローバッテリーを検知し、RAM のデータをフラッシュメモリに退避するものです。

このプログラムの本質は、115～125 行目のローバッテリー検出ルーチンです。ポート 7 のローバッテリーフラグをチェックします。

ポート 7 割り込みを利用することも可能ですが、割り込み処理ルーチン内でシステム BIOS を呼び出さないように配慮してください。

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** ローバッテリー検出とデータ退避サンプル1 **
005 ;
006 ; ローバッテリー状態を検出して、必要なデータをフラッシュメモリに退避する
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; チップの種類をアセンブラに指定
012 world     external         ; 外部メモリ用プログラム
013
014 public    main              ; ghead.asm から参照されるシンボル
015
016 extern    _game_end         ; ghead.asm への参照シンボル
017 extern    fm_wrt_ex, fm_vrf_ex, fm_prd_ex ; ghead.asm への参照シンボル
018
019
020 ; **** システム定数の定義 ****
021
022                                ; OCR(発振制御レジスタ)設定値
023 osc_rc     equ 081h          ; システムクロックに内蔵RC発振を指定
024 osc_xt     equ 082h          ; システムクロックに水晶発振を指定
025
026 LowBattChk equ 06eh          ; ローバッテリー検出フラグ(RAMバンク0)
027
028 fmflag     equ 07ch          ; フラッシュメモリ書き込み終了検出方式
029 fmbank     equ 07dh          ; フラッシュメモリバンク切替え
030 fmadd_h     equ 07eh          ; フラッシュメモリ上位アドレス
031 fmadd_l     equ 07fh          ; フラッシュメモリ下位アドレス
032
033 fmbuff     equ 080h          ; フラッシュメモリ読み書き用バッファ先頭
034
035
036 ; *** データセグメント ****
037
038          dseg                ; データセグメント開始
039
040 r0:      ds      1           ; 間接アドレッシングレジスタ r0
041 r1:      ds      1           ; 間接アドレッシングレジスタ r1
042 r2:      ds      1           ; 間接アドレッシングレジスタ r2
043 r3:      ds      1           ; 間接アドレッシングレジスタ r3
044          ds      12          ; その他のシステム予約レジスタ
045
046
047 ; *** コードセグメント ****
048
049          cseg                ; コードセグメント開始
050
051 ; *-----*
052 ; * ユーザープログラム *
053 ; *-----*
054 main:

```

```

055      call    cls          ; LCD表示イメージを消去する
056
057 loop0:                                ; テストメインループ先頭
058
059      ; アプリケーションメイン処理
060
061      ; ** [M](モード)ボタンチェック **
062      ld      P3
063      bn      acc,6,finish ; [M] ボタンが押されていたらアプリ終了
064
065      ; ** バッテリ状態チェック **
066      call    ChkBatt       ; バッテリの状態をチェック
067      bz      loop0         ; acc=0 なら バッテリ正常でループ
068
069      ; ** ローバッテリー 処理 **
070      call    prepare       ; テスト回避用のデータを用意する。
071      ; 実際のアプリケーションでは回避すべきデータを
072      ; 収集してフラッシュROM書きこみ用のバッファに配置
073      ; するようにする。
074
075      call    WriteData     ; フラッシュメモリへバッファに用意した(回避すべき)
076      ; データを書きこむ
077
078 finish:                                ; ** アプリケーション終了処理 **
079      jmp     _game_end     ; アプリケーション終了
080
081
082 ; *-----*
083
084 prepare:                                ; **** テスト回避用のデータを準備する ****
085      ; fmbuff に 10h~8fh の 128byte のデータを用意
086
087      push    PSW           ; PSW値を退避
088      set1    PSW,1         ; データRAMバンク1を選択
089
090      mov     #fmbuff,r0    ; 読み書きバッファのアドレスを r0 に
091      mov     #128,c        ; ループカウンタ(128回)
092      mov     #010h,b       ; 書き込むデータの初期値
093 loop4:
094      ld      b             ; データをバッファに置く
095      st      @r0           ;
096
097      inc     b             ; 書き込みテスト用データを変える
098
099      inc     r0            ; バッファのアドレスをインクリメント
100
101      dec     c             ; ループカウンタのデクリメント
102      ld      c
103      bnz     loop4         ; 128回の繰り返し
104
105      pop     PSW           ; PSW値を復帰
106
107      ret                  ; prepare 終わり
108
109
110 ; *-----*
111 ; * ローバッテリー状態を検出する *
112 ; * 出力: acc = 0 : 電池状態正常 *
113 ; * acc = 0ffh : ローバッテリー *
114 ; *-----*
115 ChkBatt:
116      ld      P7           ; P71の状態をチェックする
117      bn      acc,1,next3 ; 電池ないとき分岐
118
119      ; ** 電池ある **
120      mov     #0,acc        ; acc = 0
121      ret      ; ChkBatt終わり。電池ありなら acc=0 として返る
122
123 next3:
124      ; ** 電池なし **
125      mov     #0ffh,acc     ; acc = 0ffh
126      ret      ; ChkBatt終わり。電池なしなら acc=0ffh として返る

```

```

127
128 ; *-----*
129 ; * バッファのデータをフラッシュメモリへ書きこむ *
130 ; *-----*
131 WriteData: ; **** フラッシュメモリへの書き込み ****
132
133     push    PSW        ; PSW値を退避
134     set1    PSW,1      ; データRAMバンク1を選択
135
136     mov     #0,fmbank  ; フラッシュメモリのバンク指定 = 0
137     mov     #high(fmarea),fmadd_h ; 書き込み先アドレス(上位)
138     mov     #low(fmarea),fmadd_l ; 書き込み先アドレス(下位)
139     mov     #0,fmflag  ; トクルビット方式により終了を検出
140
141     clr1    PSW,1      ; データRAMバンク0を選択
142     mov     #0ffh,acc  ; ローバッテリー検出をしない(0ffh)
143     st      LowBattChk ; ローバッテリー自動検出フラグ(RAMバンク0)
144
145     pop     PSW        ; PSW値を復帰
146
147     push    OCR        ; OCR値を退避
148     mov     #osc_rc,OCR ; システムクロックを指定(RC)
149     call    fm_wrt_ex  ; BIOS「フラッシュメモリへの書き込み」
150     pop     OCR        ; OCR値を復帰
151
152     push    PSW        ; PSW値を退避
153     clr1    PSW,1      ; データRAMバンク0を選択
154     mov     #0,acc     ; ローバッテリー検出をする(0)
155     st      LowBattChk ; ローバッテリー自動検出フラグ(RAMバンク0)
156     pop     PSW        ; PSW値を復帰
157
158     ret      ; WriteData終わり
159
160
161 ; *-----*
162 ; * LCD表示イメージを消去する *
163 ; *-----*
164 cls:
165     push    OCR        ; OCR値を退避
166     mov     #osc_rc,OCR ; システムクロックを指定
167
168     mov     #0,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK0)
169     call    cls_s      ; そのバンク内のデータをクリア
170
171     mov     #1,XBNK    ; 表示用RAMのバンクアドレスを指定(BANK1)
172     call    cls_s      ; そのバンク内のデータをクリア
173     pop     OCR        ; OCR値を復帰
174
175     ret      ; cls終わり
176
177 cls_s: ; *** 表示用RAM 1BANK分の消去 ***
178     mov     #80h,r2    ; 間接アドレッシングレジスタを表示用RAMの先頭に
179     mov     #80h,b     ; ループカウンタbにループ数をセット
180 loop3:
181     mov     #0,@r2     ; アドレスをインクリメントしながら0を書き込む
182     inc     r2         ;
183     dbnz    b,loop3    ; bが0になるまで繰り返す
184
185     ret      ; cls_s終わり
186
187
188 ; *-----*
189 ; * データ保存用フラッシュメモリ領域 *
190 ; *-----*
191     org ((*-1) land 0ff80h) + 80h ; 128byte アライン
192 fmarea:
193     ; 128byte フラッシュメモリ領域を確保する
194     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
195     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
196     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
197     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
198     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

```

```
199      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
200      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
201      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
202
203 end
```


index

索引

Symbol

1 ブロック 24

A

ATTRIBUTE 61

B

BACKUP UTILITY 51

BIOS 呼び出し 12

BOOT ROM 19, 20

C

CHECK DISK 57

chip 11

CodeScape 39, 45

COLOR A 65

COMPLETE 57

COPY 59

CRC 32

cseg 11

D

DEFRAG 55

DELETE 60

Dreamcast の起動画面 19

dseg 11

DUMP 63

DUPLICATE 55

E

E2H86K 15

EVA 形式ファイル 14

EXIT 51

external 11

F

FORMAT 56

FREE 52

G

GAME 52

GAME BUFFER 53, 54

GD Workshop 39

GD Workshop の設定 44

GDUMMY.OBJ 14

GHEAD.ASM 12

GUI コメント 23

GUI コメントデータ 30

H

H00 15

H2BIN 16

HEADER OFFSET 61

HEX 15

HEX ファイルに変換 15

I

ICON BUFFER 53, 54

ICON NO. 64

ICONDATA_VMS 21, 25

INFO 53, 62

L

L86K 14

LC868700 11

M

M86K 13

MAKE 16

mem_util.elf 39

Memory Card Utility 39

Memory Card Utility の操作方法 50

N

NORMAL BUFFER 53, 54

O

OPERATION 65

org 12

Q

QUICK 57

R

RC 発振 12

RCV DATA 53

RENAME 60

RS-232C リバースケープル 39

S

SAVE DATA 54

SYSTEM CONFIG 51

U

UNFORMAT 56

UPLOAD 62

USED 52

V

VM コメント 24

VM コメントデータ 25, 29

W

world 11

X

Xmodem 68

ア

アイコン数 31
 アイコンのパターンデータ 33
 アイコンのパレットデータ 32
 アセンブル 13
 アニメーションアイコン 22, 27
 アニメーションスピード 31
 アニメーションパターン 31
 アプリケーション領域の使用状況 52

イ

インフォメーションフォーク 18, 28, 63
 インフォメーションフォークの構造 28

オ

オブジェクトファイル 13
 音声認識装置 20

カ

カラーアイコンデータの開始アドレス 26
 カラーアイコンのパターンデータ 27
 カラーアイコンのパレットデータ 27
 漢字コード 42
 間接アドレスレジスタ 11

キ

起動画面各部の名称 20

ク

クロック 12

ケ

ゲーム名 30, 34, 63
 ゲーム名キャラクタコード表 37

コ

コードセグメント 11
 コピー禁止 / 許可 61
 コマンド選択メニュー 52

シ

使用ブロック数 24
 初期化されていないメモリ 20
 初期化した日時 66

ス

ストップビット長 42

セ

セーブ時刻 24

ソ

ソートキー 30, 34

タ

タイプ A 31, 33
 タイプ B 31, 33
 タイプ C 31, 34
 ダミーの GD-ROM 44

ツ

通信速度 42
 通信ソフトのファイル転送 68
 通信プロトコルの設定 42

テ

データセグメント 11
 データ長 42
 データの種類 24
 データ保存領域の空き領域 52

ト

ドアクローズ 45

ナ

内蔵のアイコン 21

ハ

バイナリファイルに変換 16
 ハイパーターミナル 40
 ハイパーターミナルが見あたらない 40
 パリティチェック 42

ヒ

ビジュアルコメント 24
 ビジュアルコメントデータ 33
 ビジュアルコメントデータの構造 33
 ビジュアルコメントの種類 31
 ビジュアルタイプ 31
 ビジュアルメモリ 20
 ビジュアルメモリシミュレータ 16
 ビジュアルメモリの初期化 56, 64
 ビジュアルメモリへの転送 18
 ビジュアルメモリへのファイル転送 67
 ビッグエンディアン 73

フ

ファイル管理画面 22
 ファイル構造 27
 ファイル操作メニュー 59
 ファイル名 23

ファイル名を変更	60
ファイルを PC に転送	62
ファイルを消去	60
複数のファイルの選択	59
フラグメント	55
フラッシュメモリの読み書き	12
フロー制御	42

ホ

ボディーカラー	22, 65
ボリュームアイコン	21

ミ

未初期化のビジュアルメモリ	56
---------------------	----

メ

メインメニュー	20, 50
メインルーチンへのジャンプ	12

メモリ選択画面	20
メモリ選択メニュー	51

モ

モノクロアイコンデータの開始アドレス	26
モノクロアイコンのパターンデータ	26

ラ

ラベルアイコン	21, 75
---------------	--------

リ

リトルエンディアン	73
リトルエンディアン形式	73
利用済みの空き領域	52
リンク	14

ワ

割り込みベクタの定義	12
------------------	----