MR350 MKII

Data Collection Terminal

Programming Reference Manual

ユニテック・ジャパン（株）

2002.2 初版

(原典: September 2001 V1.2)

目⟩⟩⟩⟩⟩次

はじめに》

本マニュアルは、MR350 MKII とホストコンピュータもしくは PC 上でアプリケーション・プログラムの開発をされようとする皆様のためのハンドブックです。本書は I/O ファンクションコール、DOS マネージャ・ファンクションコール、ファイル・マネージャ・ファンクション・コール、そしてホスト ESC コマンドについて説明しています。

第1章)システム・カーネル)

本章は MR350MKII のシステム・カーネルを紹介しています。システム・カーネルは、アプリケーション・プログラミング・インターフェースをサービスする 6 つのサブシステム、キーボード入力、LCD 表示、通信 I/O、リアルタイム・クロック、リレー出力/デジタル入力、そしてバーコード/ワンド/磁気ストライプ/近接リーダに分かれています。

1.1)) アプリケーション・プログラミング・インターフェース)

MR350 MKII のカーネルは三つの基本モジュール、デバイスドライバ、ファイルマネージャと DOS マネージャを含んでいます。プログラマは PC DOS 環境とほとんど同じように、これらの使用可能なファンクションを呼び出すことによってアプリケーション・プログラムを設計することができます。

ターミナルの ROM ベースのオペレーティング・システムは、エミュレートされた MS-DOS ファンクションコールを提供しています。呼び出しとパラメータを与える表記は MS-DOS のものと同じです。ターミナル・サブシステム、I/O インターフェース、DOS マネージャとファイルマネージャによってサポートされるファンクションの詳細な説明は後の章で定義されています。

MR350 MKII で実行するソフトウェアは 16 ビットの Microsoft C 5.1 以降または、そして IBM PC マクロアセンブラのバージョン 1.0 以降を使用することによってプログラムすることができます。トランザクション・データはコンピュータと会話的に処理するか、あるいはファイルに保存されます。

注意: Microsoft の C を使用する場合:
プログラム実行領域が 64K 以下に割り当てられており(セクション 1.3 を参照)、そして 64K 以上のプログラムを実行させると、ランタイムエラー・メッセージ"Not enough space for environment"(環境に必要な容量がありません)が表示されます。この場合、メインプログラムに以下のステートメントを追加して下さい。

```
/* mypgm.c */
_setenvp()
{
}
main()
{.....
.....
}
```

そして、> LINK /NOE mypgm とリンクしてください。

1.2)) キーパッド・サブシステム))

キーパッド・サブシステムは、キーマトリックスをスキャンし、スキャンコードを関連するキーの値に変換し、プログラムで利用するためにその値をキーボード入力バッファに保存します。[SHIFT] キーはバッファに保存されないことに注意して下さい。これは関連するキー位置の英字と数字モードの区別をし、別なキーコードを与えるために使用されます。以下の表は各キーのキーの値を示しています。

キーの値の表))							
キー))	値))	キー))	値))	キー))	値))	キー))	値))
A))	41H))	O))	4FH))	[SP])	20H))	F1/(←))	86H))
B))	42H))	P))	50H))	0))	30H))	F2/(↴))	87H))
C))	43H))	Q))	51H))	1))	31H))	F3/(↲))	88H))
D))	44H))	R))	52H))	2))	32H))	F4/(→))	89H))
E))	45H))	S))	53H))	3))	33H))	F5/*))	8AH))
F))	46H))	T))	54H))	4))	34H))	F6/△))	8BH))
G))	47H))	U))	55H))	5))	35H))	F7/▽))	8CH))
H))	48H))	V))	56H))	6))	36H))	))	))
I))	49H))	W))	57H))	7))	37H))	))	))
J))	4AH))	X))	58H))	8))	38H))	))	))
K))	4BH))	Y))	59H))	9))	39H))	))	))
L))	4CH))	Z))	5AH))	[E])	0DH))	))	))
M))	4DH))	+))	2BH))	[C])	08H))	))	))
N))	4EH))	-))	2DH))	.))	2EH))	))	))

))

1.3) ディスプレィ・サブシステム)

このディスプレィ・サブシステムは、文字表示、文字列表示、カーソル位置セット、そしてスクリーン表示消去のインターフェース・ファンクションを提供します。ディスプレーの座標は以下のようになっています。

Min 行(Row)	Min 列(Col)	Max 行(Row)	Max 列(Col)
0	0	1	15

原点(0,0)は、常に左上角です。

1.4) 通信サブシステム)

MR350 MKII ターミナルの通信サブシステムは以下により構成されています:

1) ポイント・ツー・ポイント接続モード

2) ネットワーク処理のためのマルチポイント接続

1.4.1 ポイント・ツー・ポイント接続モード

RS-232 または RS-485 のいずれかは、**シリアルポート**としてセットされたときにポイント・ツー・ポイント・モードで使用することができます。各ポートは DOS コールによって入力または出力データについて設定することができます。ファイルを転送するには、Kermit サーバをユーザ・コマンド・メニューの"3) COM" オプションを選択するか、あるいは Ready プロンプトで"COM"をタイプすることによって起動することができ、そしてデータ接続を行うためにホスト側でも Kermit ユーティリティを実行しなければなりません。

1.4.2 マルチポイント・モード

RS-232 または RS485 のいずれかのポートは、**ホストポート**としてセットされたときに、マルチプロトコル(Unitech 社により定義される)によってマルチポイント・モードでも使用することができます。RS-485 ポートがマルチポイント・モードに指定された場合、チャンネル当たり 32 のターミナルをアクセスすることができ、一方 RS-232 ポートが選択された場合、アクセス可能なターミナルの数はホストコンピュータで使用可能な RS-232 ポートの数に制限されます。マルチポイント・ネットワークでデータ通信を行うために MR350 MKII にマルチポイント通信プロトコルも組み込まれています。

1.5)) リアルタイム・クロック・サブシステム)

このサブシステムは、MR350 MKII のシステムの日付と時刻をセットと読み取りをプログラムで行うことができます。

1.6)リレー出力とデジタル入力サブシステム)

MR350 MKII はデジタル信号入力と出力コントロールのために二つのコンタクト・リレー・ポートと 4 つのフォトカップル入力ポートを持っており、ここでピン#11/12 とピン#13/14 はジャンパ J1 から J6 の設定によって RS-232、そしてバーコード・スキャナポートに割り当てることもできます(MR350 MKII Technical Reference マニュアルをご覧下さい)。

1.7)) バーコード/磁気ストライプ/近接カード/ICC)

MR350 MKII は各種のリーダを接続するために**内部リーダ(Internal Reader)**と**外部リーダ(External Reader)**の二つのポートを持っています。

外部リーダ・ポートは、バーコード・ワンド、CCD、レーザ・ダイオード・スキャナによってバーコードを読み取るために占有され、ターミナルは Code39、Code128、Codabar、Interleaved 2 of 5、UPC と EAN の読み取りをサポートしています。

注意:

CCD およびレーザ・ダイオード・スキャナは内部スキャナ・ポートを通して接続されたときにのみサポートされます。ターミナル・ブロック・ポート#3 がスキャナ・ポートとしてセットされた場合、バーコード・ワンドまたはスロット・リーダ、磁気ストライプ・リーダそして近接リーダがサポートされます。

内蔵のリーダ・ポートはバッヂ・リーダ・ポートとして、主にバーコード・スロット・リーダ、磁気ストライプ・リーダ、非接触カード・リーダ(ウェイガンド・インターフェース)、そしてスマート・カード(ICC)・リーダの接続として設計されています。ターミナルはシングル・トラック 1, 2, 3 の磁気ストライプ・カードをサポートしています。

1.8)) ポイント・ツー・ポイント・モードでプログラムをダウンロード)

シリアルポート・インターフェースを通してポイント・ツー・ポイント・モードでターミナルに接続し、そしてターミナルにプログラムをダウンロードするために以下にリストしたステップに従って下さい。

ステップ 1: 適当なケーブルで RS-232 インターフェースを通して MR350 MKII を PC に接続します。

ステップ 2: ユーザ・コマンド・メニューを呼び出すために[F5/*]を押します。

ステップ 3: **Kermit** サーバ・モードに入るために"3) COM"のオプションを選択します。

ステップ 4: Utility ディスケットを PC に挿入します。

ステップ 5: PC で Kermit を実行します。

ステップ 6: Utility プログラム 350TEST.EXE をダウンロードするために PC から MR350 MKII に送信コマンドを使用します。

MS-Kermit> SEND <filename>

PC のディスクから MR350 MKII の RAM 領域に実行形式ファィルをダウンロード(プログラム)

MS-Kermit> GET <filename>

MR350 MKII の RAM 領域(データ)から PC のディスクにデータファイルを送信

MS-Kermit> REMOTE DIR

MR350 MKII に保存されているすべてのファイルを表示する(プログラムとデータファイル)

MS-Kermit> REMOTE DEL <file name>

MR350 MKII のプログラムまたはデータを消去します。

ステップ 7: Kermit サーバ・モードを終了して Ready モードに戻るために、[SHIFT]と[F5/*]を一緒に押します。

ステップ 8: ダウンロードした実行可能なプログラムのステップに進むために、オプション"1) RUN"を選択し、そして[F7/▽]キーを押します。そして、350TEST.EXE を実行するために[E]を押すか、Ready プロンプトで直接ファイル名(例、350TEST)をタイプします。このプログラムはバーコード・データをスキャンして、MR350 MKII から PC にデータを送ることができます。

1.9)) マルチポイント・モードでプログラムをダウンロード)

UTILITY ディスクにあるサンプルプログラム、485COM.EXE は、マルチポイント・モード環境のテストのためのものです。マルチポイント・モードで各ターミナルがユニークなアドレス ID を割り当てられ、そして PC との通信パラメータが一致していることに注意し

て下さい。

ノート: 以下の方法はマルチポイント・モードを行うためにホストポートを使用します。

1) ホストポートが RS232 の場合、PC と MR350 MKII 間の RS232 ポートを直接接続します。

2) ホストポートが RS485 の場合、RS-485 インターフェース・カードまたは RS232/422 コンバータを PC にインストールします。そして RS-485 インターフェースから MR350 MKII に RS485 ポートを通してネットワークの配線をします(注: AWG 22 または 24 のツイストペア・ケーブルを使用して下さい)。

3) 各 MR350 MKII のアドレス ID を含む通信パラメータを正しくセットします。(標準値は、9600bps、パリティなし、8 データビット、1 ストップビット、アドレス ID'A'です。)

4) PC とすべてのターミナルの電源を入れます。

5) PC で 485COM.EXE のテストプログラムを実行します。

CRT スクリーンは以下のメッセージを表示します。

Terminal type 1>350/360 2>700/870/860:

MR350 を選択するために"1"をタイプします。

COM(1-4)?:

COM1 は"1"、COM2 は"2"をタイプします。

6) スクリーンは以下を表示します。

V2.1 COM2 Address: ESC=1 NAK=3 PARA=9600,1,8,NONE

0.Send	1.Poll	A.Stop	B.BarT	C.ComT	D.DIR	E.Del	F.FxeSize
f.Font	G.Memory	H.Reset	I.FxFile	J.Exist	K.Keypad	K.Kermit	L.Dnload
M.Time	N.Buzzer	O.Auto	P.Passwd	Q.UplMode	R.TrmID	T.TrmT	U.Upload
V.DEV_T	X.Exec	3.brk	5.ChgAdr	9.Loop	@.Modem	?.320	~.UPSoff
F1.Addr	F2.Comm_P	F3.Retry	F4.Disp	F5.Shell	F6.Pktsize:	Select:	

項目

0) 文字列をメッセージとして MR350 MKII に送る。
1) 各ターミナルからデータをポールする。
A) ウォーム・スタートはすべての接続されたターミナルがレディ・モードで、

そして以前に実行していたプログラムが停止していたことを意味します。

B) バーコード・シンボルの使用可/使用不可をセット

C) 通信コントロール・テーブルをセット

D) MR350 MKII の RAM ディスクにあるファイルをリモートで読む。

E) RAM ディスクにある指定したファイルをリモートで削除

F) 実行可能領域の RAM サイズを変更(MR350 MKII では使用できません。)

f) フォント・サイズの変更(MR350 MKII では使用できません。)

G) 接続されているすべてのターミナルの RAM サイズ、実行領域のサイズ、そして空き領域の合計を得ます。

H) コールド・スタートは、すべての接続されているターミナルのシステム・パラメータを工場出荷時の値に初期化することを意味しています。

I) 現在実行しているプログラムのファイル名を得る。

J) 指定したファイルがあるかどうかをチェックする。

K) キーパッドのロック/アンロック/部分的なロックをセット(MR350 MKII では使用できません)。

k) Kermit サーバ・モードに入る(MR350 MKII では使用できません)。

L) MR350 MKII にプログラムまたはデータファイルをダウンロードする。

M) 接続されたターミナルの日付と時刻をセット。

N) ビーパのボリュームをセット。

O) 実行可能なオブジェクト・プログラムを電源投入時に自動的に起動するようにセット。

Q) アップロード・ステータスを聞く(MR350 MKII では使用できません)。

R)⟫ターミナル ID を変更。⟫

T)⟫ターミナル・コントロール・テーブルをセット⟫

U) MR350 MKII からプログラムまたはデータファイルをアップロード。

V) デバイス・コントロール・テーブルをセット。

X) リモート・ランは、ターミナルであらかじめダウンロードされた実行可能なオブジェクト・プログラムを起動することを意味します。

3) 省電力をセット(MR350 MKII では使用できません)。

5) 接続されたターミナル・アドレスをセット。

9) ループバック・テスト

@) ターミナルのダンプとモデム・コントロール

?) MR320 の設定(MR350 MKII では使用できません)

~) UPS を使用不可にする。

F1) 通信に使用可能なターミナル・アドレスをセット

F2) PC の通信パラメータをセット

F3) タイムアウト/NAK リトライ/ACK の時間間隔をセット
F4) デバッグ・モード(受信と送信した全部のデータを表示)
F5) DOS シェルへ行く
F6) 通信パケット・サイズをセット(デバッグのみ)
[ESC] 485COM.EXE を終了して、DOS プロンプトに戻る

7) すべての接続されたターミナルまたはテストするあるターミナルにアドレスをキー入力するために項目の F1) を選択します。例えば、接続されているそれぞれのターミナルのアドレス A、B と C の場合、"ABC"をタイプします。

8) プログラム 350TEST.EXE をダウンロードするために項目 L) を選択します。この方法はすべての指定されたターミナルがダウンロードされるまで繰り返されます。

9) ステップ 7 で指定したすべてのターミナルのプログラムを起動するために、項目 X) を選択して、プログラム名 350TEST を入力します。

10) データの受け取りを始めるために項目 l) を選択します。PC のスクリーンはデータがなにも収集していないことを表す "..."が現れます。あるターミナルがバーコードラベルのスキャンによってデータの入力を開始したら、PC のスクリーンには以下が現れます。

A(nn): XXXXXX

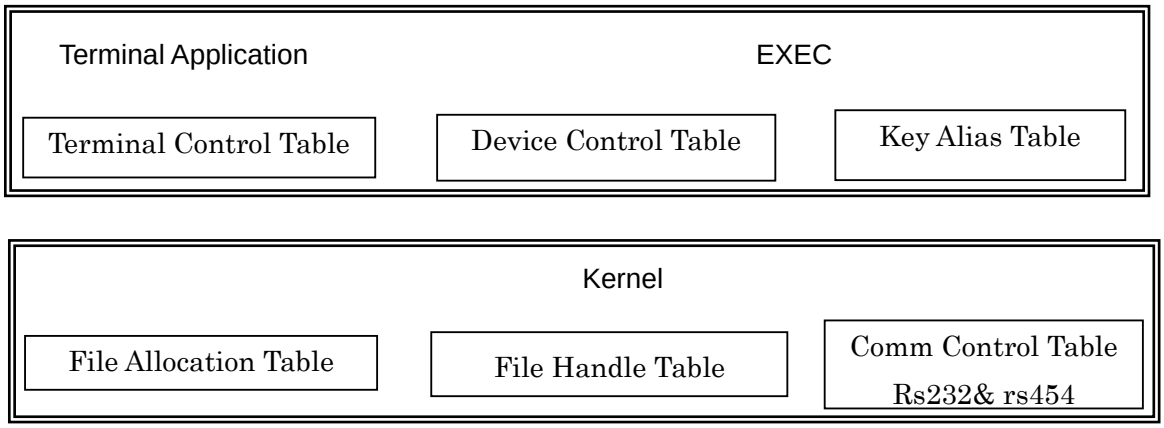
最初の文字はターミナル・アドレスを意味します。ここで、XXXXXX は接続されたバーコード入力デバイスまたは磁気ストラブ・リーダでスキャンされたデータです。そして、nn はデータ長です。

メッセージをターミナルに送るために項目 0) を選択します。PC スクリーンの指示に従ってキー入力されたかどうかには関係なく、文字パターンは **Application data: XXXXXX** の様にターミナルの LCD に表示されます。XXXXXX は PC のキーボードで入力した文字列です。

11) コールド・スタートまたはウォーム・スタートのテストを行うために H) と A)を選択することもできます。あるいは、現在のプログラムを終了するために **[ESC]** を押します。

第2章) データの構造)

MR350 MKII システムのコントロール・データ構造は以下の図に表されています。システムのカーネルは、ファイル・アロケーション・テーブル(FAT)、ファイル・ハンドル・テーブル(FHT)、通信コントロール・テーブル、デバイス・コントロール・テーブル、そしてキー・エリアス・テーブルを使用しています。以下のセクションはこれらの各コントロール・テーブルのそれぞれについて説明しています。説明は"typedef"の部分と"テーブルの標準値を含んでいます。



2.1)デバイス・コントロール・テーブル)

デバイス・コントロール・テーブルは、バーコード・スキャナポート、バッジ・リーダ・ポート、LCD ディスプレー、キーボード、そしてブザー出力情報を含む MR350 MKII の周辺機器の構成情報を含んでいます。バーコード・スキャナは、後で説明する別なデータ構造とバーコード・コントロール・テーブルによってコントロールされます。

2.2)) タイプの定義)

```
typedef struct {        BYTE scanner;
                        BYTE badge;
                        BYTE lcd_backlight;
                        BYTE buzzer;
                        BYTE keylock;
                        BYTE buzzer_volume;
        } DEV_CONFIG;
scanner:    'N'     = スキャナ・ポートを有効にする
            'F'     = スキャナ・ポートを無効にする(標準値)

badge:      'B'     = バーコード・スロット・リーダのためにバッヂ・ポートを有効(標準)
            'M'     = 磁気カード・リーダのためにバッヂ・ポートを有効にする
            'D'     = バッヂ・ポートを無効にする
```

lcd_backlight 'N'　= LCD のバックライトを ON にする
'F'　= LCD のバックライトを OFF にする(標準値)

buzzer:　**'N'**　= ブザーを ON にセット(標準値)
'F'　= ブザーを OFF にセット

Keylock:　**'N'**　= キーボードをアンロックにセット(標準値)
'K'　= キーボートをロックにセット
'P'　= キーボードの部分ロックをセット

buzzer_volume:　**'0'**　= ボリューム低(標準値)
'5'　= ボリューム中
'9'　= ボリューム高

2.3) バーコード・コントロール・テーブル)

MR350 MKII は以下のバーコード・シンボルを自動的に認識するデコード・ソフトウェアをサポートしています: Code 39、Code 39 Full ASCII、EAN-8、EAN-13、UPC-A、UPC-E、Code 128、Codabar および Interleaved 2 of 5

2.3.1 タイプの定義

```
typedef struct{        BYTE code39;
                       BYTE i2of5;
                       BYTE codabar;
                       BYTE ean_upc;
                       BYTE code128;
              } BARCODE_CONFIG;
```

code39:　**'N'**　= Code 39 のバーコード・デコード有効(標準値)
'F'　= Code 39 のバーコード・デコード無効

i2of5:　**'N'**　= Interleave 2 of 5 のバーコード・デコード有効(標準値)
'F'　= Interleave 2 of 5 のバーコード・デコード無効

codabar:　**'N'**　= Codabar のバーコード・デコード有効(標準値)
'F'　= Codabar のバーコード・デコード無効

ean_upc:　**'N'**　= EAN/UPC のバーコード・デコード有効(標準値)
'F'　= EAN/UPC のバーコード・デコード無効

code128:　**'N'**　= Code 128 のバーコード・デコード有効(標準値)
'F'　= Code 128 のバーコード・デコード無効

2.4) ホストポートの通信コントロール・テーブル)

通信コントロール・テーブルは MR350 MKII のホストポートを設定します。

通信コントロール・テーブルはホスト・システムと MR350 MKII 間のすべての通信パラメ

ータを指定します。キーパッド入力またはホストのコマンド・シーケンスを通してハード・リセット・コマンドが出された場合、標準の通信パラメータに戻されます。ホスト・システムはホスト・コマンド・シーケンスを出すことによってほとんどの MR350 MKII パラメータを設定することができます。ホスト・コマンド・シーケンスは本マニュアルにリストされています。

2.4.1 タイプの定義

```
typedef struct{     BYTE baud_rate;
                    BYTE stop_bit;
                    BYTE data_bit;
                    BYTE parity;
                    BYTE protocol;
                    BYTE address;
                    BYTR time-out;
            } COM_CONFIG;
```

MR350 MKII ターミナルは、ホストポートを経由してホストと通信します。通信の転送速度は 110 から 38.4K ボー(bps)にプログラムすることができます。

Baud_rate: '0' = 110 bits per seconds
'1' = 150
'2' = 300
'3' = 600
'4' = 1200
'5' = 2400
'6' = 4800
'7' = 9600(標準値)
'8' = 19200
'9' = 38400

stop_bit: **'1'** = 1 ストップ・ビット(標準値)
'2' = 2 ストップ・ビット

data_bit: '7' = 7 データ・ビット
'8' = 8 データ・ビット

parity: **'N'** = パリティなし(標準値)
'O' = 奇数パリティ
'E' = 偶数パリティ

protocol: **'M'** = マルチポイント(標準値)
'F' = プロトコルなし

address: **"A"** = マルチポイント・モードのターミナル・アドレス ID(標準値)
各 MR350 MKII はマルチポイントの環境で使用する場合はユニークな通信アドレスを指定しなければなりません。このアドレスはポーリングの機能を実行するためにホストあるいはコンセントレーターで使用されます。文字'A' – 'Y'と'0' – '6' が各ターミナルのアドレス ID を指定するために使用されます。

Time-out: **‘02’** = ポーリング・タイムアウト、2 サイクル(標準値)
‘02’ – ‘FF’ 16 進数フォーマット
この設定の値は通信のタイムアウトを指定します。MR350 MKII が指定した数のタイムアウト時間以内に応答を受け取らなかった場合、MR350 MKII は通信が成功せずに伝送が終了したものと見なします。タイムアウト値がゼロにセットされた場合、タイムアウトのチェックは MR350 MKII では行われません。

2.5)) ターミナル・コントロール・ケーブル(ホストポートのみについて有効))

ターミナル・コントロール・テーブルは、MR350 MKII の操作スイッチが”ターミナル・モード”にセットされている場合のみ関係します。他のすべての動作モードはターミナル・コントロール・テーブルを無視します。

2.5.1 タイプの定義

ターミナル・コントロール・テーブルは以下の typedef TERM_CONFIG によって定義されます。ひとつだけ TERM_CONFIG データ構造のインスタンスがあります。

```
Typedef struct      { char terminal_id[8];   /* terminal id */
                      BYTE online;
                      BYTE echo;
                      BYTE autolf        /* auto lF */
                      BYTE mode;
                      BYTE linepage; /* line or page block */
                      BYTE lineterm; /* line terminator */
                      BYTE pageterm; /* page terminator */
                    } TERM_CONFIG;
```

各 MR350 MKII “terminal” は ASCII 文字列によって区別されます。ターミナル識別文字列は 7 文字以内です。TERM_CONFIG テーブルの識別エントリーは C 言語で使われるように文字、ASCII_Z(hex 0)によってターミネートしなければなりません。

online: **‘R’** = Remote にセットし、データをホストポートに送信します。(標準値)
‘L’ = Local にセットし、送信しません。
echo: **‘N’** = 収集したデータを表示。
‘F’ = 収集したデータを表示しない。

上記の二つの変数、TERM_online と TERM_echo は、それぞれ送信と収集したデータの表示のコントロールに使用されます。TERM_online が **Remote** にセットされた場合、MR350 MKII はデータをホストに送信し、そうでなければ送信しません。TERM_echo が **echo** にセットされた場合、収集されたデータは MR350 MKII の LCD に表示され、そうでない場合は表示されません。

Autolf: ‘N’ = CR の後に LF の追加をしないことをセット
‘F’ = LF を追加するようにセット(標準値)

この変数は入力のスキャナ･デバイスから得られた CR にいつも LF 文字を追加することを MR350 MKII に指示します。

Mode: 'C' = Character モードにセット。
'B' = Block モードにセット(標準値)。
このパラメータは文字モードか、ブロック・モードのフリー・フォーマット操作のどちらかを指定します。前述のフォーム・キャッシュ操作は MR350 MKII がブロック・モードの場合にのみ適用可能です。

Linepage **'L'** = ライン・ブロック・モードにセット(標準値)
'P' = ページ・ブロック・モードにセット
'B' = ラインとページ・ブロック・モードの両方にセット
linepage パラメータは **mode** が'B'に指定された場合にのみ使用されます。

Lineterm: ライン・ブロック・モードの終端文字を指定します(標準値=null)。

Pageterm: ページ・ブロック・モードの終端文字を指定します(標準値=null)。

第3章) I/O) ファンクションコール)

MR350 MKII のオペレーティング・システムは、LCD ディスプレー、キーボード入力、近接/バーコード/磁気ストライプ入力、ブザー、セキュリティ・アラーム、フォトカップル入力、リレー出力、そして RS232 と RS485 のシリアルポート入力/出力をコントロールするために BIOS/DOS ファンクションをサポートしています。全体の C サンプル・プログラムは Utility ディスケットのライブラリ・ファイル”350LIB.C”に集められています。

3.1)) LCD) ディスプレー) INT10H)

00 スクリーンをクリア

```
エントリー・パラメータ:)  AH) =) 0)
戻り値:)  )  )  なし)

void) TL_clrscr())
{)
)) regs.h.ah=) 0;)
)) int86(0x10,) &regs,) &regs);)
})
)
```

01 カーソル・タイプをセット

```
エントリー・パラメータ:)  AH) =) 1)
))  )  )  AL) =) 1)  ;) ブロック・カーソルにセット)
))  )  )  ))))) 0)  ;) 下線カーソルにセット)
))  )  )  )  )  ) 3)  ;) カーソルをオフにセット)
戻り値:)  )  )  なし)
void) TL_cursor_type(int) status))
{)
)) regs.h.ah) =) 1;)
)) regs.h.al) =) (unsigned) char)status;)
)) int86(0x10,) &regs,&regs);)
})
```

02 カーソル位置セット

```
エントリー・パラメータ:  AH = 2
                         DH = 0 ～ 1       ;行
                         DL = 0 ～ 15      ;列
戻り値:                  なし

void TL_gotoxy(int x, int y);
{
  regs.h.ah = 2;
  regs.h.dh = (unsigned char)y;
  regs.h.dl = (unsigned char)x;
  int86(0x10, &regs, &regs);
}
```

03 カーソル位置を得る

```
エントリー・パラメータ:  AH = 3
戻り値:                  DH = 0 ～ 1       ;行
                         DL = 0 ～ 15      ;列

Void TL_getxy(int *x, int *y)
{
  regs.h.ah =3;
  int86(0x10, &regs, &regs);
  *y = regs.h.dh;
  *x = regs.h.dl;
}
```

04 スクリーンのスクロール

```
エントリー・パラメータ:  AH = 4
                         AH = 0   ;無効
                         AH = 1   ;有効

void TL_scroll(int status)
{
  regs.h.ah = 4;
```

```
  regs.h.al = (unsigned char)ststus;
  int86(0x10, &regs, &regs);
}
```

1A　LCD バックライトを有効/無効　INT21H

```
エントリー・パラメータ:　AH = 0x1A
                        BH = 0
                        AL = 0   ; 無効
                           1   ; 有効
戻り値:                 なし

void TL_backlight(int status)

{

  regs.h.ah = 0x1A;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 0;
  int86(0x21, &regs, &regs);
}
```

3.2　通信環境のセットアップ

MR350 MKII を通信環境におく前に、以下を決定しなければなりません。

(1) ホストポートとしてRS-422/485ポートあるいはRS-232ポートを割り付けるかどうか、そして他をシリアルポートとして割り付けるかどうか

(2) ホストポートの通信パラメータを設定。シリアルポートのプロトコルは常にないことに注意して下さい。

(3) 内部モデムインターフェースをインストールした場合、RS485 の代わりにモデムを COM1 にセットしなければなりません。

通信環境の工場出荷時の標準値は以下の通りです。

ホストポート:　RS-422/485、マルチポイント・プロトコル
シリアルポート: RS-232、プロトコル無し

いったんホストポートにプロトコルを割り付けたら、MR350 MKII のシステム・カーネル

は、選択したプロトコルに従って送信または受信データを自動的にパックまたはアンパックします。一方、シリアルポートはプロトコルがないので、文字ごとにデータを読むまたは書くために INT 34H (RS232)または INT 33H(RS-422/485)を使用しなければなりません。

注意:
ホストポート I/O サービスをコントロールしたい場合、ホストポートに対するプロトコルを”NONE”に指定し、そして RS232 または RS422/485 に対する I/O をコントロールするために INT34H または INT35H を使用しなければなりません。ポートに依存する適当なファンクションコールはホストポートとして割り当てられます(次の二つのセクションを参照)。

1C ホストポートとして COM1 または COM2 を選択

```
エントリー・パラメータ:   AH = 1C
                          BH = 0
                          AL = 1   ; ホストとしてCOM1を選択
                               2   ; ホストとしてCOM2を選択
戻り値:                   なし
```

注意:
COM ポートの一つがホストに指定されている間、他の一つはシステムによって自動的にシリアルポートとしてセットされています。

```
Void TC_select_host(int status)
{
  regs.h.ah = 0x1C;
  regs.h.al = (unsigned char)status;
    regs.h.bh = 0;
  int86(0x21, &regs, &regs);
}
```

1C ホストポートのプロトコルをセット

```
エントリー・パラメータ:   AH = 1C
                          BH = 1
                          AL = 2   ;マルチポイント(標準値)
                               3   ;プロトコル無し
戻り値:                   なし
```

```
void TC_protocol(int status)
{
  regs.h.ah = 0x1C;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 1;
  int86(0x21, &regs, &regs);
}
```

1C シリアルポートのフローコントロールをセット

システムはシリアルポートについて三つのハンドシェーキング・モード、XON/XOFF、CTS/RTS そして「無し」を提供しています。シリアルポートについてのフローコントロールのシステム標準値は「無し」です。CTS/RTS は RS-232 がシリアルポートに指定されている場合にのみ有効です。

```
エントリー・パラメータ:  AH = 1C
                         BH = 2
                         AL = 0   ; 無し
                              1   ; XON/XOFF
                              2   ; CTS/RTS
戻り値:                  なし
```

注意:
シリアルポートのハンドシェーキング・フローをコントロールしたい場合、フローコントロールを”NONE”にセットしなければなりません。

```
Void TC_flow_ctrl(int status)
{
  regs.h.ah = 0x1C;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 2;
   int86(0x21, &regs, &regs);
}
```

19　RS485 またはモデムとして COM1 ポートをセット

このファンクションコールは、RS485 シリアルポートあるいは内蔵モデムインターフェースをインストールしている場合はモデムとして COM1 ポートをセットするために使用します。通信のためにモデムを使用する前に、モデムとして COM1 ポートをセットしなければなりません。

エントリー・パラメータ:　AH = 19
　　　　　　AL = 0　;RS485としてセット
　　　　　　AL = 1　;モデムとしてセット
戻り値:　　なし

```
void TC_modem_port(int status)
{
  regs.h.ah = 0x19;
  regs.h.al = status;
  int86(0x21, &regs, &regs);
}
```

3.3　マルチポイント・プロトコル I/O のためのホストポート　(INT21H)

1C マルチポイント・アドレスのセットアップ

エントリー・パラメータ:　AH = 0x1C
　　　　　　BH = 06
　　　　　　AL = 'A'..'Y', '0'..'6'
戻り値:　　なし

```
void TC_set_address(char status)
{
  regs.h.ah = 0x1C;
  regs.h.al = ststus;
  regs.h.bh = 6;
  int86(0x21, &rergs, &regs);
}
```

1C ポーリングのタイムアウト時間をセット

```
エントリー・パラメータ:  AH =0x1C
                         BH =09
                         AL =0 – 255  ; タイムアウト・サイクルを
                                        80msとするタイムアウト時間
 戻り値:                 なし        ;;;;; AL=2の場合はタイムアウト時間160ms、
                                        AL=0の場合はタイムアウト無し
Void TC_time_out(int status)
{
  regs.h.ah =0x1C;
  regs.h.al =status;
  regs.h.bh =9;
  int86(0x21, &regs, &regs);
}
```

5F ホストポートを読む

```
エントリー・パラメータ:  AH =0x5F
戻り値:                  DS:DX =バッファ・ポインタ
                         AL =0    ; 出力完了
                         ;;;;1    ; データ無し
int TC_str_I(unsigned char *str, int wait)
{
  do {
    regs.h.ah =0x5F;
    segregs.ds =FP_SEG(str);
    regs.x.dx =FP_OFF(str);
    int86x(0x21, &regs, &regs, &segregs);
  ;;;;;} while(wait && regs.h.al)
  ;;; return(regs.h.al);
}
```

60 データ出力

```
エントリー・パラメータ:  AH = 0x60
                         DS:DX = バッファ・ポインタ
戻り値:                  AL = 0   ; 出力完了
                         AL = 1   ; バッファ・ビジー中

int TC_str_0(unsigned char *str, int wait)
{
    do {
        regs.h.ah = 0x60;
        segregs.ds = FP_SEG(str);
        regs.x.dx = FP_OFF(str);
        int86x(0x21, &regs, &regs, &segregs);
    } while(wait && regs.h.al);
    return(regs.h.al);
}
```

61 ポートがビジーかどうかをチェック

```
エントリー・パラメータ:  AH = 0x61
戻り値:                  AL = 0   ; ポートは有効
                         AL = 1   ; ポートはビジー

int TC_ready(int wait)
{
    int i;
    do {
        regs.h.ah = 0x61;
        int86(0x21, &regs, &regs);
    } while(wait && regs.h.al);
    return(regs.h.al);
}
```

3.4) RS-232 と RS-485 のためのシリアル I/O)

システムは、ポートがホストあるいはシリアルポートとして割り付けられてなくても RS-232 と RS-485 をシリアル入力/出力(文字モード I/O)を利用することができます。しかし、ポートがホストとして割り当てられている場合、そのアクティブ・プロトコルとして"NONE"を選択しなければなりません。

INT34H は RS-232 専用、そして INT33H は RS-484 専用で、両方のファンクションコールは、ホストとシリアルポートに対して働きます。

INT34H を使用する RS232 ポート・シリアル I/O

01 データ入力

```
エントリー・パラメータ:)  AH) =) 1)
戻り値:)  )  )  1)) 文字を受信した場合)
) )  )  )  )  AH) =) 0)
) )  )  )  )  AL) =) 文字データ)
) )  )  )  2)) 文字を受信しなかった場合)
) )  )  )  )  AH) =) 1)
) )  )  )  )  AL) =) 未定義)

Unsigned) char) TC_232_char_I())
{)
) ) regs.h.ah) =) 1;)
) ) int86(0x34,) &regs,) &regs);)
) ) if) (regs.h.ah) ==) 0);)
) ) ) ) return(regs,h.al);)
) ) return(255);)
})
)
```

02 データ出力

```
エントリー・パラメータ:)  AH) =) 2)
) )  )  )  AL) =) 文字データ)
戻り値:)  )  )  なし)

void) TC_232_char_O(unsigned) char) ch))
{)
) ) regs.h.ah) =) 2;)
```

```
    regs.h.al = ch;
    int86(0x34, &regs, &regs);
    }
)
```

03 RS-232 ポート有効

```
エントリー・パラメータ:   AH = 3
戻り値:                   なし

void TC_232_enable()
{
    regs.h.ah = 3;
    int86(0x34, &regs, &regs);
    }
)
```

04 RS232 ポート無効

```
エントリー・パラメータ:   AH = 4
戻り値:                   なし

void TC_232_disable()
{
    regs.h.ah = 4;
    int86(0x34, &regs, &regs);
    }
)
```

00 通信パラメータセット

```
エントリー・パラメータ:   AH = 0          BIT #
                                          76543210
                          AL = bit 0      xxxxxxx0   7データビット
                                          xxxxxxx1   8データビット
                          bit 1           xxxxxx0x   1ストップビット
                                          xxxxxx1x   2ストップビット
                          bit 2-3         xxxx00xx   パリティ無し
                                          xxxx01xx   奇数パリティ
                                          xxxx11xx   偶数パリティ
```

))))))) bit) 4-7) 0000xxxx)) 110bps)
)))))) 0001xxxx)) 150bps)
)))))) 0010xxxx)) 300bps)
)))))) 0011xxxx)) 600bps)
)))))) 0100xxxx)) 1200bps)
)))))) 0101xxxx)) 2400bps)
)))))) 0110xxxx)) 4800bps)
)))))) 0111xxxx)) 9600bps)
)))))) 1000xxxx)) 19200bps)
)))))) 1001xxxx)) 38400bps)
戻り値:))) なし)

```
void) TC_232_parameter(long) baud,) int) parity,) int) stop,) int) data))
{)
)) unsigned) char) cc=0;)
)) unsigned) int) i_baud;)
)
)) i_baud) =) (int)(baud) /) 10L);)
)) switch(i_baud);)
)) {)
)))) case) 11:cc=0x00;)) break;)
)))) case) 15:cc=0x10;)) break;)
)))) case) 30:cc=0x20;)) break;)
)))) case) 60:cc=0x30;)) break;)
)))) case) 120:cc=0x40;)) break;)
)))) case) 240:cc=0x50;)) break;)
)))) case) 480:cc=0x60;)) break;)
)))) case) 1920:cc=0x80;)) break;)
)))) case) 3840:cc=0x90;)) break;)
)))) default:cc=0x70;)) break;)
)) })
)) switch(parity))
)) {)
)))) case) 0:)) break;)
)))) case) 1:cc=cc|0x04;)) break;)
)))) case) 2:cc=cc|0x0c;)) break;)
)))) case) 3:cc=cc|0x08;)) break;)
```

```
    } }
    switch(stop)
    {
        case 1:  break;
        case 2:cc=cc|0x02;  break;
    }
    switch(data)
    {
        case 7:  break;
        case 8:cc=cc|0x01;  break;
    }
    regs.h.ah = 0;
    regs.h.al = cc;
    int86(0x34, &regs, &regs);
    }
}
```

05 RS-232 ポートの RTS 信号をセット

```
エントリー・パラメータ:   AH = 5
                          AL = 2
                          DH = 0   ;RTSをLOWにセット
                          DH = 1   ;RTSをHIGHにセット(標準値)
戻り値:                   なし

void TC_232_RTS(int rts)
{
    regs.h.ah = 5;
    regs.h.al = 2;
    regs.h.dh = (unsigned char)rts;
    int86(0x34, &regs, &regs);
    }
}
```

06 RS-232 ポートの CTS 信号を読む

```
エントリー・パラメータ:   AH = 6
                          AL = 2
戻り値:                   DH = 0   ;CTSがLOWの場合
```

```
        )          )          )          )))))1)   ;CTSがHIGHの場合)

int) TC_232_CTS())
{)
)) regs.h.ah) =) 6;)
)) regs.h.al) =) 2;)
)) int86(0x34,) &regs,) &regs);)
return((int)regs.h.ah);)
})
```

注意:)

RS-232ポートがINT)34Hの使用によってコントロールされる場合、そしてポートがホストポートとして働く場合、”NONE”としてプロトコルをセットしなければなりません。)

)

INT 33H を使用する RS-485 ポートのシリアル I/O

01 データ入力

エントリー・パラメータ) AH) =) 1)
戻り値:))) 1)) 文字を受信した場合)
))))) AH) =) 0)
))))))AL) =) 文字データ)
))) 2)) 文字を受信しなかった場合)
))))) AH) =) 1)
))))))AL) =) 未定義)

```
unsigned) char) TC_485_char_I())
{)
)) regs.h.ah) =) 1;)
)) int86(0x33,) &regs,) &regs);)
)) if(regs.h.ah) ==) 0))
)))))return(regs.h.al);)
)) return(255);)
})
```

)

02 データ出力

エントリー・パラメータ:) AH) =) 2)

AL = 文字データ
戻り値: なし

```
void TC_485_char_0(unsigned char ch)
{
  regs.h.ah = 2;
  regs.h.al = ch;
  int86(0x33, &regs, &regs);
}
```

03 シリアル I/O のために RS-485 ポートを有効にする

エントリー・パラメータ AH = 3
戻り値: なし

```
void TC_485_enable()
{
  regs.h.ah = 3;
  int86(0x33, &regs, &regs);
}
```

04 シリアル I/O のために RS-485 を無効にする

エントリー・パラメータ AH = 4
戻り値: なし

```
void TC_485_disable()
{
  regs.h.ah = 4;
  int86(0x33, &regs, &regs);
}
```

00 通信パラメータセット

エントリー・パラメータ: AH = 0 BIT #
76543210
AL = bit 0 xxxxxxx0 7データビット
xxxxxxx1 8データビット

			bit 1	xxxxxx0x	1ストップビット
				xxxxxx1x	2ストップビット
			bit 2-3	xxxx00xx	パリティ無し
				xxxx01xx	奇数パリティ
				xxxx11xx	偶数パリティ
			bit 4-7	0000xxxx	110bps
				0001xxxx	150bps
				0010xxxx	300bps
				0011xxxx	600bps
				0100xxxx	1200bps
				0101xxxx	2400bps
				0110xxxx	4800bps
				0111xxxx	9600bps
				1000xxxx	19200bps
				1001xxxx	38400bps
戻り値:			なし		

```
void TC_485_parameter(long baud, int parity, int stop, int data)
{
  unsigned char cc=0;
  unsigned int i_baud;

  i_baud = (int)(baud / 10L);
  switch(i_baud);
  {
    case 11:cc=0x00;  break;
    case 15:cc=0x10;  break;
    case 30:cc=0x20;  break;
    case 60:cc=0x30;  break;
    case 120:cc=0x40;  break;
    case 240:cc=0x50;  break;
    case 480:cc=0x60;  break;
    case 1920:cc=0x80;  break;
    case 3840:cc=0x90;  break;
    default:cc=0x70;  break;
  }
  switch(parity)
```

```
    ) ) {)
    ) ) ) ) case) 0:) ) break;)
    ) ) ) ) case) 1:cc=cc|0x04;) ) break;)
    ) ) ) ) case) 2:cc=cc|0x0c;) ) break;)
    ) ) ) ) case) 3:cc=cc|0x08;) ) break;)
    ) ) })
    ) switch(stop))
    ) {)
    ) ) ) case) 1:) ) break;)
    ) ) ) case) 2:cc=cc|0x02;) ) break;)
    ) ) })
    ) switch(data))
    ) {)
    ) ) ) case) 7:) break;)
    ) ) ) case) 8:cc=cc|0x01;) ) break;)
    ) ) })
    TD_int_dos1(0x1C,cc,1,0);)
    regs.h.ah) =) 0;)
    regs.h.al) =) cc;)
    int86(0x33,) &regs,) &regs);)
    })
)
```

05 データ送信のために RS-485 マルチバスを開く

RS-485はマルチバス・アーキテクチャであり、これは一つのRS-485) I/0ポートが一本の線(トランクライン)をアクセスすることができることを意味しています。したがって、RS-485がシリアルデータの入出力をしたい場合、他にリンクされるターミナルからのデータ送受信を防止するために最初にバスを占有しなければなりません。バスはデータ転送が終了したら開放され、他のターミナルがデータ転送に使用するために開けられます。)

エントリー・パラメータ:)) AH) =) 5)
戻り値:)))) なし)

```
void) TC_485_open())
{)
) ) regs.h.ah) =) 5;)
) ) int86(0x33,) &regs,) &regs);)
```

```
}
```

06 RS-485 マルチバスを閉じる(RS-485 バスを開放する)

```
エントリー・パラメータ:          AH = 6
戻り値:                          なし

void TC_485_close()
{
  regs.h.ah = 6;
  int86(0x33, &regs, &regs);
}
```

注意:

1) RS-485ポートがシリアル入出力(文字入出力)通信に使用される場合、アプリケーションは通信特性をセットするために先ずRS-485を有効にしなければならず、そしてシリアルI/0のためにシステムをレディ状態にして、データの読み出しや書き込みの前にバスを占有するためにそのRS-485を開きます。アプリケーションはデータパケットの転送が終了したときにRS-485ポートを閉じ、他のターミナルで使用するためにバスを開放しなければなりません。そしてアプリケーションはすべてのデータパックが送信または受信している間はRS-485を無効にしなければなりません。

```
例.     文字ベースの通信のためにRS-485ポートを有効にする
          ▽
        バスを占有するためにRS-485を開く
          ▽
        終了するまで{RS-485ポートでデータを読む/書く}を繰り返す
          ▽
        バスを開放するためにRS-485を閉じる
          ▽
        文字ベースの通信のためにRS-485ポートを無効にする
```

2) RS-422/485がINT 33Hの使用によりコントロールされ、ポートがホストとして割り当てられている場合、プロトコルは"NONE"としてセットしなければなりません。

3) RS-422/485ポートがシリアルポートして働く場合、そしてハンドシェーキングのフローがXON/XOFFの代わりにユーザプログラムによってコントロールされる場合、フローコントロールを"NONE"としてセットしなければなりません。

3.5) リレー出力/デジタル入力/ブザー/LED インジケータ)

LED インジケータ ON/OFF のセット) INT 09H

エントリー・パラメータ:) AH) =) 2)
)))) Bit) #) 76543210)
)))) AL) =) 0000xxxx) 、ここで)
)))))) X:) 1、LEDをOnにセット)
)))))))) 0、LEDをOffにセット)
)))))) Bit0:) LED1)
)))))) Bit1:) LED2)
)))))) Bit2:) LED3)
)))))) Bit3:) LED4)
戻り値:))) なし)

例:) AL=00000011) はLED1とLED2をOnにすることを意味しています。)

```
Void) TD_LED(int) led1,) int) led2,) int) led3,) int) led4))
{)
)) regs.h.ah) =) 2;)
)) regs.h.al) =) 0;)
)) if) (led1) >) 0)) regs.h.al) =) regs.h.al) |) 1;)
)) if) (led2) >) 0)) regs.h.al) =) regs.h.al) |) 2;)
)) if) (led3) >) 0)) regs.h.al) =) regs.h.al) |) 4;)
)) if) (led4) >) 0)) regs.h.al) =) regs.h.al) |) 8;)
)) int86(0x09,) &regs,) &regs);)
})
)
```

フォトカプラのレベル状態を読む) INT08H

エントリー・パラメータ:) AH) =) 1) ;) ポート1から入力を読む)
))))))) 2) ;) ポート2から入力を読む)
))))))) 3) ;) ポート3から入力を読む)
))))))) 4) ;) ポート4から入力を読む)
)))) AL) =) 0) ;) レベル状態を読む)
))))))) 1) ;) エッジ・スイッチング状態を読む)
戻り値:))) レベルにより)
)))) AL) =) 0) (LOW))

```
                                   1  (HIGH)
戻り値:                            エッジ・スイッチング状態により
                                   AL = 0  (スイッチング・エッジなし)
                                        1  (スイッチング・エッジ発生)
```

```
int TD_photocouple(int port, int type)
{
  regs.h.ah = (unsigned char)port;
  regs.h.al = (unsigned char)type;
  int86(0x08, &regs, &regs);
  return((int) regs.h.al);
}
```

リレーポートをアクティブにする/アクティブにしない INT09H

```
エントリー・パラメータ:  AH = 0   ; リレー#1を選択
                              1   ; リレー#2を選択
                         AL = 0   ; 選択したリレーコンタクトを
                                    アクティブにしない、OPEN
                              1   ; 選択したリレーコンタクトを
                                    アクティブにする、CLOSE
戻り値:                  なし
```

```
void TD_relay(int port, int status)
{
  regs.h.ah = (unsigned char)port;
  regs.h.al = (unsigned char)status;
  int86(0x09, &regs, &regs);
  return(regs.h.al);
}
```

1A ブザーのオン/オフ INT21H

```
エントリー・パラメータ:  AH = 0x1A
                         BH = 1
                         AL = 0   ; ブザーを無効にする
                              1   ; ブザーを有効にする
```

戻り値:　　なし

```
void TD_buzzer(int status)
{
  regs.h.ah = 0x1A;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 1;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
  }
```

1A ブザー音量のセット　　INT21H

エントリー・パラメータ:　AH = 0x1A
　　　　　　BH = 3
　　　　　　AL = 0　;音量をLOWにセット
　　　　　　　1　;音量をMEDIUMにセット
　　　　　　　2　;音量をHIGHにセット
戻り値:　　なし

```
void TD_beeper_vol(int status)
{
  regs.h.ah = 0x1A;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 3;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
  }
```

1B セキュリティ状態を得る　INT21H

エントリー・パラメータ:　AH = 0x1B
　　　　　　BH = 7
戻り値:　　AL = 0　;CLOSE
　　　　　　1　;OPEN

```
int TD_security_status()
{
```

```
  regs.h.ah = 0x1B;
  regs.h.bh = 7;
  int86(0x21, &regs, &regs);
  return((int) regs.h.aql);
  }
)
```

1B アラームをオン/オフ　　INT21H

```
エントリー・パラメータ:  AH = 0x1B
                         BH = 8
                         AL = 0   ; 無効
                              1   ; 有効
戻り値:                  なし

void TD_alarm(int status)
{
  regs.h.ah = 0x1B;
  regs.h.al = (unsigned char) status;
  regs.h.bh = 8;
  int86(0x21, &regs, &regs);
  }
)
```

54 ユーザ定義の周波数と時間でブザーの音量をコントロール)　INT21H

```
エントリー・パラメータ:  AH = 0x54
                         CX = 1-3000     ; 周波数(Hz)
                         DX = 1-1600     ; 時間(ミリ秒)
戻り値:                  なし

void TD_beep_user(int fz, int tm)
{
  regs.h.ah = 0x54;
  regs.x.cx = fz;
  regs.x.dx = tm;
  int86(0x21, &regs, &regs);
  }
  )
```

あらかじめ定義した周波数と時間でブザー音量をコントロール) INT35H

エントリー・パラメータ:) AX) =) 0-8) ;) 周波数指定)
)))) BX) =) 0-8) ;) 時間)
戻り値:))) なし)
)

))	)	周波数)	)	)	) 時間)
))	AX) =) 0)	200Hz)	)	)	BX) =) 0) 10ms)
))	))))) 1)	400)	)	)	))))) 1) 50)
))	))))) 2)	600)	)	)	))))) 2) 100)
))	))))) 3)	800)	)	)	))))) 3) 200)
))	))))) 4)	1K)	)	)	))))) 4) 500)
))	))))) 5)	2K)	)	)	))))) 5) 800)
))	))))) 6)	2.5K)	)	)	))))) 6) 1秒)
))	))))) 7)	3K)	)	)	))))) 7) 1.5秒)
))	)))) 8)	5K)	)	)	))))) 8) 2秒)

```
void) TD_beep(int) fz,) int) tm))
{)
)) regs.x.ax) =) fz;)
)) regs.x.bx) =) tm;)
)) int86(0x35,&regs,&regs);)
})
)
```

3.6)内部/外部リーダ・ポート:) INT21H)

二つのリーダ、**内部リーダと外部リーダ**を MR350 MKII に接続することができます。**内部リーダ**は内蔵リーダで、MR350 MKII の内部にインストールされます。**外部リーダ**はスキャナ・ポートもしくはターミナルブロックのいずれかに接続されます。内部リーダについては、磁気ストライプ・リーダまたはバーコード・スロット・リーダが接続された場合、システムは最初にカードを読んでそのタイプを検出することによって内部リーダのタイプを自動的にセットアップすることができます。ユーザが標準のタイプを使用する場合、これはバーコード・スロット・リーダです。スロット・リーダに磁気カードを通して読もうとした場合に、ターミナルはそれを検出して、スロット・リーダのタイプを自動的に磁気ストライプ・リーダに変更します。しかし、このリーダタイプの自動識別は最初に読むデータには行われません。ユーザは再度カードを読まなければなりません。

51 外部リーダ・ポートを有効/無効

エントリー・パラメータ:　AH = 0x51
　　　　　　　　AL = 1　;) 外部ポート有効
　　　　　　　　　　0　;) 外部ポート無効
戻り値:　　なし

```
void TD_set_external(int status)
{
  regs.h.ah = 0x51;
  regs.h.al = (unsigned char)status;
  int86(0x21, &regs, &regs);
}
```

18 外部スロット・リーダをセット(バーコードと磁気ストライプ・リーダのみ)

エントリー・パラメータ:　AH = 0x18
　　　　　　　　AL = 1　;) バーコード・スロット・リーダとして定義
　　　　　　　　　　0　;) 磁気ストライプ・リーダとして定義
戻り値:　　なし

```
void TD_set_external_type(int status)
{
  regs.h.ah = 0x18;
  regs.h.al = (unsigned char)status;
  int86(0x21, &regs, &regs);
}
```

50 外部リーダからデータを読む(バーコードと磁気ストライプ・リーダのみ)

エントリー・パラメータ:　AH = 0x50
戻り値:　　DS:DX = バッファ・ポインタ
　　　　　　AX = 0　;) データ入力
　　　　　　　　1　;) データ入力無し
　　　　　　スキャン方向
　　　　　　CL = 0　;) 右から左
　　　　　　　　1　;) 左から右

```
int TD_get_external(unsigned char *str, int wait, int *direction)
```

```
{
  int i;
  do
  {
    segregs.ds = FP_SEG(str);
    regs.x.dx = FP_OFF(str);
    regs.h.ah = 0x50;
    int86X(0x21, &regs, &regs, &segregs);
    *direction = regs.h.cl;
  } while(wait && regs.h.al);
  return(regs.h.al);
}
```

52 内部ポートを読む

```
エントリー・パラメータ:  AH = 0x52
戻り値:                  DS:DX = バッファ・ポインタ
                         AX = 0    ; データ入力
                              1    ; データ入力無し
                         スキャン方向
                         CH = 0    ; バーコード・データ
                         CL = 0    ; 右から左
                              1    ; 左から右
                         BL= 0x010       ; Code39
                             0x02        ; Interleaved 2 of 5
                             0x03        ; Codabar
                             0x05        ; Code 128
                             0x06        ; EAN 128
                             0x07        ; Code 93
                             0x11        ; UPC-A
                             0x12        ; UPC-E
                             0x13        ; EAN-13
                             0x14        ; EAN-8
                         CH = 1    ; 磁気データ
                         CL = 0    ; 右から左
                              2    ; 左から右
```

```
        BL = 0x01       ; トラック1
        BL = 0x02       ; トラック2/3
        BL = 'K'        ; ARK501キーパッド入力
    CH = 2      ; ウェイガンド・データ
        CL = 0      ; フォーマットされたデータ
        BL = 1      ; フォーマットされていないデータ
        BL = データ長(ビット)

Int TD_get_internal(unsigned char *str, int *direct_format, int *dev_type, int
*data_type, int wait)
{
  int i;
  do
  {
    i = TD_intdos_I(0x52, 0, str);
    *direct_format = regs.h.cl;
    *dev_type = regs.h.ch;
    *data_type = regs.h.bl;
  } while (wait && i);
  return(i);
}
```

53 内部リーダを有効/無効

```
エントリー・パラメータ:  AH = 0x53
        AL = 1      ; 内部ポート有効
        AL = 0      ; 内部ポート無効
戻り値:         なし

void TD_set_internal(int status)
{
  TD_int_dos1(0x53, (unsigned char) status,0, 0);
}
```

1F ウェイガンド・フォーマットのデコーダを有効/無効(近接リーダ)

```
エントリー・パラメータ    AH = 0x1F
```

```
                        BH) =) 5)
                        BL) =) 0)   ;) 26ビットと36ビットの両フォーマット)
                        ) ) ) ) 26) ;) 26ビットのみ)
                        ) ) ) ) 36) ;) 36ビットのみ)
                        ) ) ) 0xff) ;) フォーマットされていないデータ)
                        AL) =) 0)   ;) 無効)
                        ) ) ) ) 1)  ;) 有効)
戻り値:)                なし)

void) TD_set_weigand_status(int) status,) int) type))
{)
) ) regs.h.ah) =) 0x1f;)
) ) regs.h.al) =) (unsigned) char)status;)
) ) regs.h.bh) =) 5;)
) ) if) (type==) -1)regs.h.bl=0xff;)
) ) else) regs.h.bl) =) (unsigned) char)type;)
) ) int86(0x21,) &regs,) &regs);)
})
)
```

1F ウェイガンド・フォーマットのデコーダ・ステータスを得る

```
エントリー・パラメータ)  AH) =) 0x1F)
                        BH) =) 6)
                        BL) =) 0)   ;) 26ビットと36ビットの両フォーマット)
                        ) ) ) ) 27) ;) 26ビットのみ)
                        ) ) ) ) 37) ;) 36ビットのみ)
                        ) ) ) 0xff) ;) フォーマットされていないデータ)
戻り値:)                AL) =) 0)   ;) 無効)
                        ) ) ) ) 1)  ;) 有効)
) )

void) TD_get_weigand_status(int) type))
{)
) ) regs.h.ah) =) 0x1f;)
) ) regs.h.bh) =) 6;)
) ) if) (type==) -1)regs.h.bl=0xff;)
) ) else) regs.h.bl) =) (unsigned) char)type;)
```

```
  int86(0x21, &regs, &regs);
  return(regs.h.al);
}
```

1A 内部リーダのバーコードまたは磁気ストライプ入力を指定

```
エントリー・パラメータ:   AH = 0x1A
                          BH = 6
                          AL = 0    ; バーコード入力を指定
                          AL = 1    ; 磁気ストライプ入力を指定
戻り値:                   なし

void TD_set_internal_type(int status)
{
  regs.h.ah = 0x1A;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 6;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
}
```

1F バーコード・シンボルのデコードを有効にする

```
エントリー・パラメータ:   AH = 0x1F
                          BH = 1
                          AL = 0    ; 無効
                          AL = 1    ; 有効
                          BL = 0    ; 全部
                          BL = 1    ; Code 39
                          BL = 2    ; I 2 of 5
                          BL = 3    ; Codabar
                          BL = 4    ; EAN/UPC
                          BL = 5    ; Code 128
戻り値:                   なし

void TD_set_decode_status(int status, int type)
{
```

```
  regs.h.ah = 0x1F;
  regs.h.al = (unsigned char) status;
  regs.h.bh = 1;
  regs.h.bl = (unsigned char)type;
  int86(0x21, &regs, &regs);
}
```

3.7) その他: INT21H

1A リチウム・バッテリのレベルをチェック

```
エントリー・パラメータ:  AH = 0x1A
                         BH = 09h
戻り値:                  AL = 1   ; リチウム・バッテリ電圧低下
                         AL = 0   ; 正常

int TS_lithium_battery()
{
  regs.h.ah = 0x1A;
  regs.h.bh = 9;
  int86(0x21, &regs, &regs);
  return(regs.h.ah);
}
```

1B ターミナルのアドレス ID を得る

```
エントリー・パラメータ:  AH = 0x1B
                         BH = 6
戻り値:                  AL = アドレスID

char TC_get_address()
{
  regs.h.ah = 0x1b;
  regs.h.bh = 6;
  int86(0x21, &regs, &regs);
  return((char)regs.h.al);
}
```

25 割り込みベクトルのセット

エントリー・パラメータ:　AH) =) 0x25)
)　)　)　)　)　AL) =) 割り込み番号)
)　)　)　)　)　DS:DX) =) 割り込みルーチンのアドレス)
戻り値:)　)　)　)　なし)

```
void) TS_set_interrupt_vector(int) vect,) unsigned) int) ds,) unsigned) int) dx))
{)
   regs.h.ah) =) 0x25;)
)) regs.h.al) =) (unsigned) char)) vect;)
)) segregs.ds) =) ds;)
)) regs.x.dx) =) dx;)
)) int86x(0x21,) &regs,) &regs,) &segregs);)
})
)
```

35 割り込みベクトルを得る

エントリー・パラメータ:　AH) =) 0x35)
)　)　)　)　)　AL) =) 割り込み番号)
戻り値:)　)　)　)　ES:BX) =) 割り込みルーチンのアドレス)

```
void) TS_get_interrupt_vector(int) vect,) unsigned) int) *es,) unsigned) int) *bx))
{)
)) regs.h.ah) =) 0x35;)
)) regs.h.al) =) (unsigned) char)vect;)
)) int86x(0x21,) &regs,) &regs,) &segregs);)
)) *es) =) segregs.es;)
)) *bx) =) regs.x.bx;)
})
)
```

36 空きディスク・クラスタを得る

エントリー・パラメータ:　AH) =) 0x36)
戻り値:)　)　)　)　AH) =) 1(クラスタ当たりのセクタ数))
)　)　)　)　)　BX) =) 使用可能なクラスタ数)
)　)　)　)　)　CX) =) 1024(セクタ当たりのバイト数))

```
Long TS_free_disk()
{
  regs.h.ah = 0x36;
  int86x(0x21, &regs, &regs, &segregs);
  return((long)regs.x.bx * (long)regs.x.cx);
}
```

1A システムキー・プレス・コマンドを有効/無効: ウォーム・スタート、ユーザ・コマンド・メニューの起動、管理者モードの起動

```
エントリー・パラメータ:  AH = 0x1A
                         BH = 05
                         AL = 0   ; システムキー無効
                              1   ; システムキー有効
戻り値:                  なし

void TD_set_system_key(int status)
{
  regs.h.ah = 0x1A;
  regs.h.al = (unsigned char)status;
  regs.h.bh = 5;
  int86(0x21, &regs, &regs);
}
```

1E キーボードマップを変更

```
エントリー・パラメータ:  AH = 0x1E
                         BH = 1
                         DS:DX  = 数字と英文字ASCIIコードテーブルに
                                  対応する128バイトのキーボードマップ:
                                  未使用キーの定義はNULL
                         CX = 0x80  (128バイトのテーブル長)
戻り値:                  なし

void TD_key_map(unsigned char *str)
{
  regs.h.ah = 0x1E;
```

```
    regs.h.ah = 1;
    regs.x.cx = 0x80;
    segregs.ds = FP_SEG(str);
    regs.x.dx = FP_OFF(str);
    int86x(0x21, &regs, &regs, &segregs);
}
```

数字モードにおいてスキャンコードに対応するASCIIコード

数字キーパッドのレイアウト					ASCIIコード[スキャンコード]				
F1 (⇦	F5 *	1	2	3	86 [23]	8A [1B]	31 [13]	32 [0B]	33 [03]
F2 (⇨	F6 △	4	5	6	87 [22]	8B [1A]	34 [12]	35 [0A]	36 [02]
F3 (↵	F7 ▽	7	8	9	88 [21]	8C [19]	37 [11]	38 [09]	39 [01]
F4 (⇨	SHIFT	[C]	0	[E]	89 [20]	1F [18]	08 [10]	30 [08]	0D [00]

図3.1 ASCIIコード対スキャンコードの参照テーブル(数字モード)

英字モードにおいてスキャンコードに対応するASCIIコード

英字モードのキーパッド・レイアウト					ASCIIコード[スキャンコード]				
F1 (⇦	F5 *	QZ.	ABC	DEF	86 [23]	8A [1B]	51,5A,2E [24,25,26]	41,42,43 [27,28,29]	44,45,46 [2A,2B,2C]
F2 (⇨	F6 △	4	5	MNO	87 [22]	8B [1A]	47,48,49 [2D,2E,2F]	4A,4B,4C [30,31,32]	4D,4E,4F [33,34,35]
F3 (↵	F7 ▽	7	8	9	88 [21]	8C [19]	50,52,53 [36,37,38]	54,55,56 [39,3A,3B]	57,58,59 [3C,3D,3E]
F4 (⇨	SHIFT	[C]	0	[E]	89 [20]	1F [18]	08 [10]	2D,20,2B [1D,1E,1F]	0D [00]

図3.2 ASCIIコード対スキャンコードの参照テーブル(英字モード)

3.8) DOS マネージャ

以下の MS/DOS ファンクションコールが MR350 MKII でエミュレートされます。呼び方は MS/DOS のもの、レジスタ AH にファンクション・コードを持つ INT 21H、と同じです。パラメータの引き渡し方法は MS/DOS のものと同一です。サポートされていない DOS コ

ールはすぐに終了ステータスコードが戻ります。詳細については MS/DOS の技術資料をご覧下さい。

標準入力/出力

01 stdin を読み(キー入力がなければ待つ)、stdout へ書く

```
コントロールキーのチェック無し(ESC)
エントリー・パラメータ:  AH => 01
戻り値:                  AL => 8ビットデータ

unsigned char TS_stdin()
{
  regs.h.ah = 1;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
}
```

02 stdout 書き出し

```
エントリー・パラメータ:  AH => 02
                         DL => 8ビットデータ
戻り値:                  なし

void TS_stdout(unsigned char ch)
{
  regs.h.ah = 2;
  regs.h.dl = ch;
  int86(0x21, &regs, &regs);
  return;
}
```

03 stdaux 読み込み(COM2 RS-232 ポート)

```
コントロールキーのチェック無し(ESC)
エントリー・パラメータ:  AH => 03
戻り値:                  AL => 8ビットデータ
```

```
unsigned char TS_stdaux_in()
{
  regs.h.ah = 3;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
}
```

04 atdaux 書き出し(COM2 RS-232 ポート)

```
エントリー・パラメータ:  AH = 04
                         DL = 8ビットデータ
戻り値:                  なし

void TS_stdaux_out(unsigned char ch)
{
  regs.h.ah = 4;
  regs.h.dl = ch;
  int86(0x21, &regs, &regs);
  return;
}
```

06 stdin 読み込み/書き出し、またはレディでない場合は 0 を戻す

```
エントリー・パラメータ:  AH = 06
                         DL = 0-0xFE      ; この文字をstdoutに書き出す
                         DL = FF          ; stdinを読む
戻り値:                  stdin書き出し:  なし
                         stdin読み込み
                                  文字がある場合、ゼロ=クリア
                                  AL = 8ビットデータ
                                  文字がない場合、ゼロ=セット

unsigned char TS_stdin_out(unsigned char ch)
{
  regs.h.ah = 6;
  regs.h.dl = ch;
  int86(0x21, &regs, &regs);
```

```
  if (ch) == 0xFF)
  {
  if ((regs.x.cflag) & 0x40)) == 0)) return(regs.h.al);
  else return(0);
  }
  return(0);
  }
)
```

07 stdin 読み込み(キー入力がない場合は待つ)

```
  コントロールキーのチェック無し(ESC)
  エントリー・パラメータ:  AH) = 07
  戻り値:                  AL) = 8ビットデータ

  unsigned char TS_stdin_noecho()
  {
  regs.h.ah = 7;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
  }
  )
```

08 stdin 読み込み(キー入力がない場合は待つ)

```
  コントロールキーのチェック無し(ESC)
  エントリー・パラメータ:  AH) = 08
  戻り値:                  AL) = 8ビットデータ

  unsigned char TS_stdin_wait()
  {
  regs.h.ah = 8;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
  }
```

09 stdout に文字列を書き出し

```
  エントリー・パラメータ:  AH) = 09
```

DS:DX = セグメント:文字列のオフセット
戻り値: なし

```
void TS_stdout_string(unsigned char *str)
{
  segregs.ds = FP_SEG(str);
  regs.x.dx = FP_OFF(str);
  regs.h.ah = 9;
  int86x(0x21, &regs, &regs, &segregs);
  return;
}
```

0A キーボード・バッファ入力

エントリー・パラメータ: AH = 0A
DS:DX = 入力バッファ・エリアのポインタ
戻り値: CRで最後の文字を埋めたバッファ

```
void TS_stdin_string(unsigned char *str)
{
  segregs.ds = FP_SEG(str);
  regs.x.dx = FP_OFF(str);
  regs.h.ah = 0x0a;
  int86x(0x21, &regs, &regs, &segregs);
  return;
}
```

0B キーヒット・チェック

エントリー・パラメータ: AH = 0B
戻り値: AL = 00 、文字がない場合
AL = FF 、文字がある場合

```
Unsigned char TS_kbhit()
{
  regs.h.ah = 0x0b;
  int86(0x21, &regs, &regs);
  return(regs.h.al);
```

```
}
```

日付/時刻

以下の4つのファンクションコールは、リアルタイム・クロック・チップに直接アクセスしてシステムの時刻と日付をセット/読み出しするために使用されます。

2A システム日付を得る

```
エントリー・パラメータ:    AH => 2A
戻り値:                    AL => 週
                           CX => 年(1980..2099)
                           DH => 月(1..12)
                           DL => 日(1..31)

Void TS_get_date(int *year, int *month, int *day, int *week)
{
  TD_int_dos1(0x2a, 0,0,0);
  *year = regs.x.cx;
  *month = regs.h.dh;
  *day = regs.h.dl;
  *week = regs.h.al;
}
```

2B システムの日付をセット

```
エントリー・パラメータ:    AH => 2B
                           CX => 年(1980..2099)
                           DH => 月(1..12)
                           DL => 日(1..31)
戻り値:                    AL => 0

int TS_set_date(int year, int month, int day)
{
  regs.h/ah = 0x2b;
  regs.x.cx = year;
  regs.h.dh = month;
  regs.h.al = day;
```

```
  int86(0x21, &regs, &regs);
  return(regs.h.al);
}
```

2C システム時計を得る

```
エントリー・パラメータ:  AH = 2C
戻り値:                  CH = 時間(0..23)
                         CL = 分(0..59)
                         DH = 秒(0..59)
                         DL = 0

Void TS_get_time(int *hour, int *minute, int *second, int *hund_sec)
{
  TD_int_dos1(0x2c,0,0,0);
*hour = regs.h.ch;
*minute = regs.h.cl;
*second = regs.h.dh;
*hund_sec = regs.h.dl;
}
```

2D システム時計をセットする

```
エントリー・パラメータ:  AH = 2D
                         CH = 時間(0..23)
                         CL = 分(0..59)
                         DH = 秒(0..59)
  戻り値:                AL = 0

Void TS_set_time(int hour, int minute, int second, int hund_sec)
{
    regs.h.ch = hour;
regs.h.cl = minute;
regs.h.dh = second;
regs.h.dl = hund_sec;
regs.h.ah = 0x2d;
int86(0x21, &regs, &regs);
```

```
return(regs.h.al);
}
```

3.9 ファイルマネージャ

ファイルがダウンロードまたはアップロードされた時に、作業中のファイルはアプリケーション・コマンドによって開くことはできません。同じ結果として、ファイルがアプリケーションによって開かれた場合、ホストコンピュータは同じファイルをダウンロードまたはアップロードすることができません。この制限は再書き込みからファイルポインタを保護し、ファイルシステムの破壊を防ぎます。

3C ファイルを作成または内容の消去

ファイルを作成したときに、ファイルマネージャは一致するファイル名をファイルテーブルで探します。一致するファイルがあったら、対応するファイルハンドルが戻り、そしてファイルポインタはファイルの先頭にリセットされます。実際のファイルサイズはゼロにリセットされます。ファイルがファイルテーブルになかった場合、ファイルエントリーが割り当てられ、そしてメモリが指定されます。

```
エントリー・パラメータ    AH = 3C
                          DS:DX = セグメント:ASCII Zファイル名のオフセット
戻り値:                   成功した場合: キャリイ=クリア、AX = ハンドル
                          失敗した場合: キャリア=セット、AX = 3
```

```
int TS_create_file(char *fn)
{
  segregs.ds = FP_SEG(fn);
  regs.x.dx = FP_OFF(fn);
  regs.h.ah = 0x3C;
  int86x(0x21, &regs, &regs, &segregs);
  if ((regs.x.cflag & 0x01) ==1) return(-1);
  else return(regs.x.ax);
}
```

3D ファイルを開く

ファイルはファイルテーブルになければなりません。このファンクションはファイルハンドルを戻します。

エントリー・パラメータ:　AH = 3D
　　　　　　　　　AL = 0　;読み込みのみ
　　　　　　　　　　　1　;書き込みのみ
　　　　　　　　　　　2　;読み込みと書き込み両方
　　　　　　　　　DS:DX = セグメント:ASCIIZファイル名のオフセット
戻り値:　　　　　成功の場合:キャリィ = クリア、AX = ハンドル
　　　　　　　　　失敗の場合:キャリィ = セット、AX = 2

ファイルが開かれたときに、ファイルマネージャは一致するファイル名をファイルテーブルで探します。一致するものがあったら、対応するファイルハンドルが戻されます。現在のポインタはファイルの始めにリセットされ、現在のオフセットフィールドはゼロにセットされます。

```
int TS_open_file(char *fn, int mode)
{
  regs.h.al = (unsigned char)mode;
  segregs.ds = FP_SEG(fn);
  regs.x.dx = FP_OFF(fn);
  regs.h.ah = 0x3D;
  int86x(0x21, &0x21, &regs, &regs, &segregs);
  if ((regs.x.cflag & 0x01) ==1) return (-1);
  else return(regs.x.ax);
}

```

3E ファイルを閉じる

エントリー・パラメータ:　AH = 3E
　　　　　　　　　BX = ファイルハンドル
戻り値:　　　　　キャリィ = クリア　　;OK
　　　　　　　　　　　= セット　　;失敗

```
void TS_close_file(int hdl)
{
  regs.h.ah = 0x3e;
  regs.x.bx = hdl;
  int86(0x21, &regs, &regs);
}

```

3F ファイル読み込み

現在のアドレスから(CX)バイトDS:DXにコピーします。
現在のアドレスを(CX)バイト進めます。

エントリー・パラメータ: AH = 3F
BX = ファイルハンドル
CX = 読み込むバイト数
DS:DX = セグメント:バッファ・エリアのオフセット
戻り値: 成功した場合: キャリィ=クリア、AX = バイト数で
EOFの場合はゼロ
失敗した場合: キャリィ = セット、AX = 6

```
int TS_read_file(int hdl, int cnt, char *str)
{
  segregs.ds = FP_SEG(str);
  regs.x.dx = FP_OFF(str);
  regs.h.ah = 0x3f;
  regs.x.cx = cnt;
  regs.x.bx = hdl;
  int86(0x21, &regs, &regs, &segregs);
  if((regs.x.cflag & 0x01) == 0) return(regs.x.ax);
  else return(-1);
}
```

40 ファイル書き込み

DS:DXから(CX)バイトをファイル(BX)にコピーします。BXの現在のアドレスと最後のアドレスを更新します。CXをゼロに割り当てた場合、システムは現在のファイルポインタの位置からファイルを消去します。この機能はMSCのファンクションコールchsize()によって使用されます。

エントリー・パラメータ: AH = 40
BX = ファイルハンドル
CX = 書き込むバイト数またはゼロ
DS:DX = セグメント:バッファ・エリアのオフセット
戻り値: 成功した場合: キャリィ = クリア、
AX = 読み込むバイト数、いっぱいの場合0

失敗した場合:) キャリィ) =) セット、AX) =) 6)

```
int) TS_write_file(int) hdl,) int) cnt,) char) *str))
{)
)) segres.ds) =) FP_SEG(str);)
)) regs.x.dx) =) FP_OFF(str);)
)) regs.h.ah) =) 0x40;)
)) regs.x.bx) =) hdl;)
)) regs.x.cx) =) cnt;)
)) int86x(0x21,) &regs,) &regs,) &segregs);)
)) if) ((regs.x.cflag) &) 0x01)) ==) 0)) return(regs.x.ax);)
)) else) return(-1);)
})
)
```

41 ファイル削除

エントリー・パラメータ:) AH) =) 41)
)))) DS:DX) =) セグメント:ASCIIZファイル名のオフセット)
戻り値:))) 成功した場合:) キャリィ) =) クリア)
)))) 失敗した場合:) キャリィ) =) セット、AX) =) 2)

```
int) TS_delete_file(char) *fn))
{)
)) segregs.ds) =) FP_SEG(fn);)
)) regs.x.dx) =) FP_OFF(fn);)
   regs.h.ah) =) 0x41;)
   int86x(0x21,) &regs,) &regs,) &segregs);)
   if) ((regs.x.cflag) &) 0x01)) ==) 0)) return) (1);)
   else) return(-1);)
})
)
```

42 ファイルポインタを移動

エントリー・パラメータ:) AH) =) 42)
)))) AL) =) 0)) ;) 先頭からのオフセット)
)))))))) 1) ;) 現在位置からのオフセット)
)))))))) 2) ;) 最後からのオフセット)
)))) BX) =) ファィルハンドル)

```
))          )          )          CX) =) オフセットの上半分)
))          )          )          DX) =) オフセットの下半分)
戻り値:)    )          )          成功した場合:) キャリィ) =) クリア、)
))          )          )          AX) =) 新しい現在値の下半分)
))          )          )          DX) =) 新しい現在値の上半分)
))          )          )          失敗した場合:) キャリィ=セット、AX) =) 6)

struct) LONG_INT{)
))          long) ll;)
))          })
struct) LONG_INT1{)
))          unsigned) int) ii1,) ii2;)
))          })
union) LONG_III{)
))          struct) LONG_INT1;)
))          struct) LONG_INT1i;)
))          })
long) TS_seek_file(int) hdl,) int) type,) long) loc))
{)
)) union) LONG_III) aa;)
)
)) regs.h.ah) =) 0x42;)
)) regs.h.al) =) (unsigned) char)type;)
   regs.h.bx) =) hdl;)
)) aa.l.ll) =) loc;)
)) regs.x.cx) =) aa.i.ii2;)
)) regs.x.dx) =) aa.i.ii1;)
)) int86(0x21,) &regs,) &regs);)
)) aa.i.ii2) =) regs.x.dx;)
)) aa.i.ii1) =) regs.x.ax;)
)) if) ((regs.x.cflag) &) 0x01)) ==) 0)) return(aa.1.11.);)
)) else) rreturn(-1L);)
})
)
)
```

43 ファイル属性を得る

```
エントリー・パラメータ:　AH = 43
　　　　　　　　　　AL = 0
　　　　　　　　　　DS:DX = セグメント:ASCIIZファイル名のオフセット
戻り値:　　　　　　ファイルがある場合:　キャリィ = クリア、CX = 0
　　　　　　　　　　ファイルがない場合:　キャリィ = セット、AX = 2

int TS_check_file_exit(char *str)
{
  regs.h.ah = 0x43;
  regs.h.al = 0;
  segregs.ds = FP_SEG(str);
  regs.x.dx = FP_OFF(str);
  int86x(0x21, &regs, &regs, &segregs);
  if ((regs.x.cflag & 0x01) == 0) return(1));
  else return(0);
}
```

56 ファイル名の改名

```
エントリー・パラメータ:　AH = 56
　　　　　　　　　　DS:DX = 改名するASCIIファィル名のポインタ
　　　　　　　　　　ES:DI = 新しいファイル名のポインタ
戻り値:　　　　　　AH = 0　;　成功の場合:　キャリィ・フラグをクリア
　　　　　　　　　　　　1　;　失敗の場合:　キャリィ・フラグをセット

int TS_rename_file(char *inf, char far *out)
{
  segregs.ds = FP_SEG(inf);
  regs.x.dx = FP_OFF(inf);
  segregs.es = FP_SEG(outf);
  regs.x.di = FP_OFF(outf);
  regs.h.ah = 0x56;
  int86x(0x21, &regs, &regs, &hsegregs);
  if((regs.x.cflag & 0x01) == 0) return(regs.x.ax);
  else return(-1);
}
```

)
)

メモリ・アクセス

48 指定したパラグラフ・メモリ数を割り当てる

エントリー・パラメータ:) AH) =) 48)
)))) BX) =) セグメント数)
戻り値:))) AX) =) キャリィ・フラグがセットされた場合、)
エラーコードに割り当てられたブロックの)
セグメント・アドレス-)
BX) =) 最も大きな有効ブロック(失敗時))

49 割り当てられたメモリの空き容量

エントリー・パラメータ:) AH) =) 49)
)))) ES) =) 空いているブロックのセグメント)
戻り値:))) AX) =) エラーコード、キャリィ・フラグが)
セットされている場合)
BX) =) 最も大きい使用可能なブロック(失敗時))

4A 割り当てられたブロックの変更

エントリー・パラメータ:) ES) =) 変更するブロックのセグメント)
)))) BX) =) 希望する新しいセグメント数)
戻り値:))) AX) =) エラーコード、キャリィ・フラグが)
セットされている場合)
BX) =) 最も大きい使用可能なブロック(失敗時)、)
キャリィ・フラグがセットされている場合)
)

第4章) ホスト ESC コマンド)

ホスト通信には三つのクラスがあります。

(1) ホストが MR350 MKII にコントロール/設定コマンドを送信

MR350 MKII の設定と動作のほとんどはコントロール・コマンドを通してホスト・システムによってコントロールされます。設定コマンドは通信コントロール・テーブルのようなシステム・テーブルをセットアップするために使用されます。コントロール・コマンドは MR350 MKII の動作、MR350 MKII のリセット、アプリケーション・プログラムの実行を MR350 MKII に指示、あるいは動作に関連した多の機能の実行に使用されます。

設定コマンドは通常、初期化のプロセス中にホスト・システムによって出されます。しかし、コントロール・コマンドは通常動作もしくは復帰の動作時に任意の時点で出すことができます。

(2) ホストが MR350MKII からのデータを要求

通常、二種類のデータがホスト・システムにより要求されます。これらは MR350 MKII システム・データとアプリケーション・データです。アプリケーション・データはキーボードからの入力またはバーコード・スキャンによる情報で、一方システム・データは MR350 MKII の実行にのみ関係がある情報、例えば、ファイル名、システム・パラメータ等です。MR350 MKII は使用するプロトコルによってデータを送信します。

(3) ホストと MR350 MKII 間のファイル転送

実行形式ファイルとデータファイルはホストから MR350 MKII にダウンロードまたは MR350 MKII からホストにアップロードされます。ファィル転送はポイント・ツー・ポイントまたはマルチポイント接続でデータを転送するために特別なファイル転送プロトコルを使用します。

以下のセクションでは、ホスト通信コマンドとそれらの機能を簡単に紹介します。そして、各機能についてのプロトコルを詳しく検討します。

4.1)一般のコントロール・コマンド)

1. Hard Reset (ESC H)

ハード・リセット・コマンドは MR350 MKII の RAM メモリの内容をすべてクリアし

ます。すべての主要なハードウェア・デバイスのテストを行います。以前に MR350 MKII に保存されたプログラムまたはデータ、または以前にホストによってダウンロードされたものはメモリから消されます。標準のシステム・パラメータは Flush PROM から復元されます。ハード・リセット・コマンドは Flash ROM に保存されているプログラム・ファィルは消しません。

ハード・リセット・コマンドはパラメータあるいは値を持っていません。キーパッド呼び出しと同等なシーケンスは管理者モードへ入ることと、初期化コマンドの選択を含んでいます。

2. 中断（ESC A）

中断は「ソフトリセット」コマンドです。MR350 MKII はその実行をやめてレディ・モードに戻ります。MR350 MKII の RAM 領域に保存されたプログラムとデータは保護されます。システム・パラメータは変更されずに残ります。中断コマンドはパラメータや値を持っていません。キーパッド呼び出しと同等なシーケンスは、[SHIFT]と[F5/*]キーを同時に押すことを含んでいます。

3. 実行（ESC X ファイル名）

ホスト・システムは MR350 MKII に MR350 MKII の RAM または MR350 MKII の Flash ROM にあるプログラムを実行するように指示します。ホストは実行するプログラム名をパラメータとして実行コマンドを出します。MR350 MKII は、プログラムがあり、そして実行を開始した場合、ACK の応答で答えます。

実行コマンドは一つのパラメータ､ファイル名を持っています｡実行コマンドは MR350 MKII ワークステーション・メニューを通しても呼び出すことができます。

Flash ROM プログラムの実行(ESC X/ROM ファイル名)

ホスト・システムは MR350 MKII の Flash ROM にあるプログラムの実行を MR350 MKII に指示することができます。

4. ディレクトリ（ESC D）

ディレクトリ・コマンドは MR350 MKII に、MR350 MKII の RAM にあるファイルのリストを戻すように指示します。ディレクトリ・コマンドは、MR350 MKII のワークステーション・メニューを通して呼び出すこともできます。キーパッドから呼び出された場合、ディレクトリは LCD ディスプレーに表示されます。

Flash ROM のディレクトリ(ESC D/ROM)

ディレクトリ・コマンドはMR350 MKIIのFlash ROMにあるプログラムのリストを返すようにMR350 MKIIに指示します。

5. 消去（ESC E ファイル名）

消去コマンドはMR350 MKIIのRAMからファイルを削除します。ACK応答はファイルがあって、削除された場合に返され、それ以外はNAKがMR350 MKIIによって生成されます。消去コマンドは一つのパラメータ、ファィル名を持っています。消去コマンドはMR350 MKIIワークステーション･メニューでも呼び出すことができます。

6. 自動ブート（ESC O プログラム名）

このコマンドはMR350 MKIIの自動ブート･プログラム名を定義します。自動ブート･プログラムは電源がオフにされ、そしてオンになるたびに自動的に実行されます。

7. パスワード（ESC P パスワード）

管理者のパスワードを作成または編集します。

8. RAMサイズを得る（ESC G）

MR350 MKIIのRAMの総容量、プログラム実行メモリ、そしてRAMディスクのために空いているメモリを得ます。

Flash ROMのサイズを得る(ESC G/ROM)

MR350 MKIIのFlash ROMの総容量とRAMディスクのために有効な空きメモリを得ます。

9. 現在実行しているプログラムのファイル名を得る（EXC I ファイル名）

<ESC I> コマンドをMR350 MKIIが受信した場合、システムは現在実行しているプログラム名またはプログラムが何も実行されていないことをレポートします。

10. ファイルの存在とファイル容量をチェック（ESC J ファィル名）

MR350 MKIIにファイルがあるかどうかをチェックします。ファイルがあれば、ファイルサイズを表示します。

11. キーボード・ロックをセット（ESC K 状態）

MR350 MKIIキーボードのロック状態をセットします。三つの状態、UNLOK、LOCK、または PARTIAL LOCKがあります。

12. ターミナルのアドレスIDを変更（ESC 5 ID）

ターミナルに新しいターミナル・アドレス ID を指定します。ターミナル・アドレス ID が変わった場合、ターミナルをリセットせずにすぐに有効になります。

13. UPS の使用をやめる（ESC F）

一般に、UPS バッテリは電源アダプタが外れた場合あるいは停電時に MR350 MKII に電源を供給します。ユーザは停電の後 UPS バッテリを使用しないようにするためにこのコマンドを使用することができます。

14. ループバック・テスト（ESC 9）

ループバック・テストは通信ラインのテストに使用されます。ホストで実行されるテストプログラムはターミナルにテストデータと共にこのコマンドを送り、ターミナルはプログラムで確認するためにデータをエコーバックします。

15. ブザー・ボリューム（ESC N）

このコマンドは MR350 MKII のブザー音量をリモートで変更することができます。

16. 管理者パスワード（ESC P）

このコマンドは管理者モードで、MR350 MKII のパスワードをリモートで変更します。

17. ターミナル ID を得る（ESC R）

このコマンドはターミナル ID を得ます。標準のターミナル ID は”MR350”です。

18. ターミナル ID とバージョン番号を得る（ESC v）

このコマンドはターミナル ID とバージョン番号を得ます。標準のターミナル ID は”MR350 V4.xx” です。

4.2)設定コマンド)

ホストからの設定コマンドは、必ず標準のホスト・コマンド・シーケンス、ESC 、コマンド テーブル、に従います。コマンド・フィールドは設定する対象を指定します。テーブル・フィールドはあらかじめ定義されたフォーマットで構成されるデータを含んでいます。

1. ターミナル設定（ESC T）

ターミナル設定コマンドは以下のフォーマットです。

ESC T termtable

このコマンドはホストからデータ構造”termtable”を取り、それを MR350 MKII 内部ターミナル・コントロール・テーブルに書き込みます。”termtable”内部の構造は前のセクションで指定した TERM_CONFIG typedef (22 ページのセクション 2.5 を参照)と同じでなければなりません。

新しいターミナル・コントロール・テーブルは ESC T コマンドが正しく受信された後ですぐに有効になります。

2. 通信設定（ESC C）

通信設定コマンドは以下のフォーマットです。

ESC C comtable

このコマンドはホストからデータ構造”comtable”を取り、それを MR350 MKII の二つの内部通信コントロール・テーブルの一つに書き込みます。”comtable”中の構造は、前のセクションで指定した COM_CONFIG typedef(20 ページのセクション 2.4)と同じでなければなりません。

新しい通信コントロール・テーブルは ESC C コマンドが正しく受信された後ですぐに有効になります。MR350 MKII はその新しいパラメータで対応する通信ポートを再初期化します。

例えば、ESC C コマンドが RS232 ポートの転送速度を 9600 から 1200 に変更するように指示した場合、MR350 MKII は ESC C コマンドを受信した後ですぐに 1200 に切り換えます。次のホスト通信は 1200bps を使用します。

3. デバイス設定（ESC V）

デバイス設定コマンドは以下のフォーマットです。

ESC V devtable

このコマンドはホストからデータ構造”devtable”を取り、それを MR350 MKII 内部デバイス・コントロール・テーブルに書き込みます。”devtable”中の構造は、前のセクションで指定した DEV_CONFIG typedef(19 ページのセクション 2.1)と同じでなければなりません。

新しいデバイス・コントロール・テーブルは ESC V コマンドが正しく受信された後ですぐに有効になります。

4. バーコード・シンボルの設定) （ESC B）

バーコード・シンボル設定コマンドは以下のフォーマットです。

ESC V bartable

このコマンドはホストからデータ構造”bartable”を取り、それを MR350 MKII 内部バーコード・シンボル・コントロール・テーブルに書き込みます。”devtable”中の構造は、前のセクションで指定した BAR_CONFIG typedef(20 ページのセクション 2.3)と同じでなければなりません。

新しいデバイス・コントロール・テーブルは ESC B コマンドが正しく受信された後ですぐに有効になります。

5. 日付/時刻設定（ESC M）

日付/時刻設定コマンドは以下のフォーマットです。

ESC M datetime

このコマンドはホスト・システムが MR350 MKII のリアルタイム・クロック機能を初期化することができます。パラメータ datetime は以下の表記: yyyymmddhhmmss を持った ASCII 文字列です。

最初の 4 文字は年を表します。次の 2 文字は月を表し、1 月は 01 です。月フィールドの次のフィールドはその月の日にち、時間(24 時フォーマット)、分と秒です。

例えば、コマンド ESC M 200209262345 は、MR350 NKII の時計を 2002 年 9 月 26 日に初期化します。時刻は 11:45PM です。MR350 MKII は ESC M コマンドを正しく受け取った後すぐにリアルタイム・クロック・チップを初期化します。

4.3)ファイル転送コマンド))

1. ダウンロード（ESC L ファイル名）

ダウンロード・コマンドはバイナリの実行形式プログラムまたはデータファイルをホスト・システムから MR350 MKII に転送するために使用されます。MR350 MKII がダウンロード・コマンドを受信した場合、ACK 応答をホストに返し、そしてすぐにファイル受信状態になります。ファイル受信状態は前もって指定されたホスト・プロトコル Kermit によって決められます。ホスト・システムは ACK 応答を受信するとすぐにファイル転送を開始します。

ダウンロード・コマンドは一つのパラメータ、ファイル名を持っています。ダウンロードは MR350 MKII ワークステーション・メニューを通しても起動することができます。

アップロード・コマンドはダウンロード・コマンドと逆の機能を実行します。これはMR350 MKII からホスト・システムにデータファイルを転送するために使用されます。これはワークステーション・モードで収集したデータ検索の代表的な手段です。

2. アップロード（ESC U ファイル名）

MR350 MKII がアップロード・コマンドを受信した場合、これはファイル送信状態になり、指定されたデータファイルの送信を始めます。ファイル転送プロトコルはあらかじめ指定したホスト・プロトコル Kermit によって決められます。ホスト・システムは、アップロード・コマンドを出した後でデータファイルの受信を待ちます。

アップロード・コマンドは一つのパラメータ、ファイル名を持っています。アップロード・コマンドは MR350 MKII ワークステーション・メニューを通しても起動することができます。

4.4)マルチポイント・プロトコル)

マルチポイント動作で、MR350 MKII はホストコンピュータとの通信に非同期シリアル・マルチドロップ・プロトコルを使用します。このプロトコルを動作可能にするには、ホストと MR350 MKII との間に RS-232 と RS-485 のコンバータが必要であることにご注意下さい。ターミナル・プロトコルは、以下のフォーマットのコマンドと応答で構成されています。

記号	説明
=>	ホストからターミナルへ伝送
<=	ターミナルからホストへ伝送
ADDR	ターミナル・アドレス (A - Y, 0 - 6) + 80H
CMD	ターミナルへのネットワーク・コマンド、2 バイト、A-F、0 – 9
CS1	チェックサム、第一バイト
CS2	チェックサム、第二バイト

チェックサムは、伝送する各バイト、ADDR、そしてデータブロック長(STX と ETX を含まない)を加えることによって計算されます。CS1 は上位ニブル(4 ビット) + 40H で、CS2 は下位ニブル + 40H です。

例: A.EXE の名前のファイルを読み込むコマンド

STX ESC L A . E X E CS1 CS2 ADDR

データブロック = ESC L A . E X E (STX は含まない)

データブロック長 = 7

CS = ESC + L + A + . + E + X + E + ADDR + 7

CS1 = CS の上位ニブル + 40H

CS2 = CS の下位ニブル + 40H

ASCII データの文字とこれらの値は以下の通りです:

STX 0x02

ETX 0x03

ACK 0x06

NAK 0x15

DC1 0x11

ESC 0x1B

EOT 0x04

プロトコル制御文字を含む最大フレーム・サイズは 128 バイトです。プロトコル制御文字 STX と ETX のトランスペアレント伝送は'¥' (バックスラッシュ)文字を前に置くことによって行います。"¥"文字のトランスペアレント伝送は二つの"¥"文字を続けて送ることによって行われます。

データ伝送中のデータ変換の規則:

1) 2 バイトデータに変換される 1 バイトデータ

¥ ➔ ¥¥

00hex – 1F hex ➔ ¥ 80hex -- ¥ 9Fhex

A0hex – FFhex ➔ ¥ 20hex -- ¥7F hex

(DC hex は含まない)

(ア) 他のコードに変換せずに元のデータとして伝送される 1 バイト・データは変わらない。

ホスト伝送

伝送	フォーマット
ポール	STX、ADDR
ホストデータ	STX、 CMD、 データ、CS1、CS2、ADDR
了解	ACK
了解せず	NAK

ターミナル伝送

伝送	フォーマット
ターミナル・データ	STX、データ、CS1、CS2、ETX
了解	ACK
了解せず	NAK

4.4.1 プロトコル動作

ターミナル・プロトコルはマルチポイントのストップ・アンド・ウェイト・プロトコルと

して動作します。ステーションは 1 フレームだけを送信し、そして停止し、応答を待ちます。

以下のシナリオはリンク伝送の代表的なものです:

* ターミナルはホストに送るデータを持っていない:

  => STX ADDR

  <= EOT - もし、伝送待ちのデータがない場合

* ターミナルはホストに送るデータを持っている

  => STX ADDR

  <= STX <データ> CS1 CS2 ETX – データがある場合

  => ACK – データを正しく受信した場合

  〉NAK – エラーが発生した場合

* ホストは一つのポール・サイクルでコマンドを送信し、そして応答を受信します。そしてターミナル・コマンド応答の受信を了解します。

  => STX, CMD, パラメータ, ..., CS1, CS2, ADDR

  <= ACK – データを正しく受信、そして応答は不要、あるいは

  〉NAK – エラーが発生、もしくは送る応答データがある

  <= コマンド応答とデータ

  => ACK – コマンド応答がターミナルから送られ、そしてホストで
  正しく受信された

  NAK – コマンド応答でエラーが発生した

4.4.2 **コマンド**

以下のコマンドが、ダウンロード、診断、そしてアプリケーション・データの転送のためにサポートされています。

各コマンドとそのパラメータは送信する前の形式で作られています。応答は ADDR フィールドなしの同じフォーマットです。

STX, CMD, パラメータ, CS1, CS2, ADDR

ESC) 0 – アプリケーション・データ

=> STX ESC 0 <データ> CS1 CS2 ADDR

<= ACK または NAK

注意:

MR350 MKII は入力されてくる一つのアプリケーション・データを保持するフレームを一つだけ持っており、そしてそのフレームは保持されているデータがターミナルのアプリケーション・プログラムで取られるまで有効にはなりません。したがって、ターミナルがデータを取ったことを確認するために ACK または NAK のステータスをエコーバックされたかをターミナルでチェックすることが重要です。

ESC 5 – マルチポイント・アドレスをセット

=> STX ESC '5' <addr> CS1 CS2 ADDR　ここで、<addr>='A' – 'Y'、'0' – '6'

<= STX ESC '5' <Retcode> CS1 CS2 ETX　ここで、<Retcode>= 0x00 セット OK

0x01 エラー

=> ACK または NAK

ESC 9 – 診断テストデータを送信

=> STX ESC '9' <data> CS1 CS2 ADDR

<= STX ESC '9' <data> CS1 CS2 ETX

=> ACK または NAK

ESC A – ソフトリセット、再スタートまたは中断

このコマンドは、キーパットでコントロール-Exit が押された場合に MR350 MKII のプログラム実行を停止します。

=> STX ESC A CS1 CS2 ADDR

<= ACK または NAK

ESC B – バーコード・シンボルのデコードを有効/無効

=> STX ESC B <abcde> CS1 CS2 ADDR

ここで、<abcde>は、最初の文字が Code 39、二番目が I2 of 5、以下 Codabar、EAN/UPC、そして Code 128 の順で、個々のバーコード・シンボルをデコードを有効/無効にするための 5 文字のデータです。'N'の文字は対応するバーコードのデ

コードを有効にし、'F'はデコードを無効にします。

<= STX ESC B <Retcode> CS1 CS2 ETX

=> ACK または NAK

ESC C – 通信設定テーブルを MR350 MKII に書く

=>) STX) ESC) C) <comtable>) CS1) CS2) ADDR)

<=) STX) ESC) <Retcode>) CS1) CS2) ETX)

=>) ACKまたはNAK)

)

ESC D – RAM ディスクのディレクトリをホストに読み込む

=>) STX) ESC) D) CS1) CS2) ADDR)

<=) STX) ESC) D) <directory) data>) CS1) CS2) ETX) (またはNAK))

=>) ACKまたはNAK)

)

ESC D/DOM – Flash ROM のディレクトリをホストに読み込む

=>) STX) ESC) D/ROM) CS1) CS2) ADDR)

<=) STX) ESC) D/ROM) <directory) data>) CS1) CS2) ETX) (またはNAK))

=>) ACKまたはNAK)

)

ESC E – MR350 MKII ディレクトリからファイルを消去する

=>) STX) ESC) E) <filename>) CS1) CS2) ADDR)

<=) STX) ESC) E) <RetCode>) CS1) CS2) ETX)

)) ここで、<RetCode>)

)) 00) 消去に成功)

)) 01) ファイルがない)

=>) ACKまたはNAK)

)

ESC F – UPS バッテリの電源供給をしない

このコマンドはUPSバッテリがターミナルへの電源供給を停止するために使用します。このコマンドを使用することによって、電源がオフになった場合にUPSバッテリが上がってしまうことを防ぎます。)

=>) STX) ESC) F) CS1) CS2) ADDR)

<=) ACKまたはNAK)

)

ESC G – MR350 MKII の RAM 容量を得る

=>) STX) ESC) G) CS1) CS2) ADDR)

<=) STX) ESC) G) <xxx>) <yyy>) <zzz>) CS1) CS2) ETX)

)) ここで、<xxx>) =) 全メモリ容量(KB))

))) <yyy>) =) 使用メモリ容量(KB))

))) <zzz>) =) 空きメモリ容量(KB))

=>) ACK) または) NAK)

)

ESC G/ROM – MR350 MKII の Flash ROM 容量を得る

=>) STX) ESC) G/ROM) CS1) CS2) ADDR)

<=) STX) ESC) G/ROM) <xxx>) <zzz>) CS1) CS2) ETX)

)) ここで、<xxx>) =) 全メモリ容量(KB))

))) <zzz>) =) 空きメモリ容量(KB))

=>) ACK) または) NAK)

)

ESC H – ハードリセットと電源投入時のテストを起動する

このコマンドはMR350) MKIIターミナルをリセットし、実行中のアプリケーションを停止します。電源オンテストルーチンが実行され、正しく終了したら、MR350) MKIIは次のコマンドに対する準備ができます。)

=>) STX) ESC) H) CS1) CS2) ADDR)

<=) ACK) または) NAK)

)

ESC I – 現在実行しているファイル名を得る

=>) STX) ESC) I) CS1) CS2) ADDR)

<=) STX) ESC) I) <ファィル名>) CS1) CS2) ETX)

)) ファイル名のプログラムが実行している場合)

<=) STX) ESC) I) <01>) CS1) CS2) ETX)

)) プログラムが実行していない場合)

=>) ACK) またはNAK)

)

ESC J – ファイルの有無をチェック

=>) STX) ESC) J) <filename>) CS1) CS2) ADDR)
<=) STX) ESC) J) <Retcode>) CS1) CS2) ETX)
)) ここで、Retcode) =) 00) ファイルがある)
)))) =) 01) ファイルがない)
=>) ACK) または) NAK)
)

ESC K – キーボードロックをセット

=>) STX) ESC) K) <state>) CS1) CS2) ADDR)
)) ここで、state) =) '0') キーボード) LOCK) をセット)
)))))) =) '1') キーボードUNLOCKをセット)
)))))))) =) '2') キーボードの部分LOCKをセット)
<=) STX) ESC) K) <Retcode>) CS1) CS2) ETX)
)) ここで、Retcode) =) 00) 成功した場合)
)))) =) 01) 失敗した場合)
=>) ACK) または) NAK)
)

ESC L – MR350 MKII にファイルを転送

=>) STX) ESC) L) <filename>) CS1) CS2) ADDR)
<=) ACKまたはNAK)
)) …ファイル転送が終了するまで次の2ステップをループする)
)) =>) STX) ESC) 'Y') <data>) CS1) CS2) ADDR)
)) <=) ACK) または) NAK)
=>) STX) ESC) Z) CS1) CS2) ADDR) (データ送信終了))
<=) ACK) または) NAK)
)

ESC M – MR350 MKII に日付と時刻を書き込む

=>) STX) ESC) M) <datetime>) CS1) CS") ADDR)
)) ここで、<datetime>はASCII文字列、yyyymmddhhmmss)
<=) STX) ESC) M) <Retcode>) CS1) CS2) ETX)
=>) ACK) または) NAK)
)

ESC N – ブザー音量をセット

=>) STX) ESC) 'N') <n>) CS1) CS2) ADDR)
)) ここで、<n>) =) '0') 音量低)
))))))) =) '5') 音量中)
))))))) =) '9') 音量高)
<=) ACK) または) NAK)
)

ESC O 自動実行プログラムをセット

=>) STX) ESC) O) <program) name>) CS1) CS2) ADDR)
<=) STX) ESC) <Retcode>) CS1) CS2) ETX)
))) ここで、Retcode) =) 00) 成功した場合)
))))) =) 01) 失敗した場合)
=>) ACK) または) NAK)
)

ESC O/ROM　ROM 上のプログラムの自動実行をセット

=>) STX) ESC) O) <program) name>/ROM) CS1) CS2) ADDR)
<=) STX) ESC) <Retcode>/ROM) CS1) CS2) ETX)
))) ここで、Retcode) =) 00) 成功した場合)
))))) =) 01) 失敗した場合)
=>) ACK) または) NAK)
)

ESC P – 管理者パスワードをセット) (最大 8 文字)

=>) STX) ESC) P) <password>) CS1) CS2) ADDR)
<=) ACK) またはNAK)
)

ESC R – ターミナル ID

=>) STX) ESC) R) CS1) CS2) ADDR)
<=) STX) ESC) T) <terminal_id>) CS1) CS2) ETX)
=>) ACKまたはNAK)
)

ESC T – MR350 MKII にターミナル設定テーブルを書く

=>) STX) ESC) T) <termtable>) CS1) CS2) ADDR)
<=) STX) ESC) T) <Retcode>) CS1) CS2) ETX)
)) ここで、Retcode) =) 00) は成功)
)))) =) 01は承認されていないコマンド)
=>) ACK) または) NAK)
)

ESC U – MR350 MKII からファイルを転送

=>) STX) ESC) U) <filename>) CS1) CS2) ADDR)
<=) ACK) または) NAK) またはEOT(ファィルが見つからなかった場合))
)) ...) 転送が終了するまで次の二つのステップをループする)
)) =>) STX) ESC) 'Y') CS1) CS2) ADDR)
)) <=) STX) ESC) 'Y') <data>) CS1) CS2) ETX)
)) <=) ACK) またはNAK)
<=) STX) ESC) Z) CS1) CS2) ETX(ファイル転送の終了))
=>) ACK) または) NAK)
)

ESC V – MR350 MKII にデバイス設定テーブルを書く

=>) STX) ESC) V) <devtable>) CS1) CS2) ADDR)
<=) STX) ESC) V) <Retcode>) CS1) CS2) ETX)
=>) ACK) または) NAK)
)

ESC v – ターミナル ID とバージョン番号を得る

=>) STX) ESC) V) <devtable>) CS1) CS2) ADDR)
<=) STX) ESC) V) <terminal_id) and) ver>) CS1) CS2) ETX)
=>) ACK) または) NAK)
)

ESC X – プログラムの実行を開始

=>) STX) ESC) X) <filename>) CS1) CS2) ADDR)
<=) STX) ESC) X) <Retcode>) CS1) CS2) ETX)
)) ここで、<Retcode>)
))) 00) 開始が成功した場合)

))　　)　　01) ファイルがない場合)

=>) ACK) またはNAK)

注意: ここで、<filename> は、拡張子 .EXE なしのファイル名のみでなければなりません。

例えば、実行形式プログラムの名前が DRV30.EXE の場合、コマンドは、STX ESC X DRV30 CS1 CS2 ADDR です。

ESC X – ROM のプログラム実行を開始

=>) STX) ESC) X) <filename>/ROM) CS1) CS2) ADDR)

<=) STX) ESC) X) <Retcode>/ROM) CS1) CS2) ETX)

))　　)　　ここで、<Retcode>)

))　　)　　00) 開始が成功した場合)

))　　)　　01) ファイルがない場合)

=>) ACK) またはNAK)

注意:) ここで、<filename>) は、拡張子) .EXE) なしのファイル名のみでなければなりません。)

例えば、実行形式プログラムの名前が) DRV30.EXE) の場合、コマンドは、STX) ESC) X) DRV30) CS1) CS2) ADDR) です。)

4.5)) MV1100 指紋認識モジュールに追加された ESC コマンド)

ESC $D テンプレート・リストを得る

MV1100に保存されたすべてのテンプレートのリストは、MR350) MKIIのファイル $FP.LSTにアップデートされます。$FP.LSTの項目は$Txxxxxxxxxx.yyy) のフォーマットで、xxxxxxxxx) はテンプレートIDで、yyyはテンプレート・インデックスです。)

==>) STX) ESC) $D) CS1) CS2) ADDR))

<==) STX) ESC) $D) RESULT) CS1) CS2) ETX)

あるいは)

<==) NAK)

RESULT))　=) '0')　;) 成功)

))　　=) '1')　;) 失敗)

))　　=) '2')　;) ビジー)

))　　=) '3')　;) タイムアウト)

NAK) はコマンドフォーマットのエラーあるいはチェックサムエラーを意味します。)

)

MV1100テンプレート・リストのホストが取得するには、ESC) $Dコマンドを最初に出して、ESC)$D) コマンドが成功した場合に$FP.LST) ファイルをアップロードするためにESC) Uコマンドを出します。)

)

ESC $E テンプレート消去

コマンドはMV1100の指定したテンプレートを消去します。)

==>) STX) ESC) $E) xxxxxxxxxx.yyy) CS1) CS2) ADDR)

<==) STX) ESC) $E) RESULT) CS1) CS2) ETX)

あるいは)

<==) NAK)

xxxxxxxxxx))　　;) テンプレートID) ('1') –) '4294967294')

yyy))　　)　　;) テンプレート・インデックス('0') –) '255')

RESULT))　=) '0'　;) 成功)

))　　=) '1'　;) 失敗)

))　　=) '2'　;) ビジー)

))　　=) '3'　;) タイムアウト)

NAK) はコマンドフォーマットのエラーあるいはチェックサムエラーを意味します。)

)

ESC $F MR350 MKII にテンプレートを登録して保存

このコマンドは指定したIDを持つテンプレートをMR350) MKIIに予約されたファイル$FP.UPLに登録して保存します。)

==>) STX) ESC) $F) xxxxxxxxxx) CS1) CD2) ADDR)

<==) STX) ESC) $F) RESULT) <,QUALITY,) CONTENT>) CS1) CS2) ETX)

あるいは)

<==) NAK)

xxxxxxxxxx))　　;) テンプレートID) ('1') –) '4294967294')

RESULT))　=) '0'　;) 成功)

))　　=) '1'　;) 失敗)

))　　=) '2'　;) ビジー)

))　　=) '3'　;) タイムアウト)

RESULTが'0'の場合、MR350)MKIIはQUALITYとCONTENTフィールドを付けて応答します。)

QUALITYは、整数で、範囲は'000') –) '100'です。)

CONTENTは、整数で、範囲は'000') –) '100'です。)

NAK) はコマンドフォーマットのエラーあるいはチェックサムエラーを意味します。)
テンプレートのホストの取得については、ESC)$F) コマンドを最初に出して、そして ESC) $Fコマンドが成功した場合、$FP.UPL) をアップロードするためにESC) U) コマンドを出さなければなりません。)
)

ESC $G – MV1000 のテンプレートを登録して保存

このコマンドはMV1100で指定したIDを持つテンプレートを登録して保存します。)
==>) STX) ESC) $G) xxxxxxxxxx) CS1) CS2) ADDR)
<==) STX) ESC) $G) RESULT) <,QUALITY,) CONTENT>) CS1) CS2) ETX)
あるいは)
<==) NAK)
xxxxxxxxxx))　　;) テンプレートID) ('1') –) '4294967294')
RESULT))　=) '0')　;) 成功)
))　　=) '1')　;) 失敗)
))　　=) '2')　;) ビジー)
))　　=) '3')　;) タイムアウト)
RESULTが'0'の場合、MR350)MKIIはQUALITYとCONTENTフィールドを付けて応答します。)
QUALITYは、整数で、範囲は'000') –) '100'です。)
CONTENTは、整数で、範囲は'000') –) '100'です。)
NAK) はコマンドフォーマットのエラーあるいはチェックサムエラーを意味します。)
)

ESC $H – テンプレート検証

このコマンドはMR350) MKIIで予約された$FP.DNLファイルに保存されているテンプレートに対して指紋の検証を要求します。)

==>) STX) ESC) $H) CS1) CS2) ADDR)
<==) STX) ESC) $H) RESULT) <,INDEX,) SCORE>) CS1) CS2) ETX)
あるいは)
<==) NAK)
RESULT))　=) '0')　;) 成功)
))　　=) '1')　;) 失敗)
))　　=) '2')　;) ビジー)
))　　=) '3')　;) タイムアウト)
RESULT) =) '0'の場合、MR350) MKIIはINDEXとSCOREフィールドを付けて応答します。)
INDEXはテンプレートに一致する指標で、'000'から'255'の範囲です。)

SCOREは整数で、’000’から’100’の範囲です。
NAKはコマンドフォーマットのエラーまたはチェックサム・エラーです。
ホストがディスクに保存されたテンプレートに対して検証するには、MR350MKIIに予約されたファイル$FP.DNLにテンプレートをダウンロードするためにESCLコマンドを最初に出して、ESCLコマンドが成功した場合に指紋を検証するためにESC$Hコマンドを出さなければなりません。

ESC $I ID 検証

このコマンドはMV1100に保存された指定したIDを持つテンプレートに対して指紋を照合することを要求します。
==>STX ESC $I xxxxxxxxxx CS1 CS2 ADDR
<==STX ESC $I RESULT M,INDEX, SCORE> CS1 CS2 ETX
または
<==NAK
xxxxxxxxxx ;テンプレートID (‘1’–‘4294967294’)
RESULT =‘0’ ;成功
  =‘1’ ;失敗
  =‘2’ ;ビジー
  =‘3’ ;タイムアウト
RESULT=‘0’の場合、MR350MKIIはINDEXとSCOREフィールドを付けて応答します。
INDEXはテンプレートに一致する指標で、’000’から’255’の範囲です。
SCOREは整数で、’000’から’100’の範囲です。
NAKはコマンドフォーマットのエラーまたはチェックサム・エラーです。

ESC $L テンプレート・ダウンロード

このコマンドはMR350MKIIにあるテンプレート$FP.DNLファイルをMV1100にダウンロードし、MV1100のFlashメモリにテンプレートを保存します。
==>STX ESC $L CS1 CS2 ADDR
<==STX ESC $L RESULT CS1 CS2 ETX
または
<==NAK
RESULT =‘0’ ;成功
  =‘1’ ;失敗
  =‘2’ ;ビジー

)) =) ‘3’) ;) タイムアウト)

NAK) はコマンドフォーマットのエラーまたはチェックサム・エラーです。)

ホストのディスクに保存されているテンプレートをダウンロードするために、MR350) MKIIにあるテンプレート$FP.DNLファィルをダウンロードするには最初にESC) Lコマンドを出し、そしてESC)Lコマンドが成功した場合に、MV1100にダウンロードするためにESC) $Lコマンドを出します。)

)

ESC $S グローブ・スレッショルドをセット

このコマンドはMV1100に対してグローブ検証値をセットします。)

==>) STX) ESC) $S) THRESH) CS2) ADDR)

<==) STX) ESC) $S) RESULT) CS1) CS2) ETX)

または)

<==) NAK)

THRESH) =) ‘1’) ;) セキュリティが極めて高い)

)) =) ‘2’)) ;) セキュリティが高い)

)) =) ‘3’) ;) 普通のセキュリティ)

)) =) ‘4’) ;) セキュリティが低い)

)) =) ‘5’) ;) セキュリティが極めて低い)

RESULT)) =) ‘0’) ;) 成功)

)) =) ‘1’) ;) 失敗)

)) =) ‘2’) ;) ビジー)

)) =) ‘3’) ;) タイムアウト)

NAK) はコマンドフォーマットのエラーまたはチェックサム・エラーです。)

)

ESC $S グローブ・スレショルドを得る

このコマンドはMV1100に対してグローブ検証値を得ます。)

==>) STX) ESC) $S) CS2) ADDR)

<==) STX) ESC) $S) RESULT) <,THRESH>) CS1) CS2) ETX)

または)

<==) NAK)

RESULT)) =) ‘0’) ;) 成功)

)) =) ‘1’) ;) 失敗)

)) =) ‘2’) ;) ビジー)

)) =) ‘3’) ;) タイムアウト)

RESULT) =) '0'の場合、MR350) MKIIはTHRESHフィールドを持って応答します。)
THRESH)　=) '1')　;) セキュリティが極めて高い)
))　=) '2'))　;) セキュリティが高い)
))　=) '3')　;) 普通のセキュリティ)
))　=) '4')　;) セキュリティが低い)
))　=) '5')　;) セキュリティが極めて低い)
NAK) はコマンドフォーマットのエラーまたはチェックサム・エラーです。)
)

ESC U テンプレートのアップロード

このコマンドは指定したテンプレートをMV1100のFlashメモリからMR350) MKIIにある$FP.UPLファイルにアップロードします。)
==>) STX) ESC) $U) xxxxxxxxxx.yyy) CS1) CS2) ADDR)
<==) STX) ESC) $U) RESULT) CS1) CS2) ETX)
または)
<==) NAK)
xxxxxxxxxx))　;) テンプレートID) ('1') –) '4294967294'))
yyy)　)　;) テンプレート・インデックス('0') –) '255'))
RESULT))　=) '0')　;) 成功)
))　=) '1')　;) 失敗)
))　=) '2')　;) ビジー)
))　=) '3')　;) タイムアウト)
NAK) はコマンドフォーマットのエラーまたはチェックサム・エラーです。)
MV1100にある指定されたテンプレートをホストにアップロードするには、MR350) MKIIにあるファイルFP.UPLにテンプレートをアップロードするためにESC) $Uコマンドを最初に出して、そしてESC)$Uが成功したらファイル$FP.UPLをアップロードするためにESC) U) コマンドを出します。)
)

ESC $V バージョンを得る

このコマンドはMV1100からソフトウェアのバージョン情報を得ます。)
==>) STX) ESC) $V) CS1) CS2) ADDR)
<==) STX) ESC) $V) RESULT) <,Kx.xxx) Ay.yyy>) CS1) CS2) ETX)
または)
<==) NAK)
RESULT))　=) '0')　;) 成功)
))　=) '1')　;) 失敗)

))　　=) '2')　;) ビジー)
))　　=) '3')　;) タイムアウト)
RESULT) =) '0') の場合、MR350) MKIIはバージョン・フィールドを持って応答します。)
Kx.xxxx) Ay.yyyyはバージョン・フィールドで、x.xxxxはカーネルのバージョンを、
y.yyyyはアルゴリズムのバージョンを表します。)
NAK) はコマンドフォーマットのエラーまたはチェックサム・エラーです。)
)

第5章) プログラミングの方法)

プログラミングについては、1. MR350 MKII のプログラミング、と 2. MR350 MKII とホストコンピュータ間の通信プログラムのプログラミング、の二つの主要なパートがあります。以下のトピックはアプリケーションを速く、そして効率よく開発するためにユーザを助けるアプリケーションとユーティリティ・ライブラリについて説明しています。

5.1)) MR350) MKII のプログラミング)

MR350 MKII の OS はオープン・アーキテクチャであり、MS-DOS 互換のプラットフォームを提供しています。ユーザは入出力機能を完全にコントロールするために MR350 MKII 上で実行するプログラムを書くことができます、あるいはアプリケーション・プログラムに対する複雑なデータの保存、データの検証あるいはテーブルのルックアップ機能を持たせることができます。ユーザは標準 C、C++またはアセンブラ言語開発ツールをアプリケーション開発に使用することができます。第 3 章では、MR350 MKII の DOS/BIOS コールを詳しく説明しており、ユーザはアプリケーション・プログラムを実現するために直接この章を参照することができます。もしユーザがプログラミング言語に慣れていない場合、Unitech 社は Job Generator Pro (JobGen Pro)というプログラム生成ツールを提供しています。MR350 MKII は以下のプログラミングツールでアプリケーション・プログラムを書くことができます。

☑ JobGen Pro、DOS ベースのプログラム・ジェネレータで、プログラミングの能力はあまり必要ではありません。

☑ MS-DOS アプリケーション用 Microsoft C、バージョン 5.x、6.x、7.x、8.x

☑ MS-DOS アプリケーション用 Microsoft Visual C/C++ V1.52 以降

☑ MS-DOS アプリケーション用 Borland C/C++または Turbo C/C++

☑ MS-DOS アプリケーション用マクロアセンブラ

5.1.1 JobGen Pro によるプログラミング

JOB GENerator PROfessional (JobGen Pro)は MR350 MKII のアプリケーション開発のための MS-DOS ベースのソフトウェアです。JobGen Pro を使用するアプリケーション開発者は、プログラムコードを書かずに、トランザクション、データフィールドの属性、そして動作フローの定義をすることによって、簡単にアプリケーション・プログラムを開発することができます。JobGen Pro によって生成された実行形式のプログラムは MR350

MKII にダウンロードして実行することができます。

JobGen Pro のアプリケーションは、同じファイル名で拡張子が異なる三つのファイルで構成されています。

.CFG　ターミナルの環境、通信と I/O インターフェースの設定
.JB2　トランザクション、データフィールドと動作フローの定義
.EXE　定義したアプリケーションをリンクした後に JobGen Pro によって生成された実行プログラム

JobGen Pro の主な特徴は以下の通りです。

- ポップアップ・ウインドウを持つ会話型式のユーザ・インターフェース
- MR350 MKII のシステム設定を定義
- データフィールドとプロセスフローを定義
- MR350 MKII にルックアップ・データファイルと共に JobGen Pro アプリケーションをダウンロード
- MR350 MKII から PC へ収集したデータをアップロード
- JoBGen Pro アプリケーションの仕様を印刷
- PC 上でアプリケーションをシミュレーション

5.1.2 C/C++**よるプログラミング**

MR350 MKII のユニークな特徴は業界標準の PC でアプリケーション・プログラムの開発ができることです。プログラマは標準の C 言語や入手の簡単な Microsoft C や Borland C コンパイラを PC 上で使用することによって MR350 MKII のアプリケーションを開発します。コンパイルした実行形式のコードは、RS232/マルチポイント・ダウンロードを使用する RS485 リンクを通して MR350 MKII にダウンロードされます。

Borland/Microsoft の C/C++コンパイラは標準 C プログラミング言語のスーパーセットをサポートします。MR350 MKII は標準の ANSI C の機能を持っています。結果として、ある Microsoft の拡張または DOS 特有の機能はサポートされません。MR350 MKII ファームウェアによってサポートされる標準 C 関数の詳細は付録 A をご覧下さい。

Unitech 社はアプリケーションを開発するためにユーザにユーティリティ C ライブラリ(485LIB.C)を提供しています。350LIB.C は MR350 MKII の入出力装置のすべてとのインターフェースのライブラリを含んでいます。呼び出し方法とソースコードの詳細は第 3 章の各 BIOS/DOS コールの後ろにリストされています。このライブラリを簡単に使用するために、Unitech はユーザにサンプルプログラム(350TEST.C)を提供しています。ユーザはこのサンプルプログラムを直接変更することができます。これらのファイルのすべては

DEMO ディスクの”LIB”サブディレクトリに保存されています。

5.2)通信プログラムのプログラミング)

第 4 章では、MR350 MKII とホストコンピュータ間のマルチプロトコルについて詳しく説明しました。ユーザは通信プログラムの開発をこれに従って行うか、あるいはこれらをアプリケーション・プログラムに組み込むことができます。しかし、多くのユーザは通信プロトコルの仕様に従って通信プログラムを実現することはやや難しいでしょう。ですから、Unitech 社は詳細な通信プロトコルを知らなくてもユーザがアプリケーションを開発することのできる通信ユーティリティ・ライブラリを提供しています。デモ・デスクには、DOS バージョンと Windows バージョンの通信プログラム、通信ライブラリとサンプルプログラムが入っています。

DOS プラットフォームでは、DOS ベースの通信プログラムを開発するためにユーザに C ライブラリ(485LIB.C と SLIB.OBJ)を提供しています。また、485LIB.C を使用しているサンプルプログラム 485COM.C も提供しています。これらのすべてのファイルは DEMO ディスクのサブディレクトリ”DOSCOM”に保存されています。

Windows プラットフォームでは、Windows 通信プログラムの開発またはアプリケーション・プログラムに通信機能を組み込むために 16 ビット/32 ビット通信 DLL を提供しています。ユーザは Visual V/C++、Visual Basic または Delphi からこの DLL をコールすることができます。これらのファイルのすべては DEMO ディスクのサブディレクトリ”DLL”に保存されています。

5.3)Demo ディスクの内容)

以下の表は Demo ディスクの全体の内容を示しています。

MR350 MKII プログラミングのための C 関数ライブラリ

1) 350LIB.C/OBJ/H	MR350 MKII の C ライブラリソース/オブジェクト/ヘッダファィル
2) 350TEST.EXE	C ライブラリを使用したテストプログラムとソースコード

DOS ベース通信プログラムのプログラミングのための C ライブラリ

1) 485COM.EXE/C	通信ユーティリティ
2) 485LIB.C/OBJ	485COM.EXE の C 関数ライブラリ
3) SLIB.OBJ	アセンブラで書かれた低レベルシリアルポート I/I 関数

Windows ベース通信プログラムのプログラミングのための Windows DLL

1) 16BIT¥dll16	MS-Windows 3.1 用の 16 ビット DLL ライブラリ

16BIT¥multicom	MS-Windows 3.1 用の 16 ビット通信ユーティリティ
16BIT¥sample	16 ビット DLL を使用するための VC++(1.5)サンプルプログラム
2) 32BIT¥dll32	MS-Windows95/98/NT 用の 32 ビット DLL ライブラリ
32BIT¥multi32	MS-Windows95/98/NT 用の 32 ビット通信ユーティリティ
32BIT¥sample	32 ビット DLL を使用する VC++(5.0)サンプルプログラム
3) ¥Delphai	32 ビット DLL を使用する Delphai サンプルプログラム
4) ¥VB	32 ビット DLL を使用する Visual Basic サンプルプログラム

JobGen Lite、デモバージョン

1) ¥JGP_Lite¥disk1	JobGen Lite のディスク 1
2) ¥JGP_Lite¥disk2	JobGen Lite のディスク 2

付録 A.）MR350MKII の標準 C ライブラリ・ルーチン）

1. バッファの操作

memccpy()	memchr()	memcmp()	memicmp()
memmove()	memcpy()	memset()	movedata()

2. 文字の分類と変換

isalnum()	isalpha()	isascii()	iscntrl()
isdigit()	isgraph()	islower()	isprint()
ispunct()	isspace()	isupper()	isxdigit()
toascii()	tolower()	toupper()	

3. データ変換

atof()	atoi()	atol()	ecvt()
fcvt()	gcvt()	itoa()	ltoa()
strtod()	strtol()	strtoul()	ultoa()

4. ファイル処理

remove()	unlink()

5. ストリーム・ルーチン

clearer()	fclose()	fcloseall()	fdopen()
feof()	ferror()	fflush()	fgetc()
fgetchar()	fgetpos()	fgets()	fileno()
flushall()	fopen()	fprintf()	fputc()
fputchar()	fputs()	fread()	freopen(
fscanf()	fseek()	fsetpos()	ftell()
fwrite()	getc()	getchar()	gets()
getw()	printf()	putc()	putchar()
puts()	putw()	rewind()	scanf()
setbuf()	setvbuf()	sprintf()	sscanf()
ungetc()	vfprintf()	vprintf()	vsprintf()

6. 低レベルルーチン

close()	create()	eof()	lseek()
open()	read()	sopen()	tell()

write()

7. コンソールとポート I/O

cgets()	cprintf()	cputs()	cscanf()
getch()	getche()	inp()	inpw()
kbhit()	outp()	outpw()	putch()
ungetch()			

8. メモリ・アロケーション

alloca()	calloc()	free()	halloc()
hfree()	malloc()	msize()	realloc()
sbrk()	stackavail()		

9. プロセス・コントロール

Exit()

10. サーチとソート

bsearch()	12ind()	lsearch()	qsort()

11. 文字列操作

strcat()	strchr()	strcmp()	strcmpi()
strcpy()	strcspn()	strdup()	strerror()
stricmp()	strlen()	strlwr()	strncat()
strncmp()	strncpy()	strnicmp()	strnset()
strpbrk()	strrchr()	strrev()	strset()
strspn()	strstr()	strtok()	strupr()

12. MS-DOS インターフェース

bdos()	FP-OFF()	FP-SEG()	int86()
int86x()	intdos()	intdosx()	segread()

13. 時間

asctime()	clock()	ctime()	difftime()
ftime()	gmtime()	localtime()	mktime()
time()	tzset()		

14. その他

abs()　　div()　　labs()　　ldiv()