Wonder Witch Technical Manual

第一版

発行　　Digitalis

著　A.Watanabe

０．序文

この度は、本書「Wonder Witch Technical Manual」をご購入頂き、誠にありがとうございます。

Wonder Witch パッケージ付属のデベロッパーズマニュアル（以後、「デベロッパーズマニュアル」と略す）は、一般ユーザレベルのＣ言語プログラマーを対象として記載されています。ゆえに、ハードウェアの技術資料はほとんど無いに等しい状態であり、そのためマイコン技術者など、機械語、アセンブラレベルでの開発を行っている方々には解りにくい内容になっております。

そこで本書は、下層レイヤーレベルの設計を行っているプログラマーの方々を対象とし、Wonder Witch の技術資料としてまとめました。本書は、Ｃ言語のみならず、機械語（又はアセンブラ）による開発が行える程度の情報を記載することを目標としています。

昔ながらのマイコン技術者の方々には、他人の作ったプログラムで、尚かつソースリストが公開されていないプログラムは信頼性上、使用を避けたいと思われるでしょうが、Wonder Witch は一般ユーザ向けの「開発キット」ですので、何かと制約は大きいですが BIOS や OS を使用するようお願いします。本書もハードウェアの技術資料は記載しておりません。

本書作製の当初は、リバースエンジニアリングを行って技術情報を調べていましたが､メーカからソフトウェア使用承諾に違反するとの指摘を受けたため、リバースエンジニアリングによって得られた情報は削除しています。しかし、Wonder Witch 付属のマニュアルには、ソフトウェア作成上必要になるであろう情報が記載されていない箇所、又、曖昧な表現が使われており理解に苦しむ箇所が多々見受けられるため、メーカに問い合わせ、又、実験等を行うなどして、出きる限りの情報を調べて本書に記載しました。

本書は、執筆時間及びページ数の都合上、サンプルプログラムの作製、記載をしておりません。技術資料としては手抜きな資料になっておりますが、後々の改善点にしたいと考えております。何卒、ご了承下さい。

付属のフロッピーディスクに、執筆に当たって作製したプログラムを添付しておきます。

本書は、intel8086 系 CPU についてアセンブラによる開発を行える方が読まれることを前提として記述しました。アセンブラによる開発ができない方でも理解できるように努めて作製していますが、アセンブラを勉強し８０８６ＣＰＵの動作を理解することで本書は一層読みやすくなると思います。又、デベロッパーズマニュアルと併用することにより、Wonder Witch の仕様をより深く理解できると思います。Wonder Witch のプログラミング人生において座右の銘としてお使いして頂けたら幸いです。

最後に、本書を出版するにあたって多忙の中、多くの方にお世話になりました。
この場を借りて厚くお礼を申し上げます。
・技術情報を提供して下さったワンダーウィッチサポート係様
・出版の機会を与えて下さった DAHLIA 殿を始めとする同人サークル Digitalis の皆様
・文章のチェック及び動作検証を行っていただいた、先輩の長島さん、後輩の天方君を始めとする東京都立工業高等専門学校エレクトロニクス研究部の皆様
・この書籍をご購入頂いた読者の方々

2001.08.00
A.Watanabe

１．目次

０．序文 …… p． 1

１．目次 …… p． 3

２．システム構成 …… p． 4
２－１．メモリマップについて　Memory　p． 4
２－２．Ｉ／Ｏについて　Input / Output　p． 4
２－３．割り込みについて。　Interrupt　p． 5

３．ハードウェア及びＢＩＯＳ仕様 …… p． 7
３－１．プログラムの終了　Exit　p． 8
３－２．キー　Key　p． 9
３－３．画面　Disp　p. 12
３－４．テキスト　Text　p. 25
３－５．シリアル通信　Comm　p. 32
３－６．サウンド　Sound　p. 37
３－７．カレンダ／タイマ　Timer　p. 42
３－８．システム　System　p. 45
３－９．バンク　Bank　p. 53

４．ＯＳ仕様 …… p. 57
４－１．プロセス　Process　p. 60
４－２．ファイルシステム　Filesys　p. 64
４－３．インダイレクトライブラリ　Ilib　p. 66
４－４．ライブラリ　lib　p. 77

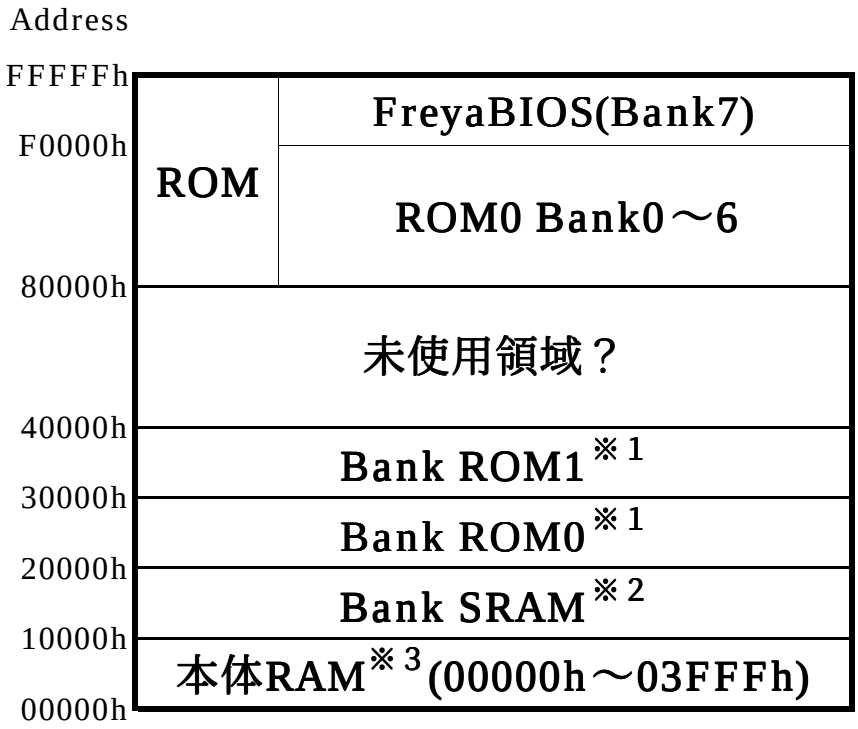
５．添付資料 …… p. 78

６．後書き …… p. 79

７．著者紹介・改訂履歴 …… p. 80

２．システム構成

２－１．メモリについて



※０　メモリーマップについてはメーカ非公開情報であるため、これ以上の詳細は不明である。

※１　フラッシュメモリは、20000h～3FFFFhにマッピングされる。
バンク0～7は、物理アドレス80000h～FFFFFhに存在する。

※２　SRAMは、10000h～1FFFFhにマッピングされる。
バンクは、以下の通り。
Bank 0　FreyaOS Soft File System
Bank 1　FreyaOS UserProsse1
Bank 2　FreyaOS UserProsse0
Bank 3　FreyaOS Work

※３　本体と記述が無いエリアは、カセットの中に配置される。

注意：　上記メモリマップは、ヘッダーファイル及びNEC社V30MZ CPUの仕様を考慮にいれた、「予想図」である。この情報を元にしたプログラムは保証しない。

２－２．Ｉ／Ｏについて

メーカ非公開情報である。ハードウェアの機能はＢＩＯＳにて提供される。
３－１項参照のこと。

２－３．割り込みについて

２－３－１．ハードウェア割り込みについて

・割り込み処理ルーチンの作製にあたって以下の点に留意しなければならない。

① BIOS内のルーチンから、'far call'命令にて呼び出されるため、retf命令にて処理を終了する必要がある。
② メーカはユーザ独自のスタック領域の確保について推奨していない。
③ BIOS内のルーチンで、汎用レジスタ及び'ds','es'レジスタが保存されるので、ユーザが独自に保存する必要はない。
④ EOI(End Of Interrupt)の発行は、BIOS内のルーチンで行うため、ユーザ側で行う必要はない。
⑤ DSレジスタには、ユーザが割り込み登録時に設定したデータセグメントが設定されている。
⑥ ESレジスタは、Ｃ言語にてファーポインタの参照等で使用されるため、値は不定である。
⑦ FreyaBIOSやＣ言語を使用しない場合は、この限りでは無い。
⑧ "/ram0"アクセス中の割り込みはSRAMバンクが切り替わっている。

・ユーザ割り込みの登録は、'SYS_INTERRUPT_SET_HOOK'ファンクションにて行う。詳細は３－８項参照。
・FreyaBIOSver1.0ではバグがあり、最新のFreyaOSにてカバーされている。このため、Ｃ言語から'sys/system.h'で定義されているファンクションを呼びこの割り込みを使用する。
・表２－３－１にハードウェア割り込みに使用される割り込みベクタについて示す。

表２－３－１　ハードウェア割り込み内容

IntVector	Trigger	優先順位	Name	Category
28h	level	最低	SEND READY	シリアル送信データエンプティ
29h	Egg	↓	KEY	キー押下検出
2Ah	Level	↓	CASETTE	ＲＴＣアラーム
2Bh	Level	↓	RECEIVE	シリアル受信データレディー
2Ch	Egg	↓	DISPLINE	描画ライン番号検出
2Dh	Egg	↓	VBLANK_COUNTUP	垂直同期タイマーカウント終了
2Eh	Egg	▼	VBLANK	垂直同期期間開始
2Fh	Egg	最高	HNLANK_COUNTUP	水平同期タイマーカウント終了

"/ram0"アクセス中の割り込みについて

ファイルシステムの"/ram0"にアクセス中はSRAM Bankが切り替わっており、ユーザプロセス用のSRAMが見えない状態になっている。このタイミングで割り込みが発生しユーザプロセス用のSRAMをバンクを切り替えずにアクセスすることで、"/ram0"中のファイルを破壊する恐れがある。

●通常プロセスで割り込みを使う場合の対策方法

1.) "pcb_get_srambank()"関数で実行しているプロセスのSRAMバンク番号を取得する
2.) バンクが違っていたら、そのバンクを保存し、プロセス用のSRAMに切り替える。
3.) 割り込み中でやりたい処理を行い、バンクを元に戻して割り込みを終了する。

●インダイレクト・ライブラリ(IL)で割り込みを使う場合の対策方法

IL中では、"pcb_get_srambank()"関数で正しい値が得られないバグがある。そのため、以下の2点の方法が提案する。

①'ram0'アクセス中は、割り込み中で処理を行わない。

"/ram0"のアクセス時間は非常に微小時間[msec単位]であり、タイマー割り込みで目立つ処理落ちは感じられなかった。次回の割り込み時に処理を遅らせることが可能である場合にこの方法が提案できる。Bank切り替えを行わないので、割り込み中の速度は速い。

②割り込み定義時に、IRAMエリアにプロセス用のSRAM Bankの値を入れておく

割り込み定義時に、BIOSの"SYS_ARROC_IRAM"関数にてIRAM領域を確保し、そこに現在実行しているプロセス用のSRAM Bankの値を入れておく。

割り込み時に、BIOSの"SYS_GET_MY_IRAM"で確保したIRAMのアドレスを取得し、そこに保存されているプロセス用のSRAM Bankの値を取得することで、"pcb_get_srambank()"関数の代わりとする。

２－３－２．FreyaＢＩＯＳファンクションについて

・FreyaBI0Sver1.0では、表２－３－２に示すファンクションコールを用意している。詳細は、３項参照こと。

表２－３－２　FreyaＢＩＯＳファンクション割り込み

IntVector	Name	Category
10h	EXIT	プログラムの終了
11h	KEY	キー
12h	DISP	画面制御
13h	TEXT	テキスト画面
14h	COMM	シリアル通信
15h	SOUND	サウンド
16h	TIMER	タイマ／カレンダ
17h	SYSTEM	システム制御
18h	BANK	バンク制御

２－３－３．FreyaＯＳファンクションについて

・Freya0Sver1.0では、表２－３－３に示すファンクションを用意している。
詳細はメーカ未公開情報のため不明である。

表２－３－３　FreyaＯＳファンクション割り込み

IntVector	Name	Category
30h	PROCESS	詳細不明　－メーカ未公開情報
31h	FILESYS	詳細不明　－メーカ未公開情報
32h	ILIB	詳細不明　－メーカ未公開情報

３．ハードウェア及びＢＩＯＳ仕様

1.) FreyaBIOS 及び、ハードウェアについて

・カートリッジ上のフラッシュメモリ中、書き換えることができない領域(プロテクト領域)に記録されていますが、モニタ機能を用いてOSの入れ替えを行うことで将来の機能拡張や修正版への差し換えが行えます。
・ソフトウェア割り込みを用いてFreyaBIOSの機能を利用します。
・ハードウェアへのアクセスは、ＢＩＯＳ，ＯＳ，ワンダーウィッチ付属のライブラリにて行います。
・FrayaBios ver 1.0は、Wonder Swan Color発売前のバージョンであるため、カラーには対応していません。カラーはＯＳが提供しており、libwwc.lib及びＣ言語のヘッダーファイルを使用することで利用可能です。詳細は、４－４項参照。

2.)CPUについて

・NEC V30MZ CPU
・供給クロック3.072MHz
・１φ／１バスサイクル
・パイプラインによる命令処理の高速化
・CMOSスタティック設計により内部システムクロックの完全停止を実現
・20bit Address Bus / 16bit Data Bus(Input / Output)

3.)メモリについて

・Frash memoly 512KByte(64KByteはBIOSが使用）
・SRAM 256KByte
・本体RAM 16KByte(BIOSが使用。主にスタック等）
・メモリマップについては、メーカ未公開情報である。
・詳細は２－１項を参照のこと。

4.)I/Oについて

・I/Oについては、メーカ未公開情報である。
・ハードウェアの機能は、ＢＩＯＳで提供される。
・詳細は２－２項を参照のこと。

5.)割り込みベクタについて

・以下の割り込みがシステムで使用されている。
　①ハードウェア割り込み（int 28h～2Fh）
　②FreyaBIOSファンクション（int 10h～18h）※1
　③FreyaOSファンクション（int 30h～32h）※2
　※１　詳細は次項以降にて説明する。
　※２　メーカ未公開情報である。
・割り込みベクタマップは、メーカ未公開情報である。
・詳細は２－３項を参照のこと。

３－１．プログラムの終了

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 10h	EXIT
引数	
	無し
返値	
	無し
処理	
	プログラムを終了する。 通常はスタートアップルーチン※1からラベル'main'※2が 呼び出され、サブルーチンmainが終了後、 スタートアップルーチンへと戻る仕様になっている。 このファンクションはスタートアップルーチンで使用されて いるため、ユーザプログラムは'ret'命令でプログラム を終了すれば問題はない。 但し、FreyaOSやスタートアップルーチンを使用しない場合、 このファンクションにてプログラムを終了する必要がある。 スターアップルーチンは、各プログラムファイルの0000hから 存在し、プログラムの実行時は0000hから実行される。 ※１　WonderWitch付属のC言語コンパイラを使用時に リンクされるオブジェクト。詳細は３－３項参照。 ※２　Ｃ言語の場合は、main()関数と置き換えて下さい。

３－２．キー制御

・WonderSwan本体にあるX,Y,A,B,Startボタンの状態を検出できる。※１
・各ボタンは以下のグループに分かれており、ハードウェアレベルでの同時検出は不可能である。

① X1(up) , X2(right) , X3(down) , X4(left)
② Y1(up) , Y2(right) , Y3(down) , Y4(left)
③ A , B , Start

・FreyaBIOSでは、VBLANKタイマー割り込み※２を使用することで、各グループを順次検出し内部ワークエリアに格納している。又、VBLANKタイマー割り込みを使用し前回検出した情報を利用してオールリピート※３がかかるようになっています。
・FreyaBIOSでは、上記の制御機能を提供する。

※１　SOUND、電源(Wonder Swan Color)ボタンの検出はできません。
※２　ＬＣＤ絵画用垂直同期信号を利用したタイマー割り込み
※３　前回、キー入力ファンクションを呼び出したときに押されていたキーと、今回呼び出したときに押されているのキーを比較することで、継続してキーが押されていることを確認する。その時、前回と同じキーが押されている場合、VBLANK割り込みを利用することで指定した遅延時間及び間隔で同じキーを返値に返す。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 11h　AH=00h　KEY_PRESS_CHECK

引数

AH　　00h

返値

AX　　キー入力状態　以下表参照

処理

キーの入力状態を得る。
キーリピート機能は使用せず、
今現在押されているキーが返値として戻る。

表３－２－１　キー入力状態返値

ビット	内容
bit 0	未使用
bit 1	Start
bit 2	a
bit 3	b
bit 4	x1(up)
bit 5	x2(right)
bit 6	x3(down)
bit 7	x4(left)
bit 8	y1(up)
bit 9	y2(right)
bit 10	y3(down)
bit 11	y4(left)
bit 12	未使用
bit 13	未使用
bit 14	未使用
bit 15	未使用

INT 11h　AH=01h　KEY_HIT_CHECK

引数

AH　　01h

返値

AX　　キー入力状態　表３－２－１参照

処理

新たに入力されたキーを返す。

INT 11h　AH=02h　KEY_WAIT

引数

AH　　02h

返値

AX　　キー入力状態　表３－２－１参照

処理

キーが入力されるまで待つ。
入力されたら、AXレジスタにその入力状態が返される。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 11h	AH=03h	KEY_SET_REPEAT
引数		
	AH	03h
	BL	キーリピート間隔(単位 1/75[sec])
	BH	キーリピート遅延(単位 1/75[sec])
返値		
	無し	
処理		
	キーリピートの遅延と間隔を指定する。	

INT 11h	AH=04h	KEY_GET_REPEAT
引数		
	AH	04h
返値		
	AL	キーリピート間隔(単位 1/75[sec])
	AH	キーリピート遅延(単位 1/75[sec])
処理		
	キーリピートの遅延と間隔を取得する。	

INT 11h	AH=05h	KEY_HIT_CHECK_WITH_REPEAT
引数		
	AH	05h
返値		
	AX	キー入力状態　表３－２－１参照
処理		
	キーリピート機能を使用し、キーの入力状態を得る。	

３－３．画面制御

●Wonder Swanの仕様

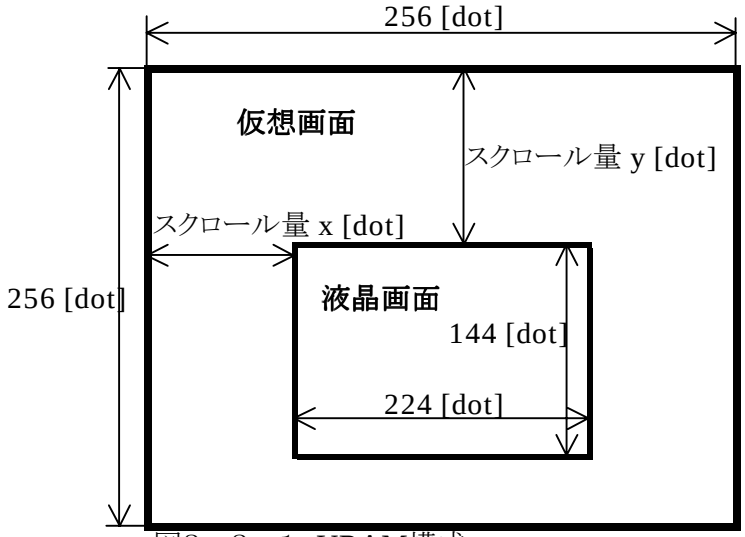


図3－3－1　VRAM構成

①画面仕様
・表示フレーム　　75.47Hz　(12,000[Hz] ÷ 159[dot])
・水平ライン数　　256 [dot]　(ブランキング期間 32[dot時間])
・数垂直ライン数　159 [dot]　(ブランキング期間 15[dot時間])
・LCD階調　　16色中8色 (Wonder Swan Colorは、4096色中241色)
※備考　　各スクリーンの右端、左端は連続されて表示される。

②キャラクター仕様
・サイズ　　8×8 [dot]
・階調　　8色中4色　(Wonder Swan Colorは、4096色中16色)
・定義可能数　　512[chr]　(Wonder Swan Colorは、1024[chr])

③スクリーン仕様
・スクリーン数　　2 [screen]　(各スクリーンは別個にスクロール量の指定が可能。)
・スプライト数　　128 [chr]　(1ライン中は32[chr]まで。)
・ウィンドウ※1　　Screen2 , Sprite共にクリッピングウィンドウ機能を有す。
・優先順位　　Sceenn1 < Sprite※2 < Screen2 < Sprite※2 < BodrColor※3
※1　dot単位で指定が可能。指定されたdotはWindowsの内側として定義される。
※2　各スプライト個別にScreen2に対する優先度を指定することができる。
※3　全てが透明色であった場合、最終的に表示する色。

④パレット
1.) Wonder Swan
・パレットテーブル　モノクロ16階調の中からスクリーンで表示する8色を選ぶ。
・スクリーン　　0～15の計16個のパレット(8色→4色の変換テーブル)を持つ。
・スプライト※1　　0～7 の計 8個のパレット(8色→4色の変換テーブル)を持つ。
2.) Wonder Swan Color
・スクリーン　　0～15の計16個のパレット(4096色→16色の変換テーブル)を持つ。
・スプライト※1　　0～7 の計 8個のパレット(4096色→16色の変換テーブル)を持つ。
※1　スプライトのパレットは、スクリーンのパレット8～15と共通。

表３－３－１　ディスプレイモードについて

ディスプレイモード	リンクするファイル		内容		
	LSI-C86	Turbo-C	Screen1	漢字	ｷｬﾗｸﾀｰ
ASCII1	crt0asc1	c0wwasc1	使用不可	使用不可	384～512
ASCII2	crt0asc2	c0wwasc2	使用可	使用不可	384～512
日本語1	crt0jpn1	c0wwjpn1	使用不可	使用可	0～512
日本語2	crt0jpn2	c0wwjpn2	使用可	使用可	0～512

・表中「キャラクター」は、使用できるキャラクターコード。
・4色ColorModeで使用する場合は、1023番までのキャラクターが使用可能。
・16色ColorModeでは、0～1023番全てキャラクターが使用可能。

表３－３－２　テキスト表示モードとディスプレイモードとの関連

テキスト表示モード	指定可能なディスプレイモード		
	Ank	AnkSjis	Sjis
ASCII1	◎	×	×
ASCII2	◎	×	×
日本語1	○	◎	○
日本語2	○	◎	○

左表の記号について
×　選択不可能
○　選択可能
◎　初期設定

・テキスト表示モードについては、３－４項、表３－４－１参照

・上記表にディスプレイモードに応じたリンクすべきスタートアップルーチンを示し、使用可能なスクリーン数、キャラクター、及び、テキスト表示モードとの関連性を示した。
・ユーザはスクリーンを使用する際、以下に示す処理を行うべきである。
① 本体の判別※1（カラー対応ソフトの場合。）
② カラーモードの設定※1（カラー対応ソフトの場合。）
　初期設定では'グレイスケール'となっている。
③ 'SCREEN_SET_VRAM'ファンクション及び'SPRITE_SET_VRAM'ファンクションを用いて、VRAMの設定を行う。通常はスタートアップルーチンがディスプレイモードに応じた設定をしているため、ユーザが行う必要は無い。※2
④ 'DISPLAY_CONTROL'ファンクションを用いて以下の設定をする。
・各スクリーン及びスプライト機能のイネーブル設定
　スタートアップルーチンでの初期設定は、スクリーン２の表示が有効となっており、ディスプレイモード'ASCII2'及び'日本語2'の場合のみ、スクリーン１についても表示が有効となっている。
・ウィンドウ機能の設定。
　スタートアップルーチンでの初期設定は、全て無効になっている。
・ボーダーカラーの設定
　スタートアップルーチンでの初期設定は、初期設定は、000<b>（カラー時0000<b>）ななっている。
・ユーザは、スプライトを使用する場合、以下の処理を行うべきである。
① 使用するフォント(PCG・RAMCG)の定義。
② 'DISPLAY_CONTROL'ファンクションを用いて、スプライト表示をイネーブルにする。
③ 'SPRITE_SET_RANGE'ファンクションを用いて、使用するスプライト番号を'表示'設定にする。それ以外の番号は'非表示'となる。

※１　FrayaBios ver 1.0は、Wonder Swan Color発売前のバージョンであるため、カラーには対応していない。カラーはＯＳが提供しており、libwwc.lib及びＣ言語のヘッダーファイルを使用することで利用可能です。詳細は、４－４項参照のこと。一部の機能はBIOSで提供される。

※２　メーカ未公開のファンクションである。スタートアップルーチンを使用しない場合は、スタートアップと同じ引数を設定することが好ましい。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=00h　DISPLAY_CONTROL

引数

AH　　00h
AX　　ディスプレイコントロール　以下表参照

返値

無し

処理

ディスプレイコントロールを設定する。

表３－３－３　ディスプレイコントロール

ビット	内容
bit 0	スクリーン１・表示・イネーブル
bit 1	スクリーン２・表示・イネーブル
bit 2	スプライト・表示・イネーブル
bit 3	スプライト・ウィンドウ・イネーブル
bit 4- 5	スクリーン２・ウィンドウ・モード
bit 8-12	ボーダー・カラー（モノクロモード時は、bit 8-11）
bit 13-15	ボーダー・カラー・パレット(カラーモードのみ)

・スクリーン２・ウィンドウ・モードについて
　0x　ディセーブル（無効）
　10　ウィンドウの内側表示
　11　ウィンドウの外側表示
・スクリーン１・表示・イネーブルは、ディスプレイモード'ank1','日本語1'モードのときは、イネーブル('1')にしてはならない。

INT 12h　AH=01h　DISPLAY_STATUS

引数

AH　　01h

返値

AX　　ディスプレイコントロール　表３－３－３参照

処理

ディスプレイコントロールを取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=02h　FONT_SET_MONODATA

引数

AH	02h
BX	設定開始キャラクタコード
CX	設定キャラクター数量
DS:DX	設定キャラクターデータ格納アドレス(単位8byte)

返値

無し

処理

キャラクターのフォントを、'Font set color'ファンクションの設定によって16byteのデータに変換しながらフォントを設定する。
フォントの書式は、各bitが各横方向のdotに対応し、各byteが各縦方向のdotに対応する。この時、'0':背景、'1':前景となる。

16色カラーモード、パックドモード時はこのファンクションを使用してはならない。

INT 12h　AH=03h　FONT_SET_COLORDATA

引数

AH	03h
BX	設定開始キャラクタコード
CX	設定キャラクター数量
DS:DX	設定キャラクターデータ格納アドレス(単位16byte)

返値

無し

処理

キャラクターのフォントを設定する。
フォントの書式は、2byte1組で横方向の8dotを指定し、それが縦方向の8dotで計16byteのデータとなる。bit0,8:1dot目、bit1,9:2dot目、……、bit7,15:8dot目のカラーコードと指定する。

16色カラーモード、パックドモード時はこのファンクションを使用してはならない。

INT 12h　AH=04h　FONT_GET_DATA

引数

AH	04h
BX	取得開始キャラクタコード
CX	取得キャラクター数量
DS:DX	設定キャラクターデータ格納アドレス(単位16byte)

返値

無し

処理

キャラクターのフォントを取得する。

16色カラーモード、パックドモード時はこのファンクションを使用してはならない。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=05h　FONT_SET_COLOR

引数

AH　05h
BX bit0,1　前景のカラーコード
BX bit2,3　背景のカラーコード

返値

無し

処理

'FONT_SET_MONODATA'ファンクションで展開するカラーコードを指定する。

INT 12h　AH=06h　FONT_GET_COLOR

引数

AH　06h

返値

AX bit0,1　前景のカラーコード
AX bit2,3　背景のカラーコード

処理

'FONT_SET_MONODATA'ファンクションで展開するカラーコードを取得する。

INT 12h　AH=07h　SCREEN_SET_CHAR

引数

AH　07h
AL　スクリーン　（0 : Screen1 / 1 : Screen2）
BH　ｙ座標先頭位置
BL　ｘ座標先頭位置
CH　ｙ座標領域のサイズ
CL　ｘ座標領域のサイズ
DS:DX　表示データ格納アドレス

返値

無し

処理

VRAMにDS:DXの示すアドレスのデータを出力する。
データ書式は以下表の通り。

表３－３－４　ＶＲＡＭデータ書式

ビット	内容
bit　0- 8	キャラクターコード bit 0～8
bit　9-12	パレット(0～15)
bit 13	キャラクターコード bit 9 (Color Modeのみ)
bit 14	横方向反転
bit 15	縦方向反転

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=08h　SCREEN_GET_CHAR

引数

AH	08h
AL	スクリーン　（0 : Screen1 / 1 : Screen2）
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CH	ｙ座標領域のサイズ
CL	ｘ座標領域のサイズ
DS:DX	表示データ格納アドレス

返値

AX

処理

VRAMのデータを取得する。
データ書式は表３－３－４の通り。

INT 12h　AH=09h　SCREEN_FILL_CHAR

引数

AH	09h
AL	スクリーン　（0 : Screen1 / 1 : Screen2）
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CH	ｙ座標領域のサイズ
CL	ｘ座標領域のサイズ
DX	表示データ　表３－３－４参照

返値

無し

処理

VRAMにデータを出力する。

INT 12h　AH=0Ah　SCREEN_FILL_ATTR

引数

AH	0Ah
AL	スクリーン　（0 : Screen1 / 1 : Screen2）
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CH	ｙ座標領域のサイズ
CL	ｘ座標領域のサイズ
DX	表示データ　表３－３－４参照
SI	表示データマスクパターン　表３－３－４参照

返値

無し

処理

VRAMにデータを出力する。
出力データは、((「現在の値」 and SI) or DX)の結果が出力される。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=0Bh　SPRITE_SET_RANGE

引数

AH　0Bh
BX　スプライト先頭番号(0～127)
CX　スプライト数量(0～127)

返値

無し

処理

表示するスプライト番号を使用する。
指定していないスプライトは、非表示となる。
BX+CXは128以下にならなければならない。

INT 12h　AH=0Ch　SPRITE_SET_CHAR

引数

AH　0Ch
BX　スプライト番号(0～127)
CX　キャラクター番号、スプライト属性　以下表参照

返値

無し

処理

スプライトのキャラクター及び属性を設定する。

表３－３－５　スプライトキャラクター・属性

ビット	内容
bit　0- 8	キャラクターコード(0～511)
bit　9-11	パレット(8～15)　８を引いた値を出力する。
bit 12	表示のスクリーン２に対する優先度。　1:優先
bit 13	ｳｨﾝﾄﾞｳのｸﾘｯﾋﾟﾝｸﾞﾀｲﾌﾟ。 0:内側 / 1:外側
bit 14	横方向反転
bit 15	縦方向反転

INT 12h　AH=0Dh　SPRITE_GET_CHAR

引数

AH　0Dh
BX　スプライト番号(0～127)

返値

AX　キャラクター番号、スプライト属性　表３－３－５参照

処理

スプライトのキャラクター及び属性を取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=0Eh　SPRITE_SET_LOCATION

引数

AH	0Eh
BX	スプライト番号(0～127)
DH	スプライト表示　ｙ座標
DL	スプライト表示　ｘ座標

返値

無し

処理

スプライトの座標を設定する。

INT 12h　AH=0Fh　SPRITE_GET_LOCATION

引数

AH	0Fh
BX	スプライト番号(0～127)

返値

AH	スプライト表示　ｙ座標
AL	スプライト表示　ｘ座標

処理

スプライトの座標を取得する。

INT 12h　AH=10h　SPRITE_SET_CHAR_LOCATION

引数

AH	10h
BX	スプライト番号(0～127)
CX	キャラクター番号、スプライト属性　表３－３－５参照
DH	スプライト表示　ｙ座標
DL	スプライト表示　ｘ座標

返値

無し

処理

スプライトのキャラクター、属性及び座標を設定する。

INT 12h　AH=11h　SPRITE_GET_CHAR_LOCATION

引数

AH	11h
BX	スプライト番号(0～127)

返値

AX	キャラクター番号、スプライト属性　表３－３－５参照
DH	スプライト表示　ｙ座標
DL	スプライト表示　ｘ座標

処理

スプライトのキャラクター、属性及び座標を取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=12h　SPRITE_SET_DATA

引数

AH	12h
BX	スプライト番号(0～127)
CX	スプライト数量
DS:DX	データ（4byte単位）

返値

無し

処理

スプライトのキャラクター、属性及び座標を一括設定する。
データは、'SPRITE_SET_CHAR_LOCATE'ファンクションの引数、CX,DXの順に格納し、内容は同等のものである。
但し、3Byte目と4Byteは逆転している。(+3:Y座標／+4:X座標)

INT 12h　AH=13h　SCREEN_SET_SCROLL

引数

AH	13h
AL	スクリーン　（0 : Screen1 / 1 : Screen2）
BH	ｙ座標スクロール量(0～255)　1dot単位
BL	ｘ座標スクロール量(0～255)　1dot単位

返値

無し

処理

LCDで表示する左上橋の座標をドット単位で設定する。

INT 12h　AH=14h　SCREEN_GET_SCROLL

引数

AH	14h
AL	スクリーン　（0 : Screen1 / 1 : Screen2）

返値

AH	ｙ座標スクロール量(0～255)　1dot単位
AL	ｘ座標スクロール量(0～255)　1dot単位

処理

LCDで表示する左上橋の座標をドット単位で取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=15h　SCREEN2_SET_WINDOW

引数

AH	15h
BH	ｙ座標先頭位置(0～255) 1dot単位
BL	ｘ座標先頭位置(0～255) 1dot単位
CH	ｙ座標領域のサイズ(0～255) 1dot単位
CL	ｘ座標領域のサイズ(0～255) 1dot単位

返値

無し

処理

クリッピングウィンドウの領域を設定する。

INT 12h　AH=16h　SCREEN2_GET_WINDOW

引数

AH	16h

返値

AH	ｙ座標先頭位置(0～255) 1dot単位
AL	ｘ座標先頭位置(0～255) 1dot単位
DH	ｙ座標領域のサイズ(0～255) 1dot単位
DL	ｘ座標領域のサイズ(0～255) 1dot単位

処理

クリッピングウィンドウの領域を取得する。

INT 12h　AH=17h　SPRITE_SET_WINDOW

引数

AH	17h
BH	ｙ座標先頭位置(0～255) 1dot単位
BL	ｘ座標先頭位置(0～255) 1dot単位
CH	ｙ座標領域のサイズ(0～255) 1dot単位
CL	ｘ座標領域のサイズ(0～255) 1dot単位

返値

無し

処理

クリッピングウィンドウの領域を設定する。

INT 12h　AH=18h　SPRITE_GET_WINDOW

引数

AH	18h

返値

AH	ｙ座標先頭位置(0～255) 1dot単位
AL	ｘ座標先頭位置(0～255) 1dot単位
DH	ｙ座標領域のサイズ(0～255) 1dot単位
DL	ｘ座標領域のサイズ(0～255) 1dot単位

処理

クリッピングウィンドウの領域を取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=19h　PALETTE_SET_COLOR

引数

AH　19h
BX　パレット番号 (0～15)
CX　色　下記表参照

返値

無し

処理

パレットの色を設定する。

表３－３－６　パレット

ビット	内容
bit 0- 3	カラーコード０の色(LCD Color 0～7)
bit 4- 7	カラーコード１の色(LCD Color 0～7)
bit 8-11	カラーコード２の色(LCD Color 0～7)
bit 12-15	カラーコード３の色(LCD Color 0～7)

カラーモード時はこのファンクションを使用してはならない。

INT 12h　AH=1Ah　PALETTE_GET_COLOR

引数

AH　1Ah
BX

返値

AX　色　表3－3－6参照

処理

パレットの色を取得する。

カラーモード時はこのファンクションを使用してはならない。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=1Bh　LCD_SET_COLOR

引数

AH　　1Bh
CX:BX　　LCD Colorの階調（4bit(0(白)～(灰)～15(黒)) × 8個）

返値

無し

処理

各LCD Color(8つ)の階調を設定する。
下位(AX bit0-3)から順にLCD Color番号0,1,…と設定される。

カラーモード時はこのファンクションを使用してはならない。

INT 12h　AH=1Ch　LCD_GET_COLOR

引数

AH　　1Ch

返値

DXAX　　LCD Colorの階調（4bit(0(白)～(灰)～15(黒)) × 8個）

処理

各LCD Color(8つ)の階調を取得する。
下位(AX bit0-3)から順にLCD Color番号0,1,…と取得する。

カラーモード時はこのファンクションを使用してはならない。

INT 12h　AH=1Dh　LCD_SET_SEGMENTS

引数

AH　　1Dh
BX　　表示セグメント　下記表参照

返値

無し

処理

LCDのセグメント表示を設定する。

表３－３－７　LCDセグメント

ビット	内容
bit　0	Sl　省電力マーク
bit　1	Hr　横モード
bit　2	Vt　縦モード
bit　3- 5	システム予約
bit　6-15	未使用

INT 12h　AH=1Eh　LCD_GET_SEGMENTS

引数

AH　　1Eh

返値

AX　　表示セグメント　表３－３－７参照

処理

LCDのセグメント表示を取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 12h　AH=1Fh　LCD_SET_SLEEP

引数

AH	1Fh
BX bit 0	LCDの電源スイッチ（0:On / 1:Off(Sleep)）

返値

無し

処理

LCDの電源状態を設定する。

INT 12h　AH=20h　LCD_GET_SLEEP

引数

AH	20h

返値

AX	LCDの電源スイッチ（0:On / 1:Off(Sleep)）

処理

LCDの電源状態を取得する。

INT 12h　AH=21h　SCREEN_SET_VRAM

引数

AH	21h
AL	スクリーン　（0 : Screen1 / 1 : Screen2）
BL	VRAMのアドレス　（メーカ未公開）

返値

無し

処理

メーカ未公開

通常、スタートアップルーチンがディスプレイモードに応じた設定を行うため、ユーザが設定する必要はない。

INT 12h　AH=22h　SPRITE_SET_VRAM

引数

AH	22h
BL	VRAMのアドレス　（メーカ未公開）

返値

無し

処理

メーカ未公開

通常、スタートアップルーチンがディスプレイモードに応じた設定を行うため、ユーザが設定する必要はない。

３－４．テキスト制御

・WonderWitchは、テキスト画面制御機能を提供している。３－３項「画面」と関連性が高いため※1、３－３項を参照のこと。以下に、テキスト制御機能の概要を記す。

① スクリーン１又は２のどちらかをテキスト画面とすることができる。
② スクリーン全体，又は一部の矩形領域をテキスト用に初期化できる。
日本語表示が可能な場合、初期化したサイズ(面積)だけキャラクターを要する。
それ以外の場合は、定義した半角フォントの分だけキャラクターを要する。
③ 文字フォントを設定できる。
④ 表３－４－１に示すテキスト表示モードモードを選ぶことができる。
⑤ テキストスクリーンに使用するパレットを任意に指定できる。
⑥ カーソル機能を提供する。

・ユーザはテキスト画面を使用する際、以下の処理を行うべきである。

① 'TEXT_SET_SCREEN'ファンクションを用いて、任意のスクリーンをテキストスクリーンを設定する。
初期設定では、スタートアップルーチンがスクリーン２をテキストスクリーンに設定している。
② 'TEXT_SET_MODE'ファンクションを用いて、表３－４－１に示すテキスト表示モードを選択する。
初期設定は、ディスプレイモード、すなわちリンクするスタートアップルーチンでテキスト表示モードは異なる。３－３項、表３－３－２参照。
③ 'TEXT_SCREEN_INIT'ファンクション又は、'TEXT_WINDOW_INIT'ファンクションを用いて、テキストスクリーン又はテキストウィンドウを初期化する。
④ 'TEXT_SET_ANK_FONT'ファンクション及び'TEXT_SET_SJIS_FONT'ファンクションを用いて、テキストのフォントを定義する。
初期設定は、FreyaBIOSが設定したフォントが定義されている。

表３－４－１　テキスト表示モードについて

モード名	数値	名称(定数)	内容		
			1Byte文字	2Byte文字	ｷｬﾗｸﾀｰ
Ank	00h	TEXT_MODE_ANK	表示可	表示不可	？～511※1
AnkSjis	01h	TEXT_MODE_ANK_SJIS	表示可	表示可	不定※1
Sjis	02h	TEXT_MODE_SJIS	表示不可※2	表示可	不定※1

・表中「キャラクター」は、テキスト機能が使用するキャラクターコード
※1　初期化ファンクションで指定する数値によって異なる。
詳細は、'TEXT_SCREEN_INIT'ファンクション及び、
'TEXT_WINDOW_INIT'ファンクション参照。
※２　BIOS内部で、1ByteのASCII Codeは、2ByteのShift-JIS Codeに変換してフォントデータを取得する。その後、LCDに表示、若しくは'TEXT_GET_FONTDATA'ファンクションでの返値に返さ

※１　３－３項「画面」を参照のこと
※２　テキスト画面制御は、画面制御機能を利用してソフトウェアで実現している。
※３　16色カラーモード,パックトモード時は、このBIOSを使用してはならない。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 13h　AH=00h　TEXT_SCREEN_INIT

引数

AH　00h

返値

無し

処理

座標(0,0)-(17,27)に対して、'TEXT_WINDOW_INIT'を行う。この時、フォントテーブルのベースは以下の様になる。

●TEXT_ANK_MODE

512から、'TEXT_SET_ANK_FONT'関数で設定したフォント数(以後「フォント数」と略す)を引いた値（初期値は128）

●その他のMODE　8　(512 － 18 × 28)

INT 13h　AH=01h　TEXT_WINDOW_INIT

引数

AH	01h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CH	ｙ座標領域のサイズ
CL	ｘ座標領域のサイズ
DX	フォントテーブルのベース

返値

無し

処理

'TEXT_SET_SCREEN'ファンクションで指定したスクリーンに対して、指定した座標をテキスト用に初期化する。この時、フォントテーブルのベースを先頭に以下の数量だけのフォントが使用される。

●TEXT_ANK_MODE　フォント数(初期値は128)

●その他のMODE　指定した領域の面積。[単位:chr]

INT 13h　AH=02h　TEXT_SET_MODE

引数

AH	02h
BX	表示モード　表３－４－１

返値

無し

処理

表示モードを設定する。

INT 13h　AH=03h　TEXT_GET_MODE

引数

AH　03h

返値

BX　表示モード　表３－４－１

処理

表示モードを取得する。

INT 13h　AH=04h　TEXT_PUT_CHAR

引数

AH	04h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CX	文字コード(S-JIS)

返値

無し

処理

指定した文字をLCDに出力する。

INT 13h　AH=05h　TEXT_PUT_STRING

引数

AH	05h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
DS:DX	文字列(終了コード=00h)

返値

無し

処理

指定した文字列をLCDに出力する。

INT 13h　AH=06h　TEXT_PUT_SUBSTRING

引数

AH	06h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
DS:DX	文字列
CX	文字数

返値

無し

処理

指定した文字列をCX文字LCDに出力する。

INT 13h　AH=07h　TEXT_PUT_NUMERIC

引数

AH	07h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CH	フォーマット　　表３－４－２
CL	表示桁数
DX	数値
DS:SI	メモリ格納時の、格納先頭アドレス。

返値

無し

処理

指定した数値をLCDに出力する。

表3－4－2　出力フォーマット

ビット	内容
bit 0	'1':16進数
bit 1	余った桁を埋める文字。'0':' '(空白) ／ '1':'0'
bit 2	'0':左詰 ／ '1':右詰
bit 3	'0':符号無し ／ '1':符号付き
bit 4	未使用
bit 5	未使用
bit 6	未使用
bit 7	'0':LCDへ出力 ／ '1':DS:SIの示すアドレスに出力

INT 13h　AH=08h　TEXT_FILL_CHAR

引数

AH	08h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CX	文字数
DX	文字コード(S-JIS)

返値

無し

処理

指定した文字を、CX分連続してLCDに出力する。

INT 13h　AH=09h　TEXT_SET_PALETTE

引数

AH	09h
BX	パレット番号

返値

無し

処理

テキスト画面で使用するパレットを設定する。

INT 13h　AH=0Ah　TEXT_GET_PALETTE

引数

AH	0Ah

返値

AX	パレット番号

処理

テキスト画面で使用するパレットを取得する。

INT 13h　AH=0Bh　TEXT_SET_ANK_FONT

引数

AH	0Bh
BL	文字コードのベース
BH	フォントのタイプ(0:単色／1:カラー(４色))
CX	フォント数
DS:DX	フォントのあるアドレス

返値

無し

処理

1Byte文字のフォントを指定する。初期値は以下となっている。
・文字コードのベース、フォントのタイプ　＝　0,単色
・フォント数、フォント　＝　128,FreyaBIOS内部のフォント

INT 13h　AH=0Ch　TEXT_SET_SJIS_FONT

引数

AH	0Ch
BX:DX	フォント解決ルーチンのファーアドレス BX=0:2Bytem文字は使用しない。

返値

無し

処理

2Byte文字のフォントを返すサブルーチンのアドレスを設定する。
サブルーチンは以下の仕様にしなければならない。
初期値は、FreyaBIOS内のルーチンが設定されている。

●引数	AX	S-JIS文字コード
●返値	CS:SI	2Byte文字のフォントデータ格納アドレス
	CyFlag	フォントデータが見つからない。

INT 13h　AH=0Dh　TEXT_GET_FONTDATA

引数

AH	0Dh
CX	文字コード
DS:DX	フォントデータ格納先アドレス (8Byte)

返値

AX	エラーコード 0:正常終了 0以外:不正な文字コードの指定

処理

FreyaBIOS内のモノクロフォントを取得する。

INT 13h　AH=0Eh　TEXT_SET_SCREEN

引数

AH　0Eh
AL　スクリーン

返値

無し

処理

テキストBIOSで使用するスクリーンを設定する。

INT 13h　AH=0Fh　TEXT_GET_SCREEN

引数

AH　0Fh

返値

AX　スクリーン

処理

テキストBIOSで使用するスクリーンを取得する。

INT 13h　AH=10h　CURSOR_DISPLAY

引数

AH　10h
AL　カーソル表示状態(0:非表示／1:表示)

返値

無し

処理

カーソルの表示有無を設定する。

INT 13h　AH=11h　CURSOR_STATUS

引数

AH　11h

返値

AX　カーソル表示状態(0:非表示／1:表示)

処理

カーソルの表示有無を取得する。

INT 13h　AH=12h　CURSOR_SET_LOCATION

引数

AH	12h
BH	ｙ座標先頭位置
BL	ｘ座標先頭位置
CH	ｙ座標領域のサイズ[chr]
CL	ｘ座標領域のサイズ[chr]

返値

無し

処理

カーソルの表示座標、大きさを指定する。

INT 13h　AH=13h　CURSOR_GET_LOCATION

引数

AH	13h

返値

AH	ｙ座標先頭位置
AL	ｘ座標先頭位置
DH	ｙ座標領域のサイズ[chr]
DL	ｘ座標領域のサイズ[chr]

処理

カーソルの表示座標、大きさを取得する。

INT 13h　AH=14h　CURSOR_SET_TYPE

引数

AH	14h
BL	使用するパレット番号
CL	点滅周期（HBLANK（1/75[sec]）単位）

返値

無し

処理

カーソルの表示タイプを指定する。
初期値は以下の通り。
・パレット番号　＝　1
・点滅周期　＝　30

INT 13h　AH=15h　CURSOR_GET_TYPE

引数

AH	15h

返値

AL	パレット番号
AH	点滅周期（HBLANK（1/75[sec]）単位）

処理

カーソルの表示タイプを取得する。

３－５．シリアル通信

・WonderSwanは、シリアル回線を搭載しており、以下の機能を有す。
　①　9600bps , 38400bpsの非同期通信
　②　タイムアウト処理
　③　オーバーランの検出
　④　キーによる通信のキャンセル
　⑤　X-modemのよるブロック送受信（メーカ未公開）
・通信は以下のプロトコルで行われる。

① プロトコル	調歩同期 , 全二重通信
② 通信速度	9600 bps / 38400 bps
③ データ長	8 bit
④ ストップビット	1 bit
⑤ エラー検出	受信オーバーラン
⑥ XON/XOFF制御	無し
⑦ モデム制御信号	無し

・FreyaBIOSは、上記の制御機能を提供する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 14h	AH=00h	COMM_OPEN
引数		
	AH	00h
返値		
	無し	
処理		
	シリアル回線を初期化し、利用を開始する。 あらかじめ、AH=09hでボーレートを設定する必要がある。	

INT 14h	AH=01h	COMM_CLOSE
引数		
	AH	01h
返値		
	無し	
処理		
	シリアル通信を終了する。	

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 14h　AH=02h　COMM_SEND_CHAR

引数

AH　02h
BL　送信データ

返値

AX　エラーコード　表３－５－１参照

処理

1byteのデータを送信する。
このファンクションで返るエラーコードは、以下の通り。
・ERR_SIO_TIMEOUT　タイムアウトの発生。
・ERR_SIO_CANSEL　キャンセルされた。

表３－５－１　エラーコード

数値	名称（定数）	内容
0000h	ERR_SIO_OK	正常終了
8100h	ERR_SIO_BUSY	ビジー状態
8101h	ERR_SIO_TIMEOUT	タイムアウトの発生
8102h	ERR_SIO_OVERRUN	オーバーランの発生
8103h	ERR_SIO_CANCEL	キャンセルされた
8104h	ERR_XM_STATECODE	メーカ未公開
8105h	ERR_XM_CANCELED	メーカ未公開
8106h	ERR_XM_BLOCK_LOST	メーカ未公開
8107h	ERR_XM_TOO_LARGE	メーカ未公開

INT 14h　AH=03h　COMM_RECEIVE_CHAR

引数

AH　03h

返値

AX　受信データ／エラーコード　表３－５－１参照

処理

1byteのデータを受信する。
このファンクションで返るエラーコードは、以下の通り。
・ERR_SIO_TIMEOUT　タイムアウトの発生。
・ERR_SIO_OVERRUN　オーバーランの発生。
・ERR_SIO_CANSEL　キャンセルされた。

INT 14h　AH=04h　COMM_RECEIVE_WITH_TIMEOUT

引数

AH　04h
CX　タイムアウト値（単位　約1/75秒）

返値

AX　受信データ／エラーコード　表３－５－１参照

処理

タイムアウト値を指定して1byteのデータを受信します。
タイムアウト判定処理は、VBLANKを利用して行われる。
以下は、'COMM_RECEIVE_CHAR'ファンクションと同等。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 14h	AH=05h	COMM_SEND_STRING
引数		
	AH	05h
	DS:DX	文字列格納アドレス（終端は、00h）
返値		
	AX	エラーコード　表３－５－１参照
処理		
	DS:DXが示すアドレスの文字列データを、送信します。 以下、'COMM_SEND_CHAR'ファンクションと同等。	

INT 14h	AH=06h	COMM_SEND_BLOCK
引数		
	AH	06h
	DS:DX	送信データ格納アドレス
	CX	送信データサイズ（単位byte）
返値		
	AX	エラーコード　表３－５－１参照
処理		
	DS:DXが示すアドレスのデータを、CX分送信します。 以下、'COMM_SEND_CHAR'ファンクションと同等。	

INT 14h	AH=07h	COMM_RECEIVE_BLOCK
引数		
	AH	07h
	DS:DX	受信データ格納アドレス
	CX	受信データサイズ（単位byte）
返値		
	DS:DXの指すアドレスに受信データが格納される。	
	DX	受信サイズ（単位byte）
	AX	エラーコード　表３－５－１参照
処理		
	DS:DXが示すアドレスに、CX分受信します。 DXには、受信したデータサイズを返す。 以下は、'COMM_RECEIVE_CHAR'ファンクションと同等。	

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 14h　AH=08h　COMM_SET_TIMEOUT

引数

AH　08h
BX　受信タイムアウト(0,即返る/0FFFFh(-1),待ち続ける)
CX　送信タイムアウト(0,即返る/0FFFFh(-1),待ち続ける)

返値

無し

処理

送受信のデフォルトのタイムアウト時間を設定する。
初期値は、'-1'（待ち続ける）となっています。
タイムアウト判定処理は、VBLANKを利用して行われる。

INT 14h　AH=09h　COMM_SET_BAUDRATE

引数

AH　09h
BX　通信速度　表３－５－２参照

返値

無し

処理

シリアル通信のボーレートを設定する。

表3－5－2　ボーレート

数値	名称（定数）	内容
0	COMM_SPEED_9600	9600[bps]
1	COMM_SPEED_38400	38400[bps]

INT 14h　AH=0Ah　COMM_GET_BAUDRATE

引数

AH　0Ah

返値

AX　通信速度　表３－５－２参照

処理

シリアル通信のボーレートを所得する。

INT 14h　AH=0Bh　COMM_SET_CANCEL_KEY

引数

AH　0Bh
BX　キャンセルキーパターン　３－２項　表３－２－１参照

返値

AX　キャンセルキーパターン　３－２項　表３－２－１参照

処理

通信をキャンセルするキーパターンを設定する。
'0'の時は、キャンセルをしない。
返値は、変更前のキーパターンが返る。

INT 14h　AH=0Ch　COMM_GET_CANCEL_KEY

引数

AH　0Ch

返値

AX　キャンセルキーパターン　３－２項　表３－２－１参照

処理

通信をキャンセルするキーパターンを所得する。
'0'の時は、キャンセルをしない。

INT 14h　AH=0Dh　COMM_XMODEM

引数

AH　0Dh
DS:BX　xmodeminfo構造体のポインタ　表3－5－3参照

返値

AX　キャンセルキーパターン　３－２項　表３－２－１参照

処理

XMODEMプロトコルによるバイナリーの送受信をする。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

表3－5－3　modeminfo構造体について

アドレス	型	内容	
0000h	word	ステータス	メーカ未公開情報
0002h	byte	モード	メーカ未公開情報
0003h	byte	リトライ	メーカ未公開情報
0004h	word	ブロック	メーカ未公開情報
0006h	word	ブロック	メーカ未公開情報
0008h	word	ブロック	メーカ未公開情報
000Ah	word	セグメント	メーカ未公開情報
000Ch	word	オフセット	メーカ未公開情報
000Eh	word	タイムアウト	メーカ未公開情報

３－６．サウンド

・WonderSwan本体は量子化数4bit(16段階)のPCMを４チャンネル搭載している。
・PCM波形は、１波長についてのサンプリング数は32個である。
・左右別々に16段階の音量が設定可能である。
・発生波長（周波数）は2048段階で指定する。
・各チャンネルには以下のモードがある。

ch1	モード無し	
ch2	PCMVoice	量子化数8bitのPCMを発生できる。
ch3	Sweep	時間による周波数変化(1shotLFO)制御ができる。
ch4	Noise	疑似乱数を用いてノイズを発生することができる。

・FreyaBIOSでは、以上の制御機能を提供する。
・PCM Voiceの音量の設定および取得は、FreyaBIOS ver1,0では実装されていない。これらの関数は、Ｃ言語ライブラリで提供される。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 15h　AH=00h　SOUND_INIT

引数

AH　00h

返値

無し

処理

サウンドを全てOffにし、
波形テーブル（音色）も0にする。

INT 15h　AH=01h　SOUND_SET_CHANNEL

引数

AH　01h
BL　サウンド出力／モードのフラグ　以下表参照。

返値

無し

処理

サウンドのチャンネルに対して、以下表の設定を行う。

表３－６－１　サウンドモード

ビット	ﾁｬﾝﾈﾙ	内容
bit 0	ch1	Output Disable(0)/Enable(1)
bit 1	ch2	Output Disable(0)/Enable(1)※２
bit 2	ch3	Output Disable(0)/Enable(1)
bit 3	ch4	Output Disable(0)/Enable(1)
bit 4	－	未使用（予約済み）※１
bit 5	ch2	PCMVoiceModeSw※２
bit 6	ch3	SweepModeSw
bit 7	ch4	NoiseModeSw

※１　必ず０を設定して下さい
※２　PCMVoiceModeSwが'1'の時は、
Output Disableの状態でも出力される。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 15h　AH=02h　SOUND_GET_CHANNEL

引数

AH　　02h

返値

BX　　サウンド出力／モードのフラグ　表３－６－１参照。

処理

現在のサウンド出力／モードの状態を得る。

INT 15h　AH=03h　SOUND_SET_OUTPUT

引数

AH　　03h
BL　　サウンド出力設定

返値

無し

処理

以下の表に示す内容を設定する。

表３－６－２　サウンド出力

ビット	内容
bit 0	内蔵Speaker Disable(0)/Enable(1)
bit 1-2	内蔵Speaker Outpur Range※１
bit 3	Stereo head phone Disable(0)/Enable(1)
bit 4-6	未使用（予約済み）※２
bit 7	Stereo head phone Connect※３

※１　サウンド出力データの11bit分、内蔵スピーカーから出力する8bitを選択する。
00<b>　D0～D7を出力
01<b>　D1～D8を出力
10<b>　D2～D9を出力
11<b>　D3～D10を出力

※２　必ず０を設定して下さい

※３　外部ヘッドホンの接続状態。接続時に'1'となる。
AH=04hのファンクションで読み込む時のみのビットである。

INT 15h　AH=04h　SOUND_GET_OUTPUT

引数

AH　　04h

返値

AX　　サウンド出力設定　表３－６－２参照。

処理

現在のサウンド出力設定の状態を得る。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 15h　AH=05h　SOUND_SET_WAVE

引数

AH　05h
AL　チャンネル（０～３）
DS:DX　波形データ格納先頭アドレス

返値

無し

処理

波形データをセットする。
波形データは、16byte１組であり、以下の書式となる。

表３－６－３　波形データ書式

アドレス		位相[rad]	内容
0byte	bit 4-7	0	・指定数値について
	bit 0-3	π/16	0h　符号：負　ﾚﾍﾞﾙ：大
1byte	bit 4-7	2π/16	～　～　～
	bit 0-3	3π/16	7h　符号：負　ﾚﾍﾞﾙ：小
…	…	…	8h　符号：正　ﾚﾍﾞﾙ：小
8byte	bit 4-7	π	～　～　～
	bit 0-3	17π/16	Fh　符号：正　ﾚﾍﾞﾙ：大
…	…	…	・データについて
16byte	bit 4-7	30π/16	ｻﾝﾌﾟﾘﾝｸﾞﾚｲﾄ = 32個／1波長
	bit 0-3	31π/16	量子化数 = 4bit(16段階)

INT 15h　AH=06h　SOUND_SET_PITCH

引数

AH　06h
AL　チャンネル（０～３）
BX　周波数(0～2047)　式３－６－１参照

返値

無し

処理

周波数を設定する。
周波数f[Hz]は、以下の式で計算できる。
nは、BXレジスタで指定すべき数値とする。

$$f\,[Hz] = \frac{3.072 \times 10^{6}}{(2048 - n) \times 32} \quad \cdots\cdots\cdots \text{式３－６－１}$$

チャンネル３のスウィープモードの時は、初期周波数となる。

INT 15h　AH=07h　SOUND_GET_PITCH

引数

AH　07h
AL　チャンネル（０～３）

返値

AX　周波数(0～2047)　式３－６－１参照

処理

周波数を取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 15h　AH=08h　SOUND_SET_VOLUME

引数

AH　　08h
AL　　チャンネル（０～３）
BL　　音量　表３－６－４参照
無し

処理

音量を設定する。
チャンネル２のＰＣＭボイスモード（8bitPCM）の時は、
8bit全てを用いて、データを指定する。サンプリングレイトは、
この１秒間にファンクションを呼ぶ回数となる。

表３－６－４　音量について

ビット	内容
bit 0-3	右の音量（0(小)～15(大)）
bit 4-7	左の音量（0(小)～15(大)）

INT 15h　AH=09h　SOUND_GET_VOLUME

引数

AH　　09h
AL　　チャンネル（０～３）

返値

AL　　音量　表３－６－４参照

処理

音量を取得する。

INT 15h　AH=0Ah　SOUND_SET_SWEEP

引数

AH　　0Ah
BL　　スウィープ値(-128～127)　式３－６－１参照
CL　　ステップタイム（0～31）　式３－６－２参照

返値

無し

処理

スウィープ機能（ﾊｰﾄﾞｳｪｱﾚﾍﾞﾙでのの1shotLFO）を設定する。
１回の変化時間をt[msec]、CLレジスタで指定値をnとすると
以下の式で計算できる。

$T\ [msec] = 2.667 \times (n + 1)$　　・・・・・・・・　式３－６－２

INT 15h　AH=0Bh　SOUND_GET_SWEEP

引数

AH　　0Bh

返値

AL　　スウィープ値(-128～127)　式３－６－１参照
AH　　ステップタイム（0～31）　式３－６－２参照

処理

スウィープ機能（ﾊｰﾄﾞｳｪｱﾚﾍﾞﾙでのの1shotLFO）を取得する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 15h　AH=0Ch　SOUND_SET_NOISE

引数

AH　　0Ch
ＢＬ　　ノイズ制御値　表３－６－５参照

返値

無し

処理

表３－６－５のとおり、チャンネル４のノイズモードを設定する。
ノイズ使用時は、bit3,4を'H'にすれば、問題はない。

表３－６－５　ノイズ制御値について

ビット	内容
bit 0-2	ノイズ形状 (恐らく、ノイズの深さと思われる。)
bit 3	ノイズ生成用カウンターリセット　詳細不明
bit 4	ノイズ生成用カウンターイネーブル　詳細不明
bit 5-7	未使用

INT 15h　AH=0Dh　SOUND_GET_NOISE

引数

AH　　0Dh

返値

AX　　ノイズ制御値　表３－６－５参照

処理

表３－６－５のとおり、チャンネル４のノイズモードを取得する。

INT 15h　AH=0Eh　SOUND_GET_RANDOM

引数

AH　　0Eh

返値

AX　　疑似乱数（0～32767）

処理

チャンネル４のノイズモードで生成される疑似乱数を取得する。
※１　ノイズ発生用ポリノミカル・カウンタの値を読み出している。
※２　この機能を利用する場合、AH=01hでサウンド・チャンネル4をイネーブルにしておく必要がある。

３－７．カレンダー／タイマー

・WonderWitch専用カートリッジには、カレンダ・時計が搭載されている。
・WonderSwan本体は、LCDの同期信号を用いたタイマー割り込みを提供する。
・FreyaBIOSでは、以上の制御機能を提供する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 16h　AH=00h　RTC_RESET

引数

AH　00h

返値

無し

処理

カセット内蔵のRTCをリセットする。
この機能は、出荷後１度だけ実行されるもので、この機能はユーザプログラムでは、呼び出してはならない機能である。

INT 16h　AH=01h　RTC_GET_DATETIME

引数

AH　01h
BX　読み込むデータ　表３－７－１参照

返値

AX　値

処理

BXで指定した内容について、データを所得する。

表３－７－１　カレンダ・時計

指定数値	名称(定数)	内容
0	RTC_YEAR	年(00h=2000年)
1	RTC_MONTH	月
2	RET_DATE	日
3	RTC_DAY_OF_WEEK	曜日(00h=???)
4	RET_HOUR	時間
5	RET_MIN	分
6	RET_SEC	秒(単位？)

INT 16h　AH=02h　RTC_SET_DATETIME

引数

AH　02h
BX　書き込むデータの種類　表３－７－１参照
CX　書き込む値

返り値

無し

処理

BXで指定した内容について、データを設定する。

INT 16h　AH=03h　RTC_SET_DATETIME_STRUCT

引数

AH　03h
DS:DX　設定するdatetime_t構造体のポインタ　以下表参照

返値

無し

処理

一括してカレンダ・時計を設定する。

表３－７－２　datetime_t構造体について

アドレス	型	内容
0000h	byte	年（00hを西暦２０００年とする。）
0001h	byte	月
0002h	byte	日
0003h	byte	曜日(00h=???)
0004h	byte	時間
0005h	byte	分
0006h	byte	秒(単位？)

INT 16h　AH=04h　RTC_GET_DATETIME_STRUCT

引数

AH　04h
DS:DX　datetime_t構造体を所得するポインタ　表3-7-2参照

返値

DS:DXの示すポインタに設定状態を返す。

処理

一括してカレンダ・時計を設定を所得する。

INT 16h　AH=05h　RTC_ENABLE_ALARM

引数

AH　05h
BL　アラーム時刻　時間
BH　アラーム時刻　分

返値

無し

処理

　アラーム（RTC割り込み）の時刻を設定する。
割り込み先は、'SYS_INTERRUPT_SET_HOOK'ファンクションで設定する。３－８項参照。

INT 16h　AH=06h　RTC_DISABLE_ALARM

引数

AH　06h

返値

無し

処理

アラームを解除する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 16h　AH=07h　TIMER_ENABLE

引数

AH　07h
AL　タイマータイプ　表３－７－３参照
BL　オートプリセット　表３－７－４参照
CX　プロセットカウンタ　表３－７－５参照

返値

無し

処理

タイマー割り込みの設定を行う。以下表参照。
割り込み先は、'SYS_INTERRUPT_SET_HOOK'ファンクションで設定する。３－８項参照。

表３－７－３　タイマータイプ

指定数値	名称（定数）	内容
0	TIMER_HBLANK	HBLANKタイマー
1	TIMER_VBLANK	VBLANKタイマー

表３－７－４　オートプリセット

指定数値	名称（定数）	内容
0	TIMER_ONESHOT	１回のみ
1	TIMER_AUTOPRESET	連続

表３－７－５　カウンタ精度

タイマータイプ	精度（有効数字3ケタ）
TIMER_HBLANK	周波数: 12.00[kHz　周期： 83.33[μs]
TIMER_VBLANK	周波数: 75.47[Hz]　周期： 13.25[ms]

INT 16h　AH=08h　TIMER_DISABLE

引数

AH　08h
AL　タイマータイプ　表３－７－３参照

返値

無し

処理

指定したタイマー割り込みを禁止する。
FreyaBIOS ver.1.0ではバグがあり、このファンクションを呼び出すとHBLANKの割り込み周期が変化するだけである。

INT 16h　AH=09h　TIMER_GET_COUNT

引数

AH　09h
AL　タイマータイプ　表３－７－３参照

返値

AX　現在のカウント値　表３－７－５参照

処理

指定したタイマーのカウンタの現在値を所得する。

３－８．システム

・WonderSwan本体は、サスペンド／リジューム機能を搭載している。
・FreyaBIOSでは、サスペンド／リジューム制御および割り込み制御機能を提供する。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=00h　SYS_INTERRUPT_SET_HOOK

引数

AH	00h
AL	割り込み要因　以下表参照
DS:BX	設定するintvector_t構造体のポインタ　以下表参照
DS:DX	設定する前の状態を返すポインタ。(DX=0の時は無し)

返値

DS:DXの示すポインタに設定前の状態を返す。

処理

　割り込み先のアドレスを設定する。
割り込み発生時は、まずBIOS内の割り込みルーチンがよばれ、
BIOS内でこのファンクションで指定したアドレスを呼び出す。
このため、登録する関数は、'retf'命令で戻る必要がある。
Ｃ言語の場合は、far関数として定義すればよい。
詳細は、２－３項を参照のこと。

表３－８－１　割り込み要因

設定値	名称（定数）	内容
0	SYS_INT_SENDREADY	送信データが空
1	SYS_INT_KEY	キー割り込み
2	SYS_INT_CASETTE	カセット割り込み※1
3	SYS_INT_RECEIVEREADY	受信データレディー
4	SYS_INT_DISPLINE	描画ライン番号検知
5	SYS_INT_TIMER_COUNTUP	タイマー割り込み※2
6	SYS_INT_VBLANK	期間開始
7	SYS_INT_HBLANK_COUNTUI	タイマー割り込み

※１　SYS_INT_CASETTEは、RTCからのアラーム割り込みで利用されるので、 設定するとアラーム機能が正しく動作しない場合がある。

※２　FreyaBIOSver1.0では、SYS_INT_TIMER_COUNTUPにバグがあるため使用できない(FreyaBIOS内部でのスタックポインタの破壊と思われる)。

表３－８－２　intvector_t構造体について

アドレス	型	内容
0000h	word	呼び出すルーチンのオフセットアドレス
0002h	word	呼び出すルーチンのセグメントアドレス
0004h	word	割り込み先で設定されるDSの値※3
0006h	word	設定必要無し。(BIOSが設定する。)

※3　0000hの場合、呼び出したプログラムのセグメントとなる。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=01h　SYS_INTERRUPT_RESET_HOOK

引数

AH　01h
AL　割り込み要因　表３－８－１参照
DS:BX　intvector_t構造体のポインタ　表３－８－２参照

返値

無し

処理

指定した割り込み要因をSYS_INTERRUPT_SET_HOOK'ファンクションで得られた初期状態値に戻す。DS:BXには、戻すべきintvector_t構造体のポインタをセットする。BX=0の場合は、割り込みハンドラをクリアする。
FreyaBIOSver,1.0では、一部、指定していない他の割り込み要因もリセットしてしまうバグがある。ライブラリ'libww.lib'内の'_SYS_INTERRUPT_SET_HOOK'関数を使うことで解決され

INT 17h　AH=02h　SYS_WAIT

引数

AH　02h
CX　待ち時間（単位約13.25[msec]）

返値

無し

処理

CXで指定された数値のみ、処理を停止する。
正確な単位および値の根拠はメーカ公開されていない。
VBLANKによるカウントが行われる。

INT 17h　AH=03h　SYS_GET_TICK_COUNT

引数

AH　03h

返値

DX:AX　起動からの経過時間（単位約13.25[msec]）

処理

起動時からの経過時間を32bit幅で所得する。
VBLANKによるカウントが行われる。

INT 17h　AH=04h　SYS_SLEEP

引数

AH　04h

返値

無し

処理

LCDをOffにして、スリープモードに入る。
'SYS_SET_AWAKE_KEY'ファンクションで設定されたキーが押されることで、スリープモードを抜ける。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h	AH=05h	SYS_SET_SLEEP_TIME
引数		
	AH	05h
	BL	無操作時にSleepModeに入るまでの時間(単位　分)
返値		
	無し	
処理		
	何も操作をしないときに、スリープモードに入るまでの時間を分単位で設定する。BL=0のときは、スリープモードに移行しない。	
INT 17h	AH=06h	SYS_GET_SLEEP_TIME
引数		
	AH	06h
返値		
	AX	無操作時にSleepModeに入るまでの時間(単位　分)
処理		
	何も操作をしないときに、スリープモードに入るまでの時間を分単位で所得する。BL=0のときは、スリープモードに移行しない。	
INT 17h	AH=07h	SYS_SET_AWAKE_KEY
引数		
	AH	07h
	BX	キーパターン　　３－２項　表３－２－１参照
返値		
	無し	
処理		
	スリープモードを解除するキーパターンを設定する。 BX=0の時は、どのキーを押してもスリープモードを解除する。	
INT 17h	AH=08h	SYS_GET_AWAKE_KEY
引数		
	AH	08h
返値		
	AX	キーパターン　　３－２項　表３－２－１参照
処理		
	スリープモードを解除するキーパターンを所得する。 BX=0の時は、どのキーを押してもスリープモードを解除する。	

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=09h　SYS_SET_KEEPALIVE_INT

引数

AH　09h
BL　割り込み要因フラグ　表３－８－３参照

返値

無し

処理

スリープ中に有効な割り込みを設定する。
デフォルトでは、キー割り込みのみが有効となっている。

表３－８－３　割り込み要因

ビット	内容
bit 0	シリアル送信データ・エンプティー
bit 1	キー割り込み
bit 2	カセット割り込み
bit 3	シリアル受信データ・レディー
bit 4	描画ライン番号検出割り込み
bit 5	VBLANLKタイマー・カウント終了
bit 6	VBLANK期間開始
bit 7	HBLANLKタイマー・カウント終了

INT 17h　AH=0Ah　SYS_GET_OWNERINFO

引数

AH　0Ah
CX　構造体のサイズ
DS:DX　格納先アドレス　表３－８－４参照

返値

AX　0　正常終了
ERR_SIO_TIMEOUT　読み込みエラー

処理

所有者情報の構造体が先頭からCX[Byte]分だけ得られます。

表３－８－４　所有者情報構造体　ownerinfo_t

型	内容
char [16]	名前　半角16文字
int	誕生日　年 西暦
char	誕生日　月 1～12
char	誕生日　日 1～31
char	性別　0：?／1：男／2：女
char	血液型　0：?／1：A／2：B／3：O／4：AB

注意　Wonder Swan Colorでは、正しい動作が得られないので、留意する。
libww.libの、wwc_sys_get_ownerinfo()関数を使うことで解決できる。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=0Bh　SYS_SUSPEND

引数

AH	0Bh
AL	コアイメージ番号（ユーザプロセス番号(0 or 1)）
DS:BX	I/Oを記録する場所　表3－8－5参照

返値

AX	0：保存完了
	1： サスペンドから復帰

処理

FreyaOS用のSRAMワークエリアにIRAMの内容全て(BIOSワークエリア、スタック、VRAM内容等)を保存する。
IOポートの状態はDS:BXで指定された場所に保存される。
このIO状態の保存先アドレスはIRAMと一緒に保存される。
その他、SS,SPレジスタも保存される
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

表３－８－５　Ｉ／Ｏポート記録の構造体

型	内容
char[2]	ヘッダ　'IO'
char[0xE0]	I/Oポート内容を記録する領域　0:記録しない／1:記録

INT 17h　AH=0Ch　SYS_RESUME

引数

AH	0Ch
AL	コアイメージ番号（ユーザプロセス番号(0 or 1)）

返値

無し

処理

'SYS_SUSPEND'関数で保存した情報を復元し、制御は復元したプロセスに戻る。
情報が保存されていない場合は、何もしない。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=0Dh　SYS_SET_REMOTE

引数

AH　0Dh
AL　0 : Remote Mode Disable
　　1 : Remote Mode Enable

返値

無し

処理

リモートモードを設定する。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

INT 17h　AH=0Eh　SYS_GET_REMOTE

引数

AH　0Eh

返値

AL　0 : Remote Mode Disable
　　1 : Remote Mode Enable

処理

リモートモードを取得する。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=0Fh　SYS_ALLOC_IRAM

引数

AH	0Fh
BX	確保したメモリの先頭アドレスを返すアドレス。 '0'の場合はBIOS内部で記憶し、 SYS_GET_MY_IRAM'ファンクションで、アドレスを
CX	メモリを要求するサイズ

返値

AX	確保したメモリの先頭アドレス。 セグメントアドレスは0000hである。 '0'の場合は、容量が足りないため確保に失敗。

処理

BIOS内部のメモリを常駐IL等のワークエリアとして確保する。
BX≠0の場合、BXの示す先には、'0'が入っている必要がある。
BX=0で呼び出した場合は、同じセグメント内でこのファンクションを呼び出してはならない。解放した後であれば、問題はない。
BX=0での記憶容量は８個までである。
CSレジスタがいわゆるハンドルとして使われている。

INT 17h　AH=10h　SYS_FREE_IRAM

引数

AH	10h
BX	解放するメモリの先頭アドレス。 セグメントアドレスは0000hである。

返値

無し

処理

BIOS内部のメモリを解放する。
一番最後に確保したワークエリアから解放しなければならない。
CSレジスタがいわゆるハンドルとして使われている。

INT 17h　AH=11h　SYS_GET_MY_IRAM

引数

AH	11h

返値

AX	確保したメモリの先頭アドレス。 セグメントアドレスは0000hである。 '0'の場合は、確保されていない。

処理

'SYS_ALLOC_IRAM'ファンクションにて、BX=0でメモリを確保した場合、BIOS内部に記憶しているアドレスを取得する。
このファンクションを呼び出した同一のセグメントでメモリが確保されている必要がある。
CSレジスタがいわゆるハンドルとして使われている。

ワンダーウィッチ　ＢＩＯＳファンクション表

INT 17h　AH=12h　SYS_GET_VERSION

引数

AH　12h

返値

AX　バージョン情報

処理

BIOSのバージョンを得る。

表３－８－３　割り込み要因

ビット	内容
bit 0-7	サブ・マイナー
bit 8-11	マイナー
bit 12-15	メジャー

INT 17h　AH=13h　SYS_SWAP

引数

AH　13h
AL　コアイメージ番号（ユーザプロセス番号(0 or 1)）

返値

AX　0：正常終了　／　1：退避しているプロセスは無い。

処理

現在のプロセスをサスペンドして、指定したプロセスをレジュームする。'SYS_SUSPEND'関数及び、'SYS_RESUME'関数を参照。
プロセスが退避されていなかった場合は、何も処理をしない。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

INT 17h　AH=14h　SYS_SET_RESUME

引数

AH　14h
AX　メーカ未公開情報

返値

無し

処理

レジューム保存状態を設定する。詳細不明。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

INT 17h　AH=15h　SYS_GET_RESUME

引数

AH　15h

返値

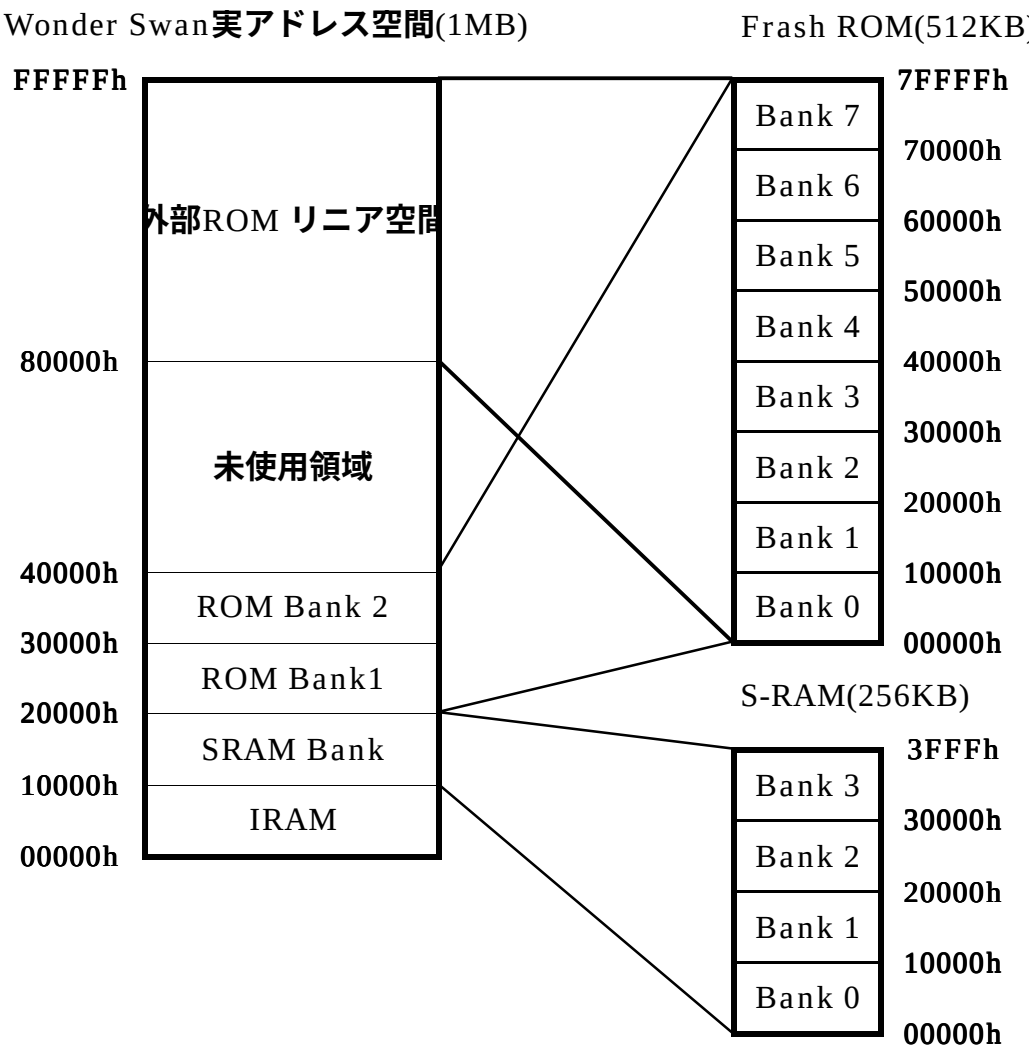
AX　メーカ未公開情報

処理

レジューム保存状態を取得する。詳細不明。
このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

３－９．バンク

・以下、Wonder Swanのメモリー構造を示す。



ワンダーウィッチ　ＢＩＯＳファンクション表

INT 18h　AH=00h　BANK_SET_MAP

引数

AH　00h
BL　バンクの種類　表3－9－1
CX　バンク番号(SRAM：0～3 ／ ROM：0～7)

返値

Bank**に、メモリをマッピングする。**

処理

バンクを設定する。

表３－８－１　割り込み要因

設定値	名称（定数）	内容
0	BANK_SRAM	SRAM (Segment = 1000h)
1	BANK_ROM0	ROM0 (Segment = 2000h)
2	BANK_ROM1	ROM1 (Segment = 3000h)

INT 18h　AH=01h　BANK_GET_MAP

引数

AH　01h
BL　バンクの種類　表3－9－1

返値

AX　バンク番号(SRAM：0～3 ／ ROM：0～7)

処理

現在のバンクを取得する。

INT 18h　AH=02h　BANK_READ_BYTE

引数

AH	02h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
DX	バンク内のオフセットアドレス	

返値

AL	データ

処理

データを1[Byte]読み込む

INT 18h　AH=03h　BANK_WRITE_BYTE

引数

AH	03h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
DX	バンク内のオフセットアドレス	
AL	データ	

返値

無し

処理

データを1[Byte]書き込む。

INT 18h　AH=04h　BANK_READ_WORD

引数

AH	04h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
DX	バンク内のオフセットアドレス	

返値

AX	データ

処理

データを2[Byte]読み込む

INT 18h　AH=05h　BANK_WRITE_WORD

引数

AH	05h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
DX	バンク内のオフセットアドレス	
CX	データ	

返値

無し

処理

データを2[Byte]書き込む。

INT 18h　AH=06h　BANK_READ_BLOCK

引数

AH	06h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
CX	データのサイズ	
DX	バンク内のオフセットアドレス	
DS:SI	データの保存先アドレス	

返値

無し

処理

データをCX[Byte]書き込む。

INT 18h　AH=07h　BANK_WRITE_BLOCK

引数

AH	07h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
CX	データのサイズ	
DX	バンク内のオフセットアドレス	
DS:SI	データの保存先アドレス	

返値

無し

処理

データをCX[Byte]読み込む。

INT 18h　AH=08h　BANK_FILL_BLOCK

引数

AH	08h	
BX	バンク番号	SRAM : 0000h～0003h ROM : 8000h～8007h
CX	データのサイズ(0 ： 64KB全て)	
DX	バンク内のオフセットアドレス	
AL	FILLデータ	

返値

無し

処理

バンクの中身を、ALで埋める。

INT 18h　AH=09h　BANK_ERASE_FLASH

引数

AH	09h
BX	バンク番号

返値

メーカ未公開

処理

メーカ未公開

このファンクションは、システムが利用するものであるため、ユーザが利用した場合の動作は保証されていない。

４．ＯＳ仕様

1.) FreyaOS について

・FreyaOSは、以下の機能を提供する。
① プロセス　４－１項参照
② ファイルシステム　４－２項参照
③ インダイレクトライフ４－３項参照
④ Ｃ言語ライブラリ　４－４項参照

2.) FreyaOS の制限事項

・FreyaOS上のプログラミングを行う場合、ユーザは以下の点に留意しなければならない
① メーカはFreyaOS及びWonder Witchパッケージに含まれるＣ言語を使用した開発を推奨しており、それ以外での開発環境はサポート対象外としている。
② FreyaOSは、プロセス制御のため全てのメモリー空間を管理しており、ユーザが独自にアクセスすることは出来ない。メモリ管理は以下の様になってい

・ROM	Bank0～5	FreyaOS File System (384 KB)
	Bank6	FreyaOS (64KB)
	Bank7	FreyaBIOS (64KB)
・SRAM	Bank0	FreyaOS File System (64KB)
	Bank1	FreyaOS User Prosess 2 (64KB)
	Bank2	FreyaOS User Process 1 (64KB)
	Bank3	FreyaOS Process (64KB)

③ コンパイル時は、TEXTセグメントは'20000<h>', DATAセグメントを'10000<h>'にする必要がある。(Wonder Witchパッケージ付属のコンパイラでは設定済みである。)
④ ユーザはディスプレイモード※１に応じたスタートアップルーチン※２をリンクしなければならない(Wonder Witchパッケージ付属のコンパイラでは、例えば、LSC-C86の場合は'crt0jpn2.obj'「日本語２」のスタートアップルーチンがリンクされる設定となっている。）。ILはこの限りでない。４－３項
⑤ ユーザは、セグメントグループ'DGRORP'の先頭にプロセス管理用構造体'ProcContext'※３を配置する必要がある。通常は、スタートアップルーチンで定義されている。

※１　３－３項「画面」参照
※２　本項7.)「スタートアップルーチンについて」を参照
※３　４－１項「プロセス」を参照

3.) FreyaOS の起動

・FreyaOSは、以下の流れに従って起動する。

① FreyaBIOSがFreyaOS起動部を呼び出す。

② カートリッジの状態をチェックし、未初期化あるいは内容が破壊されている場合に、セルフテストと初期化を行う。※1

③ 初期設定ファイルに従い、環境定数の設定及びシステムに組み込むIL(RunnableIL)を起動する。※2

④ ボタンの押下状態をチェックし、起動モードを決定する。※3

・シンプルモード('Y1'と'Y4'が同時に押されていた場合。)

cwd = /kern

ｼｪﾙ = デフォルトシェル(Meg)

・ノーマルモード('Y1'と'Y4'が同時に押されていない場合。)

cwd = /rom0

ｼｪﾙ = ユーザシェル'@shell'が'/rom0'ディレクトリにあればユーザシェルを起動、さもなくば、デフォルトシェル(Meg)を起動する。

⑤ 以降、ユーザ操作に従ってシェルが動作する。

※1 本項4.)「セルフテストと初期化」を参照

※2 本項5.)「初期設定ファイル(/rom0/init.rc)」を参照

※3 本項6.)「ユーザシェル(/rom0/@shell)」を参照

4.) セルフテストと初期化

・FreyaOSは、起動する際にSRAM上に記録されているはずの情報と、実際にSRAM上にある情報を比較して、カートリッジ内のデータの整合性をチェックする。もし、整合性が崩れている場合には、セルフテスト(自己診断プログラムの自動実行)と初期化を行
・セルフテストが行われ全て正常に終了した場合には、フラッシュメモリおよびSRAM上のファイルシステムを初期化する。記録されていた内容はBIOSとOSを除き全てクリアされる。

5.) 初期設定ファイル(/rom0/init.rc)

・FreyaOSは起動時に初期設定ファイルを読みこむ。このファイルにはシステム全体から利用される値を定義したり、システムに組みこむインダイレクトライブラリ(RunnableIL)を指定する。初期設定ファイル'init.rc'は'/rom0'ディレクトリに配置し、以下の文法にしたがって記述しなければならない。

① 環境定数定義

書式 : $var=<定義する内容>

説明 : システム全体から参照可能な環境定数varを定義する。'='の次の文字から、改行文字('\n')の直前までの内容を値として設定される。設定できる値の長さは、最大64バイトである。ライブラリ関数'getenv()'を用いて、特定の名前の環境定数に設定された値を参照できる。

② RunnableIL起動

書式 : @ilib [<引数> ...]

説明 : 指定された名前を持つインダイレクトライブラリを検索し、RunnableILとして実行する。インダイレクトライブラリは、/rom0、/kernの順に検索される。実行に失敗した場合には画面にメッセージを表示して一旦停止し、'A'ボタンを押すと続行する。

6.) ユーザシェル(/rom0/@shell)

・初期化処理が終わると、FreyaOSは標準ではデフォルトシェル(Meg)を起動するが、ユーザ独自のシェルを利用することも可能である。このようなシェルを「ユーザシェル」と呼ぶ。
・ユーザシェルを利用するには、起動したいシェル'/@shell'を、'/rom0'ディレクトリに配置する。
・初期設定ファイルを編集したり、ユーザシェルを配置すると、FreyaOSをうまく起動できなくなる場合がある。このような場合のために、初期設定ファイルの読み込みとユーザシェルの使用をスキップするモードが用意されている。これが「シンプルモー
・起動時にY1ボタンとY4ボタンを両方押しておくと、FreyaOSはシンプルモードで起動し、デフォルトシェル(MEG)が利用できる状態となる。
・シェルは、RunnableILとして実装する。

7.) スタートアップルーチンについて

・スタートアップルーチンでは主に以下の処理を行っている。※1
<"ProcIL"の"_load"関数及び、"_exec"関数にて呼ばれる。>
① 'DGRORP'の内容をSRAMに転送する。
② 呼び出し元に戻る。('retf'命令)
<"ProcIL"の"_run"関数及び、"_exec"関数にて呼ばれる。>
③ DS,ESレジスタに、SRAMのセグメント(1000h)を代入する。
④ スクロール量を全て０(VRAMの左上)に設定する。
⑤ ウィンドウサイズ全てVRAMエリア全体（(0,0)-(255,255)）に設定する。
⑥ スクリーン２をテキストスクリーンに設定する。
⑦ テキストモードを設定する。(ディスプレイモードで異なる。３－３項参照)
⑧ ＶＲＡＭを設定する。（メーカ未公開BIOSの為、詳細不明）
⑨ スプライトの表示を全てoffにする。('SPRITRE_SET_RANGE'で行われる。)
⑩ スクリーンの表示設定をする。(ディスプレイモードで異なる。３－３項参照)
⑪ サブルーチン'_main'(Ｃ言語のmain()関数)を呼び出す。
⑫ INT 10h(Exit)の呼び出し

※１ Wonder Witchパッケージ中CD-ROM付属の'crt0.asm'の解析結果である。
プロセスと関連性が高いため４－１項と併せて参考にすること。

４－１．プロセス　　*仕様調査中*

・FreyaOSでは、プログラムの単位を「プロセス」と定義している。FreyaOS自身もシステム起動時に実行される特別な「プロセス」として管理する。ユーザプログラムも例外ではない。
・プロセスは、互いに独立な記憶領域と実行状態を持ち、以下に説明する。
・FreyaOSではプロセスの実行状態を退避・復元する機能が用意されている。実行状態を退避する機能を「サスペンド」、退避されている実行状態を復元する機能を「リジューム」と呼ぶ。これにより以下の様なことが可能となる。　**（ー実行状態）**

① 通常の方法でプロセスを実行する場合、あとでリジュームできるようにそれまでに実行中だったプロセスをサスペンドできる。これにより、プロセスは他のプロセスの実効状態に影響を与えずに動作することができる。

② サスペンド機能を用いて電源を切る直前に実行を中断して状態を退避しておけば、再び電源を投入したときに状態を復元し、中断していたところから実行を再開することができる。*（スタックはどこに待避している？）*

・FreyaOSは、プロセス制御ブロック(Process Control Block 以後'PCB'と略す)を３つ確保しており、FreyaOS用に１つ、ユーザプロセス用に２つ割り振られている。PCBは各プロセス毎に一つ確保され、以下の情報を管理している。　**（ー記憶領域）**

① status　プロセスの現在の状態
0 : PCBはまだ使用されていない
1 : ロード中
2 : 実行中
3 : サスペンド中

② exit_code　プロセスの処理結果を表すコード
exit()関数を用いて、プロセスを終了した場合に設定される。
0 : 正常終了
-1 : ロードできない。

③ プロセス管理用構造体'ProcContext'※１。

注記１　①～②項は、FreyaOS上に確保さる。
注記２　③項の構造体は、SRAMの先頭に確保され、すなわち、ユーザプログラムの'DGRORP'の先頭に構造体が存在する。通常はコンパイル時にリンクするスタートアップルーチン内に構造体が存在し、スタートアップルーチン及びOSがその構造体を設定する。又、残りのSRAMは'_heap'変数('sys/process.h'及びスタートアップルーチンで定義される)にその先頭アドレスが格納され

※１　本項2.)を参照。

1.) プロセス起動の流れ

・プロセスは以下の流れに従って起動する。
<"ProcIL"の"_load"関数及び、"_exec"関数にて実行される>

① PCB割り当て
プロセスにPCBを割り当てる。
PCBに空き領域がない場合には、プロセスの起動は失敗する。
ユーザプロセス用に2つ、OS用に1つ、併せて3つ確保されている。

② 起動するプロセスの"_load0"関数の呼び出し
起動するプロセスのこの"_load0"関数内で、初期化済みデータのロードが行われる。ユーザプロセス用データ領域(SRAM:容量64KB)に、初期化済みデータを転送(ロード)※1する。返値に"_run0"関数へのファーポインタが渡され

<"ProcIL"の"_run"関数及び、"_exec"関数にて実行される>

③ 親プロセス環境の引継ぎ
親プロセスが使用しているインダイレクトライブラリ群※2、カレントワーキングディレクトリ※3がPCBに渡される。

④ コマンドラインパラメータの設定
ユーザプロセス用データ領域の空き部分にコマンドラインパラメータを格納(転送)する。

⑤ ヒープ領域の設定
最終的に残ったユーザデータ領域の先頭アドレスを、ヒープ領域の先頭アドレスとして_heap変数(sys/process.hで定義)に格納します。

⑥ 起動するプロセスの"_run0"関数の呼び出し
起動するプロセスのこの"_load0"関数内で、LCD及びスタック等の設定等を行なわれ、main()関数を呼び出す。その後、INT 10h(Exit)が呼び出され、プロセスを終了する。（４章の7項を参照。）

※１ アセンブラレベルでの表現では、'DATA'セグメントをSRAMへ一括転送をする処理のことである。（Ｃ言語のS,Dモデルのプログラムでは、データを'DATA'セグメントに配置する。）スタートアップルーチンでは、次ページ2項の'ProcContext'構造体と'DATA'セグメントを含む'DGRORP'を一括転送している。

※２ 'ilibIL','procIL'のIL構造体のファーポインタが渡される。

※３ 相対パスの基準となるディレクトリ名。

2.) プロセス管理用構造体'ProcContext' について

通常はスタートアップルーチン及びFreyaOS本体が確保、設定を行うため、ユーザは設定を行う必要はない。この構造体は'DGRORP'の先頭(1000h:0000h)に確保されている。ユーザはこの領域を参照することができる。

①構造体定義（C言語）

```
typedef struct {
        char            _id[4];             /* mark: "LCC"/"TCC"/"DMC" */
        pid_t           _pid;               /* pid of process */
        pid_t           _ppid;              /* pid of parent process */
        int             _pcbid;             /* pcbid of process */
        int             _ppcbid;            /* parent's pcbid */
        IlibIL  far     *_ilib;             /* IlibIL for this process */
        ProcIL  far     *_proc;             /* ProcIL for this process */
        FS              _cwfs;              /* current working filesystem */
        char            _currentdir[64];    /* current working directory */
        char    near    *near *_argv;       /* execution arguments */
        void    far     *_resource;         /* poiter to resource */
        void    near    *_heap;             /* poiter to free heap area */
} ProcContext;
```

②構造体説明

型	名前	内容
char	_id[4]	使用したコンパイラー※1
pid_t	_pid	プロセスID※2
pid_t	_ppid	親プロセスID※3
int	_pcbid	メーカ未公開情報
int	_ppcbid	メーカ未公開情報
IlibIL far *	_ilib	IlibIL構造体のあるアドレス※4
ProcIL far *	_proc	ProcIL構造体のあるアドレス※5
FS	_cwfs	メーカ未公開情報
char	_currentdir[64]	カレントディレクトリ※6
char near *	near *_argv	メーカ未公開情報
void far *	_resource	メーカ未公開情報
void near *	_heap	ヒープ領域※7

※1　使用したコンパイラーの種類。Ｃ言語によって引数・返値に用いられるレジスタが異なるため、その判別用に用意されてものを思われる。以下にその内容を示す。
・LSI-C86 'LCC',00h
・Turbo-C 'TCC',00h

※2　プロセスID(pid)
システム起動から現在までに起動されたプロセスのシリアルナンバー。ユーザプロセスは、1以上の正の整数が順に割り当てられ、正の整数の上限値に達した場合は、もう一度1から数え始める。OSには'0'が割り当てられる。

※3　親プロセスID(ppid)
このプロセスを起動したプロセス(親プロセス)のpidです。OSの場合は、-1となる。

※4　インダイレクトライブラリ操作IL(ilibIL)
親プロセスから引き継いだ、インライブラリ(以後'IL'と略す)操作のためのIL構造体型のあるファーアドレス。詳細は４－３－１項「IlibIL」を参照。また、ILについては、４－３項を参照。

※5　プロセス制御IL(procIL)
親プロセスから引き継いだ、プロセス制御のためのIL構造体型のあるファーアドレス。詳細は４－３－２項「ProcIL」を参照。また、ILについては、４－３項を参照。

※6　カレントワーキングディレクトリ
親プロセスから引き継いだ、相対パスの基点となるディレクトリである。この内容は、Ｃ言語ライブラリの'chdir()'関数を用いて変更することができ

※7　ヒープ領域
最終的に残ったユーザデータ領域の先頭アドレスを、ヒープ領域の先頭アドレスとして_heap変数('sys/process.h'及びスタートアップルーチンで定義)に格納します。

４－２．ファイルシステム

・FreyaOSでは、ファイルシステムを提供している。以下にその機能を示す。

① ディレクトリ機能を有す。詳細は表４－２－１を参照。FreyaOSver1.3では、ユーザが新たにディレクトリを作成することはできないが、将来の機能として現行のディレクトリの下に新たなディレクトリを作成できるような拡張性を有している。

② ブロック単位(1ブロック128Byte)によるメモリ管理

③ ファイルエントリー※１によるファイル情報の管理

④ X-modemを用いたファイルの送受信。

⑤ '/rom0'ディレクトリのみC言語ライブラリの'mmap()'関数にてファイルのある実アドレスを得ることができる。

※１ 本項1.)「ファイルエントリーについて」を参照

表４－２－１ ディレクトリについて

名称	位置 (容量)	内容
/ram0	SRAM (64KByte)	ソフトファイルシステム※1
/rom0	ROM (384KByte)	ハードファイルシステム※2
/kern	ROM (64KByte)	カーネルファイルシステム※3

※１ ソフトファイルシステム
SRAMを用いたファイルシステムである。
最大で64個のファイルを格納することができます。
主に、セーブデータや一時的使用するファイルを格納することを目的とする。
"/ram0'に格納されたプログラムを実行することはできません。

※２ ハードファイルシステム
フラッシュメモリを用いたファイルシステムである。
最大で128個のファイルを格納することができる。
主に、インダイレクトライブラリやプログラムを格納することを目的とする。
'/rom0'に格納されているファイルは、'mmap()'関数を用いてアドレスを得て、直接アクセスすることが可能である。

※３ カーネルファイルシステム
OS内部に組みこまれたファイルシステムである。
主に、デフォルトシェル(MEG)や各種ストリームデバイスのドライバが格納されている。
ユーザは'/kern'にファイルを追加できない。

1.) ファイルエントリーについて

・ファイルエントリーはファイル１つに付き以下の計64Byteの情報を持つ。

①構造体定義（Ｃ言語）

```
typedef struct stat {
        char    name[16];       /* ファイル名 */
        char    info[24];       /* ファイル情報 */
        char    far *loc;       /* ファイルの記録位置 */
        long    len;            /* ファイルの長さ */
        int     count;          /* ブロック数 */
        fmode_t mode;           /* ファイルモード */
        time_t  mtime;          /* 最終更新日時 */
        long    appid;          /* ファイルの種類 */
        char    far *resource;  /* リソースポインタ */
} fent_t, far *FS;
```

②構造体説明

名前	内容
name[24]	ファイル名。半角16文字。※1
info[24]	ファイルの説明文。半角24文字。
far *loc	実際のデータ・プログラムのあるファーポインタ。
len	ファイルの大きさ。1byte単位
count	ファイルが使用しているブロック数。128byte単位。
mode	詳細は、表４－２－２参照。
mtime	最終的な更新日時。詳細は、表４－２－３参照。
appid	現在未使用。将来の拡張機能。
fer *resource	ファイル末尾に付加されているリソース群へのポインタ。※2

※１　　ILの場合は、ファイル名の先頭を'@'にする必要がある。
※２　　詳細不明　メーカ問い合わせ中

表４－２－２　ファイルモードについて

数値	名称	内容
0001h	x	実行形式ファイル
0002h	w	書き込み可
0004h	r	読み込み可
0008h	m	mmap()の使用を禁止
0010h	s	ストリームILファイル
0020h	i	ILファイル
0040h	l	シンボリックリンク(現在未使用)
0080h	d	ディレクトリ(現在未使用)

表４－２－３　最終更新日時について

ビット	内容
bit 25 - 31	年(2000年を基準とした相対値)
bit 20 - 24	月
bit 15 - 19	日
bit 11 - 14	時間
bit 5 - 10	分
bit 0 - 4	秒(2秒単位)

４－３．インダイレクトライブラリ

・FrayaOSでは、ユーザプログラムと動的に結合し機能を呼び出すライブラリ形態を提供しており、これを「インダイレクトライブラリ(略称IL)」と称す。
・インダイレクトライブラリ(以後ILとする)は以下の特徴を有す。
　①　実行時に動的に関数（サブルーチン）のアドレスを解決する。
　②　機能セットをまとめて構造体定義した構造体を通じての関数呼び出し。
　③　IL入れ替えによる機能の拡張性。
　④　ファイル名レベルでのILの検索。
　⑤　複数のプログラムで利用するライブラリが重複してリンクせず、
　　メモリの節約に繋がる。
　⑥　大きなプログラムが小さなモジュールに分割できる。
・FrayaOS ver1.3では、標準で以下のILを実装している。
　①　IlibIL　　４－３－１項参照
　②　ProcIL　　４－３－２項参照
　③　FsIL　　４－３－３項参照
　④　StreamIL　　４－３－４項参照
　⑤　RunnableIL　　４－３－５項参照
・ユーザは、ILを作製することが出来る。（４－３－６項参照）

４－３－１．IlibIL

・インダイレクトライブラリ(IL)のアドレス解決を行うILである。
・ユーザプログラムからは、IL構造体型を指すポインタ定数'ilibIL('sys/process.h'及びスタートアップルーチンで定義される)を通じて参照することができる。(関数書式例 : '返値' = ilibIL->'関数名'('引数1','引数2',・・・))
・'/rom0'ディレクトリに'@ilib'ファイルを配置することで、ＯＳ起動時にシステムがユーザ定義の'ilibIL'を読み込む。（ユーザが作製した'IlibIL'をＯＳに組み込む。）

①構造体定義（Ｃ言語）

```
typedef struct {
        IL super;
        int (far *_open)(char far *ilname, IL far *ilbuf);
        int (far *_open_system)(char far *ilname, IL far *ilbuf);
} IlibIL;
```

②構造体説明

	int	ilibIL->_open(char far *ilname,IL far *ilbuf);	
	引数		
	char far *	ilname	ILのファイル名
	IL far *	ilbuf	IL構造体型を配置したいアドレス
	返値		
	int		エラーコード
	処理		
	ユーザ定義のILのアドレス解決を行う。実際の処理は、ilbufの示すアドレスにファーポインタを実際のメモリー空間上のアドレスに設定し直したIL構造体型を配置することを行う。 '/rom0','/kern'ディレクトリの順にファイルを検索すする。 エラーコードは以下の通り。 0000h　E_FS_SUCCESS　正常終了 0000h以外　異常終了		
	int	ilibIL->_open_system(char far *ilname,IL far *ilbuf);	
	引数		
	char far *	ilname	ILのファイル名
	IL far *	ilbuf	IL構造体型を配置したいアドレス
	返値		
	int		エラーコード
	処理		
	ユーザ定義のILのアドレス解決を行う。実際の処理は、ilbufの示すアドレスにファーポインタを実際のメモリー空間上のアドレスに設定し直したIL構造体型を配置することを行う。 /kern'ディレクトリのみにファイルを検索すする。 エラーコードは以下の通り。 0000h　E_FS_SUCCESS　正常終了 0000h以外　異常終了		

４－３－２．ProcIL

・プロセスの起動、終了、サスペンド、リジュームを制御するILである。
・ユーザプログラムからは、IL構造体型を指すポインタ定数'procIL('sys/process.h'及びスタートアップルーチンで定義される)を通じて参照することができる。(関数書式例 : '返値' = ilibIL->'関数名'('引数1','引数2',・・・))
・'/rom0'ディレクトリに'@proc'ファイルを配置することで、ＯＳ起動時にシステムがユーザ定義の'ProcIL'を読み込む。（ユーザが作製した'ProcIL'をＯＳに組み込む。）

①構造体定義（Ｃ言語）

```
typedef struct {
IL super;
        void far *(far *_load)(char far *command);
        int (far *_run)(void far *entry, int argc, char far * far *argv);
        int (far *_exec)(char far *command, int argc, char far * far *argv);
        void (far *_exit)(int exitcode);
        void (far *_yield)(void);
        int (far *_suspend)(int pcbid);
        void (far *_resume)(int pcbid);
        int (far *_swap)(int pcbid);
/*      int (far *_kill)(child);  */
} ProcIL;
```

②構造体説明

void far *	procIL->_load(char far *command);		
引数			
	char far *	command	起動したいプロセスのパス及びファイル名
返値			
	void far *		起動したいプロセスの"_run0"関数のアドレス
処理			
	指定したプロセスにPCBを割り当て、初期化済みデータをユーザプロセス用領域に転送される。		
int	procIL->_run(void far *entry,int argc,char far * far *argv);		
引数			
	void far*	entry	起動するプロセスの"_run0"関数のアドレス
	int	args	コマンドラインの文字数
	char far *	far *argv	コマンドラインの文字列
返値			
	int		起動したプロセスの返値
処理			
	子プロセスを起動する。		

構造体説明

int	procIL->_exec(char far *command, int argc, char far * far *argv);
引数	
	char far* command 起動したいプロセスのパス及びファイル名 int args コマンドラインの文字数 char far * far *argv コマンドラインの文字列
返値	
	int 起動したプロセスの返値
処理	
	procIL->_load()関数、procIL->_run()関数を呼び出す。
void	procIL->_exit(int exitcode);
引数	
	int exitcode 親プロセスに渡す返値
返値	
	無し
処理	
	プロセスを終了する。 int 10h(Exit)が呼ばれる。exit()関数や、main()からの復帰によって暗黙のうちに呼ばれる。
void	procIL->_yield(void);
引数	
	無し
返値	
	無し
処理	
	現在のプロセスを終了し、サスペンド中のユーザプロセスをリジュームする？

構造体説明

int	procIL->_suspend(int pcbid);
引数	
	int　pcbid　サスペンドしたいプロセスのpcb番号
返値	
	int　***エラーコード***
処理	
	指定したプロセスをサスペンドする。 ***エラーコードは以下の通り。*** ***0000h　E_FS_SUCCESS　正常終了*** ***0000h 以外　異常終了***
void	procIL->__resume(int pcbid);
引数	
	int　pcbid　リジュームしたいプロセスのpcb番号
返値	
	無し
処理	
	指定したプロセスをリジュームする。
int	procIL->_swap(int pcbid);
引数	
	int　pcbid　リジュームしたいプロセスのpcb番号
返値	
	int　***エラーコード***
処理	
	現在実行中のプロセスをサスペンドし、指定したプロセスをリジュームする。 ***エラーコードは以下の通り。*** ***0000h　E_FS_SUCCESS　正常終了*** ***0000h 以外　異常終了***

４－３－３．FsIL

・記録領域、記憶媒体に応じて、ファイルシステムに関連する種種の機能を提供するILである。
・ファイル関連のライブラリ関数(open()、close()、read()、write()など)(４－４項参照)が、内部的に使用するもので、ユーザプログラムが直接使用することはできない。
・各種ファイルシステムに対応したものが内蔵されている。

①構造体定義（Ｃ言語）

```
typedef struct _FsIL {
        IL super;
        fent_t far *(far *_entries)(FS fs);
        int (far *_n_entries)(FS fs);
        int (far *_getent)(FS fs, int n, fent_t far *fep);
        int (far *_findent)(FS fs, char far *fname, fent_t far *fep);
        void far *(far *_mmap)(FS fs, char far *fname);
        int (far *_open)(FS fs, char far *fname, int mode, int perms);
        int (far *_close)(int fd);
        int (far *_read)(int fd, char far *buf, int len);
        int (far *_write)(int fd, char far *buf, int len);
        long (far *_lseek)(int fd, long offset, int origin);
        int (far *_chmod)(FS fs, char far *fname, int mode);
        int (far *_freeze)(FS fs, char far *fname);
        int (far *_melt)(FS fs, char far *fname);
        int (far *_creat)(FS fs, fent_t far *fep);
        int (far *_unlink)(FS fs, char far *fname);
        int (far *_newfs)(FS fs);
        int (far *_defrag)(FS fs);
        unsigned long (far *_space)(FS fs);
} FsIL;
```

②構造体説明

・メーカ未公開情報である。

４－３－４．StreamIL

・ファイル操作を通じてアクセスされる、ストリームデバイスの機能を提供するILである
・FsIL群が内部的に使用するILで、ユーザプログラムが直接使用することはできない。
・各種ファイルシステム、通信回線、仮想キーボード、仮想コンソール用のものが内蔵されている。

①構造体定義（Ｃ言語）

```
typedef struct _fhandle_t far *fhandle_p;
typedef struct _StreamIL {
        IL super;
        int (far *_open)(fhandle_p f);
        int (far *_close)(fhandle_p f);
        int (far *_read)(fhandle_p f, char far *buf, int len);
        int (far *_write)(fhandle_p f, char far *buf, int len);
} StreamIL;
```

②構造体説明

・メーカ未公開情報である。

４－３－５．RunnableIL

・通常のプロセスと同じインターフェイスを持ち、ある種のライトウェイトプロセス的に利用されるILである。
・シェルはRunnableILとして実装する。

①構造体定義（Ｃ言語）

```
typedef struct {
        IL super;
        int (far *_exec)(int argc, char far *argv[]);
} RunnableIL;
```

②構造体説明
　通常のユーザ・プロセスと同等のコードを記述する。

４－３－６．ユーザILについて

・ユーザはILを作製することが出来る。
・ユーザがILを作製する際、以下の点に留意しなければならない。

① ファイルの先頭にはIL構造体型※1と同等の内容を配置しなければならない。
② ILは、IL構造体※2を用意しなければならない。
③ ILは、ILinfo構造体※3及び、その構造体内のポインターが示す先にデータを用意しなければならない。
④ Ｃ言語から呼ばれるため、引数はスタックに格納される。※4
⑤ Ｃ言語から呼ばれるため、返値は指定のレジスタに格納する必要がある。※4また返り値以外のレジスタは保持しなければならない。
⑥ 通常のスタートアップルーチンをコンパイル時にリンクしてはいけない。
⑦ DSレジスタの値は変更されず、呼び出し時の数値のままである。
Ｃ言語で、作製したILのセグメントグループ'DGROUP'を参照したい場合には、専用スタートアップルーチン'c0ilib.obj'をリンクし、その中の'il_get_ds()'関数を用いるなどしてDSレジスタを設定する必要がある。
⑧ ルーチンは、'callf'命令で呼ばれるため'retf'命令にて戻る必要がある。
Ｃ言語の場合は、far関数として定義すれば良い。
⑨ ILの機能群はＣ言語のライブラリで提供されるため、提供する関数群をメンバーとして持つIL構造体型※1を定義し、この定義を含むＣ言語によるヘッダーファイルを作成しなければならない。
⑩ ILは、IlibILの提供する関数にて開くときにアドレスの解決を行う。このため、IL構造体型※1，IL構造体※2，ILinfo構造体※3等で定義されるファーポインタのセグメントアドレス値は'0000h'とすれば良い。
⑪ ILは、ILinfo構造体※3のあるファーポインタを得る'_get_info()'関数を用意しなければならない。この関数はＣ言語レベルで、'far *_get_info();'と定義されている。

１．） IL構造体型について

①構造体定義（Ｃ言語）

```
typedef struct {
        IL      super;
        void    (far *UserFunction1)();          /*ユーザファンクション*/
        int     (far *UserFunction2)(int x);     /*左の2つは例である。*/
} UserIL;                                        /*ILの名前とする。*/
```

②　構造体説明

型	名前	内容
IL	super	IL構造体※２
返り値の型１	far *関数名１	関数のあるアドレス１
返り値の型２	far *関数名２	関数のあるアドレス２
（以後、関数の数だけ以下の定義が続く）		

２．） IL構造体について

①構造体定義（Ｃ言語）

```
typedef struct {
   void far *link_pos;
   int n_methods;
   ILinfo far *(far *_get_info)(void);
} IL;
```

②　構造体説明

型	名前	内容
void far *	link_pos	未使用
int	n_methods	ファンクションの数。※５
ILinfo far *	far *_get_info	_get_info()関数のある場所。

３．）ILinfo構造体について

①構造体定義（Ｃ言語）

```
typedef struct {
   char far *className;
   char far *name;
   char far *version;
   char far *description;
   char far * far *depends;
} ILinfo;
```

②構造体説明

型	名前	内容
char far *	className	未使用
char far *	name	ILの名称
char far *	version	ILのバージョン
char far *	description	未使用
char far *	far *depends	未使用

※１　IL構造体型については、本項１．を参照のこと。
※２　IL構造体については、本項２．を参照のこと。
※３　ILinfo構造体については、本項３．を参照のこと。
※４　Wandwer Witch付属のコンパイラは以下の仕様で引数、返値がセットされる。
LSI-C86 for Wonder Witchについても、関数については同様である。
(インラインアセンブラ関数は、LSI-C86試食版,製品版と同様の仕様である。)

●引数	char	word型でスタックに格納。
	int , pointer	word型でスタックに格納。
	long , far pointer	dword型でスタックに格納。
●返値	char	AX(AHは00hにする。)
	int , pointer	AX
	long , far pointer	DXAX(DXを上位16bitとする)

※５　'_get_info'関数を含む数量

４－４．ライブラリ

・FreyaOSが提供する機能は、Ｃ言語ライブラリによって提供される。
・ライブラリは、MicroSoft社のlibファイルと同一形式のファイルであるため、EXTRN,PROTO疑似命令を使用することで、Microsoft Macro Assembler等のアセンブラからでも呼び出すことが出来る。
・アセンブラから呼び出す場合、Ｃ言語コンパイラでは関数ラベルの先頭に'_'を付けるので、例えば'fopen()'関数ならば、'call _fopen'と呼ぶ必要がある。※1
・引数はスタックに格納する。格納する順番は、関数の最後の引数から順次格納する。格納した分のスタックは、呼び出したプログラムが元に戻す必要がある※１。
書式は以下の通り。

char型	2byte	(上位8bitは00hとする。)
int型	2byte	
doble型	4byte	(上位,下位の順番でpushする。)

・返り値は、レジスタに格納される。書式は以下の通り。

char型	ax	(ahは不定)
int型	ax	
doble型	dx:ax	(dx:上位16bit / ax:下位16bit)

・S-model※２なので、'near call'で関数を呼ぶ。
・C言語用の関数であるため、AX,BX,DXレジスタの内容は破壊される。
・ライブラリは、'libww.lib'及び、'libwwc.lib'で提供される。それぞれの関数は、Wonder Witch付属のDeveloper's Manualを参照。本書では省略した。

※１　関数の呼び出し例

```
;       FILE far  *_fopen(const char far *fname,const char far *mode);
;       引数
;                 ds:bx     mode      offset address
;                 ds:dx     fname     offset address
;       返値
;                 es:ax     FILE構造体のファーアドレス
        EXTURN  _fopen                ;外部関数定義
fopen   proc near                     ;
        push    bx                    ;
        push    cx                    ;
        push    dx                    ;レジスタ保存
        push    ds                    ;
        push    bx                    ;far *mode
        push    ds                    ;
        push    dx                    ;far *fname
        call    _fopen                ;関数呼び出し
        add     sp,+8                 ;スタックを戻す
        mov     es,dx                 ;es←dx(Segment address)
        pop     dx                    ;レジスタ復帰
        pop     cx                    ;
        pop     bx                    ;
        ret                           ;
fopen   endp                          ;
```

※2　LSI-C86でのコンパイルのモード。
　　プログラムは同一のセグメントに存在し、'near call'で関数が呼ばれる。

５．添付資料

本書製作に当たって、作製したプログラムソースリスト及び、サンプル等をフロッピーディスクに添付した。以下にそのディレクトリと内容を記す。

・include.asm¥　　（作製　　渡部 篤史）

アセンブラで FreyaBIOS 及び'libwwc.lib'の定数、マクロ等を定義したファイルです。
定義内容を読んで理解できる方のみお使い下さい。私がプログラムを作成する際、定義する手間を省くために作った定義ファイルです。

・sample¥il_test¥　　（作製　　渡部 篤史）

アセンブラで書かれたユーザ IL を実装し、C言語からその関数を呼び出す実験の為に作製したプログラムです。ユーザ IL を作製する際に参考にして下さい。

・その他、プログラム等の投稿があれば、添付致します。

６．後書き

本書執筆にあたって、プログラムを作って確認するという事をしていない箇所がまだ残っているため、間違えた説明をしている箇所があるかもしれません。順次確認中ですが、もし、間違えた説明をしている箇所などを発見なされた方は、お手数ですが私まで連絡頂けると有り難いです。また、説明が不足している、理解できない表現がある等の点もご教授頂けると幸いです。本業が技術者のため、修正する時間が確保できるかは難しいところですが、できる限りの修正はしていきたいと思います。

連絡先

●インターネット

・Digitals
UHL　http://www.digitalis.jp/　　（まだ完成していません。）
・私個人のホームページアドレス及び、メールアドレス。
UHL　http://homepage1.nifty.com/~sha_w
e-mail sha_w@nifty.com

●草の根ＢＢＳ

ＢＢＳ名	ＩＤ	ハンドルネーム
FSPNET	SYSOP	Ｓ．Ｗ.
PMDBBS	SHA_W	Ｓ．Ｗ.
東京 BBS　（閉鎖しました。）		
ZOB Station		Ｓ．Ｗ.

7．著者紹介・改訂履歴

著者略歴

渡部 篤史

昭和 54　　2 月 4 日生
平成 11　　東京都立工業高等専門学校 機械工学科卒業
　　半導体メーカに就職 半導体技術本部
平成 12　　同人サークル Digitalis に参加

主な作品

「蛸 Project」（都立高専エレクトロニクス研究部）
「Fm Sound Player (FSP)」（個人）
「Wonder Swan Total Sound Driver(WTD)」（Digitalis）

Wonder Witch Technical Manual

2002 年 1 月 24 日　　中途発表

発 行　Digitalis
発行者 大串 公徳
作 者　渡部 篤史

定価未定

※万一、落丁・乱丁の際にはお取り替えしいたします。