パソコンをTSS端末にするためのCプログラム(1)

武政, 尹士
佐賀大学理工学部物理学教室

正木, 延幸
佐賀大学理工学部物理学教室

https://doi.org/10.15017/1470178

出版情報：九州大学大型計算機センター広報. 24 (1), pp.19-46, 1991-01-25. 九州大学大型計算機センター
バージョン：
権利関係：

パソコンをTSS端末にするためのCプログラム(1)

*武政 尹士，*正木 延幸

1．はじめに

数年前より，大学の研究室から自分の所属する大学の計算機センターは言うにおよばず，全国７大学の共同利用大型計算機センターのマシンを自由に利用出来るようになりました．更には，これらの計算機センターを経由して，国内ばかりではなく海外の研究者と電子メールを交換したり，また直接海外のコンピュータをアクセス出来るようになっています．このような時，多くの人々は日頃使い慣れたパソコンを大型計算機のＴＳＳターミナル（端末）として利用しています．この時，パソコンを端末として利用するためのソフトウェアが必要です．これがターミナル エミュレータと呼ばれているものです．このようなターミナル エミュレータは，今日まで数多く開発されて来ました．そして，それらの中には通信速度の点を除けば，専用端末より使い易く，かつ専用端末より便利な機能を多くもっているものが発表されています．著者の一人（T.T.）も，ここ数年九州大学大型計算機センターのコンピュータＦＡＣＯＭ ７８０／２０をターゲットとして，ＮＥＣのＰＣ－９８０１シリーズのパソコン上で動作する無手順（ＴＴＹ）通信方式のインテリジェントＴＳＳ端末エミュレータを開発してきました［１］．そこでは，開発言語としてＣが用いられています．

その間に，主に東芝のダイナブックやＪ－３１００シリーズのパソコンを使用されておられる方々から，私たちの開発してきたエミュレータをそれらのパソコンに移植してしてほしい由の要請が少なからずありました．しかし，残念ながら私たちはこのような仕事が本職でありませんので，そのような要請に対して「ソース プログラムを公表していますので，それを解読して他機種でも動作するようにして下さい」とお答えしてきました．エミュレータ開発の初期の段階では，プログラミング構造も簡単であったためにこれでよかったのですが，機能が多くなるにつれてこれが難しくなってきました．

と言うのは，私たちのエミュレータ開発は最初から厳密な全体的設計を行なってプログラミングして行ったわけではなく，ユーザーの皆様からの色々な要望や，私たち自身の個人的考えを元に，後から後へと種々の機能を継足して来ました．そのために，今やソース コードは２０，０００行を超えプログラミングは迷路のように入り組んでいます．そして，現在では開発者以外の人々にとってはソース コードを入手しようとも解読が不可能に近い状態になっています．

そこで上のような要望に答える１つとして，今回より数回にわたって，ＴＴＹ型ターミナル エミュレータの基本的構造の説明や，それに付加することによって便利になる色々な機能を実現するための基本的考え方とＣ言語を用いてのプログラミング法を，実際にソースプログラムを掲載しながら，初心者向けに出来るだけやさしく，かつ丁寧に解説して行きたいと思っています．基本的エミュレータに付加する機能としては，日本語対応，ファイル転送，擬似フルスクリーン，

平成２年１１月３０日受理

* 佐賀大学 理工学部 物理学教室

受信データのスクロール，ＰＦＤ，グラフィック描画，グラフィック ハード コピー，アドレス付きの文字を端末に送ってくるグラフィック ソフトへの対応，ＵＮＩＸマシンへの対応等々があります．そして，このシリーズをこれからターミナル エミュレータ プログラムを作成されようとしておられる方や，既に所有しているターミナル プログラムを自分の環境に合うように手を加えたいと思っておられる読者の方々の一助にしたいと思っています．

なお，プログラミングにおいては高速性を追及するためにＰＣ－９８シリーズのパソコンのもつ機能に依存するプログラミングをすることがありますが，他機種にもこれらに対応する機能が見つかると思いますので，原則的にはこれから説明していく全ての機能が他機種へ移植出来るものと思っています．例えば富士通のパソコンＦＭＲシリーズにたいしては文献２が，またダイナブック等のＩＢＣ ＰＣ互換機にたいしては文献３が参考になると思います．

本稿はＭＳ－ＤＯＳとＣ言語に関して基本的な知識を既にもっておられる方を対象にして書かれています．もしそうでない方でＣ言語やこのようなプログラミングに興味を持っておられる方は，ＭＳ－ＤＯＳに関しては村瀬康治著「入門ＭＳ－ＤＯＳ」［４］を，Ｃ言語に関してはL. Hancock & M. Krieger著／三浦明美訳「Ｃ言語入門」［５］を読まれることをお勧めします．どちらの書籍も初心者を対象にわかりやすく説明した名著であります．なお，既にＢＡＳＩＣを修得されている方で，Ｃへの移行を考えられておられる人には文献６の書籍が役に立つと思われます．

第２節に，今回の開発に使用したハードウェアとソフトウェアの環境を記しています．第３節に本シリーズで作成する端末エミュレータが使用する通信制御仕様が簡単に説明されています．第４節で最も簡単なターミナル エミュレータの構造と実際のプログラミング法を解説しています．そして，第５節では第４節で作ったエミュレータを日本語対応にするためのプログラミングを説明しています．第６節では，ソース プログラムの２つのコンパイラによるコンパイル法を詳しく述べています．

２．ハードウェアとソフトウェアの環境

現在使用中のハードウェアは，ＮＥＣのＰＣ－９８０１ＶＸです．メモリーは６４０キロバイトとなっています．本体には，４メガのＲＡＭディスクが設定されています．これから紹介していくエミュレータを快適に使用するためには，ＲＡＭディスクを設定されておかれることをお薦めします．プリンターはＮＥＣのＰＣ－ＰＲ２０１Ｖです．ディスプレイは，６４０×４００ドットの専用高解像度のものが必要です（カラー ディスプレイが望ましい）．また，当然でありますが，モデムがＲＳ－２３２Ｃケーブルでもってパソコン本体と結ばれています．

これから説明していくエミュレータ プログラムでは，パソコンのモデルに依存する特殊な機能は使用しませんので，ＰＣ－９８０１シリーズの全機種（ただし，ＸＡとＬＴは除く）とＥＰＳＯＮのＰＣ－２８６とＰＣ－３８６の両シリーズの全機種で動作すると思われます．ただし，ＰＣ－９８０１（無印）とＰＣ－９８０１Ｅには漢字ＲＯＭが装着されている必要があり，ディスク ドライブは２台が接続されている必要があります．プリンターに関しても，ＰＣ－ＰＲ１０１，ＰＣ－ＰＲ２０１シリーズか，これらのエミュレーション モードを持つ全ての機種が利用可能です．

本稿で使用するオペレーティング システム（ＯＳ）はＭＳ－ＤＯＳのＶｅｒ．３．３Ｂ［7］ですが，Ｖｅｒ．３．１以上の版のものであればエミュレータは正しく動作します．ただ，ＭＳ－ＤＯＳはＶｅｒ．３．１のＰＳ９８－１２７－ＸＸＸ以降では，プリンター ドライバ（フ

ファイル名 ＰＲＩＮＴ．ＳＹＳ）とＲＳ－２３２Ｃドライバ（ファイル名 ＲＳＤＲＶ．ＳＹＳ）が別ファイルになっていますので，ＭＳ－ＤＯＳ構築ファイル"ＣＯＮＦＩＧ．ＳＹＳ"への両ドライバの登録が必要です．日本語を含むデータをホストコンピュータとやり取りするには，ＡＴＯＫ［８］や松茸［９］などの日本語入力フロントエンドプロセッサが必要となってきます．それらの使用方法ならびに"ＣＯＮＦＩＧ．ＳＹＳ"への登録方法は，それぞれのマニュアルをご覧下さい．その他の環境設定は特に必要ありませんが，"ＣＯＮＦＩＧ．ＳＹＳ"ファイルにおいて"ＢＵＦＦＥＲＳ＝２０"以上の指定を行なっておくと，ファイルの入出力が速くなります．

処理系としては，ＭＳ－ＣのＶｅｒ．４．０［１０］以上，もしくはＴｕｒｂｏ Ｃ Ｖｅｒ．１．５［１１］以上を使用して下さい．今回のソースプログラムはこれらより以下の版のコンパイラでもコンパイル出来ますが，次回以降のプログラムで上記の版よりサポートされたＣの関数を使用しますので，古いバージョンのコンパイラを使用されておられる方は，早めに新しいバージョンに乗換えておいて下さい．ソースファイルの作成や修正には，Ｍｉｆｅｓ ９８［１２］等のＭＳ－ＤＯＳ上で動くエディタが必要です．

３．パソコン ターミナルと通信仕様

今日では，ほとんど全てのパソコンにＲＳ－２３２Ｃ規格のコネクタが配置されています．ＲＳ－２３２Ｃとはコンピュータと周辺装置とのインターフェイスの規格です．このＲＳ－２３２Ｃコネクタにモデムを接続し，公衆電話回線や専用回線を通じてホストコンピュータを呼出すと大型コンピュータの専用ターミナルの代りとして，パソコンを使用することが出来ます．このようにパソコンを大型コンピュータにつないで使用することが，今日では一般化していますが，この主なる理由しては

（１） パソコンが非常に安くなったこと，
（２） パソコンをターミナルとして使用する場合，ソフトウェアでもってさまざまな便利な機能を必要に応じて追加することが出来る（インテリジェント端末化），

などが考えられます．

本シリーズでは，インテリジェント端末をパソコンで作成することを目的として，そのソフトウェアの実際の表現法について述べていく予定です．このインテリジェント端末の通信制御仕様は，表１に示すとうりです．

表１ 目的とするインテリジェント端末の通信制御仕様

通信速度	９６００ビット／秒以上
通信方式	全２重
同期方式	調歩同期
制御手順	無手順（ＴＴＹ手順）

この表に出てくるデータ通信に関する用語について，簡単に説明しておきます．

① 通信速度 ： １秒間に送ることの出来るデジタルデータの量です．単位はビット／秒（ｂｐｓ），またはキャラクタ／秒（ｃｐｓ）で表せます．

② 全２重通信 ： 通信を行なっている双方が，同時にデータを送受信出来る通信方式です．これに対して，トランシーバのように片方ずつ交互に通信する方式を半２重通信と言います．現在，パソコン通信の分野では全２重通信が一般的で半２重通信はほとんど見られません．

③ 調歩同期 ： パソコンとホストコンピュータとの通信のタイミングをとる（これを同期をとると言う）時に，情報ビットの前後にスタートビットをストップビットを付加して送信し同期をとる方式です．この方式は回路構成が簡単なために装置が安価に出来ます．よって，一般のパソコン通信でよく用いられています．

④ 無手順 ：無手順とは，キーボードよりタイプしたデータがその瞬間にホストコンピュータへ流れる方式のことです．この方式では，データのやり取りのための制御も，誤りのための制御も行なわれません．そのために「たれ流し」とか「ＴＴＹ手順」と呼ばれています．
ＴＴＹとはむかしあったテレタイプライター（Ｔｅｌｅ Ｔｙｐｅｗｒｉｔｅｒ）と言う名のホストコンピュータとの通信に関しては本質的にキーボード機能しかもたない端末名に由来しています．

通信に関する基本的な用語の内で上で説明した以外のものとして，"半２重モード"と"全２重モード"と言うのがあります．これはパソコン端末のキーボードからキーインされたデータをホストコンピュータへ送信する時の処理の違いのことです．つまり，"半２重モード"とはパソコン端末よりキーインされたデータをホストコンピュータへ送信すると同時に，パソコンのディスプレイ画面にも出力するモードのことです．これはまた"ローカル エコー バック モード"とも呼ばれています．これに対して，パソコン端末よりキーインしたデータをホストコンピュータへ送信すると，ホストコンピュータがそのデータを端末側にエコー バックするモードのことを"全２重モード"と言います．このモードではパソコン側でこの送り返されてきたデータをディスプレイ画面に出力することによって，ユーザが自分の入力したデータを確認するようになっています．このモードですと低速の通信速度でホストコンピュータと接続されている場合には，エコー バックが遅くて使い物になりません．なお，九大大型計算機センターのＭＳＰは半２重モードであり，ＵＴＳ［１３］は全２重モードです．この"半２重モード"と"全２重モード"は，上で説明したモデム間の論理的送受信路の方式である"半２重通信方式"と"全２重通信方式"とは別ものであることに注意して下さい．

今後，本シリーズにおいて出てくるデータ通信に関する用語は，そのつど出来るだけ説明するように努める予定です．しかし，原稿のページ数の制約もありますので，より詳しい説明やデータ通信一般に関する解説は他の文献ゆずることもあります．そのような時は，例えば文献１４や１５を参照されて下さい．

４．ダム ターミナル プログラム

ダム ターミナルとは単にホストコンピュータより受信したデータをディスプレイに表示し，パソコンのキーボードより入力された文字をホストコンピュータへ送信する機能しかもっていないターミナルのことです．これは図１に示されたような機能と構成でもって実現されます．しかし図１のなかに，ターミナル エミュレータの本質的なものの全てが入っています．あとはこれ

に種々の便利な機能を付加えて行くと，インテリジェント ターミナル エミュレータなるものが出来上がるわけです．

念のために，図1の流れを説明しておきます．まず第1に，ＲＳ－２３２Ｃポートを初期化してＲＳ－２３２Ｃ通信機能が動作するようにします．次にホストコンピュータからＲＳ－２３２Ｃポートにデータが送られて来ているかどうかをチェックします．もし受信データがあるとそれらの全てをＲＳ－２３２Ｃポートより読み出し，同時にそれらをディスプレイ画面に書き出します．この作業後直ちにＲＳ－２３２Ｃポートに，この作業中に送られてきたデータがないか調べに戻ります．ここまでで，ホストコンピュータから送られてくるデータの処理が出来ます．

もしＲＳ－２３２Ｃポートに受信データがない時は，パソコンのキーボードから入力されたデータがキーボード バッファにあるかどうかをチェックします．キーボードからの入力データがないときは，ＲＳ－２３２Ｃポートの受信データのチェックに戻り，図1の上半分に示されている処理を行ないます．キーボードからの入力データが有る時は，それを読み出すと同時にディスプレイにも書き出します．この作業が終ると直ちにＲＳ－２３２Ｃポートのチェックに戻ります．以下，この過程の繰返しであります．この説明でお分かりのように，今回のエミュレータは"半２重モード"（"ローカル エコー バック モード"）になっています．

このように，このエミュレータは常にＲＳ－２３２Ｃポートに受信されたデータの処理を最優先に行なっていることが分かって頂けると思います．このアルゴリズムによって高速の通信速度にも追従出来るようになります．もちろんＲＳ－２３２Ｃポートから高速にデータを読み出すような機能と同時に読み出したデータを高速にディスプレイ画面に書き出す機能をもった関数を作る必要はありますが．

上で説明した機能の他に端末プログラムとして必要なものに，ブレーク信号をホストコンピュータへ送信する機能があります．これを加えるとターミナル エミュレータ プログラムに必要な機能の全てが揃ったことになります．それらを図1に従って適当に組合せるとダム ターミナルの出来上がりです．そのためのソース リストが，リスト1からリスト4までに載せてあります．リスト1とリスト2に載っているものは，インクルード ファイルと呼ばれるファイルであります．これはコンパイル時にリスト3とリスト4のファイルに取込まれることによって，リスト1と2で定義された各種のパラメータやマクロがソースファイルにおいて置き換えられます．リスト1では一般的なパラメータの定義と関数の型の宣言がなされています．リスト2では，ＲＳ－２３２Ｃ関係のパラメータが定義されています．これらの意味は関数ｒｓ＿ｉｎｉｔ（）の説明のところで詳しく述べます．

リスト1から4までのソースプログラムをコンパイルして出来る実行形式ファイルは，例えば九大大型計算機センターのＭＳＰ用のエミュレータとして正しく動作します．また，"ｐａｒａｍ．ｈ"なるファイルで定義されているＲＳ－２３２Ｃ関係のパラメータを適当に変えることによって，ほとんど全ての計算機のダム ターミナルとして使用可能です．ただし，ホストコンピュータがエコー バック機能をもっている時は，パソコンのディスプレイ画面に同じ文字が２つならんで表示されます．この場合の解決策は次回以降の稿で説明します．

なお，第1節にも書きましたがこれから紹介していきますプログラムは高速性を追求するために，多くの関数でＰＣ－９８シリーズのＭＳ－ＤＯＳとＰＣ－９８のハードウェアに強く依存するプログラミングになっていることに注意して下さい．よって他機種に移植する時は，それぞれのハードに関する解説書とそれぞれの機種に特有なＭＳ－ＤＯＳの機能の使用法を良く調べて下さい［例えば文献２もしくは３を参照］．繰返しになりますが，本稿はエミュレータの基本的構造を解説して行くものであって，移植性には特に気を配ってはいません．一般にエミュレータの色々な機能の高速性を追求すると，どうしてもパソコンの機種に依存するプログラミングになっ

てしまいます．

以下でリスト３とリスト４の中に現れる各々の関数のもつ機能を，簡単に説明します．

リスト３で現れる関数：

（１）ｍａｉｎ（）

関数ｒｓ＿ｉｎｉｔ（）をコールしＲＳ－２３２Ｃポートの初期化とパラメータの設定を行なった後，このエミュレータの本体である関数ｅｍｕｌａｔｏｒ（）をコールします．もしＲＳ－２３２Ｃポートの初期化と設定が正しく行なわれなかった時は，ＭＳ－ＤＯＳにぬけます．

（２）ｅｍｕｌａｔｏｒ（）

このエミュレータの心臓部です．ＲＳ－２３２Ｃポートを通じてのホストコンピュータからの受信データの処理（関数ｒｅｃｉ＿ｔｓｓ（ｌｅｎ）がこの機能を担う）と，パソコン端末からの入力データの送信（関数ｓｅｎｄ＿ｔｓｓ（ｃｏｄｅ）がこの機能を担う）の制御を行なっています．この際，図１で説明したように受信データの処理を最優先で行なうようにプログラミングされています．

[CTRL] ＋ [B] でもって関数ｒｓ＿ｂｒｅａｋ（）をコールして，ホストコンピュータにブレーク信号を送信します．一方，[CTRL] ＋ [Q] でもってエミュレータを終了して，ＭＳ－ＤＯＳに戻るようになっています．ここで [CTRL] ＋ [B] とは，[CTRL] キーを押しながら [B] キーを押すことを意味します（ [CTRL] キーと [B] キーの同時押下）．

（３）ｒｅｃｉ＿ｔｓｓ（ｌｅｎ）

関数ｒｓ＿ｒｅｃｅｉｖｅ（）を用いて，ＲＳ－２３２Ｃバッファ内にある"ｌｅｎ"個の１バイトデータを"ｉｎ＿ｃｏｄｅ"に読み出します．次にその読み出したデータを関数ｓｙｓ＿ｐｕｔｃｈａｒ（ｃｏｄｅ）をコールすることにより，ディスプレイに書き出します．ただし"ｉｎ＿ｃｏｄｅ"がヌル（０ｘ０：１６進ゼロ）のときは，スキップします．（リスト３の行番号６３）

（４）ｓｅｎｄ＿ｔｓｓ（ｃｏｄｅ）

端末のキーボードより入力された１バイトデータ"ｃｏｄｅ"を関数ｓｙｓ＿ｐｕｔｃｈａｒ（ｃｏｄｅ）を用いてディスプレイに書き出すと同時に，関数ｒｓ＿ｓｅｎｄｃ（ｃｏｄｅ）をコールすることによってホストコンピュータに送信します．

リスト４で現れる関数：

（１）ｒｓ＿ｉｎｉｔ（）

Ｃの子プロセスを実行させる関数ｓｐａｗｎｖ［１０，１１］を用いて，ＭＳ－ＤＯＳの外部コマンドである"ＳＰＥＥＤ．ＥＸＥ"［７］を子プロセスとして実行させ，ＲＳ－２３２Ｃポートの初期化と，その起動を行ないます．子プロセスが正しく実行された時，この関数は"０"を返し，子プロセスが実行できなかったときは"－１"を返します．"ＳＰＥＥＤ．ＥＸＥ"に渡される各種のパラメータはヘッダファイル"ｐａｒａｍ．ｈ"

（リスト２）で定義されています．つまり，初期化&起動されるのは標準ＲＳ－２３２Ｃポート（ｒ０）であり，ボーレイト９６００ビット／秒（ｂｐｓ），キャラクタ長７ビット（ｂ７），偶パリティ（ｐｅ），ストップビット１（ｓ１），フロー制御ＯＮ（ｘｏｎ）です．これらのパラメータの詳しい意味と，パラメータの他の値での設定形式は文献７に詳しく説明されていますので，そちらを参照して下さい．
このプログラムでは"ＳＰＥＥＤ．ＥＸＥ"なるファイルは，実行形式になった本エミュレータ プログラムの入ったディスクのルートディレクトリになくてはなりません．また，システムにＲＳ－２３２ＣドライバＲＳＤＲＶ．ＳＹＳ）［７］が組込まれている必要があります．この２点を注意して下さい．

（２）ｒｓ＿ｌｅｎｇｔｈ（）

Ｃのソフトウェア割り込み関数ｉｎｔ８６ｘ［１０，１１］を用いてＭＳ－ＤＯＳのサポートするＲＳ－２３２ＣのＢＩＯＳ［Ｂａｓｉｃ Ｉｎｐｕｔ Ｏｕｔｐｕｔ Ｓｙｓｔｅｍ］をコール（割り込み番号 ０ｘ１９）することにより，ＲＳ－２３２Ｃ受信バッファ内の受信データ長を読み出し，その値を返します．この関数と次の２つの関数ｒｓ＿ｒｅｃｅｉｖｅ（）とｒｓ＿ｓｅｎｄｃ（ｃｏｄｅ）についての詳細は，文献１６を参照して下さい．

（３）ｒｓ＿ｒｅｃｅｉｖｅ（）

Ｃのソフトウェア割り込み関数ｉｎｔ８６ｘを用いてＭＳ－ＤＯＳのサポートするＲＳ－２３２ＣのＢＩＯＳをコールすること（割り込み番号 ０ｘ１９）により，ＲＳ－２３２Ｃ受信バッファにある受信データを１バイト読み出し，その値を返します．この関数は受信データがなければ受信するまで待ちますので，使用時に注意が必要です．

（４）ｒｓ＿ｓｅｎｄｃ（ｃｏｄｅ）

Ｃのソフトウェア割り込み関数ｉｎｔ８６ｘを用いてＭＳ－ＤＯＳのサポートするＲＳ－２３２ＣのＢＩＯＳをコールすること（割り込み番号 ０ｘ１９）により，送信データ"ｃｏｄｅ"をＲＳ－２３２Ｃポートに出力します．ＲＳ－２３２Ｃの初期化で，データ長が７ビットに設定されているなら，"ｓｈｉｆｔ ｉｎ（０ｘ０ｆ）"コード，もしくは"ｓｈｉｆｔ ｏｕｔ（０ｘ０ｅ）"コードの付加と，８ビットデータの７ビット変換を行なってから送信します．
ここで，"ｓｈｉｆｔ ｉｎ（ＳＩ）"コードと"ｓｈｉｆｔ ｏｕｔ（ＳＯ）"コードとは，７ビット データでローマ字（アスキー文字）と半角のカタカナが混在した時，
両者を切りわけるコードです．すなわち，ＳＩに続くコードはローマ字と，ＳＯに続くコードはカタカナと約束されています．つまり，この関数を用いるとデータ長が７ビットの場合でも，半角のカタカナの送受信が正しく行なわれるようになっています．

（５）ｒｓ＿ｂｒｅａｋ（）

Ｃのポート出力関数ｏｕｔｐ［１０，１１］を用いて，ＲＳ－２３２Ｃのポート番号"０ｘ３２"へ"０ｘ３ｆ"なるデータを出力します．これによってＲＳ－２３２Ｃポートへブレーク信号が出力されます．適当なポーズをおいて，ブレーク信号出力状態を通常状態

に戻す必要があります．これはポート番号"０ｘ３２"へ"０ｘ３７"なるデータを出力することによって行なわれます．これらのポート番号と出力データについての詳細は，ＰＣ－９８０１シリーズのハードウェア マニュアル，例えば文献１７を見て下さい．
高速のＣＰＵをもつパソコン使用時には，ｐａｕｓｅ（３０）ではポーズ時間が不足します．そのようなパソコンを使用する時は，ヘッダファイル"ｔｓｓ＿ｍｉｎ．ｈ"の内で定義されているＢＲＥＡＫ＿ＴＩＭＥを"１"より大きな適当な値に設定してください．また，この関数が呼ばれたことを確認するために，ベルを鳴らすようにしています（リスト４の行番号８３）．

（６）ｐａｕｓｅ（ｔｉｍｅｓ）

適当な時間だけポーズする関数です．機種によって，パラメータｉ＿ｍａｘの値を適当に変更して下さい．

（７）ｉｎｋｅｙ（）

Ｃの標準的なＭＳ－ＤＯＳシステムコール関数ｉｎｔｄｏｓ［１０，１１］を用いて，キーボード バッファに入力された文字を取りだし，そのコードを返します．文字が入力されていない時は，"０"が返されます．ＭＳ－ＤＯＳシステムコールについての詳細は，文献１８を見て下さい．

（８）ｃｌｓ（）

Ｃのソフトウェア割り込み関数ｉｎｔ８６［１０，１１］を用いてＭＳ－ＤＯＳの拡張システムコール（割り込み番号 ０ｘＤＣ）を行なって，ディスプレイ全体をクリアします．ＭＳ－ＤＯＳの拡張システムコールについての詳細は，文献１８と１９を見て下さい．

（９）ｓｙｓ＿ｐｕｔｃｈａｒ（ｃｏｄｅ）

Ｃのソフトウェア割り込み関数ｉｎｔ８６を用いてＭＳ－ＤＯＳの拡張システムコール（割り込み番号 ０ｘＤＣ）を行なって，ディスプレイに１バイトデータ"ｃｏｄｅ"を出力します．漢字データを出力する場合は，シフトＪＩＳコードの第１バイト，第２バイトの順に入力する必要があります．

リスト４に現れる関数の多くはＮＥＣ版のＭＳ－ＤＯＳのＢＩＯＳコールとＰＣ－９８拡張システムコールを使用しています．よって，他機種への移植の際はそれぞれの機種で上記の機能を再現するＢＩＯＳコールを使用して下さい［例えば，文献２もしくは３を参照］．また，どうしてもＭＳ－ＤＯＳの標準システムコールのみでＲＳ－２３２Ｃ関係の機能を実現しなければならない場合には，文献２０にそのサンプル プログラムが載っています．この時はパソコンの機種依存性はなくなりますが，エミュレータの高速性が失われます．

５．日本語対応バージョン

さて，第３節で作られたダム ターミナル エミュレータに，今後色々な機能を付加えてインテリジェント化していきたいと思っています．今回はその第１弾として，ダム ターミナルを日本

語対応版にしてみたいと思います．ここで言う日本語とは，漢字，ひらかな，全角記号等のいわゆる全角文字を指しています．

アスキー文字は１バイトのデータで表現されていますが，漢字等の全角文字は２バイトで１文字が表現されているため，２バイトを１文字として取り扱う必要があります．しかし，第３節で作った文字データを取り扱う関数は，受信にせよ送信にせよ全て１バイト単位でしか取り扱えません．その上，送受信のデータの中に日本語コードとアスキーコードのアルファベットが混在している場合には，そのデータの中から日本語を識別しなくてはなりません．これらのために，日本語データを取り扱う時は第３節のプログラミングに比べて少しばかり複雑になります．

漢字コードはＪＩＳ規格のもとで定められています．そこでは，漢字シフトコードＫＩ（漢字ＩＮ）とＫＯ（漢字ＯＵＴ）を使って１バイト系コードと日本語が混在しても識別出来るようになっています．すなわち，ＫＩコードの後に２バイトの日本語コードが続き，ＫＯコードによって１バイト系のコードに戻します．ＪＩＳ規格ではエスケープ（ＥＳＣ）コードに続く２バイトのコード（エスケープ シーケンス）でＫＩとＫＯが定義されています．表２に一般によく使用されている日本語ＫＩ，ＫＯコードをまとめてあります．

表２．漢字コードと対応する漢字シフトコード

漢字コード	漢字シフトコード名称	ＫＩコード	ＫＯコード
ＪＩＳ	ＪＯＩＳ	ESC, 0X24, 0X40	ESC, 0X28, EX48
	旧ＪＩＳ	ESC, 0X24, 0X40	ESC, 0X28, 0X4a
	新ＪＩＳ	ESC, 0X24, 0X42	ESC, 0X28, 0X4a
シフトＪＩＳ			

表２の最下段のカラムに記されているシフトＪＩＳコードは，ＭＳ－ＤＯＳが採用している漢字コードです．このコード体系を採用すると，日本語文字はその第１バイト目をチェックすることによって１バイト系の文字と区別することが可能になってきます．そのために，ＫＩ，ＫＯコードは必要なくなっています．

さて，パソコンのＯＳであるＭＳ－ＤＯＳが日本語コードとしてシフトＪＩＳコードを採用しているために，ホストコンピュータがシフトＪＩＳコード以外の漢字コードを採用している時は，シフトＪＩＳコードとＪＩＳコード間の変換作業が必要となってきます．つまり，ホストコンピュータが漢字コードとしてＪＩＳコードを使用している時は，ホストコンピュータからパソコンに送られてくる日本語データはＫＩコードを取除いた後，シフトＪＩＳコードに変換しないと日本語が正しくディスプレイに出力されません．逆に，パソコンから日本語データをホストコンピュータに送信する時はシフトＪＩＳコードからＪＩＳコードへの変換作業を行ない，その後漢字２バイトデータの前にＫＩコードを付けて送信することが必要となってきます．ＪＩＳコード－

シフトＪＩＳコード間の変換法は，文献２１に詳しく解説されていますから参照して下さい．

一方，九大大型計算機センターのＵＴＳ［１３］ようにホストコンピュータが漢字コードとしてシフトＪＩＳを採用している時［２２］は，１バイトデータと漢字データの違いを意識する必要がなくなります．パソコンのキーボードより入力されたデータは，何も加工することなくホストコンピュータへ送信するだけでよく，受信時も単に受取ったデータをそのままディスプレイに書き出すだけてよくなります．よって，この場合はリスト１から４までのプログラムのままで日本語のデータに正しく対応出来ます．

このような考えのもとでダム ターミナル用の関数を変更したものと，日本語データの処理のために新しく追加した関数のソースやパラメータ用のステートメントをリスト５からリスト８に載せています．つまり，リスト５にはリスト１に追加すべきステートメントを，リスト２に追加すべき変数の定義をリスト６に載せています．そして，リスト７にはリスト３に追加すべき関数と取り換えられるべき関数のソース プログラムを載せており，リスト８にはリスト４に載っているソースに追加すべき関数を載せています．

ここでは漢字シフトコードとしては，九大大型計算機センターのＭＳＰが採用しているＪＯＩＳコードを取っています．よって，ＪＯＩＳ以外の漢字コードに対応するためには，ヘッダー ファイル"ｐａｒａｍ．ｈ"の内の"ＫＩ＿ＣＯＤＥ２"と"ＫＯ＿ＣＯＤＥ２"の値を，例えば表２に従って適当なものに変える必要があります．

更に，この版では受信データと送信データが一目で区別出来るように，ｃｏｌｏｒ（ｃ）なる関数を用いて，受信データを緑の文字（リスト７の行番号１９）で，送信データを白い文字（リスト７の行番号９５）でディスプレイに表示するようにしています．

これらの新しいソースファイルをコンパイルすると，日本語のデータがアスキーデータと同じようにホストコンピュータとやり取りできます．

以下で，リスト７とリスト８に載っている関数の機能を簡単に説明していきます．

リスト７に載っている関数：

これらはリスト３に追加する関数と置き換える必要がある関数です．

（１）ｒｅｃｉ＿ｔｓｓ（ｌｅｎ）

内容が変更された関数です．
ホストコンピュータからの受信データの内に"ＥＳＣ"コードがないかをチェックし，もしあった時は，"ｆｇ＿ｒｅｃｉ＿ｅｓｃ"なる変数を"１"にセットし，"ＥＳＣ"コードが受信されたことを記憶します．（このような，ある状態を記録した変数をフラグと呼びます）．そうでない時は，関数ｏｕｔｐｕｔ＿ｃｒｔ（）の内で，ダム ターミナルの時と同じような処理をします．
一方，"ＥＳＣ"コードを受信すると，続いて２バイトの漢字識別コードが送られて来ますので，それらの余分なコードを取除く処理が必要となってきます．これは関数ｅｓｃ＿ｓｅｑ（ｃｏｄｅ）を呼ぶことによって行なわれます．

（２）ｅｓｃ＿ｓｅｑ（ｃｏｄｅ）

新しく追加された関数です．
この関数によって，"ＥＳＣ"のあとに送られて来るＫＩとＫＯに関する２バイトのコードが取除かれます．この時ＫＩコードの第２バイト目が来ると"ｆｇ＿ｒｅｃｉ＿

ｋａｎｊｉ"と"ｆｇ＿ｈｉｌｏ"なる２つのフラグを共に"１"とします（リスト７の行番号５７と行番号５８）．ＫＯのコードの第２バイト目が来ると"ｆｇ＿ｒｅｃｉ＿ｋａｎｊｉ"なるフラグを"０"に戻します（リスト７の行番号６５）．
データ長８ビットで通信している時には，ＫＩコードとＫＯコードの第２バイト目のデータである"０ｘ２４"と"０ｘ２８"が，８ビット目のビットが立てられて送られてくる可能性があります．その場合にも正しく対応出来るようしたものが，リスト７の行番号５４と行番号６２のステートメントです．

（３）ｏｕｔｐｕｔ＿ｃｒｔ（ｃｏｄｅ）

新しく追加された関数です．
受信データ"ｃｏｄｅ"がアスキー文字の時（ｆｇ＿ｒｅｃｉ＿ｋａｎｊｉ＝０）は，そのままディスプレイ画面に書き出します．一方漢字データの時は，その第１バイトをグローバル変数"ｊｃｏｄｅ＿ｈｉ"に一旦ストアし，そして次に送られてくる第２バイト目をグローバル変数"ｊｃｏｄｅ＿ｌｏ"にストアします．漢字の２バイトデータが揃ったところで，関数ｔｏ＿ｓｈｉｆｔ（）をコールすることによりＪＩＳコードをシフトＪＩＳコードへ変換し，その後ディスプレイ画面へ出力します．

（４）ｓｅｎｄ＿ｔｓｓ（ｃｏｄｅ）

内容が変更された関数です．
まず，関数ｓｊｉｓ＿ｌｓｔ（ｃｏｄｅ）をコールして，キーボードより入力された１バイトデータがシフトＪＩＳ漢字の第一バイト目であるかどうかをチェックします．もしそうであると，キーボードよりシフトＪＩＳ漢字の第２バイト目を読み込みます．そしてまずそれらをディスプレイ画面に書き出します．次にこのシフトＪＩＳ漢字の２バイトデータを関数ｔｏ＿ｊｉｓ（）を呼ぶことによってＪＩＳ漢字コードに変換します．そしてそれらをホストコンピュータへ送信するのですが，その前に関数ｔｏ＿ｋａｎｊｉ（）を呼んでＫＩコードをホストコンピュータへ送っておきます．ただし，この漢字データの直前に送られたデータが漢字であった時は，再度ＫＩコードを送らないようにしております（リスト７の行番号１０２）．
一方，入力された１バイトコードがシフトＪＩＳ漢字の１バイト目でなかった場合は，そのコードをそのままディスプレイに書くと同時にホストコンピュータへ送信します．その時，直前に送信されたデータが漢字であった時は関数ｔｏ＿ａｌｐｈａ（）をコールして前もってＫＯコードを送信しておく必要があります．

リスト８に載っている関数：

これらはリスト４に追加する関数と置き換える必要がある関数です．

（１）ｒｓ＿ｂｒｅａｋ（）

内容が変更された関数です．
ブレーク信号をホストコンピュータへ送信した後，エスケープ シーケンスと漢字に関するフラグを元にもどしておく必要があります（リスト８の行番号２０～２２）．

（２）ｔｏ＿ｓｈｉｆｔ（）

新しく追加された関数です．
ＪＩＳコードの漢字データ（ｊｃｏｄｅ＿ｈｉ＝第１バイト目，ｊｃｏｄｅ＿ｌｏ＝第２バイト目）をシフトＪＩＳコードに変換します．その結果，シフトＪＩＳコードの第１バイト目がｊｃｏｄｅ＿ｈｉに，第２バイト目がｊｃｏｄｅ＿ｌｏに返されます．

（３）ｔｏ＿ｊｉｓ（）

新しく追加された関数です．
シフトＪＩＳコードの漢字データ（ｊｃｏｄｅ＿ｈｉ＝第１バイト目，ｊｃｏｄｅ＿ｌｏ＝第２バイト目）をＪＩＳコードに変換します．その結果，ＪＩＳコードの第１バイト目がｊｃｏｄｅ＿ｈｉに，第２バイト目がｊｃｏｄｅ＿ｌｏに返されます．

（４）ｓｊｉｓ＿ｌｓｔ（ｃｏｄｅ）

新しく追加された関数です．
"ｃｏｄｅ"なるキャラクタがシフトＪＩＳコードの漢字第１バイト目であるかどうかをチェックします．もしそうであると，この関数は"１"を返し，そうでない時は"０"を返します．

（５）ｔｏ＿ｋａｎｊｉ（）

新しく追加された関数です．
送信データが漢字の時，その前に付加する３バイトのＫＩコード（エスケープ シーケンス）をホストコンピュータに送信します．

（６）ｔｏ＿ａｌｐｈａ（）

新しく追加された関数です．
送信データがアスキー文字の時，その前に付加する３バイトのＫＯコード（エスケープシーケンス）をホストコンピュータに送信します．

（７）ｃｏｌｏｒ（ｃ）

新しく追加された関数です．
Ｃのソフトウェア割り込み関数ｉｎｔ８６を用いてＭＳ－ＤＯＳの拡張システムコール（割り込み番号 ０ｘＤＣ）を行なって，文字の属性を変更します．文字の属性とは色，ブリンキングや下線など文字を修飾するための情報を言います．"ｃ"に図２に示すような８ビット情報を与えて，この関数をコールするとそれに対応した文字の属性が以後に続く文字に対して適用され，次の変更まで有効です．この点の詳細は，文献２３を参照して下さい．

６．コンパイルの方法

先ず，Ｍｉｆｅｓ ９８［１２］などのエディタを用いて，ダム ターミナルの場合リスト１～４までのプログラムを入力して下さい．その時，リスト１～４に対応してそれぞれ"ｔｓｓ＿

ｍｉｎ．ｈ", "ｐａｒａｍ．ｈ", "ｔｓｓ＿ｍｉｎ１．ｃ", "ｔｓｓ＿ｍｉｎ２．ｃ"なるファイル名を付けます．ここで最初の２つのファイル"ｔｓｓ＿ｍｉｎ．ｈ", "ｐａｒａｍ．ｈ"はインクルード ファイルであり，残りの２つのファイル"ｔｓｓ＿ｍｉｎ１．ｃ"と"ｔｓｓ＿ｍｉｎ２．ｃ"がエミュレータ本体のソースファイルです．ソースファイルが２つに分れていますので，分割コンパイルを行なう必要があります．コンパイルの方法は，１つのソースファイルでも複数のソースファイルでも変りありません．分割コンパイルは，リンクする際に複数のオブジェクトを指定するだけでよいのです．

これらのソースプログラムはＭＳ－Ｃ　Ｖｅｒ．４．０［１０］以上，もしくはＴｕｒｂｏ Ｃ　Ｖｅｒ．１．５［１１］以上のラージ モデルでコンパイルする必要があります．これは次のような理由によります．つまり，次回以降色々な便利な機能を付加えていく予定ですが，そのためにプログラム サイズがかなり大きくなります．そこでＣコンパイラのメモリモデルの１つであるスモール モデル内におさめることが難しくなってきますので，最初からラージ モデルを使用しているためです．

例えば，ＭＳ－Ｃ　Ｖｅｒ．５．１［２４］でコンパイルする時は

```
cl  /AL  /c  /J  tss_min1.c ↵
cl  /AL  /c  /J  tss_min2.c ↵
```

として下さい．ここで記号 ↵ は改行キーを押すことを意味します。"ｃｌ"はＭＳ－Ｃの３つのコンパイラ パスを実行するためのコンパイラ ドライバです．すると，２つのオブジェクト ファイル"ｔｓｓ＿ｍｉｎ１．ｏｂｊ"と"ｔｓｓ＿ｍｉｎ２．ｏｂｊ"が作られます．次にリンカ"ｌｉｎｋ．ｅｘｅ"を用いて，これら２つのオブジェクトとライブラリを結合する必要があります．つまり，

```
link  tss_min1+tss_min2, tss_min.exe, NUL, llibce ↵
```

として下さい．ここで，"ｌｌｉｂｃｅ"はラージ モデル用のライブラリです．これで"ｔｓｓ＿ｍｉｎ．ｅｘｅ"なる実行形式プログラムが出来ます．漢字対応版の場合にも，リスト１から４までのソースを第４節に記した説明に従って変更＆追加後，上と同じ方法でコンパイル出来ます．

上のような分割コンパイルによるプログラム開発を自動化するユーティリティ"ｍａｋｅ．ｅｘｅ"［１０，２４］を用いるとコンパイルとリンクを効率良く行なうことが出来ます．具体的には，
先ずエディタを用いて次のようなテキスト ファイルを作ります．

```
tss_min1.obj  :  tss_min1.c  tss_min.h  param.h
    cl  /AL  /c  /J  tss_min1.c
tss_min2.obj  :  tss_min2.c  tss_min.h  param.h
    cl  /AL  /c  /J  tss_min2.c
tss_min.exe   :  tss_min1.obj  tss_min2.obj
    link  tss_min1+tss_min2, tss_min.exe, NUL, llibce
```

そして，これを例えば"ｔｓｓ＿ｍｉｎ．ｍａｋ"なるファイル名でセーブします．そして

```
make  tss_min.mak ↵
```

と入力すると，コンパイルとリンクが一度で行なわれ，"ｔｓｓ＿ｍｉｎ．ｅｘｅ"なる実行形式プログラムが出来ます．また，この方法ですと修正が行なわれたファイルのみコンパイル／リンクが実行されますので，プログラム開発が手際良く行なわれるようになります．上で表れた，コンパイラ ドライバ（"ｃｌ"）とリンカ（"ｌｉｎｋ"）のパラメータの詳細やｍａｋｅファイルの文法等については，ＭＳ－Ｃのマニュアル［１０，２４］を参照して下さい．

Ｔｕｒｂｏ Ｃでコンパイルする時は，プロジェクト ファイルを用いて統合開発環境バージョンＴＣでコンパイルすると，ＭＳ－ＣでＭＡＫＥファイルを用いるのと同じように効率良くプログラム開発が出来ます．そのためには，まず次のようなテキスト ファイルを作ります．

```
tss_min1.c  (tss_min.h, param.h)
tss_min2.c  (tss_min.h, param.h)
```

そして，これを例えば"ｔｓｓ＿ｍｉｎ．ｐｒｊ"なるファイル名でセーブします．次に，"ｔｃ．ｅｘｅ"を立上げ，メニュー画面より"Ｐｒｏｊｅｃｔ"を選択してファイル"ｔｓｓ＿ｍｉｎ．ｐｒｊ"を登録します．こうしておいてから，[f・9]（ｍａｋｅ）キーを押すと，"ｔｓｓ＿ｍｉｎ．ｐｒｊ"の内容に従い，コンパイルとリンクが一度に行なわれます．なお，実行形式ファイル名は"ｔｓｓ＿ｍｉｎ．ｅｘｅ"となります．これらの点についての詳細は，文献１１を参照して下さい．

７．おわりに

Ｃ言語によるＴＳＳ端末エミュレータ プログラミングの解説の最初のものとして，最も基本的なエミュレータと，その１つの拡張としての漢字対応版の２つの版のエミュレータを紹介しました．今回のプログラムでは，ＲＳ－２３２Ｃポートの初期化を"ＳＰＥＥＤ"コマンドを用いて行なっています．そのために，エミュレータ実行時に"ＳＰＥＥＤ．ＥＸＥ"なるファイルが必要となっています．これはＲＳ－２３２Ｃ関係のパラメータを変更する必要が起った時に，この方法のほうが初心者の方にとって分かりやすいと思ったからです．もちろんこのＲＳ－２３２Ｃポートの初期化は，ｒｓ＿ｌｅｎｇｔｈ（）やｒｓ＿ｒｅｃｅｉｖｅ（）などの関数と同じようにＲＳ－２３２ＣのＢＩＯＳに割り込み番号０ｘ１９のソフトウェア割り込みを実行することによって実現出来ます［１６，２５］．この方法については次回詳しく説明したいと思います．また，日本語ＰＦＤ，日本語ＡＳＰＥＮ，ファイル転送（特にホストコンピュータがデータの終結コードとしてヌル（０ｘ０）を送ってこない時の対応策），エコーバックがある場合等への対応機能も次回解説する予定です．

本稿にソースプログラムの全てを掲載しておりますが，これらを打ち込むのが面倒な方には郵送によるコピー サービスを行ないます．希望される方は，フロッピィ ディスクを返信用封筒（宛名，切手付き）と一緒に下記住所あてにお送り下さい．メディアは問いません．

〒840　佐賀市本庄町一番地
佐賀大学 理工学部 物理学教室
武政 尹士

この解説シリーズがこれから自分の計算機環境に合うように，既存のエミュレータに手を加えたいと思っておられる方や，新しく自作のエミュレータを作ろうと思っておられる方たちのお役に立てば幸いです．なお，私たちはプロのプログラマでもなければ，Cの専門家でもありませんので，プログラミング上改良すべき点は多々あると思われます．また，本稿にも不十分な点や勉強不足な点が多く見られると思われます．お気づきのことがありましたら，ぜひご指摘をお願い致します．

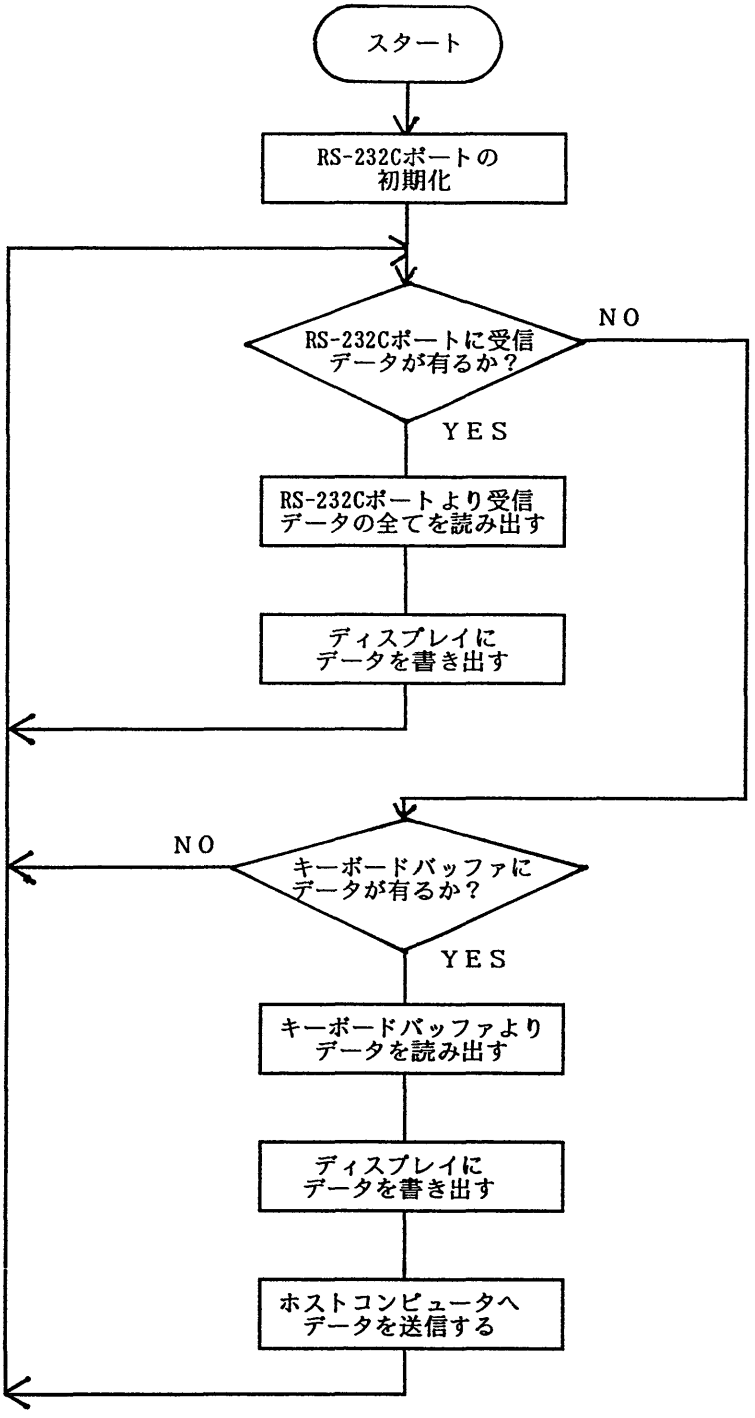
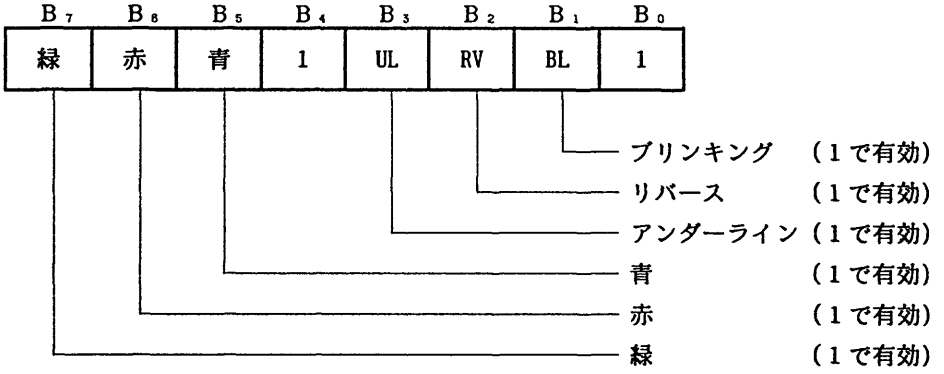


図 1．エミュレータプログラムの基本的な処理の流れ図



B7	B6	B5	
緑	赤	青	
0	0	0	黒
0	0	1	青
0	1	0	赤
0	1	1	紫
1	0	0	緑
1	0	1	水色
1	1	0	黄
1	1	1	白

例： 黄色でブリンキング ＝ １６進数 Ｄ３（２進数 １１０１００１１）

図２．文字属性を設定する８ビットデータ

参考文献

1．渡辺健次，武政尹士，「日本語ＰＦＤエミュレータ　－ ＰＣ－９８０１シリーズ対応－」，九州大学大型計算機センター広報，Ｖｏｌ．２１，Ｎｏ．５（１９８８），ｐ．３７１．
渡辺健次，武政尹士，「高速多機能ＴＳＳＰＦＤ端末エミュレータ（グラフィック機能追加）－ ＰＣ－９８０１シリーズ対応 －」，九州大学大型計算機センター広報，Ｖｏｌ．２２，Ｎｏ．１（１９８９），ｐ．３７．
武政尹士，渡辺健次，「高速多機能ＴＳＳＰＦＤ端末エミュレータ（ＵＴＳ対応版）－ＰＣ９８０１シリーズ対応」，九州大学大型計算機センター広報，Ｖｏｌ．２２，Ｎｏ．３（１９８９），ｐ．２０９．
武政尹士，渡辺健次，「ＰＣ－９８０１シリーズ対応高速多機能ＴＳＳ端末エミュレータ：ＴＳＳＰＦＤ（ＡＰＳＥＮ，パソコン間対応版）」，九州大学大型計算機センター広報，Ｖｏｌ．２３，Ｎｏ．６（１９９０），ｐ．６１１．
2．インタープログ編，「富士通ＦＭＲシリーズ 徹底解析マニュアル 増補改訂版」，（１９９０），（株）ビー・エヌ・エヌ．
3．T. Hogan／ＳＥ編集部訳編，「プログラマーのためのＰＣソースブック」，（１９９０），（株）翔泳社．
4．村瀬康治，「入門ＭＳ－ＤＯＳ」，（１９８５），（株）アスキー．
5．L. Hancock, M. Krieger／三浦明美（訳），「Ｃ言語入門」，（１９８４），（株）アスキー．
6．R. J. Traister／野中浩一（訳），「ＢＡＳＩＣ ｔｏ Ｃ」，（１９８６），（株）アスキー．
7．日本電気（株），「ＰＣ－９８００シリーズ ＭＳ－ＤＯＳ ３．３ ユーザーズ リファレンス マニュアル」，（１９８８）．
8．ジャスト システム（株），「一太郎 Ver．３．０ 活用編」，（１９８７）．
9．管理工学研究所，「日本語ワードプロセッサ 松 専門機能編」，（１９８７）．
10．アスキーマイクロソフト，「Ｍｉｃｒｏｓｏｆｔ Ｃ ユーザーズ ガイド Ｖｅｒ．４．０」（１９８７），（株）アスキー．
11．Ｂｏｒｌａｎｄ Ｉｎｔｅｒｎａｔｉｏｎａｌ Ｉｎｃ．，「Ｔｕｒｂｏ Ｃ ユーザーズ ガイド Ｖｅｒ．１．５」，（１９８７），（株）マイクロソフトウェア アソシエイツ．
12．メガソフト（株），「Ｍｉｆｅｓ ９８ Ｖｅｒ．４．０ ユーザーズ マニュアル」，（１９８８）．
13．松尾文碩，「ＦＡＣＯＭ Ｍ－７８０上のＵＴＳ － 世界最高速のＵＮＩＸ －」，九州大学大型計算機センター広報，Ｖｏｌ．２０，Ｎｏ．５（１９８７），ｐ．３８９．
14．川田良文，堤守政，安藤八郎，「端末からセンターを利用するためのデータ通信入門」，名古屋大学大型計算機センター ニュース，Ｖｏｌ．２０，Ｎｏ．２（１９８９），ｐ．１２０．
15．石坂充弘，「情報通信プロトコル」，（１９８９），（株）オーム社．
16．柿園昭俊，植松早苗，「ＭＳ－ＤＯＳ メニュー ブック（２）」，（１９８６），（株）技術評論社．
17．日本電気（株），「ＰＣ－９８００シリーズ ＰＣ－９８０１ＶＸ ハードウェア マニュアル」，（１９８６）．

１８．日本電気（株），「ＰＣ－９８００シリーズ ＭＳ－ＤＯＳ ３．３ プログラマーズ リファレンス マニュアル Ｖｏｌ．１ ＆ ２」，（１９８８）．
１９．山崎福馬，小沢真樹，「ＭＳ－ＤＯＳ プラス Ｖｅｒ．２．１／３．１完全対応」，（１９８６），（株）ビー・エヌ・エヌ．
２０．嶺巓一起，鷹野澄，「ＭＳ－ＤＯＳ」，（１９８９），共立出版（株）．
２１．アスキー書籍編集部編，「ＭＳ－ＤＯＳ プログラミング テクニック」，（１９８６），（株）アスキー．
２２．二村祥一，「ＵＴＳの日本語処理機能」，九州大学大型計算機センター広報，Ｖｏｌ．２０，Ｎｏ．５（１９８７），ｐ．４５０．
２３．藤田英時，幸田敏記，「ＰＣ－Ｔｅｃｈｋｎｏｗ ９８００」，（１９８５），（株）システムソフト．
２４．アスキーマイクロソフト，「Ｍｉｃｒｏｓｏｆｔ Ｃ ユーザーズ ガイド Ｖｅｒ．５．１」（１９８８），（株）アスキー．
２５．白田耕作，「ＰＣ－９８拡張システムコール詳説」，（１９８５），インターフェイス４月号別冊付録．

リスト１

```
 1: /*=========================================================================*
 2:  *                                                                         *
 3:  *                                                                         *
 4:  *                Terminal Emulator Program : TSS_min.H                    *
 5:  *                                                                         *
 6:  *                                                                         *
 7:  *            <<<<<   Microsoft C & Turbo C version   >>>>>               *
 8:  *                                                                         *
 9:  *                                                                         *
10:  *=========================================================================*/
11:
12:
13: /*-------------------------------------------------------------------------*/
14: /*                        入力キーコード                                   */
15: /*-------------------------------------------------------------------------*/
16:
17: #define  CTRL_B                  0x02
18: #define  CTRL_Q                  0x11
19:
20:
21: /*-------------------------------------------------------------------------*/
22: /*                  システム＆拡張システムコール用機能番号                 */
23: /*-------------------------------------------------------------------------*/
24:
25: #define  EXT_DOS                 0xdc
26: #define  RS232C_BIOS             0x19
27:
28:
29: /*-------------------------------------------------------------------------*/
30: /*                           基本的常数の定義                              */
31: /*-------------------------------------------------------------------------*/
32:
33: #define  TRUE                    1
34: #define  FALSE                   !TRUE
35:
36: #define  BREAK_TIME              1
37:
38:
39: /*-------------------------------------------------------------------------*/
40: /*                              マクロ定義                                 */
41: /*-------------------------------------------------------------------------*/
42:
43: #define  ring_bell()             sys_putchar( 0x07 )
44:
45:
46: /*-------------------------------------------------------------------------*/
47: /*                              型宣言                                     */
48: /*-------------------------------------------------------------------------*/
49:
50: typedef  unsigned char           BYTE;
51:
52:
53: /*-------------------------------------------------------------------------*/
54: /*                           関数の型宣言                                  */
55: /*-------------------------------------------------------------------------*/
56:
57: int    rs_init( void );
58: int    rs_length( void );
59: BYTE   rs_recieve( void );
60: BYTE   inkey( void );
61: void   rs_sendc( BYTE );
62: void   rs_break( void );
63: void   pause( int );
64: void   cls( void );
65: void   sys_putchar( BYTE );
66: void   emulator( void );
67: void   send_tss( BYTE );
68: void   reci_tss( int );
69:
70: /* ==================      END of TSS_min.H      ====================== */
```

リスト2

```
 1: /*=========================================================================*
 2:  *                                                                         *
 3:  *                                                                         *
 4:  *                 Terminal Emulator Program : PARAM.H                     *
 5:  *                                                                         *
 6:  *                                                                         *
 7:  *           <<<<<   Microsoft C & Turbo C version   >>>>>                 *
 8:  *                                                                         *
 9:  *                                                                         *
10:  *=========================================================================*/
11:
12:
13: /*-------------------------------------------------------------------------*/
14: /*                      R S - 2 3 2 C 関係のパラメータ                     */
15: /*-------------------------------------------------------------------------*/
16:
17: #define  SPEED_PATH          "¥¥speed.exe"
18: #define  RS_PORT_NO          "r0"
19: #define  BAUD_RATE           "9600"
20: #define  CHAR_LEN            "b7"
21: #define  PARITY              "pe"
22: #define  STOP_BIT            "s1"
23: #define  XONOFF              "xon"
24:
25: /* ======================      END of PARAM.H      ================== */
```

リスト3

```
 1: /*=========================================================================*
 2:  *                                                                         *
 3:  *                                                                         *
 4:  *              Terminal Emulator Program : TSS_min1.C                     *
 5:  *                                                                         *
 6:  *                                                                         *
 7:  *           <<<<<   Microsoft C & Turbo C version   >>>>>                 *
 8:  *                                                                         *
 9:  *                                                                         *
10:  *=========================================================================*/
11:
12:
13: #include  <dos.h>
14: #include  <process.h>
15: #include  "tss_min.h"
16: #include  "param.h"
17:
18:
19: /*     ----------------------     main()     ----------------------     */
20:
21: void  main( void )                    /*  メイン ルーチン  */
22: {
23:   if ( rs_init() == -1 )
24:     exit( 1 );
25:   else
26:     emulator();
27:   exit( 0 );
28: }
29:
30: /*     --------------------     emulator()     -------------------     */
31:
32: void  emulator( void )                /*  エミュレータ本体  */
33: {
34:   int  len;
35:   BYTE  key_code;
36:
37:     cls();
38:     while ( TRUE )  {
39:       if ( ( len = rs_length() ) != 0 )  reci_tss( len );
40:       if ( ( key_code = inkey() ) == 0 )  continue;
```

リスト３（続）

```
41:       switch ( key_code )  {
42:         case  CTRL_B :
43:               rs_break();
44:               break;
45:
46:         case  CTRL_Q :
47:               return;
48:
49:         default :
50:               send_tss( key_code );
51:       }
52:     }
53: }
54:
55: /*     ----------------     reci_tss( len )      -----------------     */
56:
57: void  reci_tss( int len )                /*  受信データの処理  */
58: {
59:   BYTE  in_code;
60:
61:     do {
62:       while ( len-- > 0 )  {
63:         if ( ( in_code = rs_receive() ) == 0x0 )  continue;
64:         sys_putchar( in_code );
65:       }
66:     }
67:     while ( ( len = rs_length() ) != 0 )  ;
68: }
69:
70: /*     -----------------     send_tss( code )     ----------------     */
71:
72: void  send_tss( BYTE code )               /*  送信データの処理  */
73: {
74:   sys_putchar( code );
75:   rs_sendc( code );
76: }
77:
78: /* ===================     END of TSS_min1.C     ===================== */
```

リスト４

```
 1: /*=========================================================================*
 2:  *                                                                         *
 3:  *                                                                         *
 4:  *              Terminal Emulator Program : TSS_min2.C                     *
 5:  *                                                                         *
 6:  *                                                                         *
 7:  *           <<<<<   Microsoft C & Turbo C version   >>>>>                 *
 8:  *                                                                         *
 9:  *                                                                         *
10:  *=========================================================================*/
11:
12:
13: #include  <dos.h>
14: #include  <process.h>
15: #include  "tss_min.h"
16: #include  "param.h"
17:
18:
19: /*     --------------------     rs_init()     --------------------     */
20:
21: int  rs_init( void )                      /*  RS-232Cの初期化  */
22: {
23:   char  *rs_param[ 8 ];
24:
25:     rs_param[0] = SPEED_PATH;
26:     rs_param[1] = RS_PORT_NO;
27:     rs_param[2] = BAUD_RATE;
```

リスト4 (続)

```
28:     rs_param[3] = CHAR_LEN;
29:     rs_param[4] = PARITY;
30:     rs_param[5] = STOP_BIT;
31:     rs_param[6] = XONOFF;
32:     rs_param[7] = "¥0";
33:     return  ( spawnv( P_WAIT, SPEED_PATH, rs_param ) == -1 ) ? -1 : 0;
34: }
35:
36: /*     ------------------     rs_length()     --------------------     */
37:
38: int  rs_length( void )                  /*  受信データ長の取得  */
39: {
40:   union REGS  inregs, outregs;
41:   struct SREGS  segregs;
42:
43:     segread( &segregs );
44:     segregs.ds  = 0x0060;
45:     inregs.h.ah = 0x04;
46:     int86x( RS232C_BIOS, &inregs, &outregs, &segregs );
47:     return  outregs.x.cx;
48: }
49:
50: /*     ------------------     rs_receive()     -------------------     */
51:
52: BYTE  rs_receive( void )                 /*  1バイト受信  */
53: {
54:   union REGS  inregs, outregs;
55:   struct SREGS  segregs;
56:
57:     segread( &segregs );
58:     segregs.ds  = 0x0060;
59:     inregs.h.ah = 0x02;
60:     int86x( RS232C_BIOS, &inregs, &outregs, &segregs );
61:     return  outregs.h.al;
62: }
63:
64: /*     -----------------     rs_sendc( code )     ----------------     */
65:
66: void  rs_sendc( BYTE code )              /*  1バイト送信  */
67: {
68:   union REGS  inregs, outregs;
69:   struct SREGS  segregs;
70:
71:     segread( &segregs );
72:     segregs.ds  = 0x60;
73:     inregs.h.ah = 0x01;
74:     inregs.h.cl = code;
75:     int86x( RS232C_BIOS, &inregs, &outregs, &segregs );
76: }
77:
78: /*     --------------------     rs_break()     --------------------     */
79:
80: void  rs_break( void )                   /*  ブレーク信号の送信  */
81: {
82:   outp( 0x32, 0x3f );
83:   ring_bell();
84:   pause( 30 * BREAK_TIME );
85:   outp( 0x32, 0x37 );
86: }
87:
88: /*     ----------------     pause( times )     --------------------     */
89:
90: void  pause( int times )                 /*  タイマー  */
91: {
92:   int  i, j;
93:   static int  i_max = 1000;
94:
95:     for ( j = 0; j < times; j++ )
96:       for ( i = 0; i < i_max; i++ )
```

リスト４（続）

```
 97:         ;
 98: }
 99:
100: /*     ----------------------     inkey()     --------------------     */
101:
102: BYTE  inkey( void )                    /*  キーボードからの入力   */
103: {                                       /*  データの取り出し       */
104:   union REGS  inregs, outregs;
105:
106:     inregs.h.ah = 0x06;
107:     inregs.h.dl = 0xff;
108:     intdos( &inregs, &outregs );
109:     return  outregs.h.al;
110: }
111:
112: /*     ---------------------     cls()     ----------------------     */
113:
114: void  cls( void )                        /*  ディスプレイ画面の全消去   */
115: {
116:   union REGS  inregs, outregs;
117:
118:     inregs.h.cl = 0x10;
119:     inregs.h.ah = 0x0a;
120:     inregs.h.dl = 2;
121:     int86( EXT_DOS, &inregs, &outregs );
122: }
123:
124: /*     --------------     sys_putchar( code )     ----------------     */
125:
126: void  sys_putchar( BYTE code )           /*  ディスプレイへの１バイト   */
127: {                                        /*  コードの出力               */
128:   union REGS  inregs, outregs;
129:
130:     inregs.h.cl = 0x10;
131:     inregs.h.ah = 0x00;
132:     inregs.h.dl = code;
133:     int86( EXT_DOS, &inregs, &outregs );
134: }
135:
136: /* ====================     END of TSS_min2.C     ====================== */
```

リスト５

```
  1: /*************************************************************************
  2:
  3:                リスト１に追加するパラメータと関数の型の宣言
  4:
  5: *************************************************************************/
  6:
  7:
  8:
  9: /*-----------------------------------------------------------------------*/
 10: /*                        入力キーコード                                 */
 11: /*-----------------------------------------------------------------------*/
 12:
 13: #define  ESC                      0x1b
 14:
 15:
 16: /*-----------------------------------------------------------------------*/
 17: /*                        表示色、表示属性コード                         */
 18: /*-----------------------------------------------------------------------*/
 19:
 20: #define  GREEN                    0x81
 21: #define  WHITE                    0xe1
 22:
 23:
```

リスト5 (続)

```
24: /*------------------------------------------------------------------------*/
25: /*                          関数の型宣言                                  */
26: /*------------------------------------------------------------------------*/
27:
28: int    esc_seq( BYTE );
29: int    sjis_1st( BYTE );
30: void   to_shift( void );
31: void   to_jis( void );
32: void   to_kanji( void );
33: void   to_alpha( void );
34: void   output_crt( BYTE );
35: void   color( int );
36:
37: /* ==================     END of TSS_min.H     ====================== */
```

リスト6

```
 1: /*************************************************************************
 2:
 3:                      リスト2に追加するパラメータ
 4:
 5: *************************************************************************/
 6:
 7:
 8: /*------------------------------------------------------------------------*/
 9: /*                      JOIS用KI, KOコード                                */
10: /*------------------------------------------------------------------------*/
11:
12: #define  KI_CODE1           0x24
13: #define  KI_CODE1_S         0xa4
14: #define  KI_CODE2           0x40
15: #define  KO_CODE1           0x28
16: #define  KO_CODE1_S         0xa8
17: #define  KO_CODE2           0x48
18:
19: /* ======================     END of PARAM.H     ================== */
```

リスト7

```
 1: /*************************************************************************
 2:
 3:                    リスト3に追加する関数と変更する関数
 4:
 5: *************************************************************************/
 6:
 7:
 8: int  fg_send_kanji = 0, fg_reci_kanji = 0, fg_hilo, fg_reci_esc = 0;
 9:
10: BYTE  jcode_hi, jcode_lo;
11:
12:
13: /*     -----------------    reci_tss( len )     -----------------     */
14:
15: void  reci_tss( int len )              /*  受信データの処理  */
16: {
17:   BYTE  in_code;
18:
19:     color( GREEN );
20:     do {
21:       while ( len-- > 0 )  {
22:         in_code = rs_receive();
23:         switch ( fg_reci_esc )  {
```

リスト7（続）

```
24:           case  0 :
25:             switch ( in_code )  {
26:               case  ESC :
27:                     fg_reci_esc = 1;
28:                     continue;
29:
30:               case  0 :                 /*  ヌルコードの読み飛ばし   */
31:                     continue;
32:
33:               default :
34:                     output_crt( in_code );
35:                     continue;
36:             }
37:
38:           case  1 :
39:                 fg_reci_esc = 0;
40:                 if ( esc_seq( in_code ) != 0 )  len = rs_length();
41:                 continue;
42:         }
43:       }
44:     }
45:     while ( ( len = rs_length() ) != 0 )  ;
46: }
47:
48: /*      -----------------      esc_seq( code )      ----------------      */
49:
50: int  esc_seq( BYTE code )                 /*  エスケープ シーケンスの   */
51: {                                         /*  処理                      */
52:   switch ( code )  {
53:     case  KI_CODE1 : ;
54:     case  KI_CODE1_S :
55:           while ( rs_length() == 0 )  ;
56:           rs_receive();                   /*  KIコード（2バイト目）    */
57:           fg_reci_kanji = 1;              /*  の読み飛ばし             */
58:           fg_hilo = 1;
59:           return  1;
60:
61:     case  KO_CODE1 : ;
62:     case  KO_CODE1_S :
63:           while ( rs_length() == 0 )  ;
64:           rs_receive();                   /*  KOコード（2バイト目）    */
65:           fg_reci_kanji = 0;              /*  の読み飛ばし             */
66:           return  1;
67:
68:     default :
69:           fg_reci_kanji = 0;
70:           return  0;
71:   }
72: }
73:
74: /*      -----------------      output_crt( code )      --------------      */
75:
76: void  output_crt( BYTE code )             /*  受信データのディスプレイ  */
77: {                                         /*  画面への出力              */
78:   if ( fg_reci_kanji == 0 )  sys_putchar( code );   /*  アスキー データ  */
79:   else {                                            /*  漢字データ  */
80:     fg_hilo = 1 - fg_hilo;
81:     if ( fg_hilo == 0 )  jcode_hi = code;
82:     else {
83:       jcode_lo = code;
84:       to_shift();
85:       sys_putchar( jcode_hi );
86:       sys_putchar( jcode_lo );
87:     }
88:   }
89: }
90:
91: /*      ----------------      send_tss( code )      -----------------      */
92:
```

リスト7 (続)

```
 93: void  send_tss( BYTE code )              /*  送信データの処理   */
 94: {
 95:   color( WHITE );
 96:   if ( sjis_1st( code ) == 1 )  {        /*  漢字データ   */
 97:     jcode_hi = code;
 98:     jcode_lo = inkey();
 99:     sys_putchar( jcode_hi );
100:     sys_putchar( jcode_lo );
101:     to_jis();
102:     if ( fg_send_kanji != 1 )  to_kanji();
103:     rs_sendc( jcode_hi );
104:     rs_sendc( jcode_lo );
105:     fg_send_kanji = 1;
106:   }
107:   else {                                 /*  アスキー データ   */
108:     sys_putchar( code );
109:     if ( fg_send_kanji != 0 )  to_alpha();
110:     rs_sendc( code );
111:     fg_send_kanji = 0;
112:   }
113: }
114:
115: /* ====================     END of TSS_min1.C     ====================== */
```

リスト8

```
  1: /*****************************************************************************
  2:
  3:                    リスト4に追加する関数と変更する関数
  4:
  5: *****************************************************************************/
  6:
  7:
  8: extern int  fg_reci_kanji, fg_hilo, fg_reci_esc;
  9: extern BYTE  jcode_hi, jcode_lo;
 10:
 11:
 12: /*     --------------------     rs_break()     --------------------     */
 13:
 14: void  rs_break( void )                  /*  ブレーク信号の送信   */
 15: {
 16:   outp( 0x32, 0x3f );
 17:   ring_bell();
 18:   pause( 30 * BREAK_TIME );
 19:   outp( 0x32, 0x37 );
 20:   fg_reci_kanji = 0;                    /*  漢字に関する   */
 21:   fg_hilo = 0;                          /*  フラグを       */
 22:   fg_reci_esc = 0;                      /*  おろす         */
 23: }
 24:
 25: /*     --------------------     to_shift()     --------------------     */
 26:
 27: void  to_shift( void )                  /*  ＪＩＳ→シフトＪＩＳ変換   */
 28: {
 29:   if ( jcode_hi % 2 == 1 )  {
 30:     jcode_lo += 0x1f;
 31:     if ( jcode_lo >= 0x7f )  jcode_lo++;
 32:   }
 33:   else  jcode_lo += 0x7e;
 34:   jcode_hi = ( jcode_hi - 0x21 >> 1 ) + 0x81;
 35:   if ( jcode_hi > 0x9f )  jcode_hi += 0x40;
 36: }
 37:
 38: /*     --------------------     to_jis()     ----------------------     */
 39:
 40: void  to_jis( void )                     /*  シフトＪＩＳ→ＪＩＳ変換   */
```

リスト８（続）

```
41: {
42:   jcode_hi -= ( jcode_hi > 0x9f ) ? 0xb1 : 0x71;
43:   jcode_hi = jcode_hi * 2 + 1;
44:   if ( jcode_lo > 0x9e )  {
45:     jcode_lo = jcode_lo - 0x7e;
46:     jcode_hi++;
47:   }
48:   else  {
49:     if ( jcode_lo > 0x7e )  jcode_lo--;
50:     jcode_lo -= 0x1f;
51:   }
52: }
53:
54: /*     ----------------     sjis_1st( code )     -----------------     */
55:
56: int  sjis_1st( BYTE code )              /*  シフトＪＩＳ漢字の  */
57: {                                       /*  第１バイト目の検出  */
58:   code &= 0xff;
59:   if ( ( code >= 0x81 && code <= 0x9f ) ||
60:        ( code >= 0xe0 && code <= 0xfc ) )
61:     return  1;
62:   else
63:     return  0;
64: }
65:
66: /*     ------------------     to_kanji()      ------------------     */
67:
68: void  to_kanji( void )                  /*  漢字ＩＮコードの送信  */
69: {
70:   rs_sendc( ESC );
71:   rs_sendc( KI_CODE1 );
72:   rs_sendc( KI_CODE2 );
73: }
74:
75: /*     ------------------     to_alpha()      ------------------     */
76:
77: void  to_alpha( void )                  /*  漢字ＯＵＴコードの送信  */
78: {
79:   rs_sendc( ESC );
80:   rs_sendc( KO_CODE1 );
81:   rs_sendc( KO_CODE2 );
82: }
83:
84: /*     --------------------     color( c )     ------------------     */
85:
86: void  color( int c )                    /*  表示文字の色、属性の設定  */
87: {
88:   union REGS  inregs, outregs;
89:
90:     inregs.h.cl = 0x10;
91:     inregs.h.ah = 0x02;
92:     inregs.h.dl = ( BYTE ) c;
93:     int86( EXT_DOS, &inregs, &outregs );
94: }
95:
96: /* ====================     END of TSS_min2.C     ====================== */
```