NI-488.2
ソフトウエア参考マニュアル
ーMS-DOS版ー

1992年12月版

部品番号 370914A-01
(日本語版)

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
(800) 433-3488 (toll-free U.S. and Canada)
Technical support fax: (512) 794-5678

Branch Offices:
Australia 03 879 9422, Austria 0662 435986, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521,
Denmark 45 76 26 00, Finland 90 527 2321, France 1 48 65 33 70,
Germany 089 714 50 93, Italy 02 48301892, Japan 03 3788 1921,
Netherlands 01720 45761, Norway 03 846866, Spain 91 640 0085,
Sweden 08 730 49 70, Switzerland 056 27 00 20, U.K. 0635 523545

著作権

商標

はじめに

このマニュアルはMS-DOS用NI-488.2ソフトウエアを詳細に説明しています。IBMパーソナルコンピュータ用のAdvanced IBM Interpretive BASIC、Microsoft Quick BASIC、Microsoft Professional Basic及びMicrosoft Cで使用するNI-488.2ルーチンとNI-488関数の全てがここで説明されています。上記以外のプログラミング言語で使用するソフトウエア及び付属マニュアルは別に用意されています。

本マニュアルの構成

本マニュアルは次のように構成されています。

- 第1章　序言ではGeneral Purpose Interface Bus (汎用インターフェース・バス、GPIBと省略)ならびに本マニュアルを紹介しています。

- 第2章 NI-488.2ソフトウエアのインストレーションと構成設定ではNI-488.2ソフトウエアをインストールし構成するための手順を説明しています。またこの章はソフトウエア構成用プログラムIBCONF及びユーザーの皆様が御自分でアプリケーション・プログラムを開発しデバッグする際に使用できるNI-488.2ソフトウエア機能の説明も含んでいます。

- 第3章 NI-488.2ソフトウエアを理解するではまずNI-488.2ルーチンとNI-488関数を紹介し、両者に共通した特徴を説明します。さらにユーザーの皆様がアプリケーションプログラムを書こうとする場合に知っておくべき一般的情報、及び各言語のプログラミングの特殊情報を説明します。

- 第4章　NI-488.2ソフトウエアの特性とルーチンではMS-DOSドライバにより使用出来るNI-488.2ルーチンの重要特性のうちで各言語に共通なものを説明し、次にこれらのルーチンの一つ一つについてプログラムの例を付して詳細に記述しています。ルーチンはアルファベット順に配列してあります。

- 第５章　NI-488ソフトウエアの特性と関数ではMS-DOSドライバにより使用出来るNI-488関数の重要な特性のうちで各言語に共通なものを説明し、次にこれらの関数の一つ一つについてプログラムの例を付して詳細に記述しています。関数はアルファベット順に配列してあります。

- 第６章　IBICではNI-488.2ルーチンとNI-488関数を利用してInterface Bus Interactive Control (IBIC)プログラムを実行する手続きを説明しています。また、これらのルーチンや関数のシンタクスと重要な特性を述べ、IBICのプログラム例を示しています。

- 第７章 Applications Monitorでは、applications monitorと言うプログラムのインストール、構成、使用方法が説明されています。Applications monitorはGPIB呼び出し手順のデバッグをユーザー各自のアプリケーション内から行うことが出来る便利な常駐プログラムです。

- 付録A マルチライン・インターフェース・メッセージにはマルチライン・インターフェース・メッセージ参照表があって、各種インターフェース機能の簡略記号とメッセージを示しています。

- 付録B 起こりやすいエラーと解決法では起こりやすいエラーとその解決策を示しています。これらのエラーは関数から返されてiberrによって示されるエラーコードの順になっています。また、この付録ではエラーコードとして返ってこないエラー状況も説明されています。

- 付録C シリアル／パラレルポートの方向変換—GPIBデバイスへにおいてはMS-DOSドライバ用のNI-488.2ソフトウエアを通じて標準のDOSプリンタ・コマンド（LPRINT, PRINT等)を使用してシリアル或いはパラレルポートからのデータをGPIB プリンタ、プロッタ、その他のデバイスに向け換える方法が説明されています。

- 付録D GPIBのオペレーションではGPIBのオペレーションを行う際にに必要な基本的な事項が説明されています。またこの付録ではGPIBの物理的、電気的特性と構成に必要な条件が述べられています。

- 付録E 使用者各位との連絡にはいくつかの書式が入っています。あらかじめこれらの表に必要事項を記入しておかれますと、ナショナルインスツルメンツの製品について当社に御連絡の場合に便利です。
- 用語解説においてはこのマニュアル中で使用されている用語がアルファベット順、アイウエオ順でリストされ、解説が加えられています。
- 索引ではこのマニュアル中の主要な用語と項目をアルファベット順にまとめ、それらが出てくるページを示しています。

表記規約

本マニュアルではテキスト中の各要素を区別するため、次の表記規約を用いています。

italic	イタリックのテキストは、強調、相互参照、及び主要な概念の導入を表わすために使用されます。
monospace	このフォントの小文字による表記はテキストなり文字がそのままキーボードから入力されるべきものであったり、コードの各部であったり、プログラミングの例であったり、或いはシンタクスの例であったりする場合に使用されます。このフォントはディスクドライブ、パス、ディレクトリ、プログラム、サブプログラム、サブルーチン、デバイス名、関数、変数、および拡張子の個有名を示すためにも使用されます。またプログラムコードのステートメントとコメントにも使用されます。
bold monospace	このフォントのボールドで小文字のテキストはコンユータが自動的にスクリーンに表示するメッセージや応答を示すのに使用されます。

italic monospace	このフォントでイタリックの小文字のテキストはその部分をユーザーが適当な言葉なり値で置き換えなければならないことを示しています。
<>	キーボード上のキーの名は＜＞の中に括られます。例えば <PageDown> の如くです。
-	２つ或いはそれ以上の数のキーの名がハイフンでつながれて＜＞中に括られていると、それらのキーを同時に幼ければいけないことを示します。例えば、<Control-Alt-Delete> の如しです。
<Enter>	キーの名は何時でも大文字で始まります。

本マニュアルで使用される用語の定義

用語	意味
BASICA	Advanced IBM Interpretive BASIC for the IBM Personal Computer
QuickBASIC	Microsoft QuickBASIC
BASIC	Microsoft Professional BASIC
C	Microsoft C

省略形

このマニュアルでは、計量単位の省略形としてメートル法の接頭語を使用しています。

接頭語	意味	累乗値
m-	milli-	10^{-3}
μ-	micro-	10^{-6}
n-	nano-	10^{-9}

このマニュアルで使用される略語を以下に示します。

dec	decimal (10進)
hex	hexadecimal (16進)
Hz	hertz (ヘルツ)
m	meters (メートル)
M	megabytes of memory (メガバイトの記憶容量)
oct	octal (8進)
sec	seconds (秒)

アクロニム (頭文字語)

本マニュアルで使用するアクロニムを以下に示します。

AC	alternating current (交流)
ANSI	American National Standards Institute (アメリカ規格協会)
ASCII	American Standard Code for Information Interchange (情報交換用米国標準コード)
CPU	central processing unit (中央処理装置)
DMA	direct memory access (直接メモリアクセス)
DVM	digital voltmeter (ディジタル電圧計)
EOI	End-or-Identify (終了或いは識別)
EOS	End-of-String (ストリング終了)
FIFO	first-in-first-out (先入れ先出し方式)
GPIB	General Purpose Interface Bus (汎用インターフェースバス)
IBIC	Interface Bus Interactive Control (対話式インターフェースバス制御)
IEEE	Institute of Electrical and Electronic Engineers (アメリカ電気・電子学会)
I/O	input/output (入出力)
ISO	International Standards Organization (国際標準化機構)
PC	personal computer (パーソナルコンピュータ)
RAM	random-access memory (ランダムアクセスメモリ)
SCPI	Standard Commands for Programmable Instruments (プログラム可能計測器用標準コマンド)
TTL	transistor-transistor logic (トランジスタトランジスタ論理回路)
ULI	universal language interface (汎用言語インターフェース)
VAC	volts alternating current (交流電圧)

注) 本マニュアルにおいてIEEE-488とIEEE-488.2はおのおのGPIBを規定するANSI/IEEE規格488.1-1987とANSI/IEEE規格488.2-1987を指しています。

関連文書

本マニュアルをお読みになる場合、以下に示す文書を参照されれば理解に役立つ場合があるかと思います。

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1987, *IEEE Standard Codes, Formats, Protocols, and Common Commands*

ユーザーの皆様よりの御連絡について

ナショナルインスツルメンツではユーザー各位より当社の製品及びマニュアルについての御感想が寄せられることを期待しております。私どもは、当社の製品を使って皆様が開発されたアプリケーションプログラムにも興味がありますし、問題がある場合は解決のお手伝いを致したいと思います。皆様が当方に御連絡の際のご便宜をはかるため、このマニュアルにはコメントと構成を書き込めばよいようになっている書式を入れてあります。巻末の付録E使用者各位との連絡の中にそれらの書式がありますので御利用ください。

目次

第 6 章

第 7 章

図

表

第１章
序言

この章では、GeneralPurpose Interface Bus (汎用インターフェースバス、GPIBと略します）と本マニュアルを皆様に御紹介いたします。ナショナルインスツルメンツの生みだした各種GPIB製品は皆様の作業環境の欠くべからざる一部としてお役にたって行くことを確信しております。

GPIBとは

GPIBとはリンクあるいはインターフェース・システムであって、結び付けられた電子機器がたがいに通信ができるようにするものです。

GPIBの歴史

はじめGPIBはヒューレットパッカード社によって設計されました。(同社ではGPIBをHP-IBの名で呼んでいます。)ヒューレットパッカード社ではこれを同社が製作するプログラム可能な計器の接続と制御のために開発したのですが、秒当り１メガバイトに達する転送速度の魅力でHP-IBはたちまちコンピュータ間の通信や周辺装置の制御等、他の応用部門で人気を獲得しました。それはやがてIEEE-488として業界で規格化され、応用範囲の広さの故に一般目的用インターフェース・バス(General Purpose Interface Bus)と呼ばれるようになりました。

ナショナルインスツルメンツはGPIBの使用をヒューレットパッカード社製でないコンピュータを使用するユーザーの間に広めました。ナショナルインスツルメンツは高性能、高速度のハードウエア・インタフェース製品とユーザーのもつ計器やコンピュータの知識とGPIBの知識の間のギャップを埋める助けとなる汎用的多機能ソフトウエア製品を専門とするメーカーです。

背景としての知識

このマニュアルはNI-488.2 MS-DOSソフトウエアの文書の一部として編集されたものであり、したがって、本書にはソフトウエアについての参考事項がまとめられています。ハードウエア固有の情報については、個々のハードウエア製品に付属した手引き書を参照してください。

NI-488.2 MS-DOS、 NI-488 MS-DOSおよびGPIB-PCは必ずしも類語ではありません。GPIB-PCはMS-DOS系パーソナルコンピュータに限って用いられるNI-488 GPIBコントローラ系ソフトウエアにはじめつけられた名です。NI-488 MS-DOSは同じパーソナルコンピュータ用NI-488 GPIBコントローラ系ソフトウエアの総称ですが、今日ではこの名のほうが一般に用いられます。NI-488.2 MS-DOSは先行した2種の製品の機能を強化したものですが、IEEE-488.2規格に100パーセント準拠したインタフェース・ボードでしか使用することができません。

第2章 NI-488.2ソフトウエアのインストレーションと構成設定

この章ではNI-488.2ソフトウエアをインストールし構成するための手順を説明しています。またこの章はソフトウエア構成用プログラムIBCONF及びユーザーの皆様が御自分でアプリケーション・プログラムを開発しデバッグする際に使用できるNI-488.2ソフトウエア機能の説明も含んでいます。

NI-488.2ソフトウエア・パッケージについて

ソフトウエアのインストレーションに先立って、これからコピーするディストリビューションディスクに含まれる各ファイルについて、それらの目的その他を理解しておくことが必要です。以下にこれらのファイルの説明をします。

各種メイン・プログラムとファイル

GPIBインターフェースをBASICA, QuickBASIC, Microsoft BASIC, あるいはMicrosoft Cによってプログラムするには次にあげるプログラムとファイルが必要となります。

- GPIB.COMはディスクオペレーティングシステムの起動によってロードされるNI-488.2ドライバです。
- BIB.Mはバイナリインターフェースファイルであって、BASICAで書かれたアプリケーションプログラムがNI-488.2ソフトウエアにアクセスできるようにします。
- QBIB.OBJはバイナリインターフェースファイルであって、QuickBASIC (4.0以降のバージョン) で書かれたアプリケーションプログラムがNI-488.2ソフトウエアにアクセスできるようにします。

- MBIB.OBJはバイナリインターフェースファイルであって、Microsoft Professional BASIC (7.0以降のバージョン) で書かれたアプリケーションプログラムがNI-488.2ソフトウエアにアクセスできるようにします。

- MCIB.OBJはバイナリインターフェースファイルであって、Microsoft C (4.0以降のバージョン) で書かれたアプリケーションプログラムがNI-488.2ソフトウエアにアクセスできるようにします。

- DECL.BASはBASICAアプリケーションプログラムのはじめに置かれるコードを含んでいる宣言ファイルです。

- QBDECL.BASはQuickBASICのアプリケーションプログラムのはじめに置かれるコードを含んでいる宣言ファイルです。

- MBDECL.BASはMicrosoft BASICのアプリケーションプログラムのはじめに置かれるコードを含んでいる宣言ファイルです。

- DECL.HはMicrosoft Cのアプリケーションと共に使用するヘッダファイルです。

その他のプログラムとファイル

下記のプログラムとファイルはNI-488.2のソフトウエアのインストレーションとテスト、ハードウエアの故障診断、及びBASICA、QuickBASIC、BASICとCのプログラミングの例示のために使用されます。

- IBDIAG.EXEはハードウエアのインストレーションのテスト用のプログラムです。このプログラムはIBDIAG.EXEを使ってインストレーション後のハードウエアが正しく機能しているか否かを確認します。

- IBTEST.EXEはNI-488.2ソフトウエアのソフトウエアをテストする場合に使用します。

- INSTALL.EXEはNI-488.2ソフトウエアをインストールしたり、ハードウエアとソフトウエアの構成をテストしたりする場合に使用する多目的プログラムで、メニュー（機能の一覧表示）にしたがって容易に使用することが出来ます。INSTALL.EXEを使ってNI-488.2をインストールした場合CONFIG.SYS (MS-DOSの構成ファイル)は自動的に更新されます。

- IBCONF.EXEはソフトウエア構成用のプログラムであり、NI-488.2ソフトウエアの構成を変更するために使用します。

- IBIC.EXEはキーボードから入力したNI-488.2の関数をインタラクティブに(対話形式で)実行する制御プログラムです。このプログラムを使うことでNI-488.2の関数を知り、計測機器やGPIB機器をプログラムし、自分でアプリケーションプログラムを開発することが出来るようになります。

- APPMON.EXEはメモリレジデント型アプリケーションズモニタープログラムであって、アプリケーションのデバッグの際に有用です。アプリケーションズモニターはGPIBソフトウエアの呼び出しから戻った際プログラムの実行を一時停止(トラップ）し、使用者が関数の引き数、バッファ、戻り値、GPIBのステータスのグローバル変数、その他関係のあるデータを点検できるようにします。

- IBTRAP.EXEはアプリケーションズモニターを構成するためのプログラムです。

- ULI.COMはUniversal Language Interface ソフトウエア・ファイルで、NI-488.2ソフトウエアパッケージのUniversal Language Interfaceオプションを使用したい場合はこのファイルが必要になります。

- DBSAMP.BAS, DQBSAMP.BAS, DMBSAMP.BAS, DCSAMP.C,及び DIBSAMP は BASICA, QuickBASIC, BASIC, C, と IBICのためのデバイス呼び出しプログラムの例です。

- BBSAMP.BAS, BQBSAMP.BAS, BMBSAMP.BAS, BCSAMP.C,及び BIBSAMPは BASICA, QuickBASIC, BASIC, C, と IBICのためのボード呼び出しプログラムの例です。

- BSAMP488.BAS, QBSAMP488.BAS, MSAMP488.BAS, CSAMP488.C, 及び SAMP488は BASICA, QuickBASIC, BASIC, C, と IBICのための488.2呼び出しプログラムの例です。さらに第4章の「NI-488.2 ソフトウエア特性とルーチン」にそのほかの例が示されています。

- DBSAMP.BAS, DQBSAMP.BAS, DMBSAMP.BAS, DCSAMP.C, および DIBSAMPはおのおのBASICA, QuickBASIC, BASIC, C, および IBIC中でのデバイス・コールの例を示すプログラムです。このほかの各プログラム例は第4章の「NI-488.2ソフトウエア特性とルーチン」中に示されています。

ディストリビューション・ディスクには種々のリードミー(Readme)ファイルが入っています。README.DOCではNI-488.2ソフトウエアのことが述べられています。その他のリードミーファイルでは各プログラム言語使用上の注意事項が述べられています。

ステップ1 ― ソフトウエアのインストレーション

NI-488.2のディストリビューション・ディスク中のINSTALL.EXEと言うプログラムがNI-488.2ソフトウエアをインストールし、テストしてくれます。以下で使用される用語で、*source disk*あるいは*source directory* はNI-488.2ディストリビューションディスクを意味します。*destination directory* はハードディスクなりフロッピーディスク上のNI-488.2ソフトウエアがインストールされる場所を指します。(通常C:*boardname*[ボード名](例えばC:\AT-GPIB)です。)また、*boot drive* と言う用語はコンピュータにスイッチを入れたり、リスタートしたりしたときにコンピュータが読み出しをするドライブのことです。

準備

インストレーションの第1段階は、使用コンピュータがフロッピーディスクからスタートするかハードディスクからスタートするかで違います。御使用のシステムに合わせて第1段階を行ってください。

- フロッピーディスクからのスタート

  MS-DOSをフロッピーディスクからスタートさせるには、デスティネーション・ディスクにディストリビューションディスクからコピーしたNI-488.2ソフトウエアを保持するに十分な空きがあることが必要です。ディストリビューションディスクの中のREADME.DOCファイルには、NI-488.2ソフトウエアの各部をインストールするに必要なディスク・スペースのリストが含まれています。

  さて、まだコンピュータにスイッチが入っていないようでしたら、スイッチを入れてスタートさせてください。次にシステム・ディスクを第1ディスクドライブ（通常ドライブA:と言います）に、ディストリビューションディスクを第2ディスクドライブ（通常ドライブB:と言います）に挿入してください。

- ハードディスクからスタートする場合

  MS-DOSをハードディスクからスタートさせるには、ハードディスク中にディストリビューションディスクからコピーしたNI-488.2ソフトウエアを保持するに十分な空きがあることが必要です。ディストリビューションディスクの中のREADME.DOCファイルには、NI-488.2ソフトウエアの各部をインストールするに必要なディスクスペースのリストが含まれています。

  もし旧バージョンのNI-488.2ソフトウエアがすでにシステム内に存在する場合は、それら旧ファイルは新ファイルのインストレーション前に除いておく必要があります。

  コンピュータにスイッチが入っていないようでしたら、スイッチを入れてスタートさせてください。次にディストリビューションディスクをフロッピーディスク・ドライブに挿入してください。

INSTALLを走らせる

コンピュータがスタートしましたらNI-488.2のディストリビューションディスクからINSTALLを走らせます。先ず次のようにキーボードに入力します。

```
y:install                    <Enter>
```

上の例で、yはディストリビューションディスクの挿入されたドライブを示します。

するとINSTALLはメインメニューを表示します。メインメニューは4種のオプション（選択肢）を示します。それらPartial GPIB Installation(部分的GPIBインストレーション)、Full GPIB Installation(GPIBの全部のインストレーション)、Diagnostics(診断)、およびReturn to DOS(復帰)です。

- Partial GPIB Installationを選択しますと、新しいメニューが現われ、NI-488.2ソフトウエアのどの部分をインストールするか選ぶことが出来ます。望む部分を選択すると、インストレーションスクリーンが現われます。使用中のディスクの空きスペースが小さい場合はこの部分インストールのオプションを選ばれるのが良いでしょう。

- Full GPIB Installationを選択した場合は直ちにインストレーションスクリーンが表示されます。このスクリーンの仕事が完了しますと、メインメニューに戻ります。

- 第3のオプション、すなわちDiagnosticsを選択しますと、ハードウエア及び／またはソフトウエアを診断するプログラムを使用することが出来るようになります。

- 最後のオプション、すなわちReturn to DOSを選択しますと、インストレーションプログラムを終了してDOSに戻ります。

注）<Escape>キーを押していただければ何時でもインストレーションプログラムを中止することが出来ます。

Partial Installation (部分インストレーション)

メニューからPartial GPIB Installationを選びますと、INSTALLプログラムが新しいメニューを表示します。このメニューには6つのオプションがあり、そのうちどれでも、あるいは好きな数だけのオプションでも（全部でも）選ぶことが出来ます。6つのオプションは次の通りです。

- Driver and Support Files (ドライバとサポート用各ファイル)。このオプションを選びますとNI-488.2ソフトウエア(GPIB.COM)及びそれをサポートする各ファイル(IBIC.EXE, IBCONF.EXE, APPMON.EXE, IBTSTB.EXE, BIBSAMP, DIBSAMP, SAMP488, およびREADME.DOC)がコピーされます。

- Microsoft C Language Interface (マイクロソフトC言語インターフェース）。このオプションを選ぶとマイクロソフトCのアプリケーションプログラムをNI-488.2ソフトウエアとインターフェースさせるのに必要な各ファイルがコピーされます。

- BASIC 7.x Language Interface (BASIC 7.Xソフトウエア 言語インターフェース）。このオプションではマイクロソフトBASIC 7.0のアプリケーションプログラムをNI-488.2ソフトウエアとインターフェースさせるのに必要な各ファイルがコピーされます。

- QuickBASIC Language Interface (QuickBASIC 言語インターフェース）。このオプションではQuickBASICのアプリケーションプログラムをNI-488.2ソフトウエアとインターフェースさせるのに必要な各ファイルがコピーされます。

- BASICA Language Interface (BASICA言語インターフェース）。このオプションではBASICAのアプリケーションプログラムをNI-488.2ソフトウエアとインターフェースさせるのに必要な各ファイルがコピーされます。

- Universal Language Interface (ユニバーサル言語インターフェース）。このオプションではアプリケーションプログラムをNI-488.2ソフトウエアのUniversal Language Interfaceとインターフェースさせるのに必要な各ファイルがコピーされます。

以上のオプションは、そのうちのいずれか、あるいは全部でも選ぶことが出来ます。選択には矢印キー及びスペースバーを使います。選択を終わりましたら<Enter>キーを押してください。INSTALLプログラムがインストレーションスクリーンを表示します。

全部のインストレーション

INSTALLが表示したスクリーンはインストールする設定値をデフォールト（省略時値）にするか、あるいは変更するかを決めるためのものです。スクリーンには3フィールドがあり、おのおののフィールドはINSTALLが今使用中のシステムに合わせて選んだデフォールトの値を持っています。いずれかのオプションを変更したい時は、矢印キーを使用してください。インストレーションを開始するには<Enter>キーを押してください。

INSTALLがここで必要とする3種のフィールドの情報はソース・ディレクトリ、デスティネーション・ディレクトリ、およびブート・ドライブです。

- デスティネーションディレクトリはINSTALLがNI-488.2ソフトウエアをコピーするドライブとディレクトリの名前です。ハードディスクにNI-488.2ソフトウエアをインストールする場合にはこのフィールドは通常C:*boardname*(ボードの名)、たとえばC:\AT-GPIBとなります。多くの場合、初めにコンピュータをスタートさせたドライブの名をこのフィールドのドライブ名とします。

- ブートドライブのフィールドにはハードディスクなりフロッピーディスクなり、このコンピュータをスタートさせたドライブの名を選びます。フロッピーディスクからスタートした場合にはこのフィールドはA:となるのが普通です。ハードディスクからスタートした場合にはC:となるのが普通です。

上記の情報を確認しましたら、<Enter>を押します。すると、INSTALLがこれらの情報に問題が無いか否かをチェックし、これらのドライブにアクセスするのに問題があれば、エラーメッセージを出し、問題のあるドライブについては情報の入力をやり直すように要求してきます。

インストレーションが完了しますと、INSTALLがCONFIG.SYSを変更するかどうか聞いてきます。これに対してyesを入力しますと、INSTALLは以下に示す行をCONFIG.SYSに付け加えます。

```
device=dir\gpib.com
```

ここで*dir*はINSTALLがNI-488.2ソフトウエアファイルをコピーしたディレクトリです。例えばAT-GPIBを使用している場合は*dir*はC:\AT-GPIBとなります。もしもNI-488ソフトエアの旧バージョンがすでにインストールされている場合は、INSTALLはCONFIG.SYS中の旧バージョンの情報を新バージョンの情報で置き換えます。

INSTALLの質問に*no*と答えますと、INSTALLはCONFIG.SYSに付け加えるべき正しいラインを知らせるメッセージを表示します。

診断

ソフトウエアのインストレーションが完了すると、スクリーンはメインメニューに戻ります。すでにGPIBインターフェースボードがコンピュータにインストールされている場合はそれからDiagnosticsを選んでください。まだGPIBインターフェースボードがインストールされていなかった場合は<Escape>を押してINSTALLを離れ、購入したボードについてきたGetting Started と言うタイトルのマニュアル中に示されるハードウエアインストレーションの手続きにしたがってボードをインストールしてください。次に、ボードがインストールされたコンピュータにスイッチを入れ、再びINSTALLを走らせ、Diagnosticsメニューを選び、以下の手続きを行ってください。

Diagnosticsメニューではハードウエアの診断（IBDIAG.EXE),ソフトウエアの診断（IBTEST.EXE),あるいは両方の診断のいずれかを

選択します。まずハードウエアの診断を選んで見ましょう。もしその際IBDIAGからエラーメッセージが返ってきましたら、次の事項をチェックして見てください。

- ハードウエアの構成(スイッチとジャンパーの設定) を調べ、以下の事柄を確かめて下さい。

  - ハードウエアの構成は先にIBDIAGの要求にしたがって入力した設定値と合致していますか？

  - このボードの構成設定値は同じコンピュータ中にある他のボードやデバイスの設定値と同じになっていませんか？その場合には、今のボードを設定し直し、それからハードウエア診断のプログラムをもう一度走らせて見てください。

- このGPIBインターフェースボードがGPIBデバイスと接続していないかどうか確認してください。IBDIAGを走らせるには、このインターフェースボードが如何なるGPIBデバイスとも接続していないことが必要です。

以上の手続きを踏んでなおかつ問題がある場合には、インターフェースボードについてきたGetting Startedと題するマニュアルの付録にある Hardware and Software Configuration Form と言うハードウエアとソフトウエアの構成について報告する書式に必要事項を全部書き込んだ上ナショナルインスツルメンツのテクニカルサポート部に連絡してください。

ハードウエア診断に合格した場合は、以下この章で述べる手続きに進んでください。

ステップ2 － ソフトウエアの構成

ソフトウエア診断テストを行うには、あらかじめNI-488.2ソフトウエアがロードされていなければなりません。もしもインストレーションを今完了したばかりで、システムのリスタートを済ませていない場合はソフトウエアはまだロードしていません。この場合、<Escape>を押してINSTALLを一旦終了し、システムのリスタートをし

ます。但し、次に述べるようにリスタートの前にIBCONF(ソフトウエア構成プログラム）を走らせる必要がある場合もあるので注意してください。

GPIBハードウエアのスイッチあるいはジャンパーの設定値を変えた後、あるいはソフトウエアのデフォールトのソフトウエア構成オプションの変更が必要な場合にはIBCONFを走らせる必要が出てきます。ハードウエアの設定値をまったく変えてなくて、デフォールトのソフトウエア構成も変更の必要がない時は、IBCONFを走らせる必要はありません。しかし、ソフトウエアの構成を見ておきたいという意味でIBCONFを走らせて見ることは構いません。

IBCONFの走らせ方、構成可能のソフトウエアオプション各種、及びそのデフォルト値については、この章の後のほうに説明がありますので参照してください。

ステップ3 － ソフトウエア診断

NI-488.2ソフトウエアをコンピュータのメモリにロードするにはコンピュータをリスタートしなければなりません。(コンピュータをフロッピーディスクからスタートした場合は、リスタートはスタートと同じフロッピー・ディスクを使用して行ってください。）普通リスタートが必要になる場合は、NI-488.2ソフトウエアを最初にインストールする時と、ハードウエアの設定値を再構成する必要が出てきた時だけです。

コンピュータのリスタート(再始動) は<Control-Alt-Delete>を同時に押し下げることにより行います。これによりNI-488.2ソフトウエアがメモリにロードされます。

NI-488.2のインストレーション後、それが正しくシステムにインストールされたか否かを確かめるため、IBTESTを走らせてください。エラーが起きた場合は次のことをチェックしてください。

- このGPIBインターフェースボードのGPIB デバイスとの接続は許されません。IBTESTを走らせるには、GPIBインターフェースボードが如何なるGPIBデバイスと接続していることも許されません。

- GPIBインターフェース・ボード上のハードウエア構成を変更した後でしたら、IBCONFを走らせてソフトウエアの現在のソフトウエア構成をチェックして、ハードウエアの設定値とマツチすることを確かめて下さい。それには、この章の前にあったステップ2のソフトウエア構成の項を参照してください。

- コンピュータをスタートしたディスク上のCONFIG.SYS が次のラインによって正しく変更されていますか？

  ```
  device=dir\gpib.com
  ```

  上のラインで、*dir* はINSTALLがNI-488.2ソフトウエアの各ファイルをコピーしたディレクトリをさします。例えば、AT-GPIBを使用している場合は、*dir* はC:\AT-GPIBとなります。

- NI-488.2ソフトウエアをインスとールし構成した後コンピュータをリスタートしましたか？これについてはこの章の前に出たステップ2のソフトウエア構成の項を参照してください。

以上の手続きを踏んでなおかつ問題がある場合には、インターフェースボードについてきたGetting Startedと題するマニュアルの付録に出ている Hardware and Software Configuration Form というハードウエアとソフトウエアの構成報告書式に必要事項を全部書き込んだ上ナショナルインスツルメンツのテクニカルサポート部に連絡してください。

もし何のエラーも起きなかった場合は、次に進んでソフトウエアの使用方法と自分でアプリケーションプログラムを開発する方法を学んでください。

ステップ4 － アプリケーション・プログラム開発

計測器との通信は対話形式の（インタラクティブな）制御によって最も容易に習うことが出来ます。GPIBディレクトリには対話形式の制御プログラムIBIC.EXEが入っています。このプログラムを使用して計測器への書込・読出が出来ます。このプログラムは使用中のGPIBデータ転送状態を常に更新し、エラーがあればその旨表示します。

このプログラムの使用には、計測器をバスに接続した後次のコマンドを入力します。

```
cd \dir
```

ここで*dir* はINSTALLがNI-488.2をコピーしたさきのディレクトリの名です。

```
IBIC
```

その後、このマニュアルの第６章　IBIC で述べられる各種関数の入力をはじめます。

IBCONF

IBCONF.EXEはスクリーンを見ながら対話形式で使用出来るプログラムです。これによってインターフェース・ボードとそれに接続されたGPIBのデバイスの構成パラメータを変更することが出来ます。また、このプログラムにはバッチ・モードも付け加えられているので。それを利用して対話形式によらないで構成を行うことも出来ます。

対話形式で使用の場合、IBCONFはディスク中のGPIBドライバのファイルから構成パラメータを読み出して画面に表示してくれます。表示されたパラメータは調べた上で使用者の特殊な要求にしたがってどれでも変更することが出来ます。変更を終えてIBCONFプログラムを終了させると、変更はディスク上のGPIBドライバのファイル中に保存されます。

IBCONFによってGPIBソフトウエアに加えられた変更をDOSアプリケーションによって使用されるメモリ・レジデントのGPIBドライバ中に移すためには２つの方法が可能です。まずコンピュータをリスタートすることによって変更したGPIBデバイスドライバをメモリにロードし直すという従来の方法があります。第２の方法はIBCONFを終了させる際に同プログラムによってメモリ中のGPIBドライバを変更するという方法です。この２番目の方法はより簡単ですが、GPIBの２つのコピー（ディスクに記憶されたものとDOSによりメモリ中にロードされたもの）の間に互換性がある場合のみ使用可能です。

IBCONFを使用するのに最も簡単な方法はディレクトリをGPIBディストリビューションファイルを含んだディレクトリに変えてから以下のコマンドを入力する方法です。

```
IBCONF
```

もし現在のディレクトリにドライバ（すなわちGPIB.COM)がない場合は、IBCONFはGPIB.COMを変更するために探します。一般に、IBCONFは次のプロセスにしたがってGPIB.COMを見出し、それを変更します。

1. もしもC:\config.sysがあってdevice=<path>gpib.comと言うフォーマットのラインを含む場合は、そのGPIB.COMファイルを使用します。

2. もしもconfig.sysファイルが現在のドライブのルートディレクトリにありdevice=<path>gpib.comと言うフォーマットのラインを含む場合は、そのGPIB.COMファイルを使用します。

3. GPIB.COMが現在のディレクトリに含まれている場合はそのファイルを使用します。

表2－1はIBCONFをスタートできるオプションの各種を示します。

表2－1 IBCONFの各オプション

IBCONFオプション	手続き
Driver	このオプションではIBCONFはgpib.comをさがす代わりに、指定されたドライバを構成します。(例 IBCONF d:\at-gpib\at-gpib.com)

(次ページに続く)

表2－1 IBCONFオプション(前ページより続く)

IBCONFオプション	手続き
-b *filename*	このオプションではバッチモードで構成を行います。この場合、IBCONFは、指定されたファイル内の構成の情報を使用してバッチモードで実行されます。(例 IBCONF -b gpib.cfg) この章で後出する *IBCONF Batch Mode* のバッチモードの項を参照のこと。
-d	これはダイナミック構成オプションです。このオプションを使用していてIBCONFを終了(Exit)させると、IBCONFはメモリにロードしたドライバを自動的に更新します。この章の後に出る*IBCONFのExit*の項を参照してください。
-f	このオプションはダイナミック構成を不可能化します。IBCONFの終了(Exit)に際し、このオプションはIBCONFがメモリにロードされたドライバを無視するようにさせます。したがってIBCONFはロードされたドライバの更新を行いません。
-e	これはエキスパートモードのオプションです。このモードではIBCONFは終了(Exit)に際して警告メッセージを出しません。この章の後で出る*IBCONFのExit*の項を参照してください。
-m	これは白黒モードのオプションです。、これを選択すると、カラーモニターの場合でもIBCONFは白黒モードで動作します。

(次ページに続く)

表2－1 IBCONFオプション(前ページより続く)

IBCONFオプション	手続き
-vb	これはBIOSを通じてビデオにアクセスするオプションです。これを選択すると、IBCONFはBIOSシステムのルーチンを使用してスクリーンの書込をするようになります。スクリーンに直接書き込む方法に比べてこの方法は遅いのですが、規格外のある種の動作環境ではこの方が互換性が良い場合があります。

IBCONFの上位レベルと下位レベル

IBCONFは上位レベルと下位レベルの両レベルで動作します。上位レベルはボード・デバイスのマップからなり、ドライバにより定義されたGPIBシステムをグラフィックに示します。下位レベルはシステムを構成するボードとデバイスを個々に説明する各スクリーンからなっています。

GPIBxボードの上位レベルのデバイス・マップ

図２－１はIBCONFの上位レベルを示します。

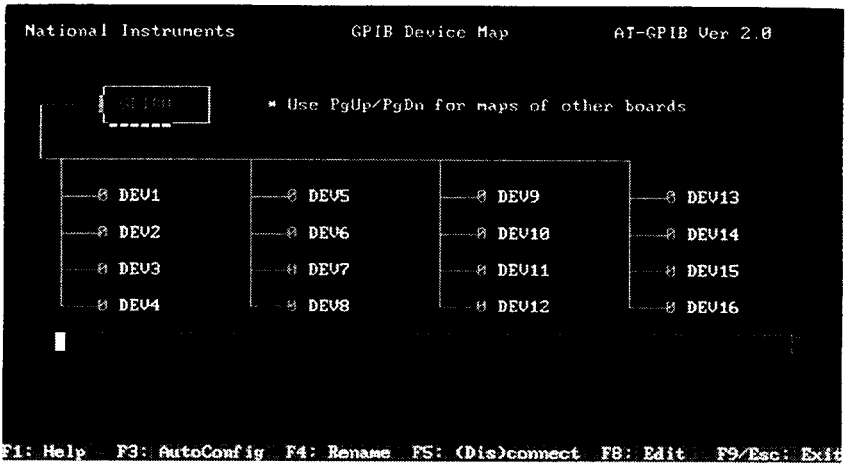


図２－１　IBCONFの上位レベル

図２－１に示されるように、IBCONFの上位レベルのスクリーンはドライバにより制御される全てのデバイスの名を表示しています。また、この図はどのデバイスがGPIBxと名付けられるインターフェースなりアクセスボードによってアクセスされるのか示しています。なお、２ボード・ドライバの場合GPIBx のx は０か１です。また、カーソル・コントロール用キーを使用してこの図の中のどのデバイスも選択出来ます。また、Microsoftのマウス、或いはそれと互換のマウスを使用して選択を行うこともできます。

上位レベルにおいては次の各オプションを選択出来ます。

- Device Maps of the Boards (各ボードのデバイス・マップ)
- Help (ヘルプ)
- Rename (名称の変更)
- (Dis)connect (電気的接続あるいは切断)
- Edit (編集)
- Output GPIB driver Configuration (GPIBドライバ構成の出力)
- Autoconfigure (自動的構成)
- Exit (終了)

各ボードのデバイス・マップ

<PageUp>及び<PageDown>を使用して異なったGPIBインターフェースボード(アクセスボードと呼ばれます）のデバイス・マップの間を移るようにしてください。マップはどのデバイスがどちらのボードに付属するかを示してくれます。デフォルトでは各GPIBインターフェースボードには同じ数のデバイスが接続されています。

ヘルプ

ファンクションキーF1を使ってIBCONFの総合的オンラインヘルプを受けることが出来ます。ヘルプの情報にはIBCONFの上位レベルに関する各機能及び一般的な用語が説明されています。

名称の変更

デバイスの名称変更にはファンクションキーF4を使います。カーソルコントロールキーによって名称変更したいデバイスを選びます。

次にF4キーを押してからデバイスの新しい名を入力します。名前には8文字を限って使用出来ます。MS-DOSでファイルに命名するときと同じ規則が適用されるのですが、拡張子（.xxx)は使用出来ません。

MS-DOSでも規定しているように、以下に示す文字は無効文字であって、使用出来ないことになっています。

```
.     "     /     \     [     ]     :
|     <     >     +     =     ;     ,
```

(16進数系で21以下のASCII文字)

また、次の予約語も使用出来ません。

CON　NUL

大文字と小文字は同じ文字として扱われます。つまり、PLOTTERはplotterと同じです。したがって、IBCONFは全ての小文字を大文字として表示します。

注）GPIBデバイスにファイル、ディレクトリ、そして／あるいはサブディレクトリに存在する名前をつけることは許されません。ファイルシステムにPLTR.DATと言うファイルがあったり、PLTRと言うサブディレクトリがあったりする場合にGPIBデバイスにPLTRと言う名をつけると問題が起こります。また、アクセスボードの名、例えばGPIB0、は変えることが出来ないということを覚えておいてください。

デバイスあるいはアクセスボードの名をあらわすストリングはアプリケーションプログラムの始めに呼び出される 関数 ibfind の最初の引き数です。ibfind の詳しい説明についてはこのマニュアルの第4章と第5章を参照してください。

電気的接続あるいは切断

あるデバイスをひとつのボードに接続したり、あるいは接続を断ったりするすることを論理上で行うにはファンクションキーF5を使います。スクリーン上のカーソルをカーソルコントロール用キーを使って該当するデバイスまで動かした後F5キーを押します。

編集

ある特定のデバイスなりボードなりの特性を編集したり調べたりする際にはファンクションキーのF8或いは<Enter>キーを使います。まずカーソル・コントロール用キーを使ってカーソルを目的のボードなりデバイスまで動かし、次にF8キーを押します。この手続きを踏むと下位レベルのIBCONFが現われ、該当のボードかデバイスの特性のリストを示します。その編集が終わりましたらファンクションキーF9か<Escape>を押して編集オプションを終了させます。

GPIBドライバ構成の出力

GPIBドライバの構成において、ドライバのテキスト版をディスクファイルに書き込むことができます。F2ファンクションキーを使用すれば、IBCONFに命令して現在使用のディレクトリ中に、GPIB.TXTと言う名のテキストファイルを作らせることができます。ファイルは使用中のGPIBドライバの内容を説明しています。これは情報をしる目的のみに使用されます。

自動構成

IBCONFに自動的に構成を行わせるにはファンクションキーのF3を使用します。IBCONFがある特定のGPIBボードの自動構成を命じられると、GPIB上の全てのリスンアドレスをチェックしてリスニングデバイスを探し出します。次に、IBCONFはボードのデバイスマップを調整して反応を示すデバイスのみが接続されているようにします。また、それらのデバイスの1次及び2次アドレスフィールドを調整してそれらがリスナとして反応したアドレスとマッチするようにします。以上の動作は数秒で完了します。このようにして接続されたデバイスにはそれぞれの機能を表わすような名を新たにつけるのがよいでしょう。

自動構成を行うには、あらかじめシステム中の全てのデバイスが接続され、スイッチが入っている状態であることが必要です。

1つ以上のボードを自動構成する際に、IBCONFは高いナンバーのインターフェースボードからデバイスを切り離してきて、低いナンバー

のボードを構成するのに使います。ですから、1つ以上のインターフェースボードがシステム中にある場合は、全てのボードを自動的に構成するように計画するべきです。この場合、自動構成はNo. 0のボードから始めて高いナンバーのボードに及ぶべきです。もしも唯一つのボードだけ自動構成すると決めた場合は、No. 0 のボードはそのままにしておき、No. 1 のボードを自動構成するようにするべきです。

注)　LabWindowsの使用中には自動構成を行わないでください。

Exit (イグジット)

IBCONFを終了するにはファンクションキーのF9或いは<Escape>キーを使います。変更を加えた後でF9を押しますと、IBCONFは Save changes? (変更を保存しますか？) と聞く表示を出します。これに対して y (yes) と答えると変更は保存され、n (no) と答えると保存されません。また、c (cancel)をタイプすると続けてIBCONFにとどまることになります。Exitについてのより詳しい説明はこの章のあとのほうの Exiting IBCONF の項にありますので参照してください。

下位レベルのデバイス／ボードの特性

図2－2にIBCONFの下位レベルが示されています。

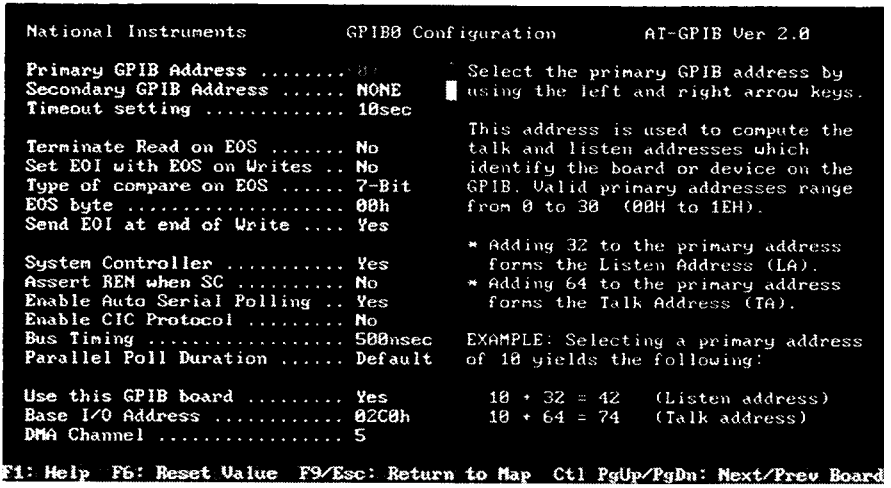


図2－2 IBCONFの下位レベル

図2－2を見てわかるように、IBCONFの下位レベルのスクリーンはボードやデバイスについて、アドレスとかタイムアウトなどの特性の現在の設定値を示します。これら下位レベルのスクリーンにアクセスするには、IBCONFの上位レベルのスクリーンにおいてボードかデバイスを選択してからファンクションキーF8或いは<Enter>キーを押します。ここに見られる構成設定の項目は、ボードやデバイスの通信その他のオプションを必要や好みにしたがって変更する手がかりとして与えられています。デバイスの設定値は、そのデバイスの機能の使用時にアクセスボードが利用出来るデバイス特性を規定しています。ボードやデバイスの設定値は、ボードの機能の使用時に利用される各特性を規定しています。

下位レベルでは次の機能を使用出来ます。

- Change Characteristics (特性の変更)
- Change Board or Device (ボードやデバイスを変える)
- Help (ヘルプ)
- Reset Value (リセット値)
- Return to Map (マップに復帰)

Change Characteristics [特性の変更]

あるデバイスなりボードのある特定の特性を変更するには、カーソルをその特性のところまで動かします。<PageUp>, <PageDown>, <Home> あるいは <End> を使って動かすことも出来ます。カーソルが目的の特性のところに来ましたら、右／左の矢印を使って望みのオプションを選ぶか、直接キーボードからオプションを入力します。スクリーンの左上のコーナーに、特性によってどちらの方法を使って選択したら良いかの指示が出ています。

Change Board or Device [ボード或いはデバイスの変更]

<Control-PageUp> と<Control-PageDown>を使って次のボード或いはデバイス、またはこの前のボード或いはデバイスに移るようにします。例えば、今DEV3を編集中であるとすると、<Control-PageUp>を押せばDEV4を編集できる状態になります。

Help [ヘルプ]

ファンクションキーF1を使えばIBCONFの総合的オンラインのヘルプを利用することが出来ます。ヘルプの情報には、IBCONFの下位レベルに関係する諸機能と一般的な用語の説明があります。

Reset Value [リセット値]

ある特性のオプションをデフォルトの値に再設定するにはファンクションキーのF6を使って行います。

Return to Map [マップに復帰する)

下位レベルにあるときは、ファンクションキーのF9か <Escape> を押せばIBCONFの上位レベルのデバイス・マップに戻ることが出来ます。

Default Configurations [デフォルトの構成]

NI-488.2 MS-DOS ソフトウエアには、工場設定のデフォルト構成値があります。例えば、32のGPIBデバイスはDEV1に始まってDEV32まで名がつけられています。これらはより実体を表わした名前に変えたほうがよいということが考えられます。例えば、ディジタルマルチメータにはMETERと言う名をつけたほうが解り易いといえます。

IBCONFを使えば、ソフトウエアの中の現在のデフォルト設定を見ることが出来ます。

IBCONFによるNI-488.2 MS-DOS ソフトウエアの変更を行わないときは、このソフトウエアのデフォルト設定が引き続き効力を持つことになります。

Default Values [デフォルト設定値]

以下がソフトウエアのデフォルト設定値です。

- 32のデバイスがあり、それらにはDEV1からDEV32にいたる符号的な名がついている。
- 4つのアクセスボードがあって、GPIB0とGPIB1、GPIB2、GPIB3と言う符号的名がつけられている。アクセスボードの名の変更は許されない。

- アクセスボードのGPIB0が可能化され、GPIB1、GPIB2、及びGPIB3は不可能化されている。

- 最初の16デバイスのGPIBアドレスはデバイス番号と相応している。例えば、DEV1はアドレス1にあるという具合である。これら16デバイスはGPIB0に付属し、それをアクセスボードとしている。

- 残りの16デバイス（すなわちDEV17からDEV32まで）はGPIB1に付属し、それをアクセスボードとしている。これらのGPIBアドレスは1から16にわたる。例えば、DEV17はアドレス1にある。

- 各GPIBインターフェースボードはそれ自身のバスのシステムコントローラ(System Controller)であり、GPIBアドレスの0を持っている。

- ENDメッセージは各データメッセージの最後のバイトと一緒にデバイスに送られる。End-Of-String (EOS) の文字は認識されない。

- I/Oと待機関数の呼び出しのタイム・リミットは、約10秒に設定されている。

- ベースI/Oアドレス、インタラプト設定、及びDMAチャネルについて、各GPIBインターフェースボードはそれ自身のデフォルト設定値を持っている。これらの設定値については、インターフェースボードに附いてきたGetting Startedと題するマニュアルによりチェックのこと。

Device and Board Characteristics [デバイスとボードの諸特性]

以下のIBCONFおけるボード及びデバイスの諸特性の説明はNI-488.2 MS-DOSドライバのあらゆるバージョンに共通して言えるものです。ある特定のバージョンのドライバに限定した情報は、インターフェースボードに附いてきたGetting Startedと題するマニュアルに含まれています。さらに、各特性についてのより行き届いた情報は、カーソルがIBCONFのスクリーン上のある特定のフィールドに置かれ

たときにスクリーン上に表示されます。以下の特性のほとんど全部はデバイスとボードの両方に当てはまります。いくらかの特性でボードだけにしか当てはまらないものがありますが、それらはその旨明記されます。

Primary GPIB Address [1次GPIBアドレス]

全てのデバイスには16進数の00から1E(10進数の0から30)の範囲内の特定数で1次アドレスが与えられます。ドライバは1次アドレスに20を加える事により自動的にリスンアドレスを形成します。また、それは1次アドレスに40をたすことによりトークアドレスを形成します。例えば、16進数の10(hex 10) はhex30のリスンアドレスとhex50のトークアドレスを持つことになります。如何なるデバイスのGPIBアドレスもそのデバイスの中で、ハードウエアスイッチかソフトウエアプログラムで設定されます。このアドレスとIBCONF中に示されたアドレスとは一致していなければなりません。デバイスの資料を参照してデバイスのアドレスについての指示を得るようにしてください。全てのGPIBボードの1次アドレスは0です。GPIBインターフェースボードにはGPIBアドレスを選ぶためのハードウエアスイッチは附いていません。

Secondary GPIB Address [2次GPIBアドレス]

拡張アドレスを使用するデバイスやボードはhex60からhex7E(10進数で96から126)の範囲に2次アドレスを持つか、2次アドレスを無効化するためにオプションのNONEを選択するかしておかなければなりません。1次アドレスの場合と同じく、あるデバイスの2次GPIBアドレスはそのデバイスの中で、ハードウエアのスイッチによるかソフトウエアのプログラムによって設定されます。このアドレスとIBCONF中にあるアドレスとは一致していなければなりません。デバイスの資料を参照して2次アドレスについての指示を得るようにしてください。IBCONFで変更していない限り、全てのボードとデバイスの2次アドレスは無効化されています。この特性についてのデフォルトのオプションはNONEです。

Timeout Settings [タイムアウト設定値]

タイムアウト値とは、ibrd, ibwrt, あるいは ibcmd の様なI/O関数がタイムアウトになる前にかけてもよい最小時間の概数です。それはまた、TIMOビットが事象マスクに設定されている場合で、ibwait 関数が復帰前に事象を待つ時間の長さであるとも言えます。もしも事象マスク内にあるSRQIビットとTIMOビットが ibwait 関数迄渡されていて、一方SRQが見い出されない場合は、この関数はタイムアウトです。より詳細な説明については、このマニュアルの第3章と第5章にあるIBWAIT関数の記述を参照してください。この特性のデフォルトのオプションは 10sec です。

Serial Poll Timeouts [シリアルポールタイムアウト(デバイス特性のみ)]

シリアルポーリングタイムアウト(デバイス特性値のみ)
このタイムアウト(制限時間) 値はドライバがシリアルポーリングの応答を待つ時間の長さを決めるものです。IEEE-488規格ではコントローラが応答バイトを待つ時間を規定していません。ドライバのデフォルトは1秒ですが、ほとんどのデバイスについてはこの時間で十分であると思われます。もしシリアルポーリングで問題があるようでしたら、タイムアウト値を長くして見てください。

Terminate READ on EOS [EOSにおいて読取を停止]

いくつかのデバイスはデータメッセージの最後のバイトにEOSバイトを送ってきます。このフィールドでyesと答えておくとGPIBボードはEOSバイトを受け取ってから読取作業を停止します。この特性では、デフォルトのオプションはnoです。

Set EOI with EOS on Write [書き込みにおいてEOSでEOIをセット]

このフィールドでyes答えておくとGPIBボードは書き込み作業中にEOSバイトを検出するとEOIラインを出します。この特性のデフォルトオプションはnoです。

Type of Compare on EOS [EOSでの比較のタイプ]

このフィールドでは、EOSバイトがあったときに行う比較のタイプを規定します。この場合、8ビット全部を比較するか、ASCIIあるいはISOフォーマットにおいて下位の7ビットだけを比較するかを明らかにすればよいのです。このフィールドは「書込においてEOSでEOIをセット」のフィールドか「EOSで読取を停止」のフィールドにyesと答えていた場合のみ有効になります。この特性のデフォルトオプションは7-bitです。

EOS Byte [EOSバイト]

いくつかのデバイスは、ある選ばれた文字が検出された場合に読取作業を停止するようプログラムすることが出来ます。改行文字（hex 0A）は通常使用されるEOSバイトです。

注)　書き込み作業中、ドライバはデータストリングの後に自動的にEOSバイトを付け加えたりはしませんので、EOSバイトはデータストリングの中に明確に付け加える必要があります。EOSバイトの指定はドライバにその値を知らせてI/O作業が正しく終了されるようにするためにのみ行われます。

この特性のデフォルトのオプションは00Hです。

Set EOI at End of Write
[書き込みの最後のバイトでEOIをセット]

いくつかのデバイスはリスナであるときにトーカがデータメッセージを終える際に最後のバイトでEOIを出すことを要求します。このフィールドにおいてyesと答えておくとGPIBインターフェースボードが最後のバイトでEOI行を出すようにします。この特性についてのデフォルトのオプションはyesです。

System Controller [システムコントローラ (ボード特性のみ)]

このフィールドはボード特性のスクリーンのみに現われます。GPIBシステムにおけるシステムコントローラは究極的にバスのコントロールを保つデバイスのことです。如何なるGPIBシステムにおいても、1つ以上のデバイスがシステムコントローラになることは出来ません。時として、例えばGPIBインターフェースボードがコンピュータのネットワークをつなげているような場合には、外のデバイスがシステムコントローラになることがあり、その様な場合にはGPIBボードはシステムコントローラには指定されません。このフィールドでnoと答えればボードはシステムコントローラに指定されず、yesと答えればボードにはシステムコントローラの能力が与えられます。一般的には、GPIBボードがシステムコントローラに指定されるべきだといえます。この特性のデフォルトオプションはyesです。

Assert REN when SC [SCならばRENを アサート (ボード特性のみ)]

このフィールドでyesと答えると、ボードにシステムコントローラの能力が与えられ、それがオンラインに置かれている限りRENを自動的にアサートします。noの答えが与えられた場合RENをアサートするためには ibsre 呼び出しの手続きが必要になります。この特性のデフォルトオプションはnoです。

Enable Auto Serial Polling [自動シリアルポーリングの可能化 (ボード特性のみ)]

このオプションはGPIBサービスリクエスト(SRQ)ラインがアサートされているときに自動シリアル・ポーリングを可能化するか不可能化するかを選択するものです。ポーリングの後で肯定的なポールのレスポンスが保存され、ibrsp デバイス関数で読み取ることが出来ます。より詳細な情報については、このマニュアルの第5章を参照してください。普通この特長はIEEE-488仕様のデバイスとは問題を起こさないのですが、もし問題があれば、このフィールドで no と答えることによりこの機能を不可能化します。この特性のデフォルトオプションはnoです。

注)　もしドライバのON SRQの特長を利用したいのなら、自動シリアルポーリングはOFFにしてください。

Enable CIC Protocol [CIC プロトコルの可能化 (ボード特性のみ)]

コントロールがボードから他のデバイスにわたされた後でデバイスレベルのNI-488呼び出しが行われた場合、このプロトコルを可能化するとボードがシリアルポールステータスバイトのhex 42を使ってSRQをアサートします。すると、その時のコントローラは、これをボードのコントロール返還要求と察することになっています。そこで、その時のコントローラがコントロールをボードに戻せば、前述のデバイス呼び出しは無事に行われます。しかし、もしもコントロールの返還がタイムアウトの時間までに行われないと、呼び出しに答えてエラーECICが出されます。CICプロトコルが不可能化されると、コントロールが戻された後でデバイス呼び出しがなされた場合ECICが直ちに返ってきます。

Bus Timing [バス・タイミング (ボード特性のみ)]

このフィールドはボードのソースとハンドシェイクを遅らせる時間T1を規定します。データがバスに置かれた後、この遅延により書き込みあるいはコマンドオペレーション中にボードがDAVをアサートする時間の最小限度が決まります。システム内のGPIBケーブルの全長が15メートル以内なら、350 nsecが適当な値です。

そのほか幾つかの因子がT1遅れ時間の決定に関与する可能性があります。(もっともこれは実際にはほとんどありえませんが。) これらの因子についてのより詳しい情報を得るには、IEEE規格488.2-1987, 5.2節をご覧ください。

このオプションのデフォルト値は500 nsec です。

Parallel Poll Duration [パラレルポール時間(ボード特性のみ)]

このフィールドでは、パラレルポーリングの進行中にドライバが待つ時間の長さを規定しています。普通のバス構成（コントローラとデバイスが同じバスに接続されている場合）では、デフォルトの時間が正しいのですが、もしバス延長器(エキステンダ）を使っている場合に透過性モードでパラレルポーリングを行う時は、ポーリング時間を10 μsecに延長してください。この延長で、バスエキステンダがアプリケーションプログラムに対して透過性で動作できるようになります。

Enable Repeat Addressing [繰り返しアドレシングの可能化(デバイス特性のみ)]

通常デバイスへのアドレシングは個々の書き込みあるいは読取作業ごとに行われます。このフィールドでnoを選択しますと、ある作業が今行われたばかりで、それをもう一度同じデバイスで繰り返す場合でも、同じアドレシングを繰り返しません。この設定は幾つかのGPIB作業を行う場合に多少の時間の節約になります。

Use This Board [このボードの使用(ボード特性のみ)]

システム内にボードが無いので、ドライバが選択されたベース（基底）アドレスにおいてボードにアクセスを試みては困ると言う場合には、このオプションでnoを選択します。このフィールドがnoに設定されていると、ドライバはボードのハードウエアにアクセスしようとしません。プログラムがボードにアクセスしようとした場合にはENEBが返されます。

Base I/O Address [ベース(基底)I/O アドレス(ボード特性のみ)]

このフィールドではGPIBボードのI/Oアドレスを規定します。これはGPIBボード自身のベースI/Oアドレスの設定値と同じでなければなりません。ベースI/Oアドレスレベルの設定の仕方については、GPIBインターフェースボードに付属してきたGetting Startedと題するマニュアル中で説明されています。マイクロチャネルやその他のシステム

では、リファレンスディスクをブートしなければI/Oアドレスを変えることは出来ません。この場合、このフィールドは読み出し専用(read-only)です。

DMA Channel [DMAチャネル (ボード特性のみ)]

このフィールドはGPIBインターフェースボードが使用するDMAチャネルを規定します。設定値はGPIBボード自身のDMAチャネル（アービトレーション(調停)レベル)設定と同じでなければなりません。DMAチャネル設定の仕方はGPIBボードに付属してきたGetting Startedと言うマニュアルの中で説明されています。マイクロチャネル及びその他幾つかのシステムでは、リファレンスディスクからブートしないとDMAチャネルを変えることは出来ません。しかしながら、IBCONFを使用すると、DMAの可能化や不可能化ができます。

Interrupt Level [インタラプト(割り込み)レベル (ボード特性のみ)]

このフィールドはGPIBインターフェースボードが使用するインタラプトラインを規定します。ここで設定するインタラプトラインのレベルはGPIBボード自身のそれと同じでなければなりません。インタラプトレベルの設定のやり方は、ボードに付属したGetting Startedと題するマニュアルに説明されています。マイクロチャネルやその他のシステムでは、リファレンスディスクをブートしなければインタラプトレベルを変えることは出来ません。しかしながら、IBCONFを使用して割り込みの可能化或いは不可能化を行うことはできます。

GPIB-PCII/IIA Mode Switch [GPIB-PCII/IIA モードスイッチ]

GPIB-PCIIとGPIB-PCIIA用インターフェースキット中のドライバは同一製品であって、どちらのボードとも動作することが出来ます。GPIB-PCII/IIA モードスイッチのフィールドでは、コンピュータシステムにインストールされたボードのタイプを選択することが出来ます。GPIB-PCIIとGPIB-PCIIA両方用のインターフェースをインストールすることも出来ます。

IBCONFの終了

全ての変更を終わった後は、ファンクションキーのF9か<Escape>キーを押すことによってIBCONFを終了させることが出来ます。いずれかのキーを押すと、「終了の前に変更を保存しますか？」と言う質問が表示されます。yesとタイプすると、変更はディスク上のファイルに書き込まれます。

IBCONFは、終了前に問題が起こる可能性がある状況が無いかどうか、例えば次のような点をチェックします。

- デバイスとアクセスボードとの間にGPIBアクセシングに関して問題が無いかどうか？
- GPIBボードがホストコンピュータの特定のアドレスに存在していないかどうか？
- タイムアウトが不可能化されているデバイスかボードが無いかどうか？

上記のような状況が存在した場合には、IBCONFはその旨を表示し、再入力をするか、あるいはそのままプログラムを終了するかどちらを選択するかを聞いてきます。

自動チェッキングを働かない様にするには、IBCONFをスタートする時に次のコマンドを入力します。

```
ibconf -e
```

また、IBCONFは、今構成したばかりでメモリにロードされているドライバがディスク上のドライバと互換性があれば、それを変更することも出来ます。もしもメモリにロードされたドライバが互換性を示しており、IBCONFが-fか-dオプションでスタートしたのであれば、「メモリにロードされたドライバを変更しますか？」と言う質問が表示されます。もしyで答えれば、メモリ中のドライバはすでに選択したパラメータを含むように変更されます。もしnを入力すれば、メモリ中のドライバは変更を受けません。上記の質問は、IBCONFがロードされたドライバを見つけることが出来ない場合、あるいはそれがドライバ・ファイルとはバージョンが違う場合、さ

らにドライバ・ファイルとは異なったインターフェースボードをサポートしている場合には表示されません。IBCONFが -d オプションでスタートした場合は、メモリ上にロードされたバージョンの構成に関する質問は表示されず、メモリ上のバージョンは、それが互換性を示す限り、自動的に構成されます。-f オプションを選んだ場合は、メモリにロードされたバージョンの構成は行われません。

変更を行いそれを保存をしたがメモリ上のドライバを変更しなかった場合は、コンピュータのリスタートを行わないと変更を生かす事が出来ません。また、ボードインタラプトレベルに変更を加えた場合、あるいはデバイス名をLPT1, LPT2, LPT3, COM1,あるいはCOM2に変えた場合はIBCONFはメモリ上のバージョンの構成を実行しません。

IBCONFのバッチモード

IBCONFのバッチモードによれば、上記の方法とは別の方法でNI-488.2ドライバの構成変更を行うことが出来ます。

注)　バッチモードでもメモリにロードされたドライバの変更を行うことが出来ます。この章の前の方の「*IBCONF*の終了」の節を参照してください。

バッチモードの場合の構成の情報は、前に作成した構成ファイルに含まれています。この構成ファイルを使用するには下記のコマンドをタイプします。

```
ibconf -b filename      <Enter>
```

上のコマンドで、*filename* は構成ファイルの名であって、たとえば sample.cfg です。-b と *filename* の間には少なくとも１つのスペースが入らなければなりません。

以下に構成ファイルの例を示し、一つ一つの項を順を追って説明します。

例 sample.cfg 構成ファイルの例

第1 デバイスの名をplotterに変える。

第2 デバイスの接続を切り離す。

第3 デバイスをボード1に接続する。

ボード0の構成を変える (gpib0)。

- 1次アドレスを2に変える(pad 2)
- タイムアウトをT30s (tmo 30sec) に設定
- EOSバイトをhex1E に設定 (eos 0x1E)
- システムコントローラ能力をNOにする (sc no)
- EOSでの比較のタイプを8-bitにする (bin 8-bit)
- GPIBバスタイミングを350 nsecに設定 (tmng 350nsec)

```
find device1 name plotter
find device2 disconnect
find device3 connect board1
find board0 pad 2 tmo 30sec eos 0x1E sc no
            bin 8-bit tmng 350nsec
```

構成ファイルは1対の項目からなる自由形式のテキスト・ファイルです。おのおのの項目はもう一つの項目と少なくとも1スペース、あるいは1ライン文字—で隔てられていなければなりません。それ

以外には何のフォーマット形式も必要ありません。1対の項目のうち第1のものは簡略記号であって、ボードがデバイスの特性を代表するかボード／デバイスのマップ構成機能（例えば名称変更、接続、接続の切り離し)でこれから構成されるものを指します。第2の項目は第1の項目（簡略記号）が設定されるべき値を表わします。

ボードなりデバイスを構成する前に、ボードやデバイスは find name# 対として見い出されなければなりません。ここで、name とは board か device で、#は構成しようとするGPIBボードかデバイスのインデックスです。(sample.cfg の例を参照してください。)

IBCONFがバッチ・モードの場合、それは1対の項目のうち第1の項目のシンタックスと第2の項目の値の範囲をチェックし、エラーを見い出した場合はそれを報告します。値の範囲に間違いがあった場合は正しい範囲を表示します。エラーが存在する状態では、ドライバの構成は実行されません。

表2－2には全ての有効な項目対が示されています。10進数あるいは16進数のいずれによっても入力できます。但し、16進数入力は前に0xを付け加える必要があります。(例えば、10進数の64に相応する16進数は0x40として入力します。)

表2－2 バッチモードのコマンド対の表

第1項目		第2項目	
簡略記号	説明	値	注
find	ボードあるいは デバイスを見い出す	board# or device#	
pad	1次GPIBアドレス	数値	1
sad	2次GPIBアドレス	数値	4
tmo	タイムアウト設定	簡略記号	2
xeos	書き込みにおいてEOS でEOIをセット	yes or no	

(次ページに続く)

表２－２ バッチ・モードのコマンド対の表 (次ページに続く)

第１項目		第２項目	
簡略記号	説明	値	注
bin	EOSでの比較のタイプ	7-bit or 8-bit	
eot	書き込みにおいて最後のバイトでEOIをセット	yes or no	
sc	システムコントローラ（ボードのみ）	yes or no	
sre	SCのときRENをアサート	yes or no	
spoll	自動シリアルポーリングを可能化	yes or no	
tmng	タイミング（ボードのみ）	2usec, 500nsec, 350nsec	
cic_prot	CICプロトコル（ボードのみ）	yes or no	
int	インタラプト設定（ボードにみ）	数値	4
port	ベースI/Oアドレス(ボードのみ)	数値	1
dma	DMAチャネル（ボードのみ）	数値	4
raddr	繰り返しアドレシング	yes or no	
name	デバイスの名称変更	デバイス名	3
connect	デバイスをボードに接続(デバイスのみ)	board#	
disconnect	デバイスのボードとの接続を切り離し（デバイスのみ）	値なし	
type	現在のボードをPC2あるいはPC2Aモードにスイッチ	PCII or PCIIA	5

(次ページに続く)

表2－2 バッチ・モードのコマンド対の表 (次ページに続く)

第1項目		第2項目	
簡略記号	説明	値	注
pplength	パラレルポーリング時間(ボードのみ)	簡略記号	6
useboard	このインターフェースボードの使用（ボードのみ）	yes or no	
spoltmo	シリアルポーリングタイムアウト(デバイスのみ)	簡略記号	2

表2－2の注

1. 正しい値を得るためには第5章の関数の説明の項を参照のこと。

2. 使用し得るタイムアウト用簡略記号は次の通り。

```
NONE,     1msec,   1sec,
10usec,   3msec,   3sec,
30usec,   10msec,  10sec,
100usec,  30msec,  30sec,
300usec,  100msec, 100sec
          300msec,
```

3. DOS名として有効で8文字以内の名。拡張子(.xxx)の使用は出来ない。

4. 有効な2次アドレス、パラレルポール可能化、DMAチャネル、あるいはインタラプトレベルのいずれかでなければ、noneと言う言葉か不可能化が必要。正しい値を得るためは第5章の関数の説明の項を参照のこと。

5. GPIB-PCII/IIAドライバにおいてのみ有効。

6. パラレルポーリング時間を設定するには、Note 2にある簡略記号の何れを使っても結構です（ただしNONEを除く）。デフォルト

のパラレルポーリング時間を設定するには、簡略記号 DEFAULT を使用してください。

ボードとデバイス特性のダイナミック再構成

アプリケーションプログラムを実行中種々の関数を呼び出して構成値をダイナミックに変化させることが出来ます。これらの関数を表2－3に示します。

表2－3 デフォルト特性を変える諸関数

特性	ダイナミックに変えるための関数名
1次GPIBアドレス	ibpad
2次GPIBアドレス	ibsad
EOSバイト	ibeos
7ビットあるいは8ビットにより EOSで行われる比較	ibeos
書き込み中にEOSでEOIをセット	ibeos
EOSで読取を停止	ibeos
書き込みの最後のバイトでEOIをセット	ibeot
ボードの割り当てを変更	ibbna
DMAを可能化あるいは不可能化	ibdma
時間制限を変更あるいは不可能化	ibtmo
システムコントロールの要求／解除	ibrsc

関数 ibconfig により上記及び他のほとんど全てのパラメータを実行時間中に、ダイナミックに設定する。この関数の説明については第5章を参照のこと。

NI-488.2ソフトウエアを使用するには

NI-488.2ソフトウエアは、高速で動作するドライバと幾つかのユティーリティプログラムから成り立っており、アプリケーションプログラムを開発したりデバッグしたりする助けになります。NI-488.2ドライバは次の3つの方法のうちのどれかでアクセスすることが出来ます。3つの方法は、NI-488の諸関数の直接使用による方法、NI-488.2のルーチンによる方法、及び、HP(ヒューレット・パッカード) スタイルのより低い性能のキャラクタI/Oドライバによる方法です。

NI-488の諸関数とNI-488.2のルーチン

NI-488.2ドライバはサブルーチンにより構造化されたデバイス・ドライバです。ナショナルインスツルメンツでは、ソフトウエアという用語をオペレーティングシステムの一部としてインストールされたドライバを指す場合に限って使用しています。NI-488.2ドライバは、他の入手可能な如何なるデバイス・ドライバより高速であり、バッファつきDMA転送を容易にこなします。また、現今のプログラム言語の使用者にとってなじみの深い構造的、階層的なプログラミングスタイルを使用しています。NI-488の諸関数とNI-488.2の種々のルーチンはこのマニュアルの第3章、第4章、及び第5章において説明されています。このドライバをアプリケーションプログラムにリンクするにはNI-488.2あるいはNI-488の言語インターフエースが必要です。ディストリビューション・ディスクには、BASICA, Microsoft QuickBASIC, Microsoft BASIC,および Microsoft Cとの言語インターフェースが含まれています。

以下に示すのは、QuickBASICを使ったハイレベルのNI-488の関数の1例で、バイトのアレーをあるデバイスに書き込む場合です。

```
data$ = "F0R2S2"
CALL ibwrt (scope%, data$)
```

ユニバーサル言語インターフェース (Universal Language Interface)

ユニバーサル・ランゲジ・インターフェース(ULI)は、Universal Language Interface Using HP-Style Calls (HPスタイルの呼び出しを使用するユニバーサル・ランゲジ・インターフェース) (パーツ番号320135-90) と題するマニュアルの中で説明されています。これはキャラクタI/OドライバによりNI-488.2ドライバにアクセスします。一方ULIへのアクセスは、ほとんどどの言語でもスプレッドシートでも使うような標準的I/Oコマンドで HPスタイルのコマンドストリングを引き渡すことにより達成出来ます。キャラクタI/Oドライバの常として、ULIはNI-488関数やNI-488.2ルーチンより遥かに遅く進行します。このドライバは多くのプログラミング言語から呼び出すことが出来ますが、それはHPのコマンドを使いなれたBASIC言語のプログラマーが用いる場合に最も適しています。ULIの使用には言語インターフェースは必要ではありません。

以下に示すのは、QuickBASICを使ったULIの機能の1例で、1つのコマンドをあるデバイスに書き込むためのものです。

```
PRINT #1, "OUTPUT 1;F0R2S2"
```

インタラクティブ・コントロール・プログラム (IBIC)

自分のGPIBシステムの使い方を覚える良い方法の一つとして、インターフェース・バス・インタラクティブ・コントロール(IBIC)と言うプログラムを使ってみることが上げられます。このプログラムについては、このマニュアルの第6章に説明があります。IBICを使用すると、アプリケーションプログラムを使わないで、キーボードによる計測器との対話の形式で、計測器をプログラムすることが出来ます。また、計測器とNI-488.2ドライバがどのようにして動作するかすぐに理解できるようになります。このプログラムは、アプリケーションプログラムの中でグローバル変数として戻されるステータス情報とおなじものを直ちに返してきます。

IBICを走らせながら、第6章中の各関数の説明を注意深く学び、一つ一つの関数の目的を完全に理解するようにしてください。オンラインのヘルプも利用してください。

アプリケーションズモニター

アプリケーションズモニターはコマンド・シーケンスの自動的なエラー検知とデバッグのため使用されるDOSメモリレジデントプログラムです。アプリケーションズモニターはこのマニュアルの第7章で説明されています。このプログラムでは、如何なるGPIBコマンドの後でも、あるいはエラーの条件があるだけでも実行を中止し、関数の引き数、データバッファ、戻された値、およびグローバル変数を調べることが出来ます。また、アプリケーションズモニターは最高255に及ぶそれまでに出されたコマンドを保存しますので、コマンドのシーケンスを辿って調べることが出来ます。このプログラムはアプリケーションプログラムの明白なエラーチェッキングコードの代わりの役割を果たします。

アプリケーションプログラム

自分で一つアプリケーションプログラムを書こうと決心された場合は、言語の適正な参考書、マニュアル（あるいは本書の第4章と第5章）を参照して諸関数の正しいシンタクスを心得ておくことが大切です。もし自分で書いたプログラムに問題があれば、アプリケーションズモニターをインストールしてください。また、アプリケーションプログラムで使っているコマンドのシーケンスをIBICによってテストしてください。

第3章
NI-488.2 ソフトウエアを理解する

この章では、まずNI-488.2ルーチンとNI-488関数を紹介し、両者に共通した特徴を説明します。さらにユーザーの皆様がアプリケーションプログラムを書こうとする場合に知っておくべき一般的情報、及び各言語のプログラミングの特殊情報を説明します。

- NI-488.2 ルーチンはANSI/IEEE-488.2-1987規格で定義されているコントローラシーケンスとプロトコルに直接付属するものです。これらは単一のデバイスのアドレスも受け付けますし、あるいは種々の関数が多数の計器に容易にアドレスできるように一連のデバイスアドレスを1つの入力パラメータとして受け付けることもします。これらのルーチンによって488.2の利点をすべて活用することが出来ます。

- NI-488 の関数はもう何年も使用されており、MS-DOS GPIBアプリケーションでは、業界の事実上の標準規格となっています。これらの関数には、ハイ・レベルのデバイス関数とロー・レベルのボード関数とがあります。

この章では、以上の他グローバル変数、エラー・コード、及び読取・書込の停止の様なプログラミングの問題、更にNI-488.2 ルーチンとNI-488関数の両方に共通したBASICA/QuickBASIC/BASIC/C によるプログラミングの書き方が説明されています。

NI-488.2 ルーチンとは

IEEE-488.2-1987規格の利点を利用できるように、NI-488.2の新しいルーチンの1群がNI-488.2のMS-DOSソフトウエアに付け加えられています。NI-488.2ルーチンはこのマニュアルの第4章で説明されています。これらのルーチンはIEEE-488.2-1987で定義されているコントローラシーケンスとプロトコルと完全な両立性を示します。

IEEE-488.2においては、データ・フォーマット、ステータス報告、コントローラの諸資格とコマンド、更に488.2タイプの計器のすべてが対応しなければならない一般コマンドのセット、が定義されており、それらがシステムの互換性と構成性をより強力なものにしています。今後生まれてくる新しいテスト用システムは皆この規格に沿って開発されるものと考えられます。また、488.2はthe Standard Commands for Programmable Instrumentation (SCPI)の基礎となっており、SCPIにしたがった計器は当然488.2規格と互換な筈です。NI-488.2のルーチンは488.2規格の持つこの様なシステムプログラミングに関する利点を活用するように開発されています。

NI-488.2 のルーチンのシンタックスは488.2規格で使用される命名規約に類似しています。これらのルーチンは特にコントローラと計器を含む完全な488.2システムの使用の際には488.2規格の利点の100パーセント利用を可能にします。これらのルーチンの中にはバス上の全てのリスナを見出し、接続された計器を構成し、サービス要求中のデバイスを見出し、SRQラインの状態を決定し、SRQがアサートされるのを待って多数のデバイスにアドレスするものがあります。IEEE-488.2のアプリケーション開発を計画されるのでしたらNI-488.2ルーチンの御使用が最善です。

タイムアウト値を構成するとか、全バス制御線をモニターするとか、幾つかのプログラミングの実際は488.2規格中に具体的には述べられていません。この様な場合には、伝統的なNI-488ボード関数をNI-488.2のルーチンと組み合わせて使用することができます。必要なNI-488のボード関数はこのマニュアルの第4章中の「NI-488呼出しとの関係」の項で説明されています。

NI-488 関数の概観

NI-488の関数はGPIBの制御作業の大部分をバックグラウンドにしているハイ・レベルの(と言うことはデバイスの）関数と完全なGPIBのコントロールを可能とするロー・レベルの（と言うことはボードの）関数とからなっています。たいていのアプリケーションプログラムの場合は数えるほどのハイ・レベル関数しか必要になりません。これらの関数は第5章で説明されています。

デバイス関数

デバイス関数は覚えやすく使い易いハイレベル関数です。これらの関数を使用すれば、GPIBのプロトコルとか、込み入ったバス制御の仕事を覚える必要がなくなります。バス制御の作業はデバイスからデータを読み取ったり書き込んだりする仕事やデバイスの現在の状況についてのポーリングなどを実行するために必要ですが、これらの関数は自動的にコマンドシーケンスを実行することによりこれらバス制御の作業を行います。デバイス関数は特定のデバイスにアクセスした後、そのデバイスに関したアドレシングとバス制御プロトコルを実行します。アクセスされたデバイスの記述子が関数の引き数の一つとなります。

ボード関数

デバイス関数に反し、ボード関数の方はローレベルで基礎的なGPIB作業を行います。ハイレベルの関数だけではアプリケーションの必要を何時でも満たすとは限りませんのでこの様な関数も必要となります。この様な場合にローレベルの関数は融通性にとんだ解決を与えています。

ボード関数はGPIBインターフェースに直接アクセスするので、バスのアドレシングと制御プロトコルが必要となります。アクセスされたボードの記述子はこれらの関数の引き数の一つとなります。

デバイス関数とボード関数に関する補足的説明

幾つかのローレベル関数が集まって一つのハイレベル関数の代替としての役割を果たすことができます。その様な場合のハイレベル関数とローレベル関数を比べてみるのもよいでしょう。シリアルポーリングを行う場合などがよい例です。ibrsp関数の説明のところで次のようなBASICで書かれたデバイス関数の例が出てきます。

```
CALL ibrsp (pltr%,status%)
```

上の例は、次のボード関数シーケンスと同格です。

```
cmd$ = "?!" + chr$(&H18) + "G"
CALL ibcmd (gpib0%,cmd$)
status$ = space$(1)
CALL ibrd (gpib0%,status$)
cmd$ = chr$(&H19) + "_"
CALL ibcmd (gpib0%,cmd$)
```

はじめに出てくるibcmd関数は第一プログラムラインで割り当てられたASCIIのコマンドストリングを送るために使われます。これらのコマンドは、Unlisten (?), 1次アドレス1を持つボードのリスンアドレス、1, Serial Poll Enable (chr$(&H18)), 及び1次アドレス7)を持つプロッタのトークアドレス(G)です。プロッタがそのステータスバイトを送るようにアドレスされ、ボードがそれを受け取るようにアドレスされたので、ibrd関数が呼び出されてバイトを読取り、それを変数ステータスで保存します。最後に、ibcmd関数が Serial Poll Disable (chr$(&H19))と Untalk (_)と言う2つのメッセージからなるコマンドを送出することによってポーリングを完了します。

上例によってハイレベルなデバイス関数の方が使い易いことが解ります。しかし、アプリケーションのシリアルポーリングルーチンがもっと複雑になってきて、幾つかのデバイスを連続してポーリングする一方他のサービシング作業も同時に行う必要が出てくると、ボード関数を駆使してしかるべきルーチンを作り出すことが必要になってきます。

ボードとデバイスを開く

NI-488の関数使用のための第1歩は使用することになるボードとデバイスのユニット記述子を手に入れることです。ユニット記述子udはibfind関数あるいは ibdev関数によって戻されたボードあるいはデバイス記述子の一般的な呼称です。デバイスのユニット記述子はある関数の最初の引き数であり、デバイス関数を規定します。ボードのユニット記述子はある関数の最初の引き数であり、ボードの関数を規定します。いくつかのNI-488関数はボード関数でもありまたデバイス関数でもありえます。

IBFIND (board or devname, dev)

ibfindはボードやデバイスと関連したユニット記述子を戻します。ibfindは他のいかなるNI-488関数より早く呼び出されなければなりません。ソフトウエアがインストールされると、一つ一つのデバイスの記述がソフトウエアからアクセス可能な内部参照表に付け加えられます。ibfind関数はソフトウエア内で規定されたgpib0, dev1あるいは scopeの様な記号名を用いるボードやデバイスのある場所を見つけます。これらの記号の名を見い出すには、IBCONFを走らせてみてください。

IBDEV (boardindex, pad, sad, tmo, eot, eos, ud)

ibdevは ibfindに代わる役目をする関数で、デバイスのユニット記述子を戻します。この関数は構成可能なパラメータの値が規定されていて、デバイスの記号化した名が重要な意味を持たない時に使用されます。

NI-488.2ルーチンと NI-488関数プログラミングの特徴

この節ではNI-488.2ルーチンの使用と NI-488関数の使用によるプログラミングの特徴を説明します。

複数ボード用ドライバ

このドライバは2つあるいはそれ以上の数のインターフェース・ボードを制御したり動作させることができます。第3－1図に示される複数ボードGPIBシステムデは、gpib0ボードが2つのデバイス(1つのオシロスコープと1つのディジタルボルトメータ)と接続されており、gpib1ボードが他の2つのデバイス(1つのプリンタと1つのプロッタ)と接続されています。このタイプのドライバは一般的に複数ボードドライバと呼ばれます。

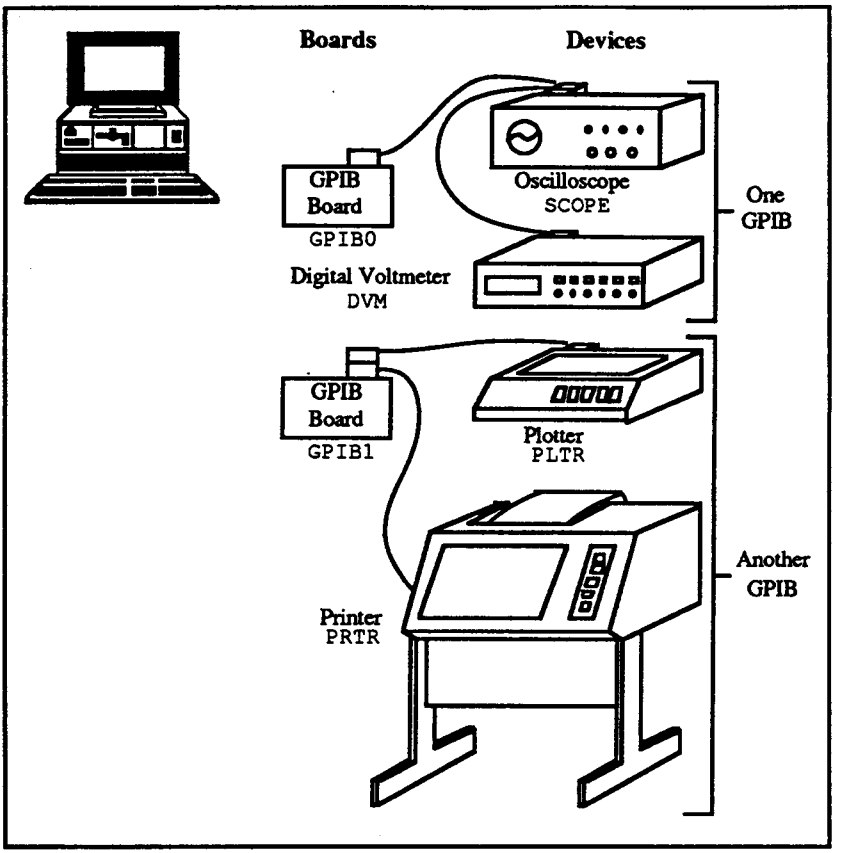


図3－1　複数ボードGPIBシステム

NI-488.2と計測器類を理解する

NI-488.2のルーチンとNI-488関数、及び計測器類のコマンドを覚えるには、キーボードによって対話形式で行うのが最善です。IBIC (Interface Bus Interactive Control－インターフェースバス対話制御)プログラムを用いるとNI-488.2ルーチンとNI-488関数をキーボードから入力することができます。計器類の制御は容易に行うことができ、アプリケーションプログラム無しでもステータスとエラー情報を得

ることができます。IBICは第6章において一歩一歩手順にしたがって説明されています。

プログラミングの一般的説明

以下のファシリティまたはオペレーションはプログラミングのあらゆるオプションに共通するものです。

- ステータス・ワード(ibsta)
- エラー・コード(iberr)
- カウント変数(ibcnt, ibcntl)
- 読取・書取停止

NI-488.2 MS-DOSソフトウエアの能力を十分に活用するには、上記の課題を十分に理解することが大切です。

次の数節にわたってステータス・ワード(ibsta), エラー変数(iberr), およびカウント変数(ibcnt, ibcntl)が説明されます。これらの変数は一つの関数が終わるごとに更新されて今アクセスされたばかりのデバイスやボードのステータスを示します。

ステータス・ワード— ibsta

全ての関数はGPIB およびGPIBインターフェースボードの状態についての情報を含むステータス・ワードを戻してきます。ステータス・ワードで報告された条件はテストしてその後のプロセスについての決定の資料とすることができます。ステータス・ワードは変数ibstaに含まれて戻されます。BASICA以外の如何なる言語を使用してもステータス・ワードを関数のかたちで(サブルーチンを使わず)呼び出すことができ、ステータス・ワードは関数の整数の形で戻されます。

ステータス・ワードは16ビットを含んでおり、そのうち14ビットは有意です。ビット値の(1)は該当する条件が成立していることを意

味し、ビット値の(0）は該当する条件が成立していないことを意味します。

表3－1には条件と、おのおのの条件をテストするビットの位置が示されています。

表3－1 ステータス・ワード(ibsta)レイアウト

簡略記号	ビット位置	16進数値	関数タイプ	内容
ERR	15	8000	dev, brd	GPIBエラー
TIMO	14	4000	dev, brd	時間制限超過
END	13	2000	dev, brd	ENDかEOSを検出
SRQI	12	1000	brd	SRQインタラプトを受信
RQS	11	800	dev	デバイスがサービス要求中
EVENT	10	400	brd	DTAS またはDCAS事象が生起。
SPOLL	9	200	brd	ボードはコントローラによりシリアルポーリングを受けている。
CMPL	8	100	dev, brd	I/O完了
LOK	7	80	brd	ロックアウト状態
REM	6	40	brd	リモート状態
CIC	5	20	brd	コントローラインチャージ
ATN	4	10	brd	ATNアサート中
TACS	3	8	brd	トーカ
LACS	2	4	brd	リスナ
DTAS	1	2	brd	デバイストリガ状態
DCAS	0	1	brd	デバイスクリア状態

BASICA/QuickBASIC/BASIC/Cの宣言ファイルはibstaとiberrステータスバイトの各ビットの簡略記号を規定しています。例えば、QuickBASICでプログラムを作成しているのであれば、次の2つの呼び出しは同じ効果を持ちます。

- `IF IBSTA% AND TACS THEN PRINT "TALK ADDRESS"`
- `IF IBSTA% AND &H0008 THEN PRINT "TALK ADDRESS"`

各ステータス・ワードとその条件の説明は以下の通りです。

ERR (dev, brd)　如何なる呼び出しにせよそれがエラーを起こした場合にERRがステータス・ワードにセットされます。その場合のエラーの特定はiberr変数を調べることによってなされます。ERRの後で、如何なる呼び出しにせよ、それがエラーを起こさなかった場合はERRはクリアされます。

注) 呼び出しの後では何時もエラーチェックをすることをお奨めします。エラーチェックは明確なそのための手段を使用してもよいし、アプリケーション・　モニターにより行っても結構です。アプリケーションプログラムの中で早期に起こり、気がつかなかったものはもっと後のインストラクションまで表面化しないこともあり得ます。この様な場合はエラーの発見はより難しくなります。

TIMO (dev, brd)　TIMOはタイムアウトが起こったか否かを示します。TIMOは、ibwaitの後に、もしもibwaitマスクパラメータのTIMOビットがセットされ、また待ちが時間制限を超えた場合にステータス・ワードにセットされます。TIMOはまたいかなる同期I/O関数の後でも(たとえばibrd, ibwrt とかibcmdなど) 呼び出し中にタイムアウトになった場合にセットされます。TIMOは上記以外の如何なる場合でもステータス・ワードの中でクリアされます。

END (dev, brd)

ENDはEOIラインからのENDメッセージを受け取ったこと、あるいは、ソフトウエアがEOSバイトの時読取関数を終了するように構成されている場合で読取関数のあとでEOSバイトを受け取ったことを示します。GPIBボードがibgts関数の結果としてシャドウハンドシエクを行っている間に、他の如何なる関数も、もしもENDかEOSメッセージがその呼び出しの前か最中に起これば、ENDビットセットを伴ったステータス・ワードを戻す可能性があります。ENDは如何なるI/O動作にせよそれが始まった時に、ステータス・ワードの中でクリアされます。

アプリケーションプログラムによっては読取動作を終了する際の入出力読取終了モードをはっきり知る必要があるものがあります。終了を示すモードとしてはEOI自体、EOS文字自体、またはEOIとEOS文字の併用によるものがあります。ibconfig関数を使用すればEOIを受信したときだけENDビットをセットするモードを可能化することができます。EOS文字だけで入出力動作が完了する場合にはENDビットはセットされません。その場合アプリケーションプログラムは受信したバッファの最終バイトをチェックしてそれがEOS文字であることを確認することになります。

SRQI (brd)

SRQIはあるデバイスがサービスを要求していることを示します。SRQIはGPIBボードがCICである時、GPIB SRQラインがアサートされており、自動シリアルポーリング資格が不可能化されていればステータス・ワードでセットされます。SRQIはGPIBボードがCICであることをやめれば、あるいはGPIB SRQラインがアサートをやめれば、クリアされます。

RQS (dev)

RQS はデバイス関数のステータス・ワードにおいてのみ現われ、デバイスがサービスを要求していることを示します。RQSはデバイスのシリアルポーリングステータスバイトにおいてhex 40のビット

がアサートされるとステータス・ワードでセットされます。ステータスバイトを得るシリアルポーリングはibrspにより起こされるか、または、自動シリアルポーリングが可能化されている場合は、自動的にソフトウエアによって起こされます。ibrspがRQSの原因となったシリアルポーリングステータスバイトを読み取ったときRQSはクリアされます。ibwaitはシリアルポーリングに答えるデバイスの場合のみ使用されるべきです。

EVENT (brd)　このビットはトーカ／リスナ用アプリケーションプログラム(これらはGPIBインターフェースがコントローラでないアプリケーションプログラムです）中で使用され、GPIBデバイスクリアのコマンドおよびGET (Group Execute Trigger) コマンドを監視します。トーカ／リスナ用アプリケーションプログラムはデバイスクリアとトリガ事象の生起の順序を記録する必要のある場合があります。ibstaに置ける通常のDCASビットやDTASビットはどちらの事象が先に起こったかを示すことが出来無いので不十分です。

EVENTビットはデフォルトでは不可能化されています。したがって、EVENTビットを使用したい場合は、ibconfig関数を使用してこのビットを可能化してください。このビットを可能化しますと、DCASとDTASは不可能化されます。その状態で一つの事象が起こった時は。EVENTビットがセットされ、進行中の入出力動作は全て中止されます。そのあとでアプリケーションプログラムがibeventを呼び出して、この事象が起こったかを知ることが出来ます。

SPOLL (brd)　このビットはトーカ／リスナアプリケーションプログラムにおいて、何時GPIBボードがコントローラによりシリアルポーリングを受けたかを知るために使用されます。SPOLLビットはデフォルトでは不可能化されていますので、使用の場合はibconfigにより可能化してください。このビッ

トを可能化すると、それはボードがポーリングを受けたときにセットします（但しこれはその時にボードがサービスを要求していた場合に限ります)。より詳細な情報については、第5章のibrsvの説明を見てください。SPOLLのステータスワードからのクリアは、もしそれが待機マスク中にセットされている場合は、ibwait呼び出しのあとで呼び出しの後必ず起こります。また、ibrsv呼び出しの後で直ちに起こります。

CMPL (dev, brd)　CMPLは現在未完のI/O作業の状態を示します。I/Oが完了したときは何時でもCMPLがセットされます。I/Oが現に進行中の場合はCMPLはクリアされています。

LOK (brd)　LOK はボードがロックアウトの状態にあるか否かを示します。LOKがセットされている間は、EnableLocalのルーチンと ibloc関数は該当するボードについては作用しません。LOKはGPIBボードあるいは他のコントローラが発したローカルロックアウト(ILO)メッセージを検出した場合は何時でもセットされます。LOKは、システムコントローラが Remote Enable (REN) GPIBラインのアサートをやめたときにクリアされます。

REM (brd)　REMはボードがリモート状態にあるか否かを示します。REMはRemote Enable (REN) GPIBラインがアサートされており、GPIBボードあるいは他のコントローラがGPIBのリスンアドレスを送出したことを検出した時は何時でもセットされます。REMはRENのアサートが止んだ時、あるいは、リスナとしてのGPIBボードがGPIBボードあるいは他のコントローラが送出したGo to Local (GTL)コマンドを検出した時、あるいはステータス・ワード中のLOKビットがクリアされているときにibloc関数が呼び出された場合にクリアされます。

CIC (brd) CICはGPIBボードがシステムコントローラであるか否かを示します。CICはGPIBボードがシステムコントローラである時、あるいは他のコントローラがコントロールをGPIBボードに渡した時SendIFCルーチンあるいはibsic関数が実行されれば何時でもセットされます。CICはシステムコントローラからのInterface Clear (IFC)をGPIBボードが検出した時、あるいはGPIBボードが他のデバイスにコントロールを渡した時にクリアされます。

ATN (brd) ATNはGPIB Attention (ATN)ラインの状態を示します。ATNはGPIB ATNラインがアサートされれば何時でもセットされ、ATNラインのアサートが止められるとクリアされます。

TACS (brd) TACSはGPIBボードがトーカとしてアドレスされているか否かを示します。TACSはGPIBボード、あるいは他のコントローラが送ったトークアドレス(そして、可能化されていれば2次アドレス）をGPIBボード自身が検出した時は何時でもセットされます。TACSは、GPIBボードがUntalk (UNT)コマンドを検出した時、それ自身のリスンアドレスを検出した時、それ自身のトークアドレス以外のトークアドレスを検出した時、あるいはInterface Clear (IFC)を検出した時にクリアされます。

LACS (brd) LACSはGPIBボードがリスナとしてアドレスされているか否かを示します。LACSはGPIBボード、あるいは他のコントローラが送った自身のリスンアドレス（そして、可能化しているのであれば、2次アドレス）をGPIBボード自身が検出した時は何時でもセットされます。またLACSは、ibgts関数の結果としてGPIBボードがシャドウハンドシェイクをした時にもセットされます。LACSは、GPIBボードがUnlisten (UNL)コマンド、それ自身のトークアドレス、あるいはInterface Clear (IFC)を検出した時、あるいはibgtsがシャドウハンドシェイクなしで呼び出された時にクリアされます。

DTAS (brd)　DTAS はGPIBボードがデバイストリガコマンドを検出したか否かを示します。DTASはGPIBボードがリスナとして他のコントローラが送出したGroup Execute Trigger (GET)コマンドを検出した時には何時でもセットされます。DTASはDTASビットが ibwaitマスクパラメータにセットされいる場合になんらかの呼び出しが ibwait呼び出しのすぐ後でなされるならば、ステータス・ワードの中でクリアされます。

DCAS (brd)　DCAS はGPIBボードがデバイス・クリアのコマンドを検出したか否かを示します。DCASは、GPIBボードが他のコントローラが送り出したDevice Clear (DCL)コマンドを検出した時は何時でも、あるいはリスナとしてのGPIBボードが他のコントローラが送り出したSelected Device Clear (SDC)コマンドを検出した時は何時でもセットされます。DCASはDCASビットが ibwaitのマスクパラメータの中にセットされていれば ibwaitのすぐ後の呼び出しによって、あるいは読出しあるいは書込み直後の呼び出しによってステータス・ワードの中でクリアされます。

関数呼び出しがENEBあるいは EDVR のようなエラーを戻してきたときはERRビット以外の全てのステータスビットはクリアされます。これはこれらのエラーコードがGPIBボードのステータスを得ることは不可能であることを示しているからです。

エラー変数—iberr

ERRビットがステータスワード中にセットされると、それはGPIBエラーが起きていることを意味します。その前になされたGPIB呼び出しがERRビットが中にセットされている ibsta値を戻してきた場合は、次の iberr解釈が成立します。これらのエラーコードは変数 iberrに含まれて戻されます。表3－2にはこれらのエラーコードが一括して示されています。

表3－2 GPIBエラーコード

示唆された簡略記号	10進数値	説明
EDVR	0	DOSエラー
ECIC	1	関数はGPIBボードがCICであることを必要としている。
ENOL	2	書込ハンドシェイクエラー(例えばリスナ無し)
EADR	3	GPIBボードが正しくアドレスされていない。
EARG	4	無効引き数を関数呼び出しに使用
ESAC	5	GPIBボードがシステムコントローラでなく、条件を満たしていない。
EABO	6	I/Oオペレーション中止(タイムアウト)
ENEB	7	GPIBボード存在せず。
EDMA	8	Windows 3のみ。仮想DMAデバイスエラー
EOIP	10	非同期I/O進行中
ECAP	11	オペレーション資格無し
EFSO	12	ファイルシステムエラー
EBUS	14	GPIBバスエラー
ESTB	15	ボールステータスバイトの待ち行列のあふれ
ESRQ	16	SRQがONの位置でつかえている。
ETAB	20	表に問題あり。

以下に各エラーの説明と、エラーが起こりうる条件の幾つかの記述がなされています。

EDVR (0)　　ibfindの呼び出しで送られたデバイスかボード名がソフトウエア中で構成されていなかった場合にEDVRが戻されてきます。この場合、変数

ibcntはDOSエラーコード2［デバイス見当たらず］を含みます。この問題を解決するにはibfindの引き数を有効なボードあるいはデバイス名で置き換えるか，IBCONFユティリティーを使ってソフトウエアを再構成し、名前を認識できるようにすることです。EDVRはまたいかなる関数呼び出しでも、無効なユニット記述子が渡された場合に戻されてきます。この場合には、変数ibcntはDOSエラーコード6［無効ハンドル］を含みます。この場合の解決法は、ibfindの呼び出しを確実に行い、成功のうちに完了しておくことです。また、ibonlを第2の引き数0で呼び出し、brdをofflineにした後では　デバイスあるいはボードを呼び出したり使用する前にはibfindが必要であることを覚えておいてください。

EDVRは、ソフトウエア(GPIB.COM)がインストールされていない場合にも戻されてきます。この場合の解決法にはルートディレクトリ中のCONFIG.SYSファイルをチェックして次のラインが含まれていることを確認してください。

```
DEVICE=dir\GPIB.COM
```

上のラインで *dir* はGPIB.COMファイルを含んでいるディレクトリです。(例えば *dir* は C:\AT-GPIB)

ECIC (1)

ECICはボードがCICでないときに次のボード関数あるいはルーチンのひとつが行われた時にECICが戻されます。

- NI-488.2ルーチンのどれか
- コマンドバイトをGPIBバスに加えるボード関数の何れか—ibcmd, ibcmda, ibln, ibrpp
- ibcac, ibgts
- GPIBに影響を与えるデバイス関数の何れか

GPIBが常にCICであることが要求される場合には、これらの呼び出しを行う前にかならずSendIFCかibsicを呼んでInterface Clearを送り、コマンドバイトTCT(hex 09, Take Control)を送らないようにすることで問題を解決できます。CICが2つあるいはそれ以上ある場合には、これらの呼び出しを行う前に　CICビットがいつでもステータスワードibstaの中に現われるようにすることが解決策になります。その通りでない場合には、ibwait (CIC)呼び出しを行って、ボードにコントロールが渡されるまでそれ以上のプロセスを延期させることができます。

ENOL (2)　ENOLは通常書込作業が行われようとしたときで、リスナがアドレスされていないときに起こります。デバイス書込の場合、このエラーは　ソフトウエアにおいてこのデバイスのために構成したGPIBアドレスがバスに接続したデバイスのGPIBアドレスの何れともマッチしない場合、GPIBケーブルがデバイスに接続されていない場合、あるいはデバイスに電源が入っていない場合であることを意味します。この様な状況を正す方法としては、適正なデバイスをGPIBに従属させること、既に従属させたデバイスのアドレスを変更すること、ibpad (そして必要ならばibsadも）呼び出しで構成したアドレスとデバイスのスイッチ設定をマッチさせること、あるいはIBCONF構成ユーティリティを使ってソフトウエア中のデバイスに正しいGPIBアドレスを与えるようにすることが挙げられます。

ボード関数の中では、ボードibwrt関数が実行される前に、デバイスにアドレスするためにibcmdが一般に必要です。ibcmdでは、引き数のストリングの中に正しいリスンアドレスがあり、そのあとにUnlisten (hex 3F)コマンドが続いていないことが重要です。

GPIBボードがCICでない場合、コントローラが読取の呼び出しが継続中にATNをアサートしたときにも起こる可能性があります。解決策としては読取バイトカウントをコントローラが予期するカウントまで下げること、あるいは、問題をコントローラ側で解決することの何れかが挙げられます。

EADR (3)

GPIBボードがCICであって読取と書込関数の前に自身にアドレスしない 場合にEADRが起こります。このエラーはデバイス関数について起こることはまずありません。解決には、SendDataBytes あるいは RcvRespMsgに先立って正しいトークあるいはリスンアドレスをSendCmds, SendSetup, あるいは ReceiveSetupを使って間違い無く送っておくこと、あるいはibwrt か ibrdをibcmdに先立って送っておくことです。

また、GPIBのATNラインのアサートが解かれた後でシャドウハンドシェイクが要求されたときにibgts がEADRを戻してきます。この場合にはシャドウハンドシェイクは不可能なのでそれを知らせるためにエラーが戻されるわけです。ibcmd呼び出し直後を除いてibgtsの呼び出しはまず避けたほうがよいのです。(ibcmdはATNのアサートを起こします。)

EARG (4)

関数呼び出しに無効の引き数を使用するとEARGが起こります。次のような例があります。

0から 17 の範囲外の値でibtmoを呼び出した時第2パラメータの上位バイト内にセットされた無意味なビットによってibeosが呼ばれた場合。

ibpadかibsadが無効のアドレスで呼び出された場合

ibppcが無効のパラレルポーリング 構成で呼び出された場合

有効なデバイス記述子を持つボード関数 あるいは有効なボード記述子を持つデバイス関数。

注）記述子が無効であるとEDVRが戻されます。

ESAC (5)

GPIBボードがシステムコントローラの能力を持たないときにSendIFC, ibsic, EnableRemote, あるいは ibsreの呼び出しを行うとESACが起こります。この問題を解決するにはGPIBボードにシステムコントローラの資格を与えます。これはibrscを呼び出すかIBCONFを使用して構成を行い、ソフトウエアの中でその資格を持たせます。

EABO (6)

EABOはI/Oがキャンセルされたことを意味します。通常このキャンセルはタイムアウトの結果起こります。また ibstopが呼ばれているとかCICがDevice Clearのメッセージを受け取った場合にもEABAが起こります。

タイムアウトによるエラーの場合、I/Oが進行中に起こるタイムアウトはibtmoによってタイムアウトの時間を延長すれば解決できます。しかし、実際にはI/Oがつかえる(リスナがハンドシェイクを継続しない場合とかトーカがトークを中止する場合）とかタイムアウトになった呼び出しのバイトカウントが相手のデバイスが予期したより多かったりする場合のほうがより頻繁に起こります。問題を防ぐためにはデータ転送の両側で予期すべきバイトカウントについて了解していることが大切です。できるなら、早期終了の際に役立つようトーカがENDメッセージを用いるようにするとよいでしょう。

ENEB (7)

ENEBが起こるのはGPIBボードが構成プログラムで指定したI/Oアドレスに存在しない場合です。これは実際にボードがシステムに差し込まれていない時、構成の場合に指定されたI/Oアドレスが実際のボードの設定と整合しない場合、あるいはシステムとベースI/Oアドレスとが合わない場合に起こります。構成の際に指定された値が実際のボード

の設定値とマッチしない場合は、ソフトウエアを再構成するか、ボードのスイッチの方を変えて構成値に合う様にします。

EDMA (8)　EDMAは、Windows 3の386 Enhanced モードにおいて、Windowsが問題を内蔵する仮想DMAデバイスを使用した場合のみに起こります。なぜ仮想DMAデバイスに問題がありうるかについては以下に示す2つの理由があります。

- Windowsのバージョン3をGPIBボード用に可能化したDMAとともに使用する場合、ナショナルインスツルメンツ製の置き換え仮想DMAデバイス`nivdmad.386`を使用していないとこのエラーが起こります。DMA使用については、*Using Your NI-488.2 Software with Microsoft Windows* (ナショナルインスツルメンツ部品番号320319-01)に置き換え仮想DMAデバイス使用の詳細な説明がありますので、参照してください。

- Windowsのバージョン3.1かバージョン3.0を`nivdmad.386`とともに使用していてEDMAが起こった場合は、このエラーは仮想DMAが未知の状態にあり、DMA転送が出来ないおそれがあるということを意味します。この様な場合は滅多に起こらないのですが、仮想DMAデバイスは、システム中のもう一つのデバイスのドライバが仮想DMAデバイスを正しく使用していないと、不安定になる場合があります。この問題を解決する只一つの方法はWindowsをリスタートすることです。こうすることによって仮想DMAデバイスは既知の状態に戻ります。

EOIP (10)　EOIPはある非同期I/O オペレーションが終了しないうちに他の呼び出しが行われたときに起こります。非同期I/Oの際には`ibstop`, `ibwait`, および`ibonl`の呼び出しのみが許されます。他の呼び出しを行うとEOIPが戻ってきます。

3種の非同期I/O呼び出し(ibcmda, ibrda, および ibwrta)はI/Oオペレーションを行っている間に他の関数(非GPIB関数)を動作させるアプリケーションを実行できるようにデザインされています。非同期I/O呼び出しが始まると、デバイスやアクセスボードの関係するGPIB呼び出しはI/Oが完了し、GPIBドライバとアプリケーションが再び同期化されるまでは行うことはできません。

再同期化は次の3関数のうちの一つを使用して行うことができます。

注） 再同期化はもどされたibstaがCMPLを含む場合のみ成功といわれます。

- ibwait - ドライバとアプリケーションが同期される。
- ibstop - 非同期I/Oがキャンセルされ、ドライバとアプリケーションが同期される。
- ibonl - 非同期I/Oがキャンセルされ、インターフェースがリセットされ、ドライバとアプリケーションが同期される。

これらの外に非同期I/O中に許されるGPIB呼び出しはibwait関数だけです(マスクは任意選択)。デバイスやアクセスボードの関係する他の全てのGPIB呼び出しはEOIPエラーを戻すことになります。

ECAP (11) ECAPはソフトウエア内である特定の資格が不可能化されている時にその資格を利用するための呼び出しがなされた場合に起こります。

EFSO (12) ibrdf あるいはibwrtf呼び出しがファイルオペレーションの実行で問題があった場合にEFSOが起こります。このエラーは関数を開くこと、作成

すること、検索すること　、書き込むこと、あるいはアクセスしたファイルを閉じることができない場合を示しています。この状態における特定のDOSエラーコードはibcnt中に含まれています。

EBUS (14)　幾つかのデバイス関数が全てアドレシングと他のバス管理用コマンドバイトを送る場合、デバイスはこれらのコマンドバイトを構成プログラムあるいはibtmoにより規定された制限時間中に受け入れるように期待されています。これらのコマンドバイトを送る途中にタイムアウトになった時にEBUSが起こります。通常の動作条件の下では、どのGPIBデバイスが異常に遅いコマンドを受けているのかを見い出してそのデバイスとの問題を解決することが必要です。もしコマンドのハンドシェイキングが遅い方が望ましい状況が存在する場合は、構成プログラムかibtmo関数を使用してボードの時間制限を延長します。

ESTB (15)　ESTBはibrsp関数の使用中のみに起こります。ESTBは自動シリアルポーリングで受け取った1つあるいはそれ以上のシリアルポールステータスバイトが保存する場所が不足のため棄てられたことを意味しています。幾つかの古いステータスバイトが入手可能です。しかしながら、最も古いものはibrsp呼び出しで戻されます。もしアプリケーションがたった1つのステータスバイトが足りなくても困るという場合は、IBCONFを使って自動シリアルポーリングを不可能化してください。

ESRQ (16)　ESRQ は WaitSRQ ルーチンか ibwait関数の使用中のみに起こります。ESRQ はGPIB SRQラインがつかえているのでRQSを待つわけにいかないことを意味します。通常この状況はソフトウエアが気付いていないデバイスがSRQをアサートしている場合に起こります。ソフトウエアがこのデバイスに気付いていないので、デバイスはシリアルポーリングの対象にならず、SRQはいつまでも残っていることになります。その他の ESRQ の原因としてはGPIBバス・テスターあるいは類似の装置がSRQラインを強制的にアサートしている場合とSRQラインに関係したケーブルのトラブルの場合があります。ESRQ シグナルがあらわれることはたしかにGPIB上の問題ではありますが、これはGPIBの動作には影響を与えません。(もっとも、この条件が続いている間はRQSラインの信頼性がなくなります。)

ETAB (20)　ETAB はFindLstn と FindRQSルーチン及びibevent関数の使用中のみ起こります。ETABはこれらの関数が使用する表に問題があることを意味します。FindLstn の場合ではETABはエラー状態ではなく、その表には見い出されたあらゆるリスナのアドレスをいれるだけの余地が無いというアドバイス的メッセージにすぎません。FindRQSの場合には、ETABは表の中のデバイスでサービスを要求中のものは１つもないという意味のメッセージです。ibeventの場合は、事象の待ち行列のあふれ出しのため、最も近くに起こった事象が失われてしまったということです。これを防ぐためには、ibeventをしばしば使用して、待ち行列を空にしてください。

カウント変数— ibcnt と ibcntl

カウント変数は読取、書込、あるいはコマンド関数の一つ一つが終わるごとに実際に動作中に転送されたバイトの数によって更新されます。また、多数のNI-488.2ルーチンによっても更新されます。

ibcntは整数値(16ビット幅)で、ibcntlは長い整数値(32ビット幅)です。

読取・書込の停止

IEEE-488規格ではデバイスによる(データ) メッセージの最後のバイトを識別するのに2つの方法を規定しています。これら2つの方法はトーカが如何なる長さのメッセージでも、リスナがあらかじめ転送バイトの数を知らなくても、送ることができるようにします。これら2つの方法は次の通りです。

- ENDメッセージこの方法ではトーカは最後のデータバイトを転送すると同時にEOI (End Or Identify) をアサートします。すると、リスナはEOIについてきたデータメッセージを検出した時バイトの値にかかわらず読取を止めるようになっています。

- エンドオブストリングキャラクタ (EOS)　この方法ではトーカはデータストリングの終で特別のキャラクタを使います。IEEE-488.2では、このEOSキャラクタははっきりとニュー・ライン(NL, ASCII 10, hex 0A)キャラクタとして指定されています。リスナはこのキャラクタを検出するとデータ受取を停止するようにあらかじめデザインされています。7ビットのASCIIキャラクタか、8ビット全部の2進法バイトか何れかを使用できます。

NI-488.2ソフトウエアは停止パラメータの値にしたがって読取と書込を停止します。読取の場合は、STOPendが読取作業をENDメッセージを受けたところで中止します。停止パラメータがキャラクタの場合は、読取作業はそのキャラクタを受け取った時点で停止されます。書込の場合は、停止パラメータはNLend, DABend, あるいはNULLendの何れかです。NLendは最後のデータバイトが送出された後NLをENDと共に送ります。DABendは最後に送るデータバイトとともにENDを送ります。NULLendはデータバイトをENDメッセージ無しで送ります。

BASICA/QuickBASIC/BASIC/Cのプログラミングについての情報

次にBASICA, QuickBASIC, BASIC, 及びCでドライバ関数をプログラムする際の情報について論じます。

注）　この節で使用される*BASICA*という用語はIBMパーソナルコンピュータ用のAdvanced IBM Interpretive BASICをさします。*QuickBASIC*という用語は Microsoft QuickBASICを指します。*BASIC*という用語はMicrosoft Professional BASICを指します。用語*C*は Microsoft Cのことです。

ここで示すプログラミングの情報は、特に指定しない限り、BASICA, QuickBASIC, およびCのあらゆるバージョンに共通して使用できるものです。

BASICA の各ファイル

NI-488ソフトウエアのディストリビューションディスクはBASICAのプログラミングに関連するファイルとして次のものを含んでいます。

- DECL.BAS は初期化コードを含むファイルです。
- BIB.MはアプリケーションプログラムがドライバにアクセスするためのBASICA言語インターフェースです。
- DBSAMP.BASはデバイス呼び出し使用のサンプルプログラムです。
- BBSAMP.BASはボード呼び出し使用のサンプルプログラムです。
- BSAMP488.BASは488.2ルーチン使用のサンプルプログラムです。

QuickBASIC用各ファイル

NI-488ソフトウエアのディストリビューションディスクは QuickBASIC (バージョン4.0以降) のプログラミングに関連するファイルとして次のものを含んでいます。

- QBDECL.BASは QuickBASIC の初期化コードです。
- QBIB.OBJはQuickBASIC言語インターフェースです。
- DQBSAMP.BASはデバイス関数使用のサンプルプログラムです。
- BQBSAMP.BASはボード関数使用のサンプルプログラムです。
- QSAMP488.BASは488.2ルーチン使用のサンプルプログラムです。

BASIC用各ファイル

NI-488ソフトウエアのディストリビューションディスクはMicrosoft Professional BASIC (バージョン7.0) のプログラミングに関連するファイルとして次のものを含んでいます。

- MBDECL.BASはBASICの初期化コードです。
- MBIB.OBJは BASIC言語のインターフェースです。
- DMBSAMP.BASはデバイス関数使用のサンプルプログラムです。
- BMBSAMP.BASはボード関数使用のサンプルプログラムです。
- MSAMP488.BAS は488.2ルーチン使用のサンプルプログラムです。

C言語用各ファイル

NI-488ソフトウエアのディストリビューションディスクはMicrosoft Cのバージョン5.0以降のプログラミングに関連するファイルとして次のものを含んでいます。

- DECL.Hは役にたつ変数と定数宣言を含むファイルです。
- MCIB.OBJはアプリケーションプログラムがドライバにアクセスできるようにするインターフェースです。
- DCSAMP.Cはデバイス呼び出し使用のサンプルプログラムです。
- BCSAMP.Cはボード呼び出し使用のサンプルプログラムです。
- CSAMP488.Cは488.2関数使用のサンプルプログラムです。

BASICAによるプログラミングの準備

BASICAでは、アプリケーションプログラムの主体を走らせる前にBASICAの言語コードブロックを実行する必要があります。DECL.BASというファイルはこのコードブロックを含んでいます。

DECL.BASファイルをアプリケーションプログラムの始めにおいて下さい。この場合、ラインナンバーはBASICA MERGEコマンドを使って適当に調整してください。DECL.BASにはBASICA言語インターフェースファイルのBIB.Mをメモリにロードするコードが含まれています。BIB.Mは現に使用中のディレクトリの中に存在している必要があります。

コードブロック中で決められている関数と変数の名称は、もしもそれらがアプリケーションプログラム中で使用されている名とかち合う場合には、変更することができます。名称の置き換えはどこにも矛盾が無いように注意深く行う必要があります。この章で示されるいろいろの指示は名称の置き換えが必要でなく、始めの名前がそのまま使用される場合を想定しています。

この章の始めに述べられているように、GPIBのステータス、エラー、及びカウント情報は、それぞれ変数ibsta%, 変数iberr%, 及び変数 ibcnt%として戻されます。

BASICAでは、全ての関数の引き数は整数かストリング変数のどちらかです。これらの値は関数の呼び出しがなされる前に与えられなければなりません。DECL.BASの1行目のCLEARステートメントはBASICA言語インターフェースBIB.Mをロードした後の使用可能メモリの量を示す定数を含みます。大多数のユーザーにとってはこのコンスタントは正確ですが、メモリが非常に小さいシステムのユーザーにとってはこの定数が不正確な場合があり得ます。BASICAをスタートした後でフリーのメモリが60,000バイト以下であると報告された場合には定数をより小さな値に調整する必要があります。

QuickBASICによるプログラミングの準備

QBDECL.BASはアプリケーションプログラムの始めに含まれていなければなりません。QBDECL.BASはアプリケーションプログラムに使用されうるすべてのNI-488サブルーチンと関数を規定しています。

この章の始めに述べたように、GPIBステータス、エラー、及びカウント情報は、それぞれ変数のibsta%, iberr%, 及びibcnt%とibcnt1%として戻されます。

QBIB.OBJファイルは、QuickBASIC言語の NI-488.2 MS-DOSソフトウエアとのインターフェースです。コンパイルしたQuickBASICアプリケーションプログラムを QBIB.OBJとリンクしてください。リンクにはDOSのコマンドプロンプトからQuickBASICのLinkerをスタートすることができます。あるいはQuickBASICの環境を設定して、その環境の中からプログラムを走らせることもできます。

QuickBASICの環境の中でEXEファイルを作成する場合には、オブジェクトモジュールライブラリを創り出さなければなりません。QBIB.LIBというオブジェクトモジュールライブラリを創るためには、次のコマンドを入力してください。

```
LIB QBIB.LIB + QBIB.OBJ;
```

QuickBASIC環境の設定には、まず言語インターフェースによりQuickLibraryを作成します。例えば、QBIB.QLBというQuickLibraryを創るためには、次のコマンドを入力します。

```
LINK /Q QBIB.OBJ,,,BQLB40.LIB;
```

あるQuickLibraryを創るのに、他のメーカー製のライブラリをナショナルインスツルメンツの言語インターフェースと共に使用して創ってみようと思われる場合があるかもしれません。以下に示すのはMicrosoft Library QB.LIB を QBIB.OBJ と共に使って1つのQuickLibraryを創る例です。

```
LINK /Q QB.LIB + QBIB.OBJ,QBIB,,BQLB45.LIB;
```

QuickBASIC 4.0を使用している場合は、上のコマンドでBQLB45.LIBをBQLB40.LIBに置き換えてください。次に、このQuickLibraryをつかってQuickBASICを走らせるには、以下のように入力してください。

```
QB /L QBIB.QLB
```

MS-DOSからプログラムを走らせるには、Microsoft QuickBASICのマニュアルの中にあるプログラムのコンパイリングとリンキングに関連する指示にしたがってください。

BASICのプログラミングのための準備

アプリケーションプログラムはその始めの部分にMBDECL.BASを含んでいなければなりません。MBDECL.BASはアプリケーションプログラムの中で使用できる全てのNI-488のサブルーチンと関数を規定します。

この章の始めに述べたように、GPIBステータス、エラー、及びカウント情報は、それぞれ変数のibsta%, iberr%, 及びibcnt%とibcntl%として戻されます

MBIB.OBJファイルは、BASIC言語の NI-488.2 MS-DOSソフトウエアとのインターフェースです。コンパイルしたBASICアプリケーショ

ンプログラムを MBIB.OBJとリンクしてください。リンクにはDOSのコマンドプロンプトからBASICのLinkerをスタートすることができます。あるいはQBXの環境を設定して、その環境の中からプログラムを走らせることができます。

QBXの環境の中でEXEファイルを作成する場合には、オブジェクトモジュールライブラリを創り出さなければなりません。MBIB.LIBというオブジェクトモジュールライブラリを作るためには、次のコマンドを入力してください。

```
LIB MBIB.LIB + MBIB.OBJ;
```

QBX環境の設定には、まず言語インターフェースにより1つのLibraryを作成します。例えば、MBIB.LIBというLibraryを創るためには、次のコマンドを入力します。

```
LINK /Q MBIB.OBJ,,,QBXQLB.LIB;
```

あるLibraryを創るのに、他のメーカー製のライブラリをナショナルインスツルメンツの言語インターフェースと共に使用して創ってみようと思われる場合があるかもしれません。以下に示すのは Microsoft Library QBX.LIB を MBIB.OBJ と共に使って1つのQuickLibraryを創る例です。

```
LINK /Q QBX.LIB + MBIB.OBJ,MBIB,,QBXQLB.LIB;
```

このQuickLibraryを使ってBASICを走らせるには、次のコマンドを入力してください。

```
QBX /L MBIB.QLB
```

MS-DOSからプログラムを走らせるには、Microsoft Professional BASICのマニュアルの中にあるプログラムのコンパイリングとリンキングに関連する指示にしたがってください。

Cプログラミングのための準備

アプリケーションプログラムの始めの部分に次のCステートメントが含まれるようにしてください。

```
#include "decl.h"
```

MCIB.OBJファイルはNI-488.2 MS-DOS ソフトウエアとのMicrosoft C言語インターフェースです。Cで書かれ、コンパイルされたアプリケーションプログラムは、MCIB.OBJとリンクされ、実行できる形のファイルとなります。

"ON SRQ"(SRQ使用中)資格

GPIB SRQ信号がアサートされた時はいつでもNI-488.2 MS-DOSドライバの割込みを行うことができます。割込みが起きると、プログラムはユーザーに規定されたサービスルーチンを実行します。ルーチンはSRQの発生源を決定し、適当な動作を行います。GPIB SRQラインの活動はコンピュータのライトペンによりモニターされます。サービスルーチンが完了するとコントロールは実行文中の次に続くステートメントに渡されます。BASICA/QuickBASIC/BASIC およびCのコード例を以下に示します。

注） ON PEN (ペン使用中)が正しく機能するようにするためには、構成パラメータの Enable Auto Serial Polling を不可能化します (メニューでNOを選択します)。また、同じライトペン割込みを使うマウスが多いので、マウスの不可能化が必要になることも有り得ます。

BASICA/QuickBASIC/BASICの"ON SRQ" (SRQ使用中)資格

BASICA/QuickBASIC/BASICプログラムはGPIB SRQ信号により割込みがかけられることがあります。割込みが起こると、プログラムからユーザーが規定したサービスルーチンに移ることがあります。

NI-488.2 MS-DOSドライバはBASICA/QuickBASIC/BASICの "ON PEN"(ペン使用中) ステートメントを使ってSRQ割込みをインタセプトしてユーザーのプログラムで使用できるようにします。"ON PEN" の動作についての完全な情報を得るためにはIBMのBASIC Reference マニュアル、マイクロソフトのQuickBASICマニュアル、及びマイクロソフトのProfessional BASICマニュアルを開き、"ON PEN" Statementの項を読んでください。"ON PEN" Statementの項にあるライトペンについての情報はすべてGPIB SRQ信号の場合に当てはまります.

BASIC及びQuickBASICで "ON SRQ" 資格を実現するためのコード構造は次の通りです。

```
   10 ON PEN GOSUB XXXX
   20 REM Enable ON SRQ capability.
   30 PEN ON
         .
         .
         .
XXXX  REM ON SRQ Service Routine
            .
            .
            .
         RETURN
```

BASICA と QuickBASIC における特殊な "ON PEN" 機能は非BASIC系の言語では普通見られないものです。

Cの"ON SRQ" (SRQ使用中)資格

Cプログラムは GPIB SRQ信号によって割込をかけられることがあります。割込は何らかのGPIB関数の動作が完了したときに起こります。割込が起こると、プログラムはユーザーに規定されたルーチンに移ります。1つのGPIB関数が完了すると,特種関数の ibsrqが NI-488.2 MS-DOS ドライバをしてSRQの有無をチェックさせます。ibsrqはボード関数でユーザー規定のサービスルーチンのアドレスを渡します。このサービスルーチンはSRQがアサートがなされると実行されます。

Cにおいて"ON SRQ"資格を実現するためのコード構造は次の通りです。

```
void far srqroutine()
  {
          .
          .
          .
  }
main() {

   int board;

   board = 0;

   ibsrq(srqroutine);

          .
          .
          .

}
```

続く第4章及び第5章においては、それぞれNI-488.2ルーチンとNI-488関数を詳細に説明します。これらの章ではルーチンと関数を一つ一つ説明し、コードとプログラムの例を示します。

第4章
NI-488.2ソフトウエアの特性とルーチン

この章ではMS-DOSドライバにより使用出来るNI-488.2ルーチンの重要特性のうちで各言語に共通なものを説明し、次にこれらのルーチンの一つ一つについてプログラムの例を付して詳細に記述しています。ルーチンはアルファベット順に配列してあります

概説

先行したIEEE-488.1規格に比べて、IEEE-488.2-1987規格ではコントローラがGPIBを制御する仕組、この規格に合ったデバイスであれば理解出来る筈の標準メッセージ、デバイスエラーや他のステータス情報が報告される際の種々のメカニズム、および本規格に沿ったデバイスのうちの何れがバスに接続されているかを見出してそれらのデバイスを構成するために用いる種々なプロトコルが詳細にきちんと説明されています。

多くのナショナルインスツルメンツのGPIBインターフェースボードが最近のバージョンアップにより、より厳しくなったIEEE-488.2規格に100パーセント互換性を持つようになり、IEEE-488.2プロトコルに完全に合致するようになりました。また。ソフトウエアにはルーチンが加えられました。これらのルーチンを使用すれば、IEEE-488.2規格書に有るものと極めて類似し、規格中のコマンドとデータのシーケンスに正しくしたがったプログラミングインターフェースを得ることが出来ます。

ナショナルインスツルメンツのNI-488.2ルーチンを488.2規格にしたがったデバイスと共に使用すれば、計器の動きの予測が簡単になり、プログラミングの間違いを減少できます。更に、異なった会社が製作した計器でも、同じようなやり方でプログラムすることが出来るようになります。

プログラミングについての一般情報

NI-488.2ルーチンではIEEE-488.2規格のコントローラプロトコルと手続きを使用します。ルーチンの呼び出しシンタクスはIEEE-488.2規格文書中に挙げられている実用化例に極めて類似するように作成されています。

NI-488.2で使用出来るルーチンは以下に示す各種ルーチンから成っています。これらルーチンの果たす種々の機能は次の各グループに分けることが出来ます。

- 単純なデバイスの入出力

  - Send
  - Receive

- 複数デバイスの入出力

  - SendList

- 単純なデバイスの制御

  - Trigger
  - DevClear
  - ReadStatusByte
  - PPoll
  - PPollConfig
  - PPollUnconfig
  - PassControl

- 複数デバイスの制御

  - TriggerList
  - DevClearList
  - EnableRemote
  - EnableLocal
  - FindRQS
  - AllSpoll

- バス管理

  - ResetSys
  - SendIFC
  - FindLstn
  - TestSRQ
  - WaitSRQ
  - TestSys
  - SendLLO
  - SetRWLS
  - GenerateREQF
  - GenerateREQT
  - GotoMultAddr

- 低レベルの入出力

  - SendCmds
  - SendDataBytes
  - SendSetup
  - RcvRespMsg
  - ReceiveSetup

単純なデバイス入出力のルーチンは個々のGPIBデバイスについて読取・書込を行うことが出来ます。

複数デバイス入出力ルーチンは一つのメッセージ伝送を行うだけで同一メッセージを複数のリスナに読込むことが出来ます。

単純なデバイスコントロールルーチンは種々のバス管理用命令を個々のデバイスに送ります。

複数デバイスのコントロールルーチンは同一のメッセージによってバス管理用命令を複数のデバイスに送ります。

バス管理ルーチンはシステム全体における諸機能の実行、あるいはシステム全体の状態の報告を行います。

低レベルの入出力ルーチンは、特殊な状況において高レベルルーチンをより詳細な個々の命令に分けるときに使用します。

あらゆるルーチンにおいて、最初のパラメータはコンピュータにインストールされたGPIBインターフェースボードを選択するためのボード番号です。普通GPIBインターフェースボードは一つだけインストールされている場合が多く、その様な場合のボード番号は0です。したがって、通常NI-488.2のルーチンの最初の引き数は0です。

ルーチンが唯一つのデバイスに働きかける場合は、第2のパラメータはデバイスのGPIBアドレスを表わす1つの整数となります。通常一つのデバイスが1次アドレスのみを選択する場合には、デバイスの1次GPIBアドレスに当たる0から30の範囲内の単純な整数を引き渡すことになります。もっと特殊なケースでデバイスが1次アドレスと2次アドレスを持つことがありますが、その様な場合には、2つのアドレスを一つの整数で、つまり1次アドレスを下位のバイトで、2次アドレスを上位のバイトで表わします。QuickBASICやBASICの場合は、この整数は&H6103のような形をとります。ここでデバイスの1次アドレスは3で、2次アドレスは hex61 ということです。

注) C言語の場合にはヘッディングファイルのDECL.Hの MakeAddr(p, s)を使用して1次及び2次アドレスを一つの正しい形にすることが出来ます。

複数のデバイスを動かすルーチンの場合は、これらデバイスのアドレスを含む整数のアレーが第2のパラメータとなります。これらのアドレスの一つ一つは上記の単一のデバイスのルーチンの場合と同じようにして形成されます。但し、これらは一つの整数アレーのなかの連続する要素として置かれ、このアレーはそれの終了を示すため特別の値NOADDRを最後に伴います。

入出力ルーチンの場合は1つのバッファ引き数を含みます。言語によっては、この外に、1つのカウント引き数を含みます。カウント引き数がある場合は、カウント引き数は1つの整数、あるいは(その言語内で許される場合は) 1つの long 型整数、として示されます。

ルーチンによっては、おのおのの特殊な必要に応じるため、上記以外のパラメータが必要になります。

NI-488.2ルーチンとNI-488呼出しとの関係

NI-488.2ルーチンにはIEEE-488.2で規定されたコントローラの諸手続きとプロトコルが完全なセットとして揃っています。しかし、時として、IEEE-488.2規格外のより込み入ったやり方でGPIBをコントロールする必要が出てきます。この様な場合の例として、次のような状況があります。

- 488.2規格に合わないデバイスと通信したい場合
- 種々の低レベルのボード構成を変更したい場合
- バスを通常の方法と異なったやり方で管理したい場合

ナショナルインスツルメンツのNI-488ボード関数の旧バージョンはNI-488.2のルーチンのシーケンスと互換であり、シーケンスの何処に挿入して使用しても通用します。例えば、タイムアウト用の値を変更したい時、NI-488.2ルーチンのシーケンスの中からibtmoの呼び出しをかけることが出来ます。また、任意のGPIBのラインの状態を監視するためiblinesを呼び出すことが出来る等の例も挙げられます。NI-488.2ルーチンのシーケンス内からこれらの呼び出しをかける場合は、通常必要とするibfindの呼び出しは不要です。　唯、ボード番号をNI-488ボード関数の最初のパラメータとして使用してください。この様にすれば、シーケンス中のあらゆる呼び出しは、NI-488.2であるとNI-488であるとを問わず、最初のパラメータとして、同じボード番号(通常0）を有するようになります。この様にして、必要に応じてNI-488.2のプログラム内から呼び出しを行えば、その時の状況とかデバイスが規格に合致していないとか、特殊な場合の要求にも、確実かつ容易に応えることが出来ます。

タイムアウト

NI-488.2ルーチンのほとんど、特にコマンドシーケンスあるいはデータメッセージの転送に関したルーチンはNI-488の呼び出しのタイムアウトと同じタイムアウト機構で規制されています。ドライバでは、あらかじめ10秒のタイムアウトをデフォルト値として構成されています。したがって、全ての入出力は、10秒以内に終了しないとタイムアウトエラーにかかります。更に、この期間には、WaitSRQルーチンが、

"no SRQ" 表示が戻されるまで待機します。デフォルトのタイムアウト値は IBCONF ユーティリティにより変更することが出来ます。また、NI-488ボードの関数の ibtmo を使用して、プログラムによるタイムアウ期間の変更をすることも出来ます。ibtmo 関数の詳細については第5章を御覧ください。

きめられた入出力ならびにWait のタイムアウト期間に関係なく、それらよりはるかに短いタイムアウト期間をシリアルポーリングの応答に適用することが出来ます。短いタイムアウト期間はシリアルポーリングを行う際いつでも発現することになります。この場合デバイスは普通ポーリングには速やかに応答するので、ある応答しないデバイスのために長い入出力タイムアウトまで待つ必要が無くなります。

BASICA/QuickBASIC/BASIC/C の NI-488.2 ルーチン

表4－1、4－2、及び4－3は、おのおのBASICA, QuickBASIC, BASIC およびCについて、各NI-488.2ルーチンの呼び出しシンタクスを示し、簡単にルーチンを説明したものです。

表4－1 BASICA の NI-488.2ルーチン

呼び出しシンタクス	内容
AllSpoll (board%,addresslist%(0), resultlist%(0))	全てのデバイスのシリアルポーリング
DevClear (board%,address%)	単一デバイスをクリアする
DevClearList (board%, addresslist%(0))	複数デバイスをクリアする
EnableLocal (board%, addresslist%(0))	一つのデバイスの計器盤からの操作を可能化

(次ページに続く)

表4－1 BASICAのNI-488.2ルーチン(前ページよりの続き)

呼び出しシンタクス	内容
EnableRemote (board%, addresslist%(0))	デバイスの遠端[コンピュータのキーボード]よりのGPIBプログラミング可能化
FindLstn (board%,addresslist%(0), resultlist%(0),limit%)	全てのリスナを見出す
FindRQS (board%,addresslist%(0), result%)	どのデバイスがサービス要求中か決定
PassControl (board%,address%)	コントローラ資格を持つ他のデバイスにコントロールを移す
PPoll (board%,result%)	パラレルポーリングを行う
PPollConfig (board%,address%, dataline%,sense%)	パラレルポーリングのためにデバイスを構成する
PPollUnconfig (board%, addresslist%(0))	デバイスのパラレルポーリング用構成を解く
RcvRespMsg (board%,data$, termination%)	既にアドレスしたデバイスからデータバイトを読み取る
ReadStatusByte (board%, address%,result%)	単一のデバイスをシリアルポーリングしてそのステータスバイトを得る
Receive (board%,address%,data$, termination%)	一つのGPIBデバイスからデータバイトを読み取る

(次ページに続く)

表4－1 BASICA の NI-488.2ルーチン(前ページよりの続き)

呼び出しシンタクス	内容
ReceiveSetup (board%,address%)	ある特定のデバイスがデータバイトを送出出来るように、またGPIBボードがそれらデータを読み取ることが出来るように準備する
ResetSys (board%, addresslist%(0))	一つのGPIBシステムを3レベルに初期化する
Send (board%,address%,data$, eotmode%)	GPIBデータバイトを単一のデバイスに送る
SendCmds (board%,commands$)	GPIBコマンドバイトを送る
SendDataBytes (board%, data$,eotmode%)	データバイトを既にアドレスしたデバイスに送る
SendIFC (board%)	IFCでGPIBインターフェース機能をクリアする
SendList (board%,addresslist%(0), data$,eotmode%)	データバイトを複数のGPIBデバイスに送る
SendLLO (board%)	ローカルロックアウトのメッセージを全てのデバイスに送る
SendSetUp (board%, addresslist%(0))	特定のデバイスをデータバイトを受信するよう準備する
SetRWLS (board%,addresslist%)	特定のデバイスをリモートモードでロックアウト状態にする

(次ページに続く)

表4－1 BASICA のNI-488.2ルーチン(前ページよりの続き)

呼び出しシンタクス	内容
TestSRQ (board%,result%)	SRQラインの現在の状態を決定する
TestSys (board%,addresslist%, resultlist%(0))	デバイスが自己テストを行うようにする
Trigger (board%,address%)	単一のデバイスをトリガする
Triggerlist (board%, addresslist%(0))	複数のデバイスをトリガする
WaitSRQ (board%,result%)	デバイスがサービス要求(SRQ)をアサートするまで待機する

表4－2 QuickBASIC/BASICのNI-488.2 ルーチン

呼び出しシンタクス	内容
AllSpoll (board%,addresslist%(), resultlist%())	全てのデバイスのシリアルポーリングをする
DevClear (board%,address%)	単一のデバイスをクリアする
DevClearList (board%, addresslist%())	複数のデバイスをクリアする
EnableLocal (board%, addresslist%())	デバイスの計器盤からの操作を可能化
EnableRemote (board%, addresslist%())	デバイスの遠端[コンピュータキーボード] よりのプログラミングを可能化

(次ページに続く)

QuickBASIC/BASIC の NI-488.2 ルーチン (前ページより続く)

呼び出しシンタクス	内容
FindLstn (board%,addresslist%(), resultlist%(),limit%)	全てのリスナを見出す
FindRQS (board%,addresslist%(), result%)	どのデバイスがサービス要求中かを決定
PassControl (board%,address%)	他のコントローラの資格を持つデバイスにコントロールを移す
PPoll (board%,result%)	パラレルポーリングを行う
PPollConfig (board%,address%, dataline%,sense%)	パラレルポーリングのためにデバイスを構成
PPollUnconfig (board%, addresslist%())	デバイスのパラレルポーリングのための構成を解く
RcvRespMsg (board%,data$, termination%)	既にアドレスしたデバイスからバイトを読み取る
ReadStatusByte (board%, address%,result%)	単一デバイスのシリアルポーリングを行いそのステータスバイトを得る
Receive (board%,address%, data$,termination%)	一つのGPIBデバイスからデータバイトを読み取る
ReceiveSetup (board%,address%)	ある特定のデバイスがデータバイトを送出できるように、またGPIBボードがそれらデータバイトを読取ることが出来るように準備する
ResetSys (board%,addresslist%())	3レベルにおいてGPIBシステムを初期化する
Send (board%,address%,data$, eotmode%)	データバイトを単一のGPIBデバイスに送る

(次ページに続く)

QuickBASIC/BASIC の NI-488.2 ルーチン (前ページより続く)

呼び出しシンタクス	内容
SendCmds (board%,commands$)	GPIBコマンドバイトを送出する
SendDataBytes (board%,data$, eotmode%)	データバイトを既にアドレスしたデバイスに送る
SendIFC (board%)	IFCでGPIBインターフェース機能をクリアする
SendList (board%,addresslist%(), data$,eotmode%)	データバイトを複数のGPIBデバイスに送る
SendLLO (board%)	ローカルロックアウトメッセージを全てのデバイスに送る
SendSetUp (board%, addresslist%())	特定のデバイスがデータバイトを受信できるように準備する
SetRWLS (board%,addresslist%)	特定のデバイスをリモートモードにし、ロックアウト状態にする
TestSRQ (board%,result%)	SRQラインの現在の状態を決定する
TestSys (board%,addresslist%, resultlist%())	デバイスが自己テストを行うようにする
Trigger (board%,address%)	単一デバイスをトリガする
Triggerlist (board%, addresslist%())	複数のデバイスをトリガする
WaitSRQ (board%,result%)	サービス要求 (SRQ) がアサートされるまで待機する

表4－3 CのNI-488.2ルーチン

呼び出しシンタクス	内容
AllSpoll (board,addresslist, resultlist)	全てのデバイスをシリアルプーリングする
DevClear (board,address)	単一のデバイスをクリアする
DevClearList (board,addresslist)	複数のデバイスをクリアする
EnableLocal (board,addresslist)	デバイスの計器盤よりの操作を可能化する
EnableRemote (board,addresslist)	遠端[コンピュータキーボード] よりのGPIBプログラミングを可能化する
FindLstn (board,addresslist, resultlist,limit)	全てのリスナを見出す
FindRQS (board,addresslist, result)	どのデバイスがサービスを要求しているかを決定する
GenerateREQF (board, addr)	サービス要求をキャンセル
Generate REQT (board, addr)	サービスを要求
GotoMultAddr (board, type, addrfunc,spollfunc)	1次または2次アドレスの複数サポートを可能化
PassControl (board,address)	コントローラの資格を持つ他のデバイスにコントロールを移す
PPoll (board,result)	パラレルポーリングを行う
PPollConfig (board,address, dataline,sense)	パラレルポーリングが行えるようにあるデバイスを構成する

(次ページに続く)

表4－3 CのNI-488.2ルーチン(前ページより続く)

呼び出しシンタクス	内容
PPollUnconfig (board, addresslist)	デバイスのパラレルポーリング用の構成を解く
RcvRespMsg (board,data, termination)	既にアドレスしたデバイスからデータバイトを読み取る
ReadStatusByte (board,address, result)	ある単一デバイスをシリアスポーリングしてそのステータスバイトを得る
Receive (board,address,count, termination)	一つのGPIBデバイスからデータバイトを読み取る
ReceiveSetup (board,address)	ある特定のデバイスがデータバイトを送出できるように、またGPIBボードがそれらバイトを読み取れるように準備する
ResetSys (board,addresslist)	一つのGPIBシステムを3レベルに初期化する
Send (board,address,data, eotmode)	データバイトをある単一のGPIBデバイスに送る
SendCmds (board,commands,count)	GPIBコマンドバイトを送る
SendDataBytes (board,data, count,eotmode)	データバイトを既にアドレスしたデバイスに送る
SendIFC (board)	IFCでGPIBインターフェース機能をクリアする
SendList (board,addresslist, data,count,eotmode)	複数のGPIBデバイスにデータバイトを送る

(次ページに続く)

表4－3 CのNI-488.2ルーチン(前ページより続く)

呼び出しシンタクス	内容
SendLLO (board)	全てのデバイスにローカルロックアウトメッセージを送る
SendSetUp (board,addresslist)	特定のデバイスがデータバイトを受信できるように準備する
SetRWLS (board,addresslist)	特定のデバイスをリモートモードにし、ロックアウト状態にする
TestSRQ (board,result)	SRQラインの現在の状態を決定する
TestSys (board,addresslist, resultlist)	デバイスが自己テストを行うようにする
Trigger (board,address)	ある単一のデバイスをトリガする
Triggerlist (board,addresslist)	複数のデバイスをトリガする
WaitSRQ (board,result)	デバイスがサービス要求(SRQ)をアサートするまで待機する

NI-488.2の各ルーチンの説明

以下この章の終まで、一つ一つのNI-488.2ルーチンを実例と共に説明します。これらのルーチンは、参照しやすいようにアルファベット順に整理されています。

AllSpoll　AllSpoll

目的　全てのデバイスのシリアルポーリング

フォーマット

BASICA

```
CALL AllSpoll (board%, addresslist% (0), resultlist% (0))
```

QuickBASIC/BASIC

```
CALL AllSpoll (board%, addresslist% (), resultlist% ())
```

C

```
void AllSpoll (short board, unsigned short addresslist [],
               unsigned short resultlist [])
```

boardはボード番号を規定します。GPIBデバイスでアドレスアレーの中にアドレスが含まれているものはシリアルポーリングされ、それらの応答はresultlistアレーの中の該当するエレメントの中に記憶されます。パラメータのaddresslistは任意のサイズのアドレス整数のアレーであって、NOADDRの値によって終了させられます。

ある特記されたデバイスがポーリングに応答せずにタイムアウトになった場合は、iberrのエラーコードEABOが戻されます。その場合ibcntにタイムアウトになったデバイスのインデクスが含まれています。

AllSpollは一般的な適用性を持つルーチンで、GPIBデバイスの数に関わらずシリアルポーリングすることが出来ますが、唯1つのGPIBデバイスをポーリングする際はReadStatusByteルーチンを使用してください。

AllSpoll　(前ページより続く)　AllSpoll

例

ボード0に接続され、GPIBアドレスが8と9の2つのデバイスをシリアルポーリングする。

BASICA

```
70  DIM addresslist%(3)
71  DIM resultlist%(2)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100 CALL AllSpoll (board%, addresslist% (0),
                   resultlist%(0))
```

QuickBASIC/BASIC

```
70  DIM addresslist%(3)
71  DIM resultlist%(2)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100 CALL AllSpoll (board%, addresslist% (),
                   resultlist%())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
unsigned short resultlist[2];
AllSpoll (0, addresslist, resultlist);
```

DevClear　DevClear

目的　単一のデバイスをクリアする。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL DevClear (board%, address%)
```

C

```
void DevClear (short board, unsigned short address)
```

boardはボード番号を指定します。GPIBのSelected Device Clear (SDC) [選択したデバイスをクリア] メッセージは該当するデバイスのアドレスへ送られます。パラメータのaddressはその下位バイトの中にこれからクリアするデバイスの1次GPIBアドレスを含んでいます。この場合上位バイトの方は、2次アドレスが無い限りは0です。0でない場合は適当な2次アドレスが必要です。addressがNOADDRという定数を含む場合は Universal Device Clear [全てのデバイスをクリア] というメッセージがGPIB上の全てのデバイスに送られます。

DevClearルーチンは、ただ一つだけのGPIBデバイスをクリアするか、あるいは全てのデバイスをクリアする場合に用いられます。複数の特定のGPIBデバイスを単一のメッセージでクリアしたい場合は、DevClearListルーチンを使用してください。

例

ボード0に接続されたディジタル電圧計をクリアする。電圧計の1次GPIBアドレスは9であり、2次GPIBアドレスは97である。

BASICA/QuickBASIC/BASIC

```
80  board% = 0
90  address% = 9 + 256*97
100  CALL DevClear (board%, address%)
```

DevClear　　(前ページより続く)　　DevClear

C

```
DevClear (0, MakeAddr (9, 97));

/* In C, a macro has been defined in the header
 * file DECL.H, MakeAddr(p, s), which can be
 * used to pack the primary and secondary
 * addresses into the correct form.
 */
```

[Cではマクロはヘッダファイルの DECL.H において定義されている。MakeAddr(p,s) は1次と2次のアドレスを正しい形式にまとめるために使用することが出来る。]

DevClearList　DevClearList

目的　複数のデバイスをクリアする。

フォーマット

BASICA

```
CALL DevClearList (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL DevClearList (board%, addresslist% ())
```

C

```
void DevClearList (short board, unsigned short addresslist [])
```

boardはボード番号を指定します。このルーチンでGPIBデバイスでそのアドレスがアドレスアレーに含まれるものはクリアすることが出来ます。パラメータの addresslist は任意のサイズのアドレス整数のアレーであり、NOADDR値によって終了されます。

DevClearList ルーチンは一般的適用性を持つので、如何なる数のGPIBデバイスのクリアにも使えるわけですが、通常唯1つのGfPIBデバイスをクリアする際は、DevClear ルーチンを使用してください。

アレーにNOADDR値しか含まれていない場合、あるいはC言語においてNULL value [空値]が引き渡された場合は、汎用性を持つDevice Clear [DCL] メッセージが送られます。

DevClearList　(前ページより続く)　DevClearList

例

ボード0に接続された2つのGPIBデバイスをクリアする。2つのデバイスのGPIBアドレスは8と9である。

BASICA

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL DevClearList (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL DevClearList (0, addresslist% ())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
DevClearList (0, addresslist);
```

EnableLocal　EnableLocal

目的　デバイスの計器盤より行う操作の可能化

フォーマット

BASICA

```
CALL EnableLocal (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL EnableLocal (board%, addresslist% ())
```

C

```
void EnableLocal (short board, unsigned short addresslist [])
```

boardはボード番号を規定します。GPIBデバイスでaddresslistアレーにアドレスが含まれているものはこのルーチンでローカルモードにされます。この時、これらのデバイスはリスナとしてアドレスされ、GPIBコマンドのGo To Local [GTL] を受け取ります。パラメータのaddresslistは任意のサイズのアドレス整数のアレーであり、NOADDR値を受けたときに終了されます。

アレーがNOADDR値しか含まない場合、あるいはC言語で　NULL値がaddresslistとして引き渡された場合、Remote Enable (REN)のアサートが解除され、全てのGPIBデバイスは直ちにローカルモードに変わります。

EnableLocal　(前ページより続く)　EnableLocal

例

GPIBアドレスが8と9である2つのデバイスをローカルモードにする。

BASICA

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL EnableLocal (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL EnableLocal (0, addresslist% ())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
EnableLocal (0, addresslist);
```

EnableRemote　EnableRemote

目的　デバイスのリモートGPIBプログラミングを可能化する。

フォーマット

BASICA

```
CALL EnableRemote (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL EnableRemote (board%, addresslist% ())
```

C

```
void EnableRemote (short board, unsigned short addresslist [])
```

board はボード番号を規定します。GPIBデバイスでアドレスが addresslist アレーに含まれているものはこのルーチンでリモートモードにされます。この時Remote Enable (REN) がアサートされ、デバイスはリスナとしてアドレスされます。パラメータの addresslist は任意のサイズのアドレス整数のアレーで、NOADDR 値が出たときに終了されます。

アレーが NOADDR 値しか含まない場合、あるいはC言語において NULL 値が addresslist として引き渡された場合、Remote Enable (REN) がアサートされます。

EnableRemote　(前ページより続く)　EnableRemote

例

GPIBアドレスが8と9である2つのデバイスをリモートモードにする。

BASICA/QuickBASIC/BASIC

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL EnableRemote (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL EnableRemote (0, addresslist% ())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
EnableRemote (0, addresslist);
```

FindLstn　　FindLstn

目的　全てのリスナを見つける。

フォーマット

BASICA

```
CALL FindLstn (board%, addresslist% (0), resultlist% (0),
               limit%)
```

QuickBASIC/BASIC

```
CALL FindLstn (board%, addresslist% (), resultlist% (),
               limit%)
```

C

```
void FindLstn (short board, unsigned short addresslist [ ],
               unsigned short resultlist [ ],
               unsigned short limit)
```

boardはボード番号を規定します。addresslistは1次GPIBアドレスを含んでおり、NOADDR値が現われたときに終了されます。このルーチンでは、リスナを見つけるため、これらのアドレスが次々とテストされます。リスナが見つかりますと、そのアドレスがresultlistに記入されます。もしリスナがある特定の1次アドレスにおいて見い出されない場合は、その1次アドレスに関連する全ての2次アドレスがテストされ、見つかれば、それらのリスナがresultlistに記入されます。limit引き数は、resultlistアレーに記入するリスナのアドレスの数を規定します。もしバス上のリスナの数が規定された数より多い場合には、limit数が検出された後でリストされた数が削られ、iberrでETABが示されます。変数ibcntにはresultlistに記入されたアドレス数が含まれています。

FindLstn　(前ページより続く)　FindLstn

どの1次アドレスもリスナとして応答する複数の2次アドレスを持つと考えられるので、一般にresultlistアレーはaddresslistアレーより大きいものになる筈です。何れにせよ、resultlistは、limitが起りうる最大数であれば、アレーのあふれの制限は無いので、そのアレーはある限りのリスナにも対処することが出来ます。

ほとんど全てのGPIBデバイスはリスナに成りうるので、このルーチンは普通ある特定のアドレスに存在するデバイスを検出する目的で使われます。デバイスが見つかりますと、通常識別用メッセージを送って、見つかったデバイスが如何なるデバイスであるかを決定することが出来ます。

例

アドレス8、9及び10のデバイスのうちの何れがGPIB上に存在するかを決定する。

BASICA

```
70  DIM addresslist%(4)
`         Because there are three primary GPIB
`         addresses, in the worst case 93
`         separate GPIB devices could be detected
`         at all the secondary addresses.  In
`         this example, we are assuming that we
`         know that there are at most 5 devices
`         connected to the GPIB.
```

[1次GPIBアドレスが3つあるので、最悪の場合には93の別々のGPIBデバイスが全ての2次アドレスで検出される可能性がある。今の例では、最高で5つのデバイスしかGPIBに接続されていないことが解っている場合を想定している。]

```
75  DIM resultlist%(5)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
```

FindLstn　(前ページより続く)　FindLstn

```
92  addresslist% (2) = 10
93  addresslist% (3) = NOADDR
94  limit% = 5
100  CALL FindLstn (board%, addresslist% (0), resultlist
                    (0), limit%)
```

この呼び出しの後に得られたresultlist% ()には次の値が含まれていた。

```
resultlist% (0) 9
resultlist% (1) 10 + 96*256
resultlist% (2) 10 + 99*256
```

これらの結果によると3つのGPIBデバイスが検出された。その内の1つはアドレス9で見出された。このデバイスは2次アドレスを持っていない。1次アドレス8ではデバイスは1つも見出されなかった。アドレス10では、2次アドレスを持つデバイスが2つ見出された。この場合では、1次GPIBアドレス8、9と10しかテストされていないので、まだ外にもGPIBデバイスが他のアドレスで接続されている可能性がある。

QuickBASIC/BASIC

```
DIM addresslist%(4)
DIM resultlist%(5)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = 10
addresslist% (3) = NOADDR
limit% = 5
CALL FindLstn (0, addresslist% (), resultlist% (),
limit%)
```

C

```
unsigned short addresslist[4] = {8, 9, 10, NOADDR};
unsigned short resultlist[5];
FindLstn (0, addresslist, resultlist, 5);
```

FindRQS　FindRQS

目的　どのデバイスがサービスを要求しているかを決定する。

フォーマット

BASICA

```
CALL FindRQS (board%, addresslist% (0), result%)
```

QuickBASIC/BASIC

```
CALL FindRQS (board%, addresslist% (), result%)
```

C

```
void FindRQS (short board, unsigned short addresslist [],
              unsigned short *result)
```

boardはボード番号を規定します。addresslistには1次GPIBアドレスのリストを含んでおり、NOADDR値によって終了させられます。指定されたデバイスはaddresslistの初めから順にシリアルポーリングされます。シリアルポーリングはSRQをアサートしているデバイスが見つかるまで続けられ、デバイスが見つかると、そのステータスバイトが変数resultで戻されます。更に、addresslistにあるデバイスのアドレスのインデックスがグローバル変数ibcntで戻されます。

指定されたデバイスの何れもがサービスを要求していない場合には、エラーコードのETABがiberrで戻されます。この場合ibcntがNOADDR記入のインデックスを含んでいます。

デバイスがシリアルポーリングに応答中にタイムアウトになった場合はiberrでエラーコードのEABOが戻されます。この場合、ibcntでタイムアウトになったデバイスのインデックスが示されます。

FindRQS　(前ページより続く)　FindRQS

例

アドレス8、9及び10で、どのデバイスがサービスを要求しているかを決定する。

BASICA

```
70  DIM addresslist%(4)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = 10
93  addresslist% (3) = NOADDR
100  CALL FindRQS (board%, addresslist% (0), result%)
```

この呼び出しの後result%が&H40（シリアルポーリングの応答）を含んでおり、ibcntが2という値を含んでいたとすると、これはaddresslist$(2)にあるデバイスがリスト中のデバイスで最初にSRQアサート中であることを見出されたものであることを意味しています。

QuickBASIC/BASIC

```
DIM addresslist%(4)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = 10
addresslist% (3) = NOADDR
CALL FindRQS (0, addresslist% (), result%)
```

C

```
unsigned short addresslist[3] = {8, 9, 10, NOADDR};
unsigned short result;
FindRQS (0, addresslist, &result);
```

GenerateREQF　GenerateREQF

目的　GenerateREQTが生成したサービス要求をキャンセル。

フォーマット

C

```
void GenerateREQF (short board, unsigned short addr);
```

boardはボード番号を指定します。addrはもはやサービスを要求していないシミュレートされたデバイスの5ビットGPIBアドレスを指定します。ドライバはSRQ線のアサートを中止する時期がわかるように現在サービスを要求中のシミュレートされたデバイスの監視を続けます。

この関数はシミュレートしたデバイスのうちの一つがコントローラ・イン・チャージのサービスを受ける必要がなくなった時に使用します。addrパラメータはもはやサービスを必要としないシミュレートしたデバイスのアドレスです。ドライバは(もし他のシミュレートしたデバイスの何れもがサービスを要求していなければ) SRQ線のアサートを終了するので、そのシミュレートしたデバイスがポールされた時にRSVビットをセットしなくなります。

シミュレートしたデバイスがシリアルポールされた後、ドライバはポールされたデバイスに自動的にGenerateREQFを行います。シミュレートしたデバイスがまだポールを受けていないが、もはやサービスを必要としていない場合にこの関数を使用してください。

例:

この機能の完全な例は*GotoMultAddr*のところに出ています。

GenerateREQT　GenerateREQT

目的　GPIBコントローラ・イン・チャージよりのサービスを要求。

フォーマット

C

```
void GenerateREQT (short board, unsigned short addr);
```

boardはボード番号を指定します。addrはサービスを要求しているシミュレートしたデバイスの5ビットGPIBアドレスを指定します。ドライバはSRQ線のアサートを中止時期がわかるように現在サービスを要求中のシミュレートしたデバイスの監視を続けます。

この関数はシミュレートしたデバイスのうちの一つがコントローラ・イン・チャージのサービスを受ける必要がある時に使用します。addrパラメータはもはやサービスを必要とするシミュレートしたデバイスのアドレスです。ドライバはSRQ線をアサートします。コントローラは、SRQがアサートされていると認めると、デバイスのシリアルポーリングを行います。つぎにspollfuncファンクションが正しいシリアルポーリング応答バイトを返します。ドライバは、応答バイトをコントローラに送る前にRSV(Request Service) ビットを1にセットします。

例:

この関数の完全な例は*GotoMultAddr*のところに出ています。

GotoMultAddr　GotoMultAddr

目的　1次或いは2次アドレスの複数サポートを可能化する。

フォーマット

C

```
void GotoMultAddr (short board,
                   unsigned short type,
                   unsigned short (_far _loadds
                       *addrfunc) (),
                   unsigned short (_far _loadds
                       *spollfunc) () )
```

boardはボード番号を指定します。typeは定数MultAddrPrimaryかMultAddrSecondary(言語インターフェースincludeファイルで定義されている）でなければなりません。このパラメータを使って2つの複数アドレスモードの何れかを選択してください。プログラムは複数1次GPIBアドレスか、同じ1次アドレスの複数2次GPIBアドレスかの何れかをシミュレートします。両方ともシミュレートすることは出来ません。

addrfuncはユーザーによって与えられたアドレス選択ファンクションのアドレスでなければなりません。ドライバは1次または2次（これはtypeによって変わります）GPIBアドレスがバス上にある時は何時でもこのファンクションを呼び出します。このファンクションは与えられたアドレスがシミュレートされたアドレスの一つであるか否かを決定し、ドライバにTRUE(1)かFALSE(0)を返すことになります。より詳しくは、以下のアドレス選択ファンクションの説明を読んでください。

spollfuncはユーザーによって与えられたシリアルポーリング応答ファンクションのアドレスでなければなりません。ドライバはシミュレートしたデバイスの一つがシリアルポールされた時は何時でもこのファンクションを呼び出します。このファンクションは与えられたアドレスに対するシリアルポーリングの応答を返し、次にこの応

GotoMultAddr　(前ページより続く)　GotoMultAddr

答バイトをコントローラに送ります。より詳しくは、以下のシリアルポーリング応答ファンクションについての説明を参照してください。

複数のGPIBアドレスをシミュレートするアプリケーションを始める時は各インターフェースボード毎にこのファンクションを呼び出して下さい。ドライバは、この呼び出しにより、typeフィールドによって、複数1次或いは2次アドレスモードになります。このモードを不可能化するには、ibonl関数を0か1の値で呼び出します。ドライバがユーザーのアプリケーションコードのポインタ (addrfuncポインタとspollfuncポインタ)を保持しているので、アプリケーションが終了する前には0の値でibonlを呼び出す必要があります。これらのポインターをクリアしないでアプリケーションが出てしまうと、これらのファンクションが記憶装置から除かれた後でドライバがそれらに呼び出しをかけることがあり得ます。その場合コンピュータが動かなくなってしまいます。

このファンクションを作用させるには、インターフェースボードに対するハードウエア割込が可能化されている必要があります。IBCONFとかibconfig関数の呼び出しで割込が不可能化されていてこのファンクションの呼び出しを行うと、ECAPエラーが返ってきます。このエラーコードについては、第3章のNI-488.2ソフトウエアを理解するを参照してください。

複数アドレスモードが可能化されるた後、ドライバはアプリケーションからの情報が必要であると、addrfuncと spollfuncファンクションを呼び出します。これらは割込時に実行されるので、これらを書く時には特に注意を払う必要があります。割込時にファンクションを書く場合に守らなければならない規則を以下に示します。

- 割込呼び出しからなるべく早く戻ること。大きな計算はシステムがほかの割込活動(例えばシステムクロックを保つことなど)を行う妨げとなるので行ってはならない。

GotoMultAddr　(前ページより続く) GotoMultAddr

- 再入可能と解っているファンクションのほかは呼び出してはならない。この種のファンクションにはDOSとBIOS関数、スタンダードなCのライブラリ関数、Windows関数等があります。

- 大きなスタックスペースを使ってはならない。ユーザーの関数に与えられたスタックでは約512バイトを使用することが出来る。その上にランタイムスタックあふれの検査を不可能化することも出来る。これはMicrosoft Cのコンパイラの/Gsオプションを使って行うことが出来る。

- アプリケーションがMicrosoft Windows 3用に書かれているのであれば、コードとデータセグメントが記憶装置中にFIXEDになっていなければならない。これは自分のアプリケーションのモジュール定義ファイルの中で行うことが出来る。

アドレス選択ファンクション：

ドライバは1次或いは2次GPIBアドレスがバス上にある時は何時でもアドレス選択ファンクションを呼び出します。ドライバはGPIBアドレスをファンクションに引き渡します。ファンクションは与えられたアドレスを受け付けるか否かを決定します。ファンクションがアドレスを受け付けると、インターフェースボードはその与えられたGPIBアドレスを使用します。そうすると、アプリケーションプログラムはまるで与えられたGPIBアドレスを持ったデバイスであるかのようにバスとの読み取り書き込みが行えるようになります。

アドレス選択ファンクションの関数プロトタイプを示します。

```
unsigned short _far _loadds addrfunc (short board,
                                      unsigned short type,
                                      unsigned short addr)
```

GotoMultAddr　(前ページより続く)　GotoMultAddr

このファンクションが出る(終了する)時は_far宣言がコンパイラにfar復帰を生成するように告げます。_loadds宣言はコンパイラにData Segment[データセグメント]レジスタにアプリケーションのデフォルトセグメントをロードするよう告げます。これでこのファンクションはアプリケーションのグローバル変数にアクセスすることが出来ます。

ドライバは次のパラメータをアドレス選択ファンクションに引き渡します。

boardはGPIBアドレスが存在するインターフェースボードの指標です。このパラメータはアプリケーションが複数のバス、言い替えれば複数のインターフェースボード、を使用する場合のみ有用です。

typeはバス上のアドレスのタイプを指し、トークアドレス、リスンアドレス、或いはシリアルポーリングモード中のトークアドレスの何れかです。これらのタイプは、言語インターフェースのincludeファイルで定義されているMultAddrListen, MultAddrTalk,及びMultAddrSerialPollと言う定数によって定義されています。

addrは現在バス上に存在する5ビットのGPIBアドレスです。

このファンクションは、与えられたアドレスを受け付けると0以外の値を返します。与えられたアドレスの受け入れを拒否する時は0を返します。

シリアルポール応答ファンクション

インターフェースボードがコントローラ・イン・チャージからシリアルポーリングを受けると、ドライバはシリアルポール応答ファンクションを呼び出します。ドライバはポーリングを受けているデバイスのGPIBアドレスをこのファンクションに引き渡します。ファンクションは8ビットのシリアルポール応答バイトを返し、応答バイトはコントローラに送られます。シミュレートしたデバイスがポー

GotoMultAddr　(前ページより続く) GotoMultAddr

リングの時GenerateREQT関数によってサービスを要求している時は、ドライバは応答バイトをコントローラに送る前に、Request Service (サービス要求、RSVと省略) ビットを1にセットします。

シリアルポーリングが行われると、コントローラはポールするデバイスのトークアドレスを送り出します。次に、シリアルポール応答ファンクションの呼び出しの直前に、アドレス選択ファンクション呼び出しが行われます。この呼び出しはMultAddrSerialPollにtypeをセットして行われます。この様にアドレス選択ファンクションがシリアルポール応答ファンクションより先に呼び出されているので、コントローラがアプリケーションプログラムによりシミュレートされていないデバイスをポールしようとすると、アドレス選択ファンクションが0を返すことによりそのシリアルポーリングを拒否します。その結果として、ドライバによるシリアルポール応答ファンクションの呼び出しは起こらなくなります。

シリアルポール応答ファンクションの関数プロトタイプを次に示します。

```
unsigned short _far _loadds spollfunc (short board,
                                       unsigned short addr)
```

ドライバがこのファンクションに引き渡すパラメータは次の通りです。

boardはGPIBアドレスが存在するインターフェースボードの指標です。このパラメータは、アプリケーションが複数のバス、言い替えれば複数のインターフェースボード、を使用している場合のみ有用です。

addrは現在バス上に存在する5ビットGPIBアドレスです。

このファンクションはコントローラに送られる8ビットのシリアルポール応答バイトを返します。

GotoMultAddr　(前ページより続く)　GotoMultAddr

例：

C

次のプログラム例は4つのGPIBデバイスをシミュレートしています。シミュレートされているGPIBデバイスは1次アドレスの1、3、24及び30にあります。

```
#include <stdio.h>
#include <string.h>
#include <process.h>
#include <bios.h>
#include "c:\at-gpib\c\decl.h"

#define TRUE  1
#define FALSE 0

#define LAD  0x20  /* listen address mask */
#define TAD  0x40  /* talk address mask   */

#define BUFSIZE 512

/*
 *    Globals.
 */
short addressed = FALSE;
unsigned short address;
char buffer[ BUFSIZE + 2 ];

/*
 *  This function implements the "address
 *  selection" call-back function.  It it used
 *  to validate the addresses that are seen on
 *  the bus.  If GPIB addresses 1, 3, 24, or 30
 *  are seen on the bus, this function returns
```

GotoMultAddr　(前ページより続く)　GotoMultAddr

```
 *  TRUE.  This means this application should
 *  simulate those devices.
 */
unsigned short _far _loadds addrfunc(short board,
                                     unsigned short type,
                                     unsigned short addr)
{
   if ((addr == 1) || (addr == 3) ||
       (addr == 24) || (addr == 30)) {
      /*
       * If the program is about to be serial polled then
       * accept the address, so that the "spollfunc"
       * is called to return the serial poll response
       * byte.
       */
      if (type == MultAddrSerialPoll) {
         return (TRUE);
      }
      /*
       * If this is a listen address, then set the
       * global "addressed" to TRUE and store the
       * listen address in the global "address".
       */
      else if (type == MultAddrListen) {
         addressed = TRUE;
         address = (LAD | addr);
         return (TRUE);
      }
      /*
       * If this is a talk address, then set the
       * global "addressed" to TRUE and store the
       * talk address in the global "address".
       */
      else if (type == MultAddrTalk) {
         addressed = TRUE;
         address = (TAD | addr);
         return (TRUE);
      }
   }
```

GotoMultAddr　(前ページより続く)　GotoMultAddr

```
    /* Return FALSE since you do not claim this
     * address.
     */
    return (FALSE);

}   /* end of addrfunc */

/*
 * This function implements the "Serial Poll Response"
 * call-back function. It always returns the GPIB
 * address of the simulated device as the serial poll
 * response byte.
 */
unsigned short _far _loadds spollfunc(short board,
                                      unsigned short addr)
{
    return (addr);
}

/*
 * This program is an example of how to simulate
 * multiple GPIB addresses. The program waits in a
 * loop until one of its simulated addresses is
 * present on the bus.  It then reads data or writes
 * data for the simulated device.  If you press any
 * key, the program terminates.
 */
short _cdecl main ( void )

{
    short testing;
    short SimulatedAddress;

    addressed = FALSE;
    testing = TRUE;
```

GotoMultAddr (前ページより続く) GotoMultAddr

```
/*
 * Enable multiple primary GPIB addresses for
 * interface board #0. Pass the address of the
 * "address selection" function (addrfunc) and the
 * "serial poll response" function (spollfunc).
 */
GotoMultAddr(0, MultAddrPrimary, addrfunc,
             spollfunc);
if (ibsta & ERR) {
   printf("Error calling GotoMultAddr.\n");
   ibonl(0, 0);
   exit(1);
}

/*
 * This is the main loop. Stay here until any
 * key is pressed on the keyboard.
 */
while (testing) {
   printf("\nWaiting to be addressed....\n");
   /*
    * Check for any key to be pressed.
    */
   while (addressed == FALSE) {
      if (_bios_keybrd(_KEYBRD_READY)) {
         testing = FALSE;
         break;
      }
}

addressed = FALSE;
SimulatedAddress = address;

/*
 * As long as you did not press a key to exit,
 * then the program must be addressed to talk or
 * listen.
 */
```

GotoMultAddr　(前ページより続く)　GotoMultAddr

```
if (testing == TRUE) {
   /*
    * If the address is a listen address, then
    * read in data byte for the simulated
    * device. After reading in the bytes, call
    * GenerateREQT to request service for the
    * simulated device.
    */
   if ((SimulatedAddress & (LAD | TAD)) == LAD) {
      printf("Address %d is listening.\n",
             (SimulatedAddress & ~LAD));

      /*
       * Read a buffer for the given device.
       */
      RcvRespMsg(0, buffer,
            (unsigned long)BUFSIZE, STOPend);
      if (ibsta & ERR) {
         printf("Error from RcvRespMsg.\n");
         ibonl(0, 0);
         exit(1);
      }
      /*
       * Put a NUL byte at the end of the buffer
       * and call printf to output the buffer
       * to the screen.
       */
      buffer[ibcntl] = '\0';
      printf("Received '%s' for PAD %d\n",
             buffer,
             (SimulatedAddress & ~LAD));

      /*
       * Now assert SRQ to request service for
       * the simulated device.
       */
      GenerateREQT(0,(SimulatedAddress & ~LAD));
   }
```

GotoMultAddr　(前ページより続く)　GotoMultAddr

```
            /*
             * If the address in a talk address, then
             * output a buffer containing the GPIB
             * address of the simulated device.  Then
             * call GenerateREQF for the simulated
             * device.  It no longer requires service
             * from the Controller.
             */
            else if ((SimulatedAddress & (LAD | TAD))
                        == TAD) {
               printf("Address %d talking.\n",
                       (SimulatedAddress & ~TAD));

               sprintf(buffer,
                  "Data from GPIB address %d.",
                  (SimulatedAddress & ~TAD));

               SendDataBytes(0, buffer,
                  (unsigned long)strlen(buffer), DABend);
            if (ibsta & ERR) {
               printf("Error from SendDataBytes.\n");
               ibonl(0, 0);
               exit(1);
            }

            GenerateREQF(0,(SimulatedAddress & ~TAD));
         }
         else {
            printf("NOT talk or listen addressed.\n");
            ibonl(0, 0);
            exit(1);
         }
      }
   }
```

GotoMultAddr (前ページより続く) GotoMultAddr

```
    /*
     * Be certain that you call ibonl with a value
     * of 0 before exiting the program.
     */
    ibonl(0, 0);

    return 0;

}   /* end of main */
```

PassControl　PassControl

目的　コントロールを他のコントローラの資格のあるデバイスに移す。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL PassControl (board%, address%)
```

C

```
void PassControl (short board, unsigned short address)
```

boardはボード番号を規定します。このルーチンでは GPIB Device Take Controlメッセージがアドレスを示されたデバイスに送られます。パラメータのaddressはコントロールが与えられるデバイスの1次GPIBアドレスをその下位バイトに含んでいます。上位バイトの方は、デバイスに2次アドレスが無い場合は、0になります。そうでない場合は適当な2次アドレスを含まねばなりません。

例

コントロールをボード0に接続したコントローラに移す。コントローラの1次GPIBアドレスは9である。

BASICA/QuickBASIC/BASIC

```
80  board% = 0
90  address% = 9
100  CALL PassControl (board%, address%)
```

C

```
    PassControl (0, 9);
```

PPoll　PPoll

目的　パラレルポーリングを行う。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL PPoll (board%, result%)
```

C

```
void PPoll (short board, unsigned short *result)
```

boardはボード番号を指定します。このルーチンはパラレルポーリングを行い、8ビットの結果を result に記憶します。結果の上位の8ビットのみが影響を受けます。BASICA、QuickBASIC、およびBASICでは上位のバイトの内容は呼び出し以前のものと変わりがありません。

ポーリングの結果得られた各ビットはパラレルポーリング用に構成された各デバイスのステータス情報の1ビットを戻してきます。各ビットの状態（0か1）及びこれら状態の解釈はもっとも近い時期にデバイスに送られたパラレルポーリング構成とおのおののデバイスの状態により決まります。

例

ボード0においてパラレルポーリングを行う。

BASICA/QuickBASIC/BASIC

```
80  board% = 0
100  CALL PPoll (board%, result%)
```

C

```
unsigned short result;
PPoll (0, &result);
```

PPollConfig　PPollConfig

目的　一つのデバイスをパラレルポーリング用に構成する。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL PPollConfig (board%, address%, dataline%, sense%)
```

C

```
void PPollConfig (short board, unsigned short address,
                  unsigned short dataline,
                  unsigned short sense)
```

boardはボード番号を規定します。このルーチンはaddressに示されるGPIBデバイスを、パラメータのdatalineとsenseの示すところにしたがって、パラレルポーリング用に構成します。datalineはデバイスが応答すべきデータのライン(1－8)です。senseはデータのラインがアサートする、あるいはアサートを止めるときの条件を示します。デバイスはsenseの値(0あるいは1)をおのおののステータスビットと比較し、その結果にしたがって応答することになっています。

デバイスはパラレルポーリング用の構成を各個に自分自身で行うことも出来ます。この場合には、デバイスはコントローラがデバイスを構成しようという働きかけを無視します。PPollConfigまたはPPollUnconfigを使用する場合は、あらかじめデバイスが個々に構成出来るのか遠端から構成出来るのか確かめてから行ってください。

PPollConfig　(前ページより続く)　PPollConfig

例

ボード0にアドレス8において接続するデバイスを構成してそれがデータライン5においてセンス0の時に応答するように(即ちそのデバイスのステータスが0の時にラインをアサートし、1の時にアサートを止めるように)する。

BASICA/QuickBASIC/BASIC

```
70  address% = 8
80  board% = 0
90  dataline% = 5
91  sense% = 0
100  CALL PPollConfig (board%, address%, dataline%,
                       sense%)
```

C

```
PPollConfig (0, 8, 5, 0);
```

PPollUnconfig　PPollUnconfig

目的　デバイスをパラレルポーリングのための構成から解除する。

フォーマット

BASICA

```
CALL PPollUnconfig (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL PPollUnconfig (board%, addresslist% ())
```

C

```
void PPollUnconfig (short board, unsigned short addresslist [])
```

board はボード番号を規定します。このルーチンにより、GPIBデバイスでアドレスがアドレスアレーにあるものは、パラレルポーリングのための構成を解除されます。つまり、このルーチンを行うと、これらのデバイスはポーリングに参加しなくなります。パラメータの addresslist は任意のサイズのアドレス整数のアレーであって、NOADDR 値により終了させられます。

アレーが NOADDR 値しか含まない場合、あるいはC言語の中でNULLが現われた場合には、GPIB Parallel Poll Unconfigure (PPU) メッセージが送られ、全てのデバイスが構成から解除されます。

PPollUnconfig　(前ページより続く) PPollUnconfig

例

ボード0に接続されており、GPIBアドレスがそれぞれ8と9である2つのデバイスをパラレルポーリングのための構成から解く。

BASICA

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL PPollUnconfig (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL PPollUnconfig (0, addresslist% ())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
PPollUnconfig (0, addresslist);
```

RcvRespMsg　RcvRespMsg

目的　既にアドレスされたデバイスからデータバイトを読取る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL RcvRespMsg (board%, data$, termination%)
```

C

```
void RcvRespMsg (short board, unsigned char data [],
                 unsigned long count,
                 unsigned short termination)
```

boardはボード番号を指定します。countのデータバイトまではGPIBから読み出され、あらかじめ割り当てられたストリングであるdata中に入れられます。BASICA, QuickBASICおよびBASICにおいては、データの量はストリングの長さから推定されます。ストリングはあらかじめ適当な長さに割り当てておく必要があります。C言語においてcount引き数はlong型です。但し、integer値および変数も引き渡すことが出来ます。terminationはデータの終点を示す方法のフラグで、もしそれが0と16進数(hex) 00FFの間の数であれば該当する16進数が代表するASCII文字が終了文字とみなされます。この文字が検出されると読取は停止されます。terminationが定数のSTOPend (これはヘッダファイルのDECL.BAS, QBDECL.BAS, MBDECL.BAS, あるいはDECL.Hで定義されています)の場合は、読取はEOIが検出された時に停止されます。

SendCmdsによってトーカとリスナがアドレスされていることを前提として使用することになっています。したがって、RcvRespMsgにはアドレシングのステップは含まれていません。普通Receiveルーチンがアドレシングの全シーケンス及びそれに続くデータバイトの受信を行う目的で使用されます。

RcvRespMsg　(前ページより続く)　RcvRespMsg

例

既にアドレスしてあるトーカより100バイトを受信する。伝送は改行(linefeed)文字が検出されたときに終了するものとする。

BASICA

```
80  board% = 0
90  data$ = space$(100)
91  termination% = &H0A
100  CALL RcvRespMsg (board%, data$, termination%)
```

QuickBASIC/BASIC

```
DIM data AS STRING*100
CALL RcvRespMsg (0, data$, &H0A)
```

C

```
unsigned char data[100];
RcvRespMsg (0, data, 100, '\n');
```

ReadStatusByte　ReadStatusByte

目的　単一のデバイスのシリアルポーリングを行いそのステータスバイトを得る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL ReadStatusByte (board%, address%, result%)
```

C

```
void ReadStatusByte (short board, unsigned short address,
                     unsigned short *result)
```

boardはボード番号を規定します。このルーチンでは指定されたデバイスはシリアルポーリングされ、そのステータスバイトは変数resultの中に入れられます。ステータスバイトの上位バイトは零拡張です。

例

アドレス8にあるデバイスをシリアルポーリングしてステータスバイトを戻す。

BASICA/QuickBASIC/BASIC

```
80  board% = 0
90  address% = 8
100  CALL ReadStatusByte (board%, address%, result%)
```

C

```
unsigned short result;
ReadStatusByte (0, 8, &result);
```

Receive　Receive

目的　一つのGPIBデバイスからデータバイトを読み取る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL Receive (board%, address%, data$, termination%)
```

C

```
void Receive (short board, unsigned short address,
              unsigned char data [], unsigned long count,
              unsigned short termination)
```

boardはボード番号を規定します。このルーチンでは、指定されたGPIBデバイスがアドレスされ、countデータバイトまでがそのデバイスから読み取られ、あらかじめ割り当てられたストリングのdataに入れられます。countの値はBASICAでは整数ですが、QuickBASIC, BASIC, およびCにおいてはlong型です。しかしながら、long値であっても、整数値と諸変数も引き渡すことが出来ます。terminationはデータの終点指示方法のフラグで、もしそれが0と16進数(hex) 00FFの間の数であればその16進数が代表するASCII文字が終了文字とみなされます。この文字が検出されると読取は停止されます。terminationが定数のSTOPend (これはヘッダファイルのDECL.BAS, QBDECL.BAS, MBDECL.BAS, あるいはDECL.Hで定義されています)の場合は、読取はENDが検出された時に停止されます。

Receive　(前ページより続く)　Receive

例

アドレス8にあるデバイスから100バイトを受信する。伝送はENDが検出された時に停止するものとする。

BASICA/QuickBASIC/BASIC

```
80  board% = 0
81  address% = 8
90  data$ = space$(100)
91  termination% = STOPend
100  CALL Receive (board%, address%, data$, termination%)
```

C

```
unsigned char data[100];
Receive (0, 8, data, 100, STOPend);
```

ReceiveSetup　ReceiveSetup

目的　ある特定のデバイスをデータバイトを送出するよう準備させ、またGPIBインターフェースボードにそれらを読み取る準備をさせる。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL ReceiveSetup (board%, address%)
```

C

```
void ReceiveSetup (short board, unsigned short address)
```

boardはボード番号を指定します。このルーチンは特定されたGPIBデバイスをトーカとしてアドレスし、特定されたボードをリスナとしてアドレスします。このルーチンの後では、ふつうRcvRespMsgの様なルーチンを呼び出して実際にトーカからのデータ転送をすることなどが行われます。

このルーチンは最初にいくつかのデバイスがデータを受信できるよう準備し、つぎにRcvRespMsgを繰り返し呼び出して多数のデータブロックを受信する場合に便利です。こうするとブロックの度毎にデバイスにアドレスし直す必要が無くなります。この方法の代わりに、Receiveルーチンを使って最初のブロックを送信し、次に残りのブロック全てをRcvRespMsgによって送ることも出来ます。

ReceiveSetup （前ページより続き） ReceiveSetup

例

アドレス8にあるGPIBデバイスをボード0にデータバイトを送れるように準備する。次にデバイスから100バイトのmessagesを受取り、それを1つのストリング内に記憶する。メッセージはENDが来た時に停止させる。

BASICA

```
80  board% = 0
90  address = 8
94  messages$ = space$(100)
100  CALL ReceiveSetup (board%, address%)
130  termination% = STOPEND
140  CALL RcvRespMsg (board%, messages$, termination%)
```

QuickBASIC/BASIC

```
DIM message AS STRING*100
CALL ReceiveSetup (0, 80)
CALL RcvRespMsg (0, message$, STOPEND)
```

C

```
unsigned char message[100];
ReceiveSetup (0, 8)
RcvRespMsg (0, message, 100, STOPEND);
```

ResetSys　ResetSys

目的　GPIBシステムを3レベルで初期化する。

フォーマット

BASICA

```
CALL ResetSys (board%, addresslist%(0))
```

QuickBASIC/BASIC

```
CALL ResetSys (board%, addresslist%())
```

C

```
void ResetSys (short board, unsigned short addresslist [])
```

boardはボード番号を指定します。GPIBシステムは次の3レベルで初期化されます。

バスの初期化	Remote Enable (REN), 次に Interface Clear (IFC) がアサートされます。その結果、全てのデバイスはアドレスされていない状態になり、GPIBインターフェースボード（システムコントローラ）がコントローラインチャージ (CIC) になります。
メッセージ交換初期化	Device Clear (DCL)メッセージが全ての接続されたデバイスに送られます。その結果として、全ての488.2規格にあったデバイスは次のReset (RST) メッセージを受信できるようになります。
デバイス初期化	*RST メッセージが addresslist 引き数にアドレスが含まれる全てのデバイスに送られます。その結果として、一つ一つのデバイスの中のデバイス特有の機能が初期化されます。

ResetSys　(前ページより続く)　ResetSys

例

アドレス8、9及び10にあるデバイスを含むGPIBシステムを完全にリセットする。

BASICA

```
70  DIM addresslist%(4)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = 10
93  addresslist% (3) = NOADDR
100  CALL ResetSys (board%, addresslist%(0))
```

QuickBASIC/BASIC

```
DIM addresslist%(4)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = 10
addresslist% (3) = NOADDR
CALL ResetSys (0, addresslist%())
```

C

```
unsigned short addresslist[4] = {8, 9, 10, NOADDR};
ResetSys (0, addresslist);
```

Send　Send

目的　データバイトを単一のGPIBデバイスに送る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL Send (board%, address%, data$, eotmode%)
```

C

```
void Send (short board, unsigned short address,
           unsigned char data [], unsigned long count
           unsigned short eotmode)
```

boardはボード番号を指定します。このルーチンでは特定されたGPIBデバイスはリスナとしてアドレスされ、特定されたボードはトーカとしてアドレスされます。そしてdata中のcountのデータバイトが送出されます。countの値はBASICAでは整数です。一方QuickBASIC, BASIC, およびCにおいてはcountの値はlong型です。しかし、これらの言語でlong型であっても、整数値や諸変数も引き渡すことが出来ます。eotmodeはデータの終結をリスナに示す信号方法のフラグです。eotmodeは次の定数の何れかに設定しなければなりません。

NLend　はデータバイトの後にNL(linefeed)をEOIと共に送れという意味です。

DABend　はストリングの最後のデータバイトと共にEOIを送れという意味です。

NULLend　は転送の終了を示す印をつけるなという意味です。

これらの定数はヘッダファイルのDECL.BAS, QBDECL.BAS, MBDECL.BAS, およびDECL.Hの中で定義されています。

Send　(前ページより続く)　Send

例

アドレス8にあるGPIBデバイスに識別用の質問を送る。データ転送はENDを伴った改行文字で終了させる。

BASICA

```
80  board% = 0
81  address% = 8
90  data$ = "*IDN?"
91  eotmode% = NLend
100  CALL Send (board%, address%, data$, eotmode%)
```

QuickBASIC/BASIC

```
CALL Send (0, 8, "*IDN?", NLend)
```

C

```
Send (0, 8, "*IDN?", 5, NLend);
```

SendCmds　SendCmds

目的　GPIBコマンドバイトを送る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL SendCmds (board%, commands$)
```

C

```
void SendCmds (short board, unsigned char commands [],
               unsigned long count)
```

boardはボード番号を指定します。commandsにはGPIBに送られるコマンドバイトが含まれています。送られるストリング中のバイトの数は引き数のcountにより示されます。countの値はBASICAでは整数であり、QuickBASIC, BASIC, 及びCにおいては long型です。但し、これらの言語では、countがlong値であっても、整数値や変数も送ることが出来ます。

GPIBのオペレーションにはSendCmdsは普通必要ではありません。SendCmdsは他のルーチンが持っていない特殊なコマンドシーケンスをGPIBに送る必要のある場合に用います。

SendCmds　(前ページより続く)　SendCmds

例

アドレス0のコントローラがアドレス8と9にある2つのGPIBデバイスを同時にトリガしてそれらをローカルモードに変えるようにする。

BASICA

```
80  board% = 0
90  commands$ = chr$(&3F)+chr$(&40)+chr$(&H28)+chr$(&H29)
    +chr$(&H04)+chr$(&H01)
100  CALL SendCmds (board%, commands$)
```

QuickBASIC/BASIC

```
CALL SendCmds (0,
chr$(&3F)+chr$(&40)+chr$(&H28)+chr$(&H29)
+chr$(&H04)+chr$(&H01))
```

C

```
SendCmds (0, "\x3F\x40\x28\x29\x04\x01", 6);
```

SendDataBytes　SendDataBytes

目的　既にアドレスされたデバイスにデータバイトを送る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL SendDataBytes (board%, data$, eotmode%)
```

C

```
void SendDataBytes (short board, unsigned char data [],
                    unsigned long count,
                    unsigned short eotmode)
```

boardはボード番号を指定します。dataはGPIB送られるべきデータバイトを含んでいます。送られるストリング中のバイトの数は引き数の count により示されます。 count の値はBASICAでは整数であり、QuickBASIC, BASIC, 及び Cではlong 型です。但し、これらの言語では、countが long値であっても、整数値や変数も引き渡すことが出来ます。eotmode はデータの終結をリスナに示す信号方法を示すフラグです。eotmode は次の定数の何れかに設定しなければなりません。

NLend　はデータバイトの後にNL(linefeed)をEOIと共に送れという意味です。

DABend　はストリングの最後のデータバイトと共にEOIを送れという意味です。

NULLend　は転送の終了を示す印をつけるなという意味です。

これらの定数はヘッダファイルのDECL.BAS, QBDECL.BAS, MBDECL.BAS, および DECL.H の中で定義されています。

SendDataBytes (前ページより続く) SendDataBytes

SendDataBytesの使用に際してはあらかじめSendSetup, Send, あるいはSendCmdsの様な関数を呼び出して全てのGPIBリスナにアドレスしておく必要があります。したがって、このルーチンはGPIB管理の各ステップのうちアドレシングのステップを省略するときに使用します。アドレシングの全手続きを行った上でデータバイトを伝送したい場合は、普通Sendルーチンを用います。

例

全てのアドレスしたリスナに識別用の質問を送る。伝送はENDを伴った改行文字に出会ったときに終了される。

BASICA

```
80  board% = 0
90  data$ = "*IDN?"
91  eotmode% = NLend
100  CALL SendDataBytes (board%, data$, eotmode%)
```

QuickBASIC/BASIC

```
CALL SendDataBytes (0, "*IDN?", NLend)
```

C

```
SendDataBytes (0, "*IDN?", 5, NLend);
```

SendIFC　SendIFC

目的　IFCによりGPIBインターフェース機能をクリアする。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL SendIFC (board%)
```

C

```
void SendIFC (short board)
```

boardはボード番号を指定します。このルーチンはGPIB Device IFC [GPIB デバイスインターフェースをクリア]のメッセージを出し、全ての接続したデバイスのインターフェース機能をクリアした状態に戻します。

このルーチンはGPIBの初期化手続きの一部として行います。これはGPIBインターフェースボードを強制的にGPIBのコントローラとすることにより、全ての接続されたデバイスをアドレスされていない状態にし、またインターフェース機能を非活動の状態にします。

例

ボード0に接続されたデバイスのインターフェース機能をクリアする。

BASICA

```
80  board% = 0
100  CALL SendIFC (board%)
```

QuickBASIC/BASIC

```
CALL SendIFC (0)
```

C

```
SendIFC (0);
```

SendList　SendList

目的　複数のGPIBデバイスにデータバイトを送る。

フォーマット

BASICA

```
CALL SendList (board%, addresslist%(0), data$, eotmode%)
```

QuickBASIC/BASIC

```
CALL SendList (board%, addresslist%(), data$, eotmode%)
```

C

```
void SendList (short board, unsigned short addresslist [],
               unsigned char data [], unsigned long count,
               unsigned short eotmode)
```

boardはボード番号を指定します。addresslistは1次GPIBアドレスのリストを含んでおり、NOADDR値により終了させられます。GPIBデバイスでアドレスがこのアドレスアレーに入っているものはリスナとしてアドレスされます。指定されたボードはトーカとしてアドレスされます。そしてcountのデータバイトが送られます。count値はBASICAにおいては整数で、QuickBASIC, BASIC, およびCにおいてはlong型です。しかしながら、countがlong値であっても、整数値や変数を引き渡すことは可能です。eotmodeはデータの終末をリスナに知らせる信号方法のフラグです。eotmodeは次の3つの定数のうちの一つに設定しなければなりません。

NLend　はデータバイトのの値にNL(linefeed)をEOIと共に送れという意味です。

DABend　はストリングの最後のデータバイトと共にEOIを送れという意味です。

NULLend　は転送の終了を示す印をつけるなという意味です。

SendList　(前ページより続き)　SendList

これらの定数はヘッダファイルのDECL.BAS, QBDECL.BAS, MBDECL.BAS, および DECL.H の中で定義されています。

このルーチンは Send ルーチンと似ていますが、複数のリスナに唯一度の伝送でデータを送ることが出来るところが違っています。

例

アドレス8と9にあるGPIBデバイスに識別のための問い合わせを行う。伝送はEOIを伴った改行文字(linefeed)により終了せしめるものとする。

BASICA

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
93  data$ = "*IDN?"
94  eotmode% = NLend
100  CALL SendList (board%, addresslist%(0), data$,
eotmode%)
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL SendList (0, addresslist%(), "*IDN?", NLend)
```

C

```
unsigned addresslist[3] = {8, 9, NOADDR};
SendList (0, addresslist, "*IDN?", 5, NLend);
```

SendLLO　SendLLO

目的　すべてのデバイスにLocal Lockout [ローカルロックアウト] メッセージを送る。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL SendLLO (board%)
```

C

```
void SendLLO (short board)
```

boardはボード番号を指定します。このルーチンでは全てのデバイスにLocal Lockoutメッセージを送り、各デバイスが別々にローカルになりリモートの状態を選べないようにします。Local Lockoutが発効すると、コントローラによる外はその状態を変えることは出来ません。デバイスのローカルとリモートの状態の間の変換はコントローラにより適当なGPIBメッセージを送ることにより行うことができます。

SendLLOはローカルとリモート状態の関係した特殊な状況、特にすべてのデバイスをロックアウトして、ローカルプログラミングの状態にする場合において使用されます。デバイスをリモートモードでロックアウトした[Remote Mode With Lockout]状態にしたい場合には、SetRWLSルーチンを使います。

SendLLO　(前ページより続き)　SendLLO

例

Local Lockout［ローカルロックアウト］メッセージをボード0に接続したすべてのデバイスに送る。

BASICA

```
80  board% = 0
100  CALL SendLLO (board%)
```

QuickBASIC/BASIC

```
CALL SendLLO (0)
```

C

```
SendLLO (0);
```

SendSetup　SendSetup

目的　特定のデバイスがデータバイトを受信できるよう準備する。

フォーマット

BASICA

```
CALL SendSetup (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL SendSetup (board%, addresslist% ())
```

C

```
void SendSetup (short board, unsigned short addresslist [])
```

boardはボード番号を指定します。このルーチンはGPIBデバイスでアドレスがaddresslist のアレーに含まれるものをリスナとしてアドレスし、指定されたボードをトーカとしてアドレスします。このルーチンの後では、通常 SendDataBytes のようなルーチンを呼び出して実際にデータをリスナに転送することがおこなわれます。パラメータの addresslist は任意のサイズのアドレス整数のアレーであって、NOADDR値により終了させられます。

このコマンドはデータ伝送の準備としてまずデバイスにアドレスし、つぎに SendDataBytes の複数呼び出しによってデータの複数ブロックを送る場合に有用です。こうすると、ブロック間でアドレシングを繰り返す手間が省けます。この方法の代わりとして、Sendルーチンを使用して最初のブロックを送り、次に以下のブロックをSendDataBytesを利用して送ることも出来ます。

SendSetup　(前ページより続き)　SendSetup

例

アドレス8と9にあるGPIBデバイスをデータバイトを受信するように準備する。次に、ストリングアレーに記憶した5つのメッセージを両方のデバイスに送る。最後のメッセージの終のバイトを送る時にEOIを一緒に送る。

BASICA

```
70  DIM addresslist%(3)
75  DIM messages$(5)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
94  messages$(0) = "Message 0"
95  messages$(1) = "Message 1"
96  messages$(2) = "Message 2"
97  messages$(3) = "Message 3"
98  messages$(4) = "Message 4"
100  CALL SendSetup (board%, addresslist% (0))
110  for i% = 0 to 3
120    CALL SendDataBytes (board%, messages$(i%), NULLEND)
130  NEXT i%
140  CALL SendDataBytes (board%, messages$(4), NLend)
```

SendSetup　(前ページより続き)　SendSetup

QuickBASIC/BASIC

```
DIM addresslist%(3)
DIM messages$(5)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
messages$(0) = "Message 0"
messages$(1) = "Message 1"
messages$(2) = "Message 2"
messages$(3) = "Message 3"
messages$(4) = "Message 4"
CALL SendSetup (0, addresslist% ())
FOR i% = 0 to 3
  CALL SendDataBytes (board%, messages$(i%), NULLEND)
NEXT i%
CALL SendDataBytes (board%, messages$(4), NLend)
```

C

```
int i;
unsigned short addresslist[3] = {8, 9, NOADDR};
unsigned char *messages[5] = {
     "Message 0",
     "Message 1",
     "Message 2",
     "Message 3",
     "Message 4"  };
SendSetup (0, addresslist)
for (i = 0; i < 4; i++)
     SendDataBytes (0, messages[i],
                    strlen (messages[i]),NULLend);
SendDataBytes (0, messages[4], strlen (messages[4])NLend);
```

SetRWLS　SetRWLS

目的　特定のデバイスを Remote With Lockout [リモートモードでロックアウト] した状態にする。

フォーマット

BASICA

```
CALL SetRWLS (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL SetRWLS (board%, addresslist% ())
```

C

```
void SetRWLS (short board, unsigned short addresslist [])
```

boardはボード番号を指定します。このルーチンはGPIBデバイスのうちで addresslist アレー中にアドレスが含まれているものを Remote Enable (REN) のアサートによりリモートモードにし、これらのデバイスをリスナとしてアドレスします。更に、すべてのデバイスをロックアウトの状態 [Lockout State] にすることにより、デバイスが、コントローラを通さずに個々にローカルプログラミングのモードになることを防ぎます。パラメータの addresslist は任意のサイズのアドレス整数のアレーであって、NOADDR 値により終了されます。

SetRWLS　(前ページより続く)　SetRWLS

例

アドレス8と9にあるGPIBデバイスを Remote With Lockout State [リモートモードでロックアウトした状態] にする。

BASICA

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL SetRWLS (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL SetRWLS (0, addresslist% ())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
SetRWLS (0, addresslist);
```

TestSRQ　TestSRQ

目的　SRQラインの現在の状態を決定する。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL TestSRQ (board%, result%)
```

C

```
void TestSRQ (short board, short *result)
```

boardはボード番号を指定します。このルーチンでは、GPIBのSRQラインがアサートされている場合は変数の結果に1の値を置きます。SRQラインがアサートされていない場合は0を置きます。

このルーチンはフォーマットがWaitSRQに似ていますが、WaitSRQがSRQの生起を待ち、その間プログラムの実行を一時停止するのに対し、TestSRQルーチンは直ちに現在のSRQの状態を戻してくる点で異なっています。

例

SRQの現在の状態を決定する。

BASICA

```
80  board% = 0
100  CALL TestSRQ (board%, result%)
105  IF result% = 1 then
110       'SRQ is asserted
111  ELSE
112       'No SRQ at this time
113  END IF
```

TestSRQ　(前ページより続く)　TestSRQ

QuickBASIC/BASIC

```
CALL TestSRQ (0, result%)
IF result% = 1 then
    ` SRQ is asserted
ELSE
    ` No SRQ at this time
END IF
```

C

```
short result;
TestSRQ (0, &result);
if (result == 1)
  {  /* SRQ is asserted */  }
else
  {  /* No SRQ at this time */  }
```

TestSys　TestSys

目的　デバイスの自己テストを行わせる。

フォーマット

BASICA

```
CALL TestSys (board%, addresslist% (0), resultlist% (0))
```

QuickBASIC/BASIC

```
CALL TestSys (board%, addresslist% (), resultlist% ())
```

C

```
void TestSys (short board, unsigned short addresslist [],
              unsigned short resultlist [])
```

boardはボード番号を指定します。このルーチンによってアドレスアレーにアドレスがあるGPIBデバイスは自己テストを行うようにという同一メッセージを同時に受け取ります。これに対し、各デバイスはおのおののテストの結果を示す整数コードを戻してきます。これらのコードは resultlist アレーの中の該当する要素の中に置かれます。IEEE-488.2規格により結果コード0はデバイスがテストに合格したことを示します。0以外のすべての値はテストの結果がエラーであることを示します。変数 ibcnt にはテストに不合格であったデバイスの数が含まれいます。パラメータの addresslist は任意のサイズのアドレス整数のアレーであって、NOADDR 値により終了させられます。

TestSys　(前ページより続く)　TestSys

例

GPIBアドレス8と9にあり、ボード0に接続している2つのデバイスに自己テストを行うように命令する。

BASICA/QuickBASIC/BASIC

```
70  DIM addresslist%(3)
71  DIM resultlist%(2)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL TestSys (board%, addresslist% (0),
                   resultlist%(0))
101  ` If any of the results are non-zero, the
102  ` corresponding device has failed the test.
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
unsigned short resultlist[2];
TestSys (0, addresslist, resultlist);
```

Trigger　Trigger

目的　単一のデバイスをトリガする。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL Trigger (board%, address%)
```

C

```
void Trigger (short board, unsigned short address)
```

boardはボード番号を規定します。このルーチンは指定されたアドレスにあるデバイスにGPIBコマンドのGroup Execute Trigger (GET) を送ります。パラメータのaddressはクリアすべきデバイスの1次GPIBアドレスをその下位バイトに含みます。デバイスに2次アドレスが無い場合は上位バイトは0です。2次アドレスがあれば、上位バイトは適当な2次アドレスを含みます。アドレスがNOADDRであればGroup Execute Triggerはアドレシング無しで送られ、それまでにアドレスされたすべてのリスナをトリガします。

Triggerルーチンは唯1つのGPIBデバイスをトリガする場合に使用します。同一のメッセージを送って幾つかの特定のGPIBデバイスをトリガしたい場合はTriggerListルーチンを使用してください。

例

ボード0に接続され、1次GPIBアドレスが9であり、2次GPIBアドレスが97であるディジタル電圧計をトリガする。

BASICA

```
80  board% = 0
90  address% = 9 + 256*97
100  CALL Trigger (board%, address%)
```

Trigger　(前ページより続く)　Trigger

QuickBASIC/BASIC

```
CALL Trigger (0, 9 + 256*97)
```

C

```
Trigger (0, MakeAddr (9, 97));

/* In C, a macro has been defined in the header
 * file DECL.H, MakeAddr(p,s), which can be
 * used to pack the primary and secondary
 * addresses into the correct form.
 */
```

[C ではマクロはヘッダファイルの DECL.H において定義されている。MakeAddr(p,s) は1次と2次のアドレスを正しい形式にまとめるために使用することが出来る。]

TriggerList　　TriggerList

目的　複数のデバイスをトリガする。

フォーマット

BASICA

```
CALL TriggerList (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
CALL TriggerList (board%, addresslist% ())
```

C

```
void TriggerList (short board, unsigned short addresslist [])
```

boardはボード番号を指定します。このルーチンはGPIBデバイスでアドレスがアドレスアレーにあるものを同時にトリガします。パラメータのaddresslistは任意のサイズのアドレス整数のアレーであって、NOADDR値により終了させられます。もしもこのアレーがNOADDRしか含んでいない場合、あるいはC言語においてNULL値が引き渡された場合は、Group Execute Triggerメッセージはアドレシング無しで送られ、それまでにアドレスされたすべてのリスナをトリガします。

TriggerListルーチンは如何なる数のGPIBデバイスのトリガにも使用出来る一般性を持っていますが、唯一つのGPIBデバイスをトリガしたい場合は普通Triggerルーチンを使用してください。

TriggerList　(前ページより続く)　TriggerList

例

ボード0に接続され、GPIBアドレスの8と9にある2つのデバイスを同時にトリガする。

BASICA

```
70  DIM addresslist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = NOADDR
100  CALL TriggerList (board%, addresslist% (0))
```

QuickBASIC/BASIC

```
DIM addresslist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = NOADDR
CALL TriggerList (0, addresslist% ())
```

C

```
unsigned short addresslist[3] = {8, 9, NOADDR};
TriggerList (0, addresslist);
```

WaitSRQ　WaitSRQ

目的　デバイスがService Request [サービス要求] をアサートするまで待機する。

フォーマット

BASICA/QuickBASIC/BASIC

```
CALL WaitSRQ (board%, result%)
```

C

```
void WaitSRQ (short board, short *result)
```

boardはボード番号を指定します。このルーチンは指定されたボードに接続されたあるGPIBデバイスがService Request (SRQ)ラインをアサートするまでプログラムの実行を一時停止するために使用されます。タイムアウト前にSRQが起こりますと変数結果が1にセットされます。SRQが検出されないうちにタイムアウトになった場合は、結果は0にセットされます。

WaitSRQはフォーマットにおいてTestSRQに似ています。しかし、TestSRQがSRQの状態を直ちに報告するのに比べて、WaitSRQはタイムアウト期間中はプログラムの実行を一時停止してSRQが起こるのを待つという点で異なっています。

WaitSRQ　　(前ページより続く)　　WaitSRQ

例

一つのGPIBデバイスがサービスを要求するのを待ち、要求の後にアドレス8、9と10の3つのデバイスの何れがサービスを要求したかを決定する。

BASICA

```
70  DIM addresslist%(4), resultlist%(3)
80  board% = 0
90  addresslist% (0) = 8
91  addresslist% (1) = 9
92  addresslist% (2) = 10
93  addresslist% (3) = NOADDR
100  CALL WaitSRQ (board%, result%)
105  IF result% = 1 THEN
110       CALL AllSpoll (board%, addresslist% (0),
                         resultlist%(0))
120  END IF
130  ' resultlist%() now contains the serial
131  ' poll responses for the three devices.
```

QuickBASIC/BASIC

```
DIM addresslist%(4), resultlist%(3)
addresslist% (0) = 8
addresslist% (1) = 9
addresslist% (2) = 10
addresslist% (3) = NOADDR
CALL WaitSRQ (0, result%)
IF result% = 1 then
    CALL AllSpoll (0, addresslist% (), resultlist%())
END IF
```

C

```
unsigned short addresslist[4] = {8, 9, 10, NOADDR};
unsigned short resultlist[3];
int result;
WaitSRQ (0, &result);
if (result == 1)
    AllSpoll (0, addresslist, resultlist);
```

BASICA/QuickBASIC/BASIC/Cによるプログラミングの例

IEEE-488.2-1987規格の優れた特長はNI-488.2のルーチンの使用によって100パーセント生かすことが出来ます。これらのルーチンはIEEE-488.2で規定されたコントローラコマンドとプロトコルと完全な互換性を持っています。

NI-488.2ルーチンは容易に習得、使用できます。大多数のアプリケーションプログラムの使用には、ほんの数種のルーチンしか必要になりません。

ここではユーザーの皆様が御自身のパーソナルコンピュータによりNI-488.2のルーチンを使用する場合を考慮して、代表的なIEEE-488.2規格の計測器をプログラムする際に利用可能なプログラミングの手順を例示しました。これらのアプリケーションプログラムはBASICA, QuickBASIC, BASIC, およびCによって書かれています。対象となっている計測器はディジタル電圧計［digital voltmeter (DVM)］です。ディジタル電圧計であるということ以上には特別な指定をしていません(例えば、別にDVMのメーカー名は指定していません）。ここで目的とするところは、如何にドライバを使用してNI-488.2プログラミングを行い、シーケンスをコントロールするかを説明することであって、これらのシーケンスを決定する方法を説明することではありません。

注)　各ステップの詳細な説明については、ユーザー各位に購入をいただいたインターフェースボードに付属している "Getting Started with..." と題するマニュアルの第3章 *Writing an Advanced Program Using NI-488.2 Routines* を参照してください。

1. ディストリビューションディスクのファイルからNI-488.2ルーチンの定義をロードします。

2. Controller State (CACS). IEEE-488 バス及びインターフェースボードのコントローラ回路を初期化して、各デバイスとのIEEE-488インターフェースが静止状態にあるようにします。また、インターフェースボードがコントローラインチャージであり Active Controller State (CACS) であるようにします。

3. 全てのリスナを検出します。

   a. IEEE-488バスに接続している可能性のある全てのIEEE-488 1次アドレスを含むアレーを作ります。

   b. もし接続しているデバイスがある場合は、その、あるいはそれらのデバイスを探し出します。

4. おのおののデバイスに識別のための問い合わせを送ります。

5. 計測器を次のようにして初期化します。

   a. マルチメータをクリアします。

   b. メータにIEEE-488.2 Reset [リセット] コマンドを送ります。

6. メータにオートレンジング(AUTO)で交流電圧 (VAC)を測定すること、測定を始める前 ((TRIGGER 2) にコントローラからトリガが来るのを待つこと、また測定が完了してメータが測定結果を送りだす準備が出来た時(*SRE 16) IEEE-488サービス要求ライン(SRQ)をアサートすることを命令します。

7. 測定の度毎に、

   a. TRIGGER [トリガ]コマンドをマルチメータに送ります。VAL1コマンドはメータに次のトリガの読みをIEEE-488.2出力バッファに送るように命令します。

   b. DVMがサービス要求 (SRQ) をアサートして測定値を読み出す準備が出来たことを知らせるのを待ちます。

   c. ステータスバイトを読んで測定したデータが有効であるか、あるいは誤った条件が存在するかを判断します。「メッセージあり (MAV) ビット (ステータスバイトの第4ビット) をチェックすることによりこれが解ります。

   d. データが有効であった場合はDVM から10バイトを読取ります。

8. セッションを終ります。

NI-488.2 ソフトウエアではボード4枚までの使用が可能です。4枚のボードはアプリケーションプログラムの中で番号を与えることにより区別します。第1のボードには（0）、第2のボードには（1）の番号を与える等となります。2枚以上のボードが御使用中のコンピュータにインストールしてあり、どのボードが0でどれが1、2、3などか解らない場合は、構成用ユーティリティのibconfを走らせて見てください。ibconf はボード番号とボードのベースアドレスとの関係を知らせてくれるので、ボードをベースアドレスにより識別することが出来ます。ibconf を走らせて利用するには、このマニュアルの第2章に示す情報を参照してください。

BASICAプログラム例—NI-488.2のルーチン

```
100  REM
110  REM  You must merge this code with DECL.BAS.
120       dim instruments% (31)
130       dim result% (31)
140       READING$ = Space$(30)
150       boardindex% = 0
160  REM
170       CLS
180  REM
190  REM  Your interface board must be the Controller-In-
200  REM  Charge to perform the Find All Listeners protocol.
210  REM
220       Call SendIFC (boardindex%)
230       msg$ = "SendIFC Error"
240       If ibsta% and EERR THEN GOSUB 4000 : STOP
250  REM
260  REM  Create an array with all of the valid GPIB primary
270  REM  addresses. This array will be given to the Find All
280  REM  Listeners protocol.
290  REM
300       For k% = 0 to 30
310          instruments% (k%) = k%
320       Next k%
330  REM
340       instruments%(31) = NOADDR%
350  REM
360  REM  Find all of the listeners on the bus.
370  REM
380       Print "Finding all listeners on the bus..."
390       limit% = 31
400  REM
410       CALL FindLstn (boardindex%, instruments%(0),
                         result%(0), limit%)
420       msg$ = "FindLstn Error"
430       If ibsta% and EERR then GOSUB 4000 : STOP
440  REM
450       num.listeners% = ibcnt% - 1
460       Print "No. of instruments found = ", num.listeners%
470  REM
480  REM  Now send the *IDN? command to each of the devices
490  REM  that you found.
500  REM
510  REM  The GPIB board is at address 0 by default.  Your
520  REM  GPIB board does not respond to *IDN?, so skip it.
```

```
530  REM
540       For k% = 1 to num.listeners%
550           cmd$ = "*IDN?"
560           call Send(boardindex%, result%(k%), cmd$, NLend%)
570           msg$ = "Send Error"
580           If ibsta% and EERR then GOSUB 4000 : STOP
590  REM
600           call Receive(boardindex%,result%(k%),Reading$,
                        STOPend%)
610           msg$ = "Receive Error"
620           If ibsta% and EERR then GOSUB 4000 : STOP
630  REM
640           pad% = result%(k%) and &HFF
650           print "The instrument at address "; pad%; " is: ",
                    left$(Reading$, IBCNT%)
660           If left$(Reading$, 9)="FLUKE, 45" then GOTO 2000
670       Next k%
680  REM
690       Print "Did not find the Fluke!" : STOP

2000 REM  Device Found.
2010 REM
2020      Print "**** We found the Fluke 45 ****"
2030      fluke% = result%(k%)
2040 REM
2050 REM  Reset the Fluke.
2060 REM
2070      Call DevClear (boardindex%, fluke%)
2080      msg$ = "DevClear Error"
2090      If ibsta% and EERR then GOSUB 4000  :  STOP
2100 REM
2110      cmd$ = "*RST"
2120      Call Send(boardindex%, fluke%, cmd$, NLend%)
2130      msg$ = "Send *RST Error"
2140      If ibsta% and EERR then GOSUB 4000 : STOP
2150 REM
2160 REM  Set up for a test.  Allow the Fluke to assert
2170 REM  SRQ when it has a message to send.
2180 REM
2190      cmd$ = "VAC; AUTO; TRIGGER 2; *SRE 16"
2200      Call Send(boardindex%, fluke%, cmd$, NLend%)
2210      msg$ = "Send Setup Error"
2220      If ibsta% and EERR then GOSUB 4000 : STOP
2230 REM
2240      sum = 0
2250 REM
2260 REM  Trigger the Fluke.
```

```
2270 REM
2280      For m% = 1 to 10
2290          cmd$ = "*TRG; VAL1?"
2300          Call Send(boardindex%, fluke%, cmd$, NLend%)
2310          msg$ = "Send Trigger Error"
2320          If ibsta% and EERR then GOSUB 4000 : STOP
2330 REM
2340 REM  Wait for the Fluke to assert SRQ, meaning it is ready
2350 REM  with the measurement.
2360 REM
2370          Call WaitSRQ(boardindex%, SRQasserted%)
2380          msg$ = "WaitSRQ Error"
2390          If SRQasserted% = 0 then GOSUB 4000 : STOP
2400 REM
2410 REM  Read its status byte. Be sure that the Message
2420 REM  Available bit is set.
2430 REM
2440          Call ReadStatusByte(boardindex%, fluke%,
                                  status%)
2450          msg$ = "ReadStatusByte Error"
2460          If ibsta% and EERR then GOSUB 4000 : STOP
2470 REM
2480          msg$ = "Improper Status Byte"
2490          If (status% and &H010) <> &H010 then GOSUB 4000
2500          If (status% and &H010) <> &H010 then Print
                 "Status byte: "; status% : STOP
2510 REM
2520 REM  Read the measurement.
2530 REM
2540          READING$ = Space$(30)
2550          Call Receive (boardindex%, fluke%, Reading$,
                            STOPend%)
2560          msg$ = "Receive Error"
2570          If ibsta% and EERR then GOSUB 4000 : STOP
2580 REM
2590          Reading$ = left$(Reading$, IBCNT%)
2600          Print "Reading: "; Reading$
2610 REM
2620          sum = sum + val(Reading$)
2630      Next m%
2640 REM
2650      Print "The average of the 10 readings is ", sum/10
2660 REM
2670 REM  Call the ibonl function to disable the hardware and
2680 REM  software.
2690 REM
2700      v% = 0 : CALL ibonl (boardindex%,v%) : STOP
```

```
2710   END

4000 REM  Print Status Variables.
4010 REM
4020      locate 15,1
4030      Print msg$
4040 REM
4050      Print "ibsta=&H"; hex$(ibsta%); "< ";
4060      If ibsta% and EERR  then print " ERR";
4070      If ibsta% and TIMO then print " TIMO";
4080      If ibsta% and EEND then print " END";
4090      If ibsta% and SRQI then print " SRQI";
4100      If ibsta% and RQS  then print " RQS";
4110      If ibsta% and CMPL then print " CMPL";
4120      If ibsta% and LOK  then print " LOK";
4130      If ibsta% and RREM then print " REM";
4140      If ibsta% and CIC  then print " CIC";
4150      If ibsta% and AATN then print " ATN";
4160      If ibsta% and TACS then print " TACS";
4170      If ibsta% and LACS then print " LACS";
4180      If ibsta% and DTAS then print " DTAS";
4190      If ibsta% and DCAS then print " DCAS";
4200      Print ">"
4210 REM
4220      Print "iberr="; iberr%;
4230      If iberr% = EDVR then print " EDVR <DOS Error>"
4240      If iberr% = ECIC then print " ECIC <Not CIC>"
4250      If iberr% = ENOL then print " ENOL <No Listener>"
4260      If iberr% = EADR then print " EADR <Address error>"
4270      If iberr% = EARG then print " EARG <Invalid argument>"
4280      If iberr% = ESAC then print " ESAC <Not Sys Ctrlr>"
4290      If iberr% = EABO then print " EABO <Op. aborted>"
4300      If iberr% = ENEB then print " ENEB <No GPIB board>"
4310      If iberr% = EOIP then print " EOIP <Async I/O in prg>"
4320      If iberr% = ECAP then print " ECAP <No capability>"
4330      If iberr% = EFSO then print " EFSO <File sys. error>"
4340      If iberr% = EBUS then print " EBUS <Command error>"
4350      If iberr% = ESTB then print " ESTB <Status byte lost>"
4360      If iberr% = ESRQ then print " ESRQ <SRQ stuck on>"
4370      If iberr% = ETAB then print " ETAB <Table Overflow>"
4380 REM
4390      Print "ibcnt="; ibcnt% : RETURN
4400 REM
4410 REM  Call the ibonl function to disable the hardware and
4420 REM  software.
4430 REM
4440      ud% = 0 : v%= 0 : call ibonl(ud%, v%) : RETURN
```

QuickBASICプログラム例—NI-488.2 ルーチン

```
REM $include: 'qbdecl.bas'

declare sub gpiberr (msg$)

dim instruments% (31)
dim result% (30)
dim READING as STRING * 30

CLS

'  Your GPIB board must be the Controller-In-Charge to
'  perform the Find All Listeners protocol.

Call SendIFC (0)
If ibsta% and EERR then
     call gpiberr ("SendIFC Error")
     stop
End If

'  Create an array with all of the valid GPIB primary
'  addresses.  This array will be given to the Find All
'  Listeners protocol.

For k% = 0 to 30
     instruments% (k%) = k%
Next k%
instruments%(31) = NOADDR

'  Find all of the listeners on the bus.

Print "Finding all listeners on the bus..."

Call FindLstn (0, instruments%(), result%(), 31)
If ibsta% and EERR then
     call gpiberr("FindLstn Error")
     stop
End If
num.listeners% = ibcnt% - 1

Print "No. of instruments found = ", num.listeners%

'  Now send the *IDN? command to each of the devices that
'  you found.
'
```

```
'  The GPIB board is at address 0 by default.  Your GPIB
'  boarddoes not respond to *IDN?, so skip it.

   For k% = 1 to num.listeners%
        call Send(0, result%(k%), "*IDN?", NLend)
        if ibsta% and EERR then
            call gpiberr("Send Error")
            stop
        end if

        call Receive(0,result%(k%),Reading$, STOPend)
        if ibsta% and EERR then
            call gpiberr("Receive Error")
            stop
        end if

        pad% = result%(k%) and &HFF
        print "The instrument at address "; pad%; " is: ", _
              left$ (Reading$, IBCNT%)

        if left$(Reading$, 9)="FLUKE, 45" then
                fluke% = result%(k%)
                print "**** We found the Fluke 45 ****"
                goto found
        end if
   Next k%
   Print "Did not find the Fluke!"
   Stop

found:

'  Reset the Fluke.

   call DevClear (0, fluke%)
   If ibsta% and EERR then
        call gpiberr("DevClear Error")
        stop
   End If

   Call Send(0, fluke%, "*RST", NLend)
   If ibsta% and EERR then
        call gpiberr("Send *RST Error")
        stop
   End If
```

```
'  Set up for a test.  Allow the Fluke to assert SRQ when it
'  has a message to send.

   call Send(0, fluke%, "VAC; AUTO; TRIGGER 2; *SRE 16", NLend)
   If ibsta% and EERR then
        call gpiberr("Send Setup Error")
        stop
   End If

   sum = 0
   for m% = 1 to 10
        '  Trigger the Fluke.

           call Send(0, fluke%, "*TRG; VAL1?", NLend)
           if ibsta% and EERR then
                call gpiberr("Send Trigger Error")
                stop
           end if

        '  Wait for the Fluke to assert SRQ, meaning it is
        '  ready with the measurement.

           call WaitSRQ(0, SRQasserted%)
           if SRQasserted% = 0 then
                call gpiberr("WaitSRQ Error")
                stop
           end if

        '  Read its status byte.  Make sure that the MAV
        '  (Message Available) bit is set.

           call ReadStatusByte (0, fluke%, status%)
           if ibsta% and EERR then
                call gpiberr("ReadStatusByte Error")
                stop
           end if

           if (status% and &H010) <> &H010 then
                call gpiberr("Improper Status Byte")
                print "Status Byte: "; status%
                stop
           end if

        '  Read the measurement.
```

```
            call Receive (0, fluke%, Reading$, STOPend)
            if ibsta% and EERR then
                 call gpiberr("Receive Error")
                 stop
            end if

           Reading$ = left$(Reading$, IBCNT%)
           print "Reading: "; Reading$

           sum = sum + val (Reading$)
      Next m%

      Print "The average of the 10 readings is ", sum/10

'   Call the ibonl function to disable the hardware and software.

    Call ibonl(0,0)

End
'   This routine prints the result of the status variables.

sub gpiberr(msg$) static

locate 15,1
print msg$

    print "ibsta=&H"; hex$(ibsta%); "< ";
    if ibsta% and EERR then print " ERR";
    if ibsta% and TIMO then print " TIMO";
    if ibsta% and EEND then print " END";
    if ibsta% and SRQI then print " SRQI";
    if ibsta% and RQS  then print " RQS";
    if ibsta% and CMPL then print " CMPL";
    if ibsta% and LOK  then print " LOK";
    if ibsta% and RREM then print " REM";
    if ibsta% and CIC  then print " CIC";
    if ibsta% and AATN then print " ATN";
    if ibsta% and TACS then print " TACS";
    if ibsta% and LACS then print " LACS";
    if ibsta% and DTAS then print " DTAS";
    if ibsta% and DCAS then print " DCAS";
    print ">"

    print "iberr="; iberr%;
    if iberr% = EDVR then print " EDVR <DOS Error>"
    if iberr% = ECIC then print " ECIC <Not CIC>"
    if iberr% = ENOL then print " ENOL <No Listener>"
```

```
   if iberr% = EADR then print " EADR <Address error>"
   if iberr% = EARG then print " EARG <Invalid argument>"
   if iberr% = ESAC then print " ESAC <Not Sys Ctrlr>"
   if iberr% = EABO then print " EABO <Op. aborted>"
   if iberr% = ENEB then print " ENEB <No GPIB board>"
   if iberr% = EOIP then print " EOIP <Async I/O in prg>"
   if iberr% = ECAP then print " ECAP <No capability>"
   if iberr% = EFSO then print " EFSO <File sys. error>"
   if iberr% = EBUS then print " EBUS <Command error>"
   if iberr% = ESTB then print " ESTB <Status byte lost>"
   if iberr% = ESRQ then print " ESRQ <SRQ stuck on>"
   if iberr% = ETAB then print " ETAB <Table Overflow>"

   print "ibcnt="; ibcnt%

' Call the ibonl function to disable the hardware and
software.

   call ibonl(0, 0)

end sub
```

Microsoft BASIC プログラム例—NI-488.2ルーチン

```
   REM $include: 'mbdecl.bas'

   DECLARE SUB gpiberr(msg$)

   DIM instruments% (31)
   DIM result% (30)
   DIM READING as STRING * 30

   CLS

'  Your GPIB board must be the Controller-In-Charge to
'  perform the Find All Listeners protocol.

   Call SendIFC(0)
   If ibsta% and EERR then
        call gpiberr ("SendIFC Error")
        stop
   End If

'  Create an array with all of the valid GPIB primary
'  addresses.  This array will be given to the Find All
'  Listeners protocol.

   For k% = 0 to 30
        instruments% (k%) = k%
   Next k%
   instruments%(31) = NOADDR

'  Find all of the listeners on the bus.

   Print "Finding all listeners on the bus..."

   Call FindLstn (0, instruments%(), result%(), 31)
   If ibsta% and EERR then
        call gpiberr("FindLstn Error")
        stop
   End If
   num.listeners% = ibcnt% - 1

   Print "No. of instruments found = ", num.listeners%
```

```
'  Now send the *IDN? command to each of the devices that you
'  found.
'
'  The GPIB board is at address 0 by default.  Your GPIB
board
'  does not respond to *IDN?, so skip it.

   For k% = 1 to num.listeners%
        call Send(0, result%(k%), "*IDN?", NLend)
        if ibsta% and EERR then
            call gpiberr("Send Error")
            stop
        end if
        call Receive(0,result%(k%),Reading$, STOPend)
        if ibsta% and EERR then
            call gpiberr("Receive Error")
            stop
        end if

        pad% = result%(k%) and &HFF
        print "The instrument at address "; pad%; " is: ", left$
              (Reading$, IBCNT%)

        if left$(Reading$, 9)="FLUKE, 45" then
                fluke% = result%(k%)
                print "**** We found the Fluke 45 ****"
                goto found
        end if
   Next k%
   Print "Did not find the Fluke!"
   Stop

found:

'  Reset the Fluke.

   Call DevClear (0, fluke%)
   If ibsta% and EERR then
        call gpiberr("DevClear Error")
        stop
   End If

   Call Send(0, fluke%, "*RST", NLend)
   If ibsta% and EERR then
        call gpiberr("Send *RST Error")
        stop
   End If
```

```
'  Set up for a test.  Allow the Fluke to assert SRQ when it
'  has a message to send.

   Call Send(0, fluke%, "VAC; AUTO; TRIGGER 2; *SRE 16", NLend)
   If ibsta% and EERR then
        call gpiberr("Send Setup Error")
        stop
   End If

   sum = 0
   For m% = 1 to 10

        '  Trigger the Fluke.

        call Send(0, fluke%, "*TRG; VAL1?", NLend)
        if ibsta% and EERR then
             call gpiberr("Send Trigger Error")
             stop
        end if

        '  Wait for the Fluke to assert SRQ, meaning it is
        '  ready with the measurement.

        call WaitSRQ(0, SRQasserted%)
        if SRQasserted% = 0 then
             call gpiberr("WaitSRQ Error")
             stop
        end if

        '  Read its status byte.  Make sure that the MAV
        '  (Message Available) bit is set.

        call ReadStatusByte (0, fluke%, status%)
        if ibsta% and EERR then
             call gpiberr("ReadStatusByte Error")
             stop
        end if

        if (status% and &H010) <> &H010 then
             call gpiberr("Improper Status Byte")
             print "Status Byte: "; status%
             stop
        end if

        '  Read the measurement.
```

```
        call Receive (0, fluke%, Reading$, STOPend)
        if ibsta% and EERR then
             call gpiberr("Receive Error")
             stop
        end if

        Reading$ = left$(Reading$, IBCNT%)
        print "Reading: "; Reading$

        sum = sum + val (Reading$)
   Next m%

   Print "The average of the 10 readings is ", sum/10

'  Call the ibonl function to disable the hardware and
'  software.

   Call ibonl(0,0)
End


'  This routine prints the result of the status variables.

Sub gpiberr(msg$) static

   Locate 15,1
   Print msg$

   Print "ibsta=&H"; hex$(ibsta%); "< ";
   If ibsta% and EERR then print " ERR";
   If ibsta% and TIMO then print " TIMO";
   If ibsta% and EEND then print " END";
   If ibsta% and SRQI then print " SRQI";
   If ibsta% and RQS then print " RQS";
   If ibsta% and CMPL then print " CMPL";
   If ibsta% and LOK then print " LOK";
   If ibsta% and RREM then print " REM";
   If ibsta% and CIC then print " CIC";
   If ibsta% and AATN then print " ATN";
   If ibsta% and TACS then print " TACS";
   If ibsta% and LACS then print " LACS";
   If ibsta% and DTAS then print " DTAS";
   If ibsta% and DCAS then print " DCAS";
   Print ">"
```

```
   Print "iberr="; iberr%;
   If iberr% = EDVR then print " EDVR <DOS Error>"
   If iberr% = ECIC then print " ECIC <Not CIC>"
   If iberr% = ENOL then print " ENOL <No Listener>"
   If iberr% = EADR then print " EADR <Address error>"
   If iberr% = EARG then print " EARG <Invalid argument>"
   If iberr% = ESAC then print " ESAC <Not Sys Ctrlr>"
   If iberr% = EABO then print " EABO <Op. aborted>"
   If iberr% = ENEB then print " ENEB <No GPIB board>"
   If iberr% = EOIP then print " EOIP <Async I/O in prg>"
   If iberr% = ECAP then print " ECAP <No capability>"
   If iberr% = EFSO then print " EFSO <File sys. error>"
   If iberr% = EBUS then print " EBUS <Command error>"
   If iberr% = ESTB then print " ESTB <Status byte lost>"
   If iberr% = ESRQ then print " ESRQ <SRQ stuck on>"
   If iberr% = ETAB then print " ETAB <Table Overflow>"

   Print "ibcnt="; ibcnt%

'  Call the ibonl function to disable the hardware and
'  software.

   CALL IBONL(0, 0)

End Sub
```

Cプログラムの例—NI-488.2 ルーチン

```
/*
 * Link this program with mcib.obj
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "decl.h"

/* Position of the Message Available bit. */

#define  MAVbit   0x10

   char          buffer[101];
   int           loop, m, num_listeners, SRQasserted;
   double        sum;
   unsigned int  instruments[32], result[31], fluke,
     statusByte;

void gpiberr(char *msg);            /* gpib error function */


main() {

   system("cls");

/* Your GPIB board must be the Controller-In-Charge to
 * perform the Find All Listeners protocol.
 */
   SendIFC(0);
   if (ibsta & ERR) {
        gpiberr ("SendIFC Error");
        exit(1);
   }


/* Create an array with all of the valid GPIB primary
 * addresses.  This array will be given to the Find All
 * Listeners protocol.
 */
```

```
   for (loop = 0; loop <= 30; loop++) {
      instruments[loop] = loop;
   }
   instruments[31] = NOADDR;  /* Mark the end of the array.*/


/* Find all of the listeners on the bus.
 */
   printf("Finding all listeners on the bus...\n");

   FindLstn(0, instruments, result, 31);
   if (ibsta & ERR) {
        gpiberr("FindLstn Error");
        exit(1);
   }

   num_listeners = ibcnt - 1;

   printf("Number of instruments found = %d\n", num_listeners);

/* Now send the *IDN? command to each of the devices that you
 * found.
 *
 * The GPIB board is at address 0 by default.  Your GPIB
 * board does not respond to *IDN?, so skip it.
 */
   for (loop = 1; loop <= num_listeners; loop++) {
      Send(0, result[loop], "*IDN?", 5L, NLend);
      if (ibsta & ERR) {
           gpiberr("Send Error");
           exit(1);
      }

      Receive(0, result[loop], buffer, 10L, STOPend);
      if (ibsta & ERR) {
           gpiberr("Receive Error");
           exit(1);
      }

      buffer[ibcnt] = '\0';
      printf("The instrument at address %d is a %s\n",
        GetPAD(result[loop]),
               buffer);
```

```
      if (strncmp(buffer, "FLUKE, 45", 9) == 0) {
         fluke = result[loop];
         printf("**** We found the Fluke ****\n");
         break;
      }
   }

   if (loop > num_listeners) {
        printf("Did not find the Fluke!\n");
        exit(1);
   }

/* Reset the Fluke.
 */
   DevClear(0, fluke);
   if (ibsta & ERR) {
        gpiberr("DevClear Error");
        exit(1);
   }

   Send(0, fluke, "*RST", 4L, NLend);
   if (ibsta & ERR) {
        gpiberr("Send *RST Error");
        exit(1);
   }

/* Set up for a test.  Allow the Fluke to assert SRQ when it
 * has a message to send.
 */
   Send(0, fluke, "VAC; AUTO; TRIGGER 2; *SRE 16", 29L, NLend);
   if (ibsta & ERR) {
      gpiberr("Send Setup Error");
      exit(1);
   }

   sum = 0.0;
   for (m=0; m < 10 ; m++) {

        /*  Trigger the Fluke.
         */
         Send(0, fluke, "*TRG; VAL1?", 11L, NLend);
         if (ibsta & ERR) {
            gpiberr("Send Trigger Error");
            exit(1);
         }
```

```
        /*  Wait for the Fluke to assert SRQ, meaning it is
         *  ready with the measurement.
         */
        WaitSRQ(0, &SRQasserted);
        if (!SRQasserted) {
            printf("SRQ is not asserted. The Fluke is not ");
            printf("ready.\n");
            exit(1);
        }

        /*  Read its status byte.  Be sure that the MAV
         *  (Message Available) bit is set.
         */
         ReadStatusByte(0, fluke, &statusByte);
         if (ibsta & ERR) {
            gpiberr("ReadStatusByte Error");
            exit(1);
         }

         if (!(statusByte & MAVbit)) {
             gpiberr("Improper Status Byte");
             printf(" Status Byte = 0x%x\n", statusByte);
             exit(1);
         }

        /*  Read the measurement.
         */
         Receive(0, fluke, buffer, 10L, STOPend);
         if (ibsta & ERR) {
            gpiberr("Receive Error");
            exit(1);
         }

         buffer[ibcnt] = '\0';

         printf("Reading :  %s\n", buffer);
         sum = sum + atof(buffer);
   }

   printf("   The average of the 10 readings is :  ");
   printf("   %f\n", sum/10);

/* Call the ibonl function to disable the hardware and
/* software.
 */
```

```
    ibonl (0,0);

}


void gpiberr(char *msg) {

    printf ("%s\n", msg);

    printf ( "ibsta=&H%x ", ibsta, "< ");
    if (ibsta & ERR ) printf (" ERR");
    if (ibsta & TIMO) printf (" TIMO");
    if (ibsta & END ) printf (" END");
    if (ibsta & SRQI) printf (" SRQI");
    if (ibsta & RQS ) printf (" RQS");
    if (ibsta & CMPL) printf (" CMPL");
    if (ibsta & LOK ) printf (" LOK");
    if (ibsta & REM ) printf (" REM");
    if (ibsta & CIC ) printf (" CIC");
    if (ibsta & ATN ) printf (" ATN");
    if (ibsta & TACS) printf (" TACS");
    if (ibsta & LACS) printf (" LACS");
    if (ibsta & DTAS) printf (" DTAS");
    if (ibsta & DCAS) printf (" DCAS");
    printf (">\n");

    printf ("iberr= %d", iberr);
    if (iberr == EDVR) printf (" EDVR <DOS Error>\n");
    if (iberr == ECIC) printf (" ECIC <Not CIC>\n");
    if (iberr == ENOL) printf (" ENOL <No Listener>\n");
    if (iberr == EADR) printf (" EADR <Address error>\n");
    if (iberr == EARG) printf (" EARG <Invalid argument>\n");
    if (iberr == ESAC) printf (" ESAC <Not Sys Ctrlr>\n");
    if (iberr == EABO) printf (" EABO <Op. aborted>\n");
    if (iberr == ENEB) printf (" ENEB <No GPIB board>\n");
    if (iberr == EOIP) printf (" EOIP <Async I/O in prg>\n");
    if (iberr == ECAP) printf (" ECAP <No capability>\n");
    if (iberr == EFSO) printf (" EFSO <File sys. error>\n");
    if (iberr == EBUS) printf (" EBUS <Command error>\n");
    if (iberr == ESTB) printf (" ESTB <Status byte lost>\n");
    if (iberr == ESRQ) printf (" ESRQ <SRQ stuck on>\n");
    if (iberr == ETAB) printf (" ETAB <Table Overflow>\n");

    printf ("ibcnt= %d\n", ibcntl);
    printf ("\n");
```

```
/* Call the ibonl function to disable the hardware and
 * software.
*/

   ibonl (0,0);

   }
```

第5章
NI-488 ソフトウエア特性と関数

この章ではMS-DOSドライバにより使用出来るNI-488関数の重要な特性のうちで各言語に共通なものを説明し、次にこれらの関数の一つ一つについてプログラムの例を付して詳細に記述しています。関数はアルファベット順に配列してあります。

BASICA, QuickBASIC, BASIC, あるいはC以外の言語をご注文のユーザーには別のマニュアルをお送りします。

一般的なプログラミング情報

以下に示す手段や動作は全てのプログラミング言語に共通して用いられます。

- ステータスワード(ibsta)
- エラーコード(iberr)
- カウント変数(ibcnt と ibcntl)
- 読取・書込の終了
- デバイス関数
- 自動シリアルポーリング

NI-488.2 MS-DOS ドライバの能力を100パーセント活用するには上記の題目を完全に理解していることが必要です。これらのうち、ステータスワード(ibsta)、エラー変数 (iberr)、及びカウント変数 (ibcnt と ibcntl)については第3章の説明を参照してください。これらの変数は関数の呼び出し毎に更新され、今アクセスしたばかりのデバイスなりボードの状態を示します。

デバイス関数

ある関数の中のユニットデスクリプタがインターフェースボードでなく、あるデバイスを指定している場合にその関数はデバイス関数であると言われます。(ボードとデバイスについては第３章の説明を参照してください。) デバイス関数の行う動作のうち幾つかは全てのデバイス関数に共通したものなので、これらの動作は十分に理解しておく必要があります。

GPIBインターフェースボードを１枚しか使用していない単一ボード構成において、あるプログラムが最初のデバイス関数を実行する場合、ソフトウエアは Interface Clear (IFC)という１ラインのインターフェースメッセージを送ってIEEE-488バスを初期化します。またIEEE-488バスの Remote Enable (REN)ラインもアサートされ、マルチラインメッセージである Local Lockout (LLO) が IEEE-488 バス上の全てのデバイスに送られてそれらをロックアウトの状態にします。更に、デバイスはいったんリスナとしてアドレスされ、ある関数が実行される前にそのアドレシングを解除される場合もありますが、これはデバイスをキーボードからプログラムできるリモートプログラムモードにたもつためのステップです。

ボードを使用するための構成でもプロセスは上記の場合と同じですが、IEEE-488バス上の最初のデバイスがあるデバイス関数によりアクセスされるときに複数のIEEE-488バスの各々がそれぞれ関連したGPIBインターフェースボードによって初期化される点が異なっています。

以上の説明はGPIBボードがIEEE-488バスのシステムコントローラであると言う通常の構成を前提としています。GPIBインターフェースボードはシステムコントローラである限りコントローラインチャージ(CIC)になることが出来ます。もしもGPIBボードがCICでないとか、CICになることが出来ないと、それはデバイス関数を実行することが出来ません。この様な状況でIEEE-488バスによりデバイス関数を呼び出しますと ECIC エラーコードが返ってきます。

ソフトウエアは構成ユーティリティ `IBCONF`か構成関数 `ibconfig` を使って現在の CIC にコントロールを要求するよう構成することも出来ます。CIC 要求構成のオプションが構成され、デバイス関数が呼び出された時にGPIBインターフェースボードがCICでなかった場

合、ボードはSRQラインをアサートし、CICからのシリアルポーリングに対してステータスレスポンスバイト(hex 42) を引き渡して現在のCICにサービスを要求します。

ソフトウエアはそのときのボードのタイムアウト値にしたがってGPIBインターフェースボードにコントロールが引き渡されるのを待ちます。ボードがタイムアウト期間中にCICになれたかった場合はECICエラーコードが返されます。例えば、システムコントローラである筈のGPIBインターフェースボードが実はソフトウエアのインストレーション時にシステムコントロールとして構成されていなかった場合などに上記のエラーが起こります。

自動シリアルポーリング

自動シリアルポーリングはこのソフトウエアの特長の一つであり、ユーザーがSRQやデバイスのステータスバイトに出くわした時の面倒な処理をしないで済ませるものです。自動シリアルポーリング（あるいはオートポーリング）は構成ユーティリティの IBCONF か構成関数の ibconfig を使って可能化することが出来ます。オートポーリングが可能化されると、ソフトウエアはSRQがアサートされている時自動的にシリアルポーリングを行います。

オートポーリング手続きの一部として、ソフトウエアはシリアルポーリングの個々の肯定的応答（つまりデバイスのステータスバイトにRQSかhex 40 ビットがセットされた応答）を各々のデバイスに関連した待ち行列として記憶します。いくつかのデバイスが複数の肯定的ステータスバイトを次々と送り出すことがあるので待ち行列が必要になるわけです。あるデバイスからの肯定的な応答が受信されると、そのステータスワード（ibsta)のRQSビットがセットされます。ポーリングはSRQのアサートが解除されるかエラー条件が検出されるまで続きます。

ソフトウエアがサービス要求中のデバイスを見い出すことが出来ない場合(つまり解っているデバイスでポーリングに肯定的応答をしたものが見当たらない場合）、あるいは計測器やケーブルの不良のためSRQがそのまま動かなくなってしまった場合はGPIBシステムエラーの状態になっており、そのためデバイスのステータスワード中のRQSビットを正しく評価できなくなります。この様な場合にRQSビッ

トが待機マスクに含まれた形式で ibwait を呼び出しますとエラーコードのESRQが報告されます。このエラーが自然に直ってしまった場合には、もう一度マスクにRQSビットをふくめて ibwait を呼び出して見てもESRQエラーが報告されません。RQS待機中のESRQエラーは問題ではありますが、それ自身がGPIBの諸動作に好ましくない影響を与えることはありません。

シリアルポーリング関数の ibrsp を呼び出して、それまでに自動シリアルポーリングに対する応答が複数あったとすると、 ibrsp が最初に戻してくるのは一番早く待ち行列に入った応答です。それに続く応答も first-in-first-out (FIFO), つまり早く入ったものが早く出てくる形で戻されてきます。ibrsp が呼び出された時にステータスワードのRQSビットがセットされていなかった場合は、ibrsp がシリアルポーリングを行って返ってきた応答を取り敢えず戻してきます。

使用するアプリケーションプログラムがRQSの存在を知っている必要がある場合は、ステータスワード中のRQSビットを調べ、ibrsp を呼び出し、ステータスバイトが現われる度にそれを調べるようにせねばなりません。デバイスのシリアルポーリング応答待ち行列は、ibrsp を使って空にすることを怠っていると、古いステータスバイトで詰まってしまうことがあります。待ち行列が一杯になったのでステータスバイトを捨ててしまったという報告が必要になると、ibrsp だけがエラー条件ESTBを戻してきます。もしアプリケーションプログラムがSRQやステータスワードのことを知る必要がない場合は、自動ポーリングが起こってもそれに注意を払う必要はありません。

注)　デバイスのステータスワードのRQSビットがibrsp呼び出しの後まだセットされている場合は、応答バイト待ち行列には少なくともあと一つの応答バイトが残っています。RQSがクリアされるまではibrspを呼び出して記憶された応答バイトを取りまとめ、待ち行列の溢れがないようにする必要があります。

両立性

自動ポーリングは、幾つかの言語インターフェースにある "ON SRQ" と言う機能と両立しないことがあります。このような場合、ユーザーとドライバの両方でSRQをコントロールするわけにはいかず、いずれか一方がSRQを管理しなければなりません。ユーザーが "ON SRQ" を使用したい時は自動ポーリングは不可能化してください。[NI-488.2 ドライバの "ON SRQ" については第3章を参照してください。]

注) マウスのドライバには "ON SRQ" と使用すると問題が起こるものがあります。そのようなマウスのドライバは "ON SRQ" の使用に先立って不可能化しておいてください。

ボード関数もまた自動ポーリングと両立しません。(ボード関数については第3章を参照してください。) ボード関数の呼び出しのあとで自動ポーリングが起こると呼び出しが無効化される場合があります。例えば、あるデバイスにアドレスするために ibcmdを呼び出したあとでシリアルポーリングのため自動ポーリングで他のデバイスをアドレスすると、最初のデバイスと通信するためのアドレシングが無効になってしまいます。これを防ぐためにドライバはボード呼び出しの度に自動ポーリングを不可能化し、アプリケーションプログラムがデバイス呼び出しを行うと自動ポーリングを再開します。(正確に言うと、自動ポーリングは次のデバイス呼び出しの最後になって再開されます。

内部におけるドライバの動作

自動ポーリング中は、IEEE-488バス上でSRQがアサートされる度に全てのオンラインデバイス (ibfind あるいは ibdev によってオープンされた全てのデバイス) でGPIBインターフェースボードに接続され、サービス要求を検出したデバイスはドライバによって以下に示す事象が起こるまでシリアルポーリングされます。

- SRQのアサート解除まで。
- この前に起こった肯定的応答の後も全てのオンラインデバイスがポーリングされ、SRQもアサートされたままの場合。こうな

ると、「SRQがつかえている」状態がドライバの中に生まれます。 ibwaitがRQSを待っているあいだにこの状態になると、ESRQエラーが ibwait呼び出しに対して報告され「SRQがつかえている」状態は終了させられます。その後、「SRQがつかえている」状態が再発しますと、今回はibwaitでRQSの待機がなされるまではポーリングは起こらないようになっています。もしibwaitによるSRQ待機になると、「SRQがつかえている」状態は終了となり、新たにシリアルポーリングが行われるようになります。「SRQがつかえている」状態はSRQのアサートが解除されたときは直ちに終了します。

ハードウエア割込み

GPIBインターフェースボードの割込みがIBCONF なり ibconfig関数で不可能化されると、ドライバはドライバが呼び出される時しかSRQを検出しなくなります。また、自動ポーリングは以下に示す事象のときのみ行われます。

- デバイスの ibwait がRQSを待機している間
- デバイス関数が完了した直後であってアプリケーションプログラムに帰ろうとしている時

上記の場合以外に、ハードウエア割込が可能化されている時もドライバは、たとえ関数の動作を行っていなくても、SRQI割込に応答して自動ポーリングを行います。但し、ここに述べた自動ポーリングが起こりうる時期といえども、次のような条件の何れかが存在する場合はドライバは自動ポーリングを行いません。

- ボード呼び出しあるいはNI-488.2呼び出しの直後。この場合は、自動ポーリングはそれに続くデバイス呼び出しの後で再開されます。
- GPIBの入出力、特に非同期入出力が進行中には全てのドライバのオペレーションが終わるまでは（言い換えれば全ての非同期入出力が終わるまでは）自動ポーリングは始まりません。

- 「SRQがつかえている」状態が存在している間は自動ポーリングは起こりません。

- IBCONF あるいは ibconfig によって自動ポーリングが不可能化されている場合。(但し、 ibwait によるRQS待機中はこのibwait 呼び出しの対象であるデバイスはシリアルポーリングされます。この呼び出し中にSRQがアサートされ、否定的応答が該デバイスから受信されると、「SRQがつかえている」との答が返ってきます。

BASICA/QuickBASIC/BASIC/Cによる NI-488 の入出力関数呼び出し

最もよく使用される入出力用関数は ibrd と ibwrt です。BASICAではこれらの関数は最長で255バイトの文字列による読取・書込を行います。

また、整数入出力呼び出し (ibrdi と ibwrti)はデータストリングが255バイトより長い場合、あるいはユーザーがデータの演算処理を行いたいがibrd と ibwrtの文字バイトを整数フォーマットに直し、後にまた元に戻すというようなオーバーヘッドは避けたいというような場合に使用されます。ibrdi と ibwrti は255バイト以上が転送できるように整数配列の形でデータを引き渡します。内部では、ibwrti関数は各整数をGPIBに下位バイト—上位バイトの順で送ります。 ibrdi 関数は一連のデータバイトをGPIBから読取り、それらを整数配列に下位バイト—上位バイトの順で記憶します。

ibrdi と ibwrti のほかに、非同期関数のibrdia と ibwrtia が非同期で整数の読取・書込を行います。

ibdev, ibfind 及び ibtrap を除く全ての関数呼び出しにおいて、最初の引き数は整数変数のudです。これは呼び出しのフォーマットを示す一般ユニット記述子としての役割を果たします。関数呼び出しにおけるud はコマンドの対象としてのボードあるいはデバイスを指します。どの型のudを使用するかを決めるには、この章と第3章の*IBFIND*の説明を参照してください。

この章で説明される関数はアルファベット順に並べられています。表５－１、５－２および５－３は各々BASICA, QuickBASIC(バージョン4.0以降) /BASIC(バージョン7.0)、及びCにおいて使用されるNI-488関数のリストを示します。

表５－１ BASICA で使用されるNI-488関数

呼び出しシンタクス	内容
ibbna (ud%,bname$)	デバイスのアクセスボードを変更する
ibcac (ud%,v%)	アクティブコントローラとなる
ibclr (ud%)	指定されたデバイスをクリアする
ibcmd (ud%,cmd$)	ストリングからコマンドを送る
ibcmda (ud%,cmd$)	ストリングからコマンドを非同期で送る
ibconfig (ud%,option%,value%)	ソフトウエアを構成する
ibdev (board.index%,pad%, sad%,tmo%,eot%,eos%,ud%)	未使用で名の解っていないデバイスを開いて初期化する
ibdma (ud%,v%)	DMAの可能化／不可能化
ibeos (ud%,v%)	EOSモードを変更／不可能化する
ibeot (ud%,v%)	ENDメッセージを可能化／不可能化する
ibevent (ud%, event%)	次の事象を返す。
ibfind (udname$,ud%)	デバイスを開いてユニット記述子を戻す

(次ページに続く)

表5－1 BASICA で使用されるNI-488関数 (前ページより続く)

呼び出しシンタクス	内容
ibgts (ud%,v%)	アクティブコントローラ状態から待機状態に移る
ibist (ud%,v%)	istをセット／クリアする
iblines (board.index%,lines%)	GPIBラインのステータスを返す
ibln (ud%,pad%,sad%,listen%)	バス上のデバイスの存在をチェックする
ibloc (ud%)	ローカルモードに移る
ibonl (ud%,v%)	デバイスをオンライン／オフラインにする
ibpad (ud%,v%)	1次アドレスを変更する
ibpct (ud%)	コントロールを引き渡す
ibppc (ud%,v%)	パラレルポーリング構成
ibrd (ud%,rd$)	データをストリングに読み込む
ibrda (ud%,rd$)	データをストリングに非同期に読み込む
ibrdf (ud%,flname$)	データをファイルに読み込む
ibrdi (ud%,iarr%(0),cnt%)	データを整数配列に読み込む
ibrdia (ud%,iarr%(0),cnt%)	データを整数配列に非同期に読み込む
ibrpp (ud%,ppr%)	パラレルポーリングを行う
ibrsc (ud%,v%)	システムコントロールを要求／解除する
ibrsp (ud%,spr%)	シリアルポーリングを行いステータスバイトを返す

(次ページに続く)

表5－1 BASICA で使用されるNI-488関数 (前ページより続く)

呼び出しシンタクス	内容
ibrsv (ud%,v%)	サービスを要求する
ibsad (ud%,v%)	2次アドレスを変更する
ibsic (ud%)	IFCを送る
ibsre (ud%,v%)	RENをセット／クリアする
ibstop (ud%)	非同期オペレーションを中止する
ibtmo (ud%,v%)	時間制限を変更／不可能化する
ibtrap (mask%,v%)	アプリケーションズモニター [Applications Monitor] を構成する
ibtrg (ud%)	選択したデバイスをトリガする
ibwait (ud%,mask%)	選択した事象の生起を待つ
ibwrt (ud%,wrt$)	ストリングからのデータを書き込む
ibwrta (ud%,wrt$)	ストリングからのデータを非同期に書き込む
ibwrtf (ud%,flname$)	ファイルからのデータを書き込む
ibwrti (ud%,iarr%(0),cnt%)	整数配列からのデータを書き込む
ibwrtia (ud%,iarr%(0),cnt%)	整数配列からのデータを非同期に書き込む

表5－2 QuickBASIC/BASIC で使用されるNI-488関数

呼び出しシンタクス	内容
ibbna (ud%,bname$)	デバイスのアクセスボードを変更する
ibcac (ud%,v%)	アクティブコントローラとなる
ibclr (ud%)	指定されたデバイスをクリアする
ibcmd (ud%,cmd$)	ストリングからコマンドを送る
ibcmda (ud%,cmd$)	ストリングからコマンドを非同期に送る
ibconfig (ud%,option%,value%)	ソフトウエアを構成する
ibdev (board.index%,pad%, sad%,tmo%,eot%,eos%,ud%)	未使用で名の知られていないデバイスをオープンし初期化する
ibdma (ud%,v%)	DMAを可能化／不可能化する
ibeos (ud%,v%)	EOSモードを変更／不可能化する
ibeot (ud%,v%)	ENDメッセージを可能化／不可能化する
ibevent (ud%, event%)	次の事象を返す。
ibfind (brdname$,ud%)	デバイスを開いてユニット記述子を返す
ibgts (ud%,v%)	アクティブコントローラ状態から待機状態に移る
ibist (ud%,v%)	istをセット／クリアする
iblines (board.index%,lines%)	GPIBラインのステータスを返す

(次ページに続く)

表５－２ QuickBASIC/BASIC で使用されるNI-488関数
(前ページより続く)

呼び出しシンタクス	内容
ibln (ud%,pad%,sad%,listen%)	バス上のデバイスの存在をチェックする
ibloc (ud%)	ローカルモードに移る
ibonl (ud%,v%)	デバイスをオンライン／オフラインにする
ibpad (ud%,v%)	１次アドレスを変更する
ibpct (ud%)	コントロールを引き渡す
ibppc (ud%,v%)	パラレルポーリング構成
ibrd (ud%,rd$)	データをストリングに読み込む
ibrda (ud%,rd$)	データをストリングに非同期に読み込む
ibrdf (ud%,flname$)	データをファイルに読み込む
ibrdi (ud%,iarr%(),cnt%)	データを整数配列に読み込む
ibrdia (ud%,iarr%(),cnt%)	データを整数配列に非同期に読み込む
ibrpp (ud%,ppr%)	パラレルポーリングを行う
ibrsc (ud%,v%)	システムコントロールを要求／解除する
ibrsp (ud%,spr%)	シリアルポーリングを行いステータスバイトを返す
ibrsv (ud%,v%)	サービスを要求する
ibsad (ud%,v%)	２次アドレスを変更
ibsic (ud%)	IFCを送る

(次ページに続く)

表5－2 QuickBASIC/BASIC で使用されるNI-488関数
(前ページより続く)

呼び出しシンタクス	内容
ibsre (ud%,v%)	RENをセット/クリア
ibstop (ud%)	非同期オペレーションを中止
ibtmo (ud%,v%)	時間制限を変更／不可能化
ibtrap (mask%,v%)	アプリケーションズモニターを構成
ibtrg (ud%)	選択されたデバイスをトリガ
ibwait (ud%,mask%)	選択された事象の生起を待つ
ibwrt (ud%,wrt$)	ストリングからデータを書き込む
ibwrta (ud%,wrt$)	ストリングからデータを非同期に書き込む
ibwrtf (ud%,flname$)	ファイルからデータを書き込む
ibwrti (ud%,iarr%(),cnt%)	整数配列からデータを書き込む
ibwrtia (ud%,iarr%(),cnt%)	整数配列からデータを非同期に書き込む

表5－3 C言語で使用するNI-488関数

呼び出しシンタクス	内容
ibbna (ud,bname)	デバイスのアクセスボードを変更
ibcac (ud,v)	アクティブコントローラとなる

(次ページに続く)

表5－3 C言語で使用するNI-488関数 (前ページより続く)

呼び出しシンタクス	内容
ibclr (ud)	指定されたデバイスをクリア
ibcmd (ud,cmd,cnt)	ストリングからコマンドを送る
ibcmda (ud,cmd,cnt)	ストリングからコマンドを非同期に送る
ibconfig (ud,option,value)	ソフトウエアを構成
ud = ibdev (bd_index, pad,sad,tmo,eot,eos)	未使用で名の知られていないデバイスを開いて初期化
ibdma (ud,v)	DMAを可能化／不可能化
ibeos (ud,v)	EOSモード（読取）を変更／不可能化
ibeot (ud,v)	ENDメッセージを可能化／不可能化
ibevent (ud, &event)	次の事象を返す。
ud = ibfind (udname)	デバイスをオープンしてユニット記述子を返す
ibgts (ud,v)	アクティブコントローラ状態から待機状態に移る
ibist (ud,v)	パラレルポーリングのためにistをセット／クリア
iblines (board,lines)	GPIBラインのステータスを返す
ibln (pad,sad,listen)	バス上のデバイスの存在をチェック
ibloc (ud)	ローカルモードに移る

(次ページに続く)

表5－3 C言語で使用するNI-488関数 (前ページより続く)

呼び出しシンタクス	内容
ibonl (ud,v)	デバイスあるいはボードをオンライン/オフラインにする
ibpad (ud,v)	1次アドレスを変更
ibpct (ud)	コントロールを引き渡す
ibppc (ud,v)	パラレルポーリング用構成
ibrd (ud,rd,cnt)	ストリングにデータを読み込む
ibrda (ud,rd,cnt)	ストリングにデータを非同期に読み込む
ibrdf (ud,flname)	ファイルにデータを読み込む
ibrpp (ud,&ppr)	パラレルポーリングを行う
ibrsc (ud,v)	システムコントロールを要求／解除
ibrsp (ud,&spr)	シリアルポーリングにより得られたステータスバイトを返す
ibrsv (ud,v)	サービスを要求し、シリアルポーリングで得られたステータスバイトをセット／変更
ibsad (ud,v)	2次アドレスを変更／不可能化
ibsic (ud)	100 msecの間IFC を送る
ibsre (ud,v)	RENをセット／クリア
ibsrq (func)	SRQの「割込ルーチン」を登録する
ibstop (ud)	同期オペレーションを中止

(次ページに続く)

表５－３ C言語で使用するNI-488関数 (前ページより続く)

呼び出しシンタクス	内容
ibtmo (ud,v)	時間制限を変更／不可能化
ibtrap(mask,mode)	アプリケーションズモニターを構成
ibtrg (ud)	選択されたデバイスをトリガ
ibwait (ud,mask)	選択された事象を待つ
ibwrt (ud,wrt,cnt)	ストリングからデータを書き込む
ibwrta (ud,wrt,cnt)	ストリングからデータを非同期に書き込む
ibwrtf (ud,flname)	ファイルからデータを書き込む

NI-488 IL 関数

QuickBASIC (バージョン4.0以降) と BASIC (バージョン 7.0及び以降) はサブルーチンだけでなく関数の使用もサポートするので、言語インターフェースはNI-488の関数呼び出しも出来るように拡張されています。関数呼び出しについて幾つか重要な点を以下に示します。

- 全てのNI-488サブルーチン(ibrd, ibwrt,...)はCALLステートメントにより呼び出すことが出来ます。

- 新しく導入された関数の名は既存のサブルーチン名とまったく同じです。但し、各々の名の２番目の字はbでなく1となっています。例えば、サブルーチンの ibsic は関数の ilsic として使用することも出来ます。

- QuickBASIC のバージョン 4.x および BASIC のバージョン 7.x で書いたプログラムの中では、GPIBのサブルーチンと関数は自由に混合して使用することができます。

- インクルードファイルの QBDECL.BAS (BASICの場合は MBDECL.BAS)の中にはサブルーチンと関数の宣言の完全なリストが入っており、コンパイルのときに型のチェックを容易にするためパラメータリストの指定も揃っています。NI-488関数の呼び出しを使用するアプリケーションプログラムはいずれもこのファイルをインクルードしている必要があります。

- 大体において関数はサブルーチンと同じように動作しますが、以下に示すような点で相違を示します。このマニュアル中のサブルーチンの説明はシンタックスに限った情報を除けば関数の説明としても通用するということができます。

現行のサブルーチンと新しく導入された関数の間には次の相違があります。

- ilfind は指定されたボードあるいはデバイスに関連した記述子を返します。その後同じデバイスにアクセスする関数にはいつもこの値を使用してください。通常の用法を以下に示します。

```
ud% = ilfind ("GPIB0")
```

- ildev は未使用で名の解っていないデバイスを開き初期化します。通常の用法を次に示します。

```
ud% = ildev (0, 6, &H67, 13, 7, 0)
```

- ilcmd, ilcmda, ilrd, ilrda, ilwrt, 及び ilwrta は転送するバイトの数を指定する第3のパラメータを必要とします。関数シンタクスは以下の通りです。

```
sta% = ilcmd (ud%, cmd$, cnt%)
sta% = ilcmda (ud%, cmd$, cnt%)
sta% = ilrd (ud%, rd$, cnt%)
sta% = ilrda (ud%, rd$, cnt%)
sta% = ilwrt (ud%, wrt$, cnt%)
sta% = ilwrta (ud%, wrt$, cnt%)
```

- ilfind と ildevを除く全ての関数はibstaの値を返します。これにより次の構造が可能となります。

```
IF ilrd (ud%, rd%, cnt%) AND EERR
   THEN CALL GPIBERROR
```

NI-488のプログラムを書く

次にNI-488の関数の使用法を実例により説明します。ここではQuickBASICによるプログラムの作成をステップ毎に説明していきます。この例ではデバイス関数を使用していますが、それはデバイス関数が最も簡単な関数であり、ほとんどのアプリケーションに応用可能のものだからです。このプログラムではディジタルマルチメータ(メータ)を構成し、電圧測定の結果を10回さかのぼって読み取り、その平均値を計算します。この例では、エラーの検出と報告はApplications Monitor [アプリケーションズモニター] によって行うものと仮定しています。(これについては第7章のAPPMONの説明を参照してください。)

このプログラム例の次には、それに相当するプログラムで、ボード関数を使用した例を示します。また、この章の終には、BASIC, BASICA, 及びCの各言語により書かれたデバイス関数プログラムを例示します。

ステップ1—システムの初期化

QuickBASICのプログラムを書く第1ステップは皆様が購入されたディストリビューションディスクにあるファイルからNI-488関数の定義をロードすることで始まります。

```
REM $INCLUDE: 'QBDECL.BAS'
```

最初のNI-488関数呼び出しはデバイスを開く ibfind あるいは ibdev の呼び出しです。

```
CALL ibdev (0, 1, 0, 12, 1, 0, dmm%)
```

ibdevサブルーチンの入力引き数はボードインデックス番号(GPIB0 の場合は0)、デバイスの1次GPIBアドレス(1)、デバイスの2次GPIBアドレス(無しの場合は0)、ソフトウエアがデバイスと交信するときのタイムアウト、(3秒の場合は12)、デバイスの書込の最後のバイトと一緒にENDメッセージを送ること(可能化の場合は1)、及びEOS検出モード(不可能化の場合は0)です。ibdevの呼び出しにおいては、ソフトウエアは自動的に IFC[インターフェースクリア] メッセージを送ってGPIBを初期化し、デバイスをコンピュータのキーボードからプログラムするリモートモードにします。

ステップ2—デバイスをクリアする

デバイスを使用しようとするアプリケーションプログラムにあわせて構成するに先立って、デバイスをクリアすることをおすすめします。デバイスをクリアしますと、デバイスの内部機能が既に知られた一定の状態にリセットされます。

```
CALL ibclr(dmm%)
```

ステップ3—デバイスを構成する

ibfind と ibclr の使用により、計測器はコマンドを受信する状態になりました。マルチメータを構成するには、ibwrt 関数を使ってこのデバイスに特有のコマンドを送ります。ibwrt の最初の引き数は ibfind 関数が変数 dmm% で戻してきたメータのユニット記述子です。

```
CALL ibwrt(dmm%, "F1R0S2T4")
```

ibwrt関数はF1R0S2T4と言うバイトをメータに送ります。このようにして計測器に送られるバイトは通常計測器毎に違います。計測器に送るべきコマンドバイトについてはその計測器のユーザーマニュアルを見てください。いまの例のバイトはメータの電圧のタイプ(F1)、電圧の範囲(R0)、更新のスピード(S2)、及びトリガモード(T4)を構成するものです。

ステップ4—デバイスをトリガする

デバイスは読み取った測定値を送り出すに先立ってトリガを待機するようにセットされました。そこで、このステップでは、測定値を読みだす前にトリガコマンドを送る必要があります。

```
CALL ibtrg(dmm%)
```

ステップ5—測定を行う

構成が終わったメータは測定を行ってその結果を計器盤に示すことができます。測定結果をGPIBを通してよみとるには ibrd関数を使用します。 ibrd関数の呼び出しにおいても最初の引き数はメータのユニット記述子です。この関数の動作が完了すると、変数 rd$ が測定結果を示すこととなります。メータはGPIBを介して13バイトのデータを送ります。測定値用のスペースの割り当ては ibrd 関数の動作の前にDIM宣言を使用して行います。

```
DIM RD AS STRING * 13
CALL ibrd(dmm%, rd$)
```

変数 rd$ にはメータが測定した電圧値を示すASCIIストリングが入っています。

ステップ6—得られたデータの解析と提示

データがGPIBから得られると、プログラム言語の解析とプレゼンテーション機能の全てがデータ処理に使用できるようになります。メータからただ一つの電圧値を読み取るのでなく、10測定値を読み取ってその平均を計算することもできます。この場合、ステップ5の最後の2ラインのコードの代わりに次のコードが使用されます。

```
DIM RD AS STRING * 13
SUM = 0

FOR X = 1 to 10
```

```
    CALL ibtrg(dmm%)
    CALL ibrd(dmm%, rd$)
    SUM = SUM + VAL(rd$)
NEXT X

PRINT "The average voltage is "; SUM/10
```

作成したアプリケーションプログラムの全体

このようにして書かれたプログラムの全体を以下に示します。

```
REM $INCLUDE: 'QBDECL.BAS'

CALL ibdev (0, 1, 0, 12, 1, 0, dmm%)
CALL ibclr(dmm%)

CALL ibwrt(dmm%, "F1R0S2T4")

DIM RD AS STRING * 13
SUM = 0

FOR X = 1 to 10
      CALL ibtrg(dmm%)
      CALL ibrd(dmm%, rd$)
      SUM = SUM + VAL(rd$)
NEXT X

PRINT "The average voltage is "; SUM/X

END
```

コンパイルと連結

このようにしてQuickBASICで書かれたアプリケーションプログラムは、QuickBASICの環境内で走らせることもできるし、また独立した実行可能なプログラムとして走らせることもできます。どちらの方法を取るにせよディストリビューションディスクに含まれたQuickBASIC言語インターフェースが必要となります。以下に示すの

はQuickBASIC 4.5 で走らせるプログラムです。

プログラムをQuickBASIC環境で走らせる

アプリケーションプログラムをQuickBASIC環境で走らせるために次の各ステップを完了してください。

1. QuickBASIC言語インターフェース (QBIB.OBJ) をインストレーション時に設定したGPIBディレクトリからQuickBASICディレクトリQB45 (QuickBASiCバージョン4.5) にコピーします。

2. 次のコマンドを入力してQuickLibrary を作成します。

```
C:\QB45> LINK /Q QBIB.OBJ,,,BQLB45.LIB
```

   このコマンドによりQBIB.QLBと言う名のQuickLibrary が作成されます。

3. このQuickLibarary を含むQuickBASIC環境を入力します。このためには次のコマンドを使います。

```
C:\QB45> QB /L QBIB.QLB
```

4. アプリケーションプログラムをロードして走らせます。

独立した実行可能なプログラムを作成する

独立した実行可能なプログラムを作成するには次の各ステップを完了してください。

1. QuickBASIC言語インターフェース (QBIB.OBJ) をインストレーション時に設定したGPIBディレクトリからQuickBASICディレクトリQB45 (QuickBASiCバージョン4.5) にコピーします。

2. 次のコマンドを入力してアプリケーションプログラム (AVGVOLTS.BAS) をコンパイルしてください。

```
C:\QB45> BC /O AVGVOLTS.BAS;
```

3. 次のコマンドを入力して言語インターフェースを連結してください。

```
C:\QB45> LINK AVGVOLTS.OBJ+QBIB.OBJ;
```

このステップで AVGVOLTS.EXE と言う名の実行可能なファイルが出来上がります。

4. 次のコマンドを入力してプログラムを走らせます。

```
C:\QB45> AVGVOLTS
```

ボード関数による上記相当のプログラム

次に示すQuickBASICプログラムは今し方作成したプログラムに相当するもう一つのプログラムです。但し、先のプログラムがデバイス関数を使用したのに反し、このプログラムはボード関数を使用している点で異なっています。このプログラムでは、メータは1次アドレスが(1)であり、したがってトークアドレスは16進数 (hex) の41 (ASCII の A) でリスンアドレスは hex 21 (ASCII の !) です。GPIBインターフェースボードは1次アドレスが (0) であり、したがってトークアドレスが hex 40 (ASCII の@)でリスンアドレスが hex 20 (ASCII の <space>)です。

```
REM $INCLUDE: 'QBDECL.BAS'
CALL ibfind("gpib0", board%)
CALL ibsic(board%)
CALL ibsre(board%)

CALL ibcmd(board%, "!" + chr$(&H04))

CALL ibcmd(board%, "?_@!")
CALL ibwrt(board%, "F1R0S2T4")

DIM RD AS STRING * 13
SUM = 0
```

```
FOR X = 1 to 10
      CALL ibcmd(board%,"!" + chr$(&H08))
      CALL ibcmd(board%, "?_ A)
      CALL ibrd(board%, rd$)
      SUM = SUM + VAL(rd$)
NEXT X

PRINT "The average voltage is "; SUM/10

END
```

上記相当のBASICプログラム

BASIC言語によるプログラムは、デバイス関数によるものもボード関数によるものも、以下に示す宣言ファイル名を除いてQuickBASICによるプログラムと全く同じです。QBDECL.BASをMBDECL.BASに置き換えます。

上記相当のCプログラム(デバイス関数によるもの)

```
#include <stdio.h>
#include "decl.h"
main() {

    short dmm, x;
    double sum;
    char rd[13];

    dmm = ibdev (0, 1, 0, 12, 1, 0);
    ibclr (dmm);

    ibwrt (dmm, "F1R0S2T4", 8);
```

```
    for (sum=0, x=0; x <= 10; x++) {
        ibtrg (dmm);
        ibrd (dmm, rd, 13);
        sum = sum + atof(rd);
    }
    printf ("The average voltage is %f", sum/10);
}
```

上記相当のBASICAプログラム(デバイス関数によるもの)

BASICAを開いた後、BASICA MERGEコマンドを使ってDECL.BASをロードします。

```
MERGE DECL.BAS
100 brdindx% = 0 : pad% = 1 : sad% = 0 : tmo% = 12
110 eot% = 1 : eos% = 0
120 CALL ibdev(brdindx%,pad%,sad%,
                tmo%,eot%,eos%,dmm%)
130 CALL ibclr(dmm%)
140 REM
145 wrt$ = "F1R0S2T4"
150 CALL ibwrt(dmm%, wrt$)
160 REM
170 SUM = 0
180 rd$ = space$(16)
190 FOR X = 1 to 10
200 CALL ibtrg(dmm%)
210 CALL ibrd(dmm%, rd$)
220 SUM = SUM + VAL(rd$)
230 NEXT X
240 REM
250 PRINT "The average voltage is "; SUM/10
260 REM
270 END
```

NI-488の各関数の説明

この章の以下の部分ではNI-488の関数を一つ一つ例を上げて詳細に説明します。これらの関数は参照の便のためアルファベット順に配列してあります。

IBBNA　IBBNA

目的　デバイスのアクセスボードを変更する。

フォーマット

BASICA

```
CALL ibbna (ud%, bname$)
```

QuickBASIC/BASIC

```
CALL ibbna (ud%, bname$)
      あるいは
ilbna (ud%, bname$)
```

C

```
short ibbna (short ud, char bname [])
```

udはデバイスを指定します。bnameはそのデバイスにおける全ての呼び出しに使用されることになる新しいアクセスボードを指定します。ibbnaは構成において設定されたボードの割り当てを変更する時にのみ必要になります。

いったん割り当てられたボードはそれ以降のデバイス関数がそのデバイスから行われるときはいつでも使用され、その状態はibbnaがもう一度呼び出されるか、ibonlかibfindが呼び出されるか、またはシステムがリスタートされるまで続きます。

IBBNA (前ページより続く) IBBNA

デバイス関数の例

デバイス dvm をインターフェースボード "GPIB0" との関係を設定する。

BASICA/QuickBASIC/BASIC

```
100  devname$ = "DEV10"
110  CALL ibfind (devname$, dvm%)
120  REM  Set GPIB1 as the access board for
130  REM  device dvm%.
140  bname$ = "GPIB1"
150  CALL ibbna (dvm%, bname$)
```

QuickBASIC/BASIC

```
dvm% = ilfind ("dEV10")
sta% = ilbna (dvm%, "GPIB0")
```

C

```
dvm = ibfind ("DEV10");
ibbna (dvm, "GPIB0");
```

IBCAC　　IBCAC

目的　[ボードが] アクティブコントローラとなる

フォーマット

BASICA

```
CALL ibcac (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibcac (ud%, v%)
     あるいは
ilcac (ud%, v%)
```

C

```
short ibcac (short ud, short v)
```

udはインターフェースボードを指定します。もしvが0以外なら、GPIBボードはデータ転送オペレーションに関して、同期的にコントロールをとります。もしvが0ならば、GPIBボードは即座に（非同期的に）コントロールをとります。

GPIBボードが同期的にコントロールを得るに際し、GPIBボードは転送中のデータを損なわずにATNシグナルをアサートします。もしもデータのハンドシェイクが進行中であれば、コントロールを獲得する動作は延期されます。ハンドシェイクが進行中でなければ、コントロールの獲得は直ちに行われます。同期的なコントロールの獲得は ibrd か ibwrt の動作の完了が時間切れかエラーを伴っていれば保証されません。

非同期的なコントロール獲得は同期的なコントロール獲得が不可能と見られる場合（例えばタイムアウトエラーがあった場合)に使用するべきです。

IBCAC (前ページより続く) IBCAC

ibcacの使用はほとんどのアプリケーションでは必要ではありません。ibcmd と ibrppの様にGPIBボードがコントロールをとることを必要とする関数の場合にはGPIBボードは ibcacなしで自動的にコントロールをとります。

GPIBボードがCICでない場合は、ECICエラーとなります。

ボード関数の例

1. 転送が進行中であってもコントロールを獲得する。

BASICA/QuickBASIC/BASIC

```
100  v% = 0
110  CALL ibcac (brd0%, v%)
120  REM  ibsta% should show board is now CAC
130  REM  (CIC ATN).
```

QuickBASIC/BASIC

```
sta% = ilcac (brd0%, 0)
```

C

```
ibcac (brd0, 0);
```

IBCAC (前ページより続く) IBCAC

2. 同期的にコントロールをとり、一つの読取動作の後にATNをアサートする。

BASICA/QuickBASIC/BASIC

```
100  board$ = "GPIB0"
110  CALL ibfind (board$, brd0%)
120  CALL ibrd (brd0%, rd$)
130  v% = 1
140  CALL ibcac (brd0%, v%)
```

QuickBASIC/BASIC

```
brd0% = ilfind ("GPIB0")
sta% = ilrd (brd0%, rd$, 512)
sta% = ilcac (brd0%, 1)
```

C

```
short brd0;
brd0 = ibfind ("gpib0");
ibrd (brd0,rd,512);
ibcac (brd0,1);
```

IBCLR　　IBCLR

目的　指定されたデバイスをクリアする。

フォーマット

BASICA

```
CALL ibclr (ud%)
```

QuickBASIC/BASIC

```
CALL ibclr (ud%)
        あるいは
ilclr (ud%)
```

C

```
short ibclr (short ud)
```

udはデバイスを指定します。

ibclr関数はある指定されたデバイスの内部あるいはデバイス関数をクリアします。

ibclrはボード関数ibcmdを呼び出して指定されたアクセスボードを使って次のコマンドを送るようにします。

- アクセスボードのトークアドレス
- アクティブリスナへのアドレシングの中止(UNL)
- デバイスのリスンアドレス
- もしあれば、デバイスの2次アドレス
- 選択したデバイスのクリア(SDC)

IBCLR (前ページより続く) IBCLR

上記以外のコマンドも必要にしたがって送られます。

その他の情報については*IBCMD*の項と第3章のデバイス関数の説明を参照してください。

デバイス関数の例

デバイスvmtrをクリアする。

BASICA/QuickBASIC/BASIC

```
100  dev$ = "DEV3" 'open the voltmeter
110  CALL ibfind (dev$, vmtr%)
120  REM Clear the voltmeter
130  CALL ibclr (vmtr%)
```

QuickBASIC/BASIC

```
' Open instrument
vmtr% = ilfind ("dev3")
' Clear it
sta% = ilclr (vmtr%)
```

C

```
short vmtr;

vmtr = ibfind ("dev3"); /* open instrument */
ibclr (vmtr); /* clear it */
```

IBCMD　　IBCMD

目的　GPIBコマンドメッセージを送る。

フォーマット

BASICA

```
CALL ibcmd (ud%, cmd$)
```

QuickBASIC/BASIC

```
CALL ibcmd (ud%, cmd$)
     あるいは
ilcmd (ud%, cmd$, cnt&)
```

C

```
short ibcmd (short ud, char cmd [], unsigned long cnt)
```

udはインターフェースボードを指定します。cmd はGPIBを通して送るコマンドを含んでいます。

ibcmd関数はインターフェースメッセージ（コマンド）をGPIBを通して送るために用いられます。これらのコマンドの種類は付録Aにまとめられています。ibcmd関数はまたGPIBコントロールをデバイスに引き渡すためにも使用されます。この関数はプログラム用の命令をデバイスに送る目的では使用されません。このような命令を送るには ibrd と ibwrt関数を使用します。

ibcmdのオペレーションは次の事象のうちの一つによって終了させられます。

- 全てのコマンドの転送が成功した場合。
- エラーが検出された場合。
- 時間制限の超過。

IBCMD (前ページより続く) IBCMD

- コントロール獲得のコマンドが送られた場合。
- システムコントローラから送られたインターフェースクリア(IFC)のメッセージを受信した場合。

以上の事象のうち、最初のものを除く全ての事象のいずれかがibcmdのオペレーションを終了させた場合、転送カウントは要求されたカウントより少ないことがあります。

C言語の場合は、要求されたカウントcntはlong型整数に収まる値であれば何でも構いません。但し、コマンド転送は通常非常に小さい値です。アプリケーションプログラムからの小さい値にも問題が無いように、C言語インターフェースはlong値でなくて整数値が関数に引き渡されても問題無いようになっています。

GPIBボードがCICでないとECICエラーが起こります。GPIBボードがアクティブコントローラでないと、ボードはコマンドバイトを送る前にコントロールを獲得しATNをアサートします。この後GPIBボードはアクティブコントローラの資格を保持します。

以下に示す幾つかの例においては、GPIBコマンドとアドレスは印刷可能なASCII文字で符号化されています。種々の値は、ASCII文字で代表することが可能な限りは、ASCII文字の使用が最も簡単な方法です。ある数値に相当するASCII文字を捜したいときは付録Aを見てください。

ボード関数の例

1. 全てのリスナに対するアドレシングを停止し（UNLあるいはASCIIの?)、hex 46 (ASCIIの?)のトーカとhex 31 (ASCIIの1）のリスナにアドレスする。

   BASICA/QuickBASIC/BASIC

```
100  cmd$ = "?F1"          ' UNL TAD LAD
110  CALL ibcmd (brd0%, cmd$)
```

IBCMD　　(前ページより続く)　　IBCMD

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, "?F1", 3)
```

C

```
ibcmd (brd0, "?F1", 3);  /*  UNL TAD LAD          */
```

2. 第1例と同じであるが、但しリスナが2次アドレスのhex 6Eを有している。

BASICA/QuickBASIC/BASIC

```
100  cmd$ = "?F1n"          ' UNL TAD LAD SAD
110  CALL ibcmd (brd0%, cmd$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%,"?F1n",4)
```

C

```
ibcmd (brd0,"?F1n",4);   /*   UNL TAD LAD SAD      */
```

3. デバイスクリア(DCLあるいはhex 14)コマンドにより全てのGPIBデバイスをクリアする。

BASICA/QuickBASIC/BASIC

```
100  cmd$ = chr$(&H14)    ' DCL
110  CALL ibcmd (brd0%, cmd$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, chr$(&H14), 1)
```

IBCMD (前ページより続く) IBCMD

C

```
ibcmd (brd0, "\x14", 1); /*      DCL      */
```

4. 選択されたデバイスをクリア(SDCあるいは hex 04) コマンドにより、リスンアドレスが各々hex 21 (ASCII の !) と hex 28 (ASCII の ([左括弧]) の2つのデバイスをクリアする。

BASICA/QuickBASIC/BASIC

```
100  cmd$ = "?!(" + chr$(&H04) ' UNL LAD LAD SDC
110  CALL ibcmd (brd0%, cmd$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, "?!(" + chr$(&H04), 4)
```

C

```
ibcmd (brd0,"?!(\x04", 4); /* UNL LAD LAD SDC  */
```

5. 以前にGroup Execute Trigger (GETあるいはhex 08)のコマンドでリスンするようにアドレスされたデバイスを全てトリガする。

BASICA/QuickBASIC/BASIC

```
100  cmd$ = chr$(&H08)    ' GET
110  CALL ibcmd (brd0%, cmd$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, chr$(&H08), 1)
```

C

```
ibcmd (brd0, "\x08", 1); /*      GET            */
```

IBCMD　　(前ページより続く)　　IBCMD

6. シリアルポーリング可能化(SPEあるいはhex 18) とシリアルポーリング不可能化(SPDあるいはhex 19) コマンドを使用して(GPIBのリスンアドレスはhex 20あるいはASCIIの<space>)トークアドレスがhex 52 (ASCIIのR) にあるデバイスをシリアルポーリングする。

BASICA/QuickBASIC/BASIC

```
100  cmd$ ="R " + chr$(&H18)  'TAD MLA SPE
110  CALL ibcmd (brd0%, cmd$)
120  rd$ = space$(1)
130  CALL ibrd (brd0%, rd$)
140  REM  After checking the status byte in
150  REM  rd$, disable this device and
160  REM  unaddress it with the Untalk
170  REM  (UNT or ASCII _) command before
180  REM  polling the next one.
190  cmd$ = chr$(&H19) + "_"     ' SPD UNT
200  CALL ibcmd (brd0%, cmd$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, "R_" + chr$(&H18), 3)
rd$ = space$(1)
sta% = ilrd (brd0%, rd$, 1)
sta% = ilcmd (brd0%, chr$(&H19) + "_", 2)
```

C

```
ibcmd (brd0, "R \x18", 4);  /* TAD MLA SPE     */
ibrd (brd0, rd, 1);     /*   read one byte     */
ibcmd (brd0, "\x19_", 2);     /*   SPD UNT     */
```

IBCMDA　　IBCMDA

目的　ストリングからコマンドを非同期的送る。

フォーマット

BASICA

```
CALL ibcmda (ud%, cmd$)
```

QuickBASIC/BASIC

```
CALL ibcmda (ud%, cmd$)
     あるいは
ilcmda (ud%, cmd$, cnt&)
```

C

```
short ibcmda (short ud, char cmd [], unsigned long cnt)
```

udはインターフェースボードを指定します。cmdはGPIBを通して送られるコマンドを含みます。

ibcmda関数はインターフェースメッセージ（コマンド）をGPIBを通して伝送するために使用します。これらのコマンドのリストは付録A中に示されています。ibcmda関数はGPIBコントロールを他のデバイスに引き渡す目的でも使用されます。この関数はデバイスにプログラミング用の命令の伝送には使用されません。そのような命令とか、他のデバイス依存型の情報の伝送には ibrd と ibwrt 関数を使用します。

ibcmdaは、アプリケーションプログラムがGPIBコマンドを処理している間に他の機能も果たさなければならない場合にibcmd に代わって使用されます。ibcmdaは入出力オペレーションがスタートすると即座に戻されます。

IBCMDA　　(前ページより続く)　　IBCMDA

3種の非同期入出力関数 (ibcmda, ibrda 及び ibwrta)はアプリケーションプログラムが入出力処理を行いながらその他の (非GPIB) 関数を実行できるようにデザインされています。一度このような入出力関数の呼び出しが行われると、その入出力動作が完了してGPIBドライバとアプリケーションが再同期化されない限り、そのデバイスあるいはアクセスボードに関連したそれ以外のGPIB関数呼び出しは許されなくなります。

再同期かは次の3種の関数のうちの1つを使用して行うことが出来ます。

注)　再同期化は戻されたibstaがCMPLを含む場合のみ実現します。

- ibwait (マスクにCMPLを含む) – ドライバとアプリケーションが同期化される。
- ibstop – 非同期的な入出力がキャンセルされ、ドライバとアプリケーションが同期される。
- ibonl – 非同期入出力のキャンセル、インターフェースのリセットさ、及びドライバとアプリケーションの同期が行われる。

以上のほか、非同期的入出力中に呼び出し可能なGPIB関数としてはibwait (マスクは任意)しかありません。このデバイスあるいはアクセスボードに関して以上に挙げた以外のGPIB関数呼び出しをしてもEOIPエラーが返ってくるだけです。

GPIBボードがCICでない場合はECICエラーとなります。ボードがアクティブコントローラでなかった場合は、ボードはコントロールを獲得し、コマンドバイトを送るに先立ってATNをアサートします。それ以後ボードはアクティブコントローラの資格を保持します。IEEE-488バス上にほかのデバイスが無かった場合はENOLエラーが返されます。

IBCMDA (前ページより続く) IBCMDA

ボード関数の例

優先的な事象の生起をテストする一方で幾つかのデバイスにアドレスし、同時通信メッセージを送る。

BASICA/QuickBASIC/BASIC

```
100  REM  The interface board BRD0% at talk
110  REM  address &H40 (ASCII @), addresses
120  REM  nine Listeners at addresses &H31-
130  REM  &H39 (ASCII 1-9) to receive the
140  REM  broadcast message.
150  board$ = "GPIB0"
160  CALL ibfind (board$, brd0%)
170  cmd$ = "?@123456789"     ' UNL MTA
180                           ' LAD1...LAD9
190  CALL ibcmda (brd0%, cmd$)
200  GOSUB 1000               ' Unspecified routine
210                           ' for a high priority
220                           ' event.
230  mask% = 0                ' Set mask to return
240                           ' immediately.
250  CALL ibwait (brd0%, mask%)
260  IF ibsta% AND TIMO THEN GOTO 900 'jump on error
270  IF (ibsta% AND &H100) <> &H100 GOTO 200
280  PRINT "Asynchronous commands sent!"
290  mask% = &H4100
300  CALL ibwait (brd0%,mask%)
310  PRINT "Asynchronous transfer properly terminated"
320  ...
900  PRINT "Error" : STOP
910  END
```

IBCMDA　　(前ページより続く)　　IBCMDA

QuickBASIC/BASIC

```
brd0% = ilfind ("GPIB0")
ilcmda (brd0%,"?@123456789",11)
sta% = 0
While (sta% AND &H100) <> &H100
       CALL eventtest
       sta% = ilwait (brd0%, 0)
       IF sta% < 0 THEN GOSUB ERROR
WEND
PRINT "Asynchronous commands sent!"
sta% = ilwait (brd0%,&H4100)
PRINT "Asynchronous transfer properly terminated"
```

C

```
brd0 = ibfind ("GPIB0");
ibcmda (brd0, "?@123456789", 11);
sta = 0;
while ((sta & 0x100) != 0x100) {
        event_test();
        ibwait (brd0, 0);
        sta = ibsta;
        if (sta < 0) error ();
}
printf ("Asynchronous commands sent!");
ibwait (brd0,0x4100)
printf ("Asynchronous transfer properly terminated");
```

IBCONFIG　IBCONFIG

目的　ソフトウエアの構成パラメータを変更する。

フォーマット

BASICA

```
CALL ibconfig (ud%, option%, value%)
```

QuickBASIC/BASIC

```
CALL ibconfig (ud%, option%, value%)
        あるいは
ilconfig (ud%, option%, value%)
```

C

```
short ibconfig (short ud, unsigned short option,
                unsigned short cnt)
```

udはインターフェースボードかあるいはデバイスを指定します。optionはソフトウエア中で構成可能なアイテムを選択するために使用します。構成可能な項目はvalueの内容にしたがってセットされます。構成可能な項目のそれ以前の内容はiberrで返されます。udがGPIBインターフェースボードの記述子であれば、optionは表5－4中に示される値をとります。udがデバイス記述子の場合は、optionは表5－5中に示される値をとります。

表5－4 ボード構成の各オプション

オプション	内容
1	1次アドレス。valueはGPIBインターフェースボードの新しい1次アドレス(0-30)。*IBPAD*と付録Aを参照のこと。
2	2次アドレス。valueはボードの新しい2次アドレス(0,96-126)。*IBSAD*と付録Aを参照のこと。

(次ページに続く)

IBCONFIG　(前ページより続く)　IBCONFIG

表5－4 ボード構成の各オプション(前ページより続く)

オプション	内容
3	タイムアウト値。valueはボードの新しいタイムアウト値（0－17)。*IBTMO*を参照のこと。
4	書込オペレーションにおけるENDメッセージの可能化／不可能化。value は新しいEOTモード(0ならEND無し、0以外なら最後のバイトにENDをつけて送る）。*IBEOT*参照。
5	パラレルポーリングのための構成。valueはパラレルポーリング構成バイト(0,96-126)。*IBPPC*参照。
7	自動シリアルポーリングの可能化／不可能化。valueが0なら自動ポーリングは不可能化される。0以外なら自動ポーリングは可能化される。
8	NI-488 CIC プロトコルを使用する／使用しない。valueが0ならCIC プロトコルを使用しない。valueが0以外ならCICプロトコルを使用する。この章の初めの方の Device Functions の項を参照のこと。
9	ハードウエア割込を可能化／不可能化。valueが0ならGPIBインターフェースボード割込を不可能化、valueが0以外ならGPIBインターフェースボード割込を可能化。第2章の*IBCONF*ユーティリティプログラムの説明を参照のこと。
10	システムコントロールの要求あるいは解除。valueが0ならばシステムコントローラ資格を必要とする関数の使用は許されない。valueが0以外ならシステムコントローラ資格を必要とする関数の使用は許される。*IBRSC*を参照のこと。

(次ページに続く)

IBCONFIG (前ページより続く) IBCONFIG

表5－4 ボード構成の各オプション(前ページより続く)

オプション	内容
11	RENをアサート／解除する。valueが0以外ならIEEE-488のRENがアサートされる。valueが0ならRENは解除される。*IBSRE*を参照のこと。
12	EOS(ストリング終了）文字が検出されれば読取を停止する。valueが0以外ならEOS文字がデータの流れの中で検出された時点で読取機能が停止される。valueが0ならEOS検出が不可能化される。*IBEOS*を参照のこと。
13	EOS文字を送る時にEOIをアサート。valueが0ならEOSのときにEOIを送らない。valueが0以外ならEOIをEOSに付けて送ってやる。
14	7-/8-ビットEOSの比較機能の使用。valueが0ならEOS文字の下位の7ビットを比較に使用、valueが0以外の場合は8ビットを使用。*IBEOS*の項を参照のこと。
15	EOS(ストリング終了）文字。valueはボードの新しいEOS文字(8ビット）。*IBEOS*の項を参照のこと。
16	パラレルポーリングのリモート／ローカル構成。valueが0の場合はGPIBインターフェースボードはIEEE-488パラレルポーリング（PP)インターフェース機能のサブセットPP1（外部のコントローラによるリモート構成）を使用。valueが0以外ならば、ボードはPPのサブセットPP2（アプリケーションプログラムから行うローカル構成。この場合valueはローカルポーリング可能化メッセージ［lpe]として使用)。*IBPPC*の項を参照のこと。

(次ページに続く)

IBCONFIG　(前ページより続く)　IBCONFIG

表5－4 ボード構成の各オプション(前ページより続く)

オプション	内容
17	IEEE-488バスハンドシェイクのタイミング。valueが1ならばIEEE-488ソースハンドシェイクT1に正常タイミング(≧2 msec)を使用。valueが2であれば高速タイミング(≧500 nsec)をT1に使用。valueが3であれば超高速タイミング(≧350 nsec)を使用。
18	ディレクトメモリアクセス(DMA)転送の可能化／不可能化。valueが0であればGPIBインターフェースボードDMA転送を不可能化する。valueが0以外であればGPIBインターフェースボードDMA転送を可能化する。第2章のIBCONFユーティリティプログラムの説明を参照のこと。
19	ibrdにおけるバイトスワッピング。valueが1ならバスから読み取られた対になったバイトはibrdバッファに記憶される前にスワップされる。転送カウントは偶数でなければならず、さもなければECAPが返される。この場合バッファの最後の2バイトは無効である。ECAPが返され、バッファが奇数のアドレスで始まっている場合はバッファを偶数のアドレスで始めるようにする。valueが0であれば、ibrdにおけるバイトスワッピングは不可能化される。

(次ページに続く)

IBCONFIG (前ページより続く) IBCONFIG

表5－4 ボード構成の各オプション(前ページより続く)

オプション	内容
20	ibwrtにおけるバイトスワッピング。valueが1であれば対になったバイトはユーザーのバッファからバスに書き込まれる前にスワップされる。転送カウントは偶数でなければならず、さもなければECAPが返される。幾つかの場合にはバッファのアドレスは偶数でなければならない。ECAPが返され、バッファが奇数のアドレスで始まる場合はバッファを偶数のアドレスで始めるようにする。valueが0ならば、ibwrtにおけるバイトスワッピングは不可能化される。
21	ソフトウエアの事象待ち行列の可能化／不可能化。valueが1ならば待ち行列は可能化され、IBSTA中のEVENTビットは活動状態になり、DTASとDCASビットは非活動状態になる。待ち行列が可能化されると、「デバイスクリア」や「トリガ」メッセージは全て待ち行列に入れられ、EVENTビットがセットされ、進行中の入出力オペレーションは全て中止される。事象を検索するにはibevent関数を使用出来る。valueが0であれば待ち行列は不可能化される。この方がデフォルトである。このオプションはハードウエア割込がボードについて可能化している場合のみに使用出来る。

(次ページに続く)

IBCONFIG　　(前ページより続く)　　IBCONFIG

表5－4 ボード構成の各オプション(前ページより続く)

オプション	内容
22	IBSTAのSPOLLビットの可能化／不可能化。valueが1であればSPOLLビットは活動状態になり、インターフェースボードがサービス要求中にコントローラからシリアルポールされるとそれはセットされる。valueが0(デフォルト)であるとSPOLLは非活動状態にある。このオプションはボードに対するハードウエア割込が可能化されている場合のみ使用出来る。
23	デバイスをオンラインにする時にLLOの送出を可能化／不可能化する。valueが0(デフォルト)であると、ibfindが使用されていれば、Local Lockoutはリモートデバイスまで送られない。デバイスに自動的にLocal Lockout状態になって欲しければ、ibfindをどのデバイスに対してでも呼び出さないうちにvalueを1にセットすること。
25	ドライバがパラレルポーリングの応答を待つ時間を設定する。デフォルトの2μsecはvalueを0にセットすることによって選択できる。違った時間の設定には、ibtmo関数の制限時間設定と同じ値を使用する。合法値の範囲は0から15までである。

(次ページに続く)

IBCONFIG (前ページより続く) IBCONFIG

表5－4 ボード構成の各オプション(前ページより続く)

オプション	内容
26	End-of-String (EOS)文字を受け取った時にIBSTAにENDビットをセットすることを可能化／不可能化。valueが1(デフォルト) であれば、IBSTAのENDビットはEOI, EOS, 及びEOIプラスEOSの何れを受け取った時でもセットされる。valueが0であれば、ENDビットがセットされるのはEOI或いはEOIプラスEOSを受け取った時に限られる。EOS文字がEOIなしで受け取られるとENDビットはセットされない。このオプションは3種類の入出力終了モード(EOIのみ、EOSのみ、及びEOIプラスEOS)を区別して使用する必要があるアプリケーションの場合に有用である。

IBCONFIG　　(前ページより続く)　　IBCONFIG

表5－5 デバイス構成における各オプション

オプション	内容
1	1次アドレス。valueはデバイスの新しい1次アドレス(0-30)。*IBPAD*と付録Aを参照のこと。
2	2次アドレス。valueはデバイスの新しい2次アドレス(0, 96-126)。*IBSAD*と付録Aを参照のこと。
3	タイムアウト値。valueはデバイスの新しいタイムアウト値(0-17)。*IBTMO*の項を参照のこと。
4	書込オペレーションにおけるENDメッセージの可能化／不可能化。valueは新しいEOTモード(0はEND無し、0以外はENDを最後のバイトに付けて送る)。*IBEOT*の項を参照のこと。
6	アドレシングの繰り返し。valueが0ならアドレシングの繰り返しを不可能化、1ならアドレシングの繰り返しを可能化。第2章中のIBCONFユーティリティの説明を参照のこと。
12	このデバイスからEOS(ストリング終了)文字が検出された時に読取を停止。valueが0でなければEOSがデバイスから受信したデータの流れ中で検出されるとともに読取機能は停止。valueが0ならEOS検出は不可能化される。*IBEOS*の項を参照のこと。
13	EOSをこのデバイスに送る時にEOIをアサート。valueが0ならEOIをEOSとともに送らない。valueが0以外ならEOIをEOSに付けて送る。*IBEOS*の項を参照のこと。

(次ページに続く)

IBCONFIG　(前ページより続く)　IBCONFIG

表5－5 デバイス構成における各オプション(前ページより続く)

オプション	内容
14	7-/8-ビットEOS比較の使用。valueが0ならEOS文字の下位の7ビットを比較に使用。valueが0以外なら8ビットを使用。*IBEOS*の項を参照のこと。
15	EOS(ストリング終了）文字。valueはこのデバイスとともに使用する新しいEOS文字（8ビット）。*IBEOS*の項を参照のこと。
19	ibrdにおけるバイトスワッピング。valueが1ならば、バスから読み取られたバイト対はibrdバッファ中に記憶される前にスワップされる。転送カウントは偶数でなければならず、さもなければECAPが返される。この場合バッファの最後の2バイトは無効となる。ECAPが返されてバッファが奇数のアドレスで始まっている場合はバッファを偶数のアドレスで始めること。valueが0ならばibrdのバイトスワッピングは不可能化される。
20	ibwrtにおけるバイトスワッピング。valueが1ならバイト対はユーザーのバッファからバスに書き込まれる前にスワップされる。転送カウントは偶数でなければならず、さもなければECAPが返される。幾つかの場合にはバッファのアドレスは偶数でなければならない。もしECAPが返されバッファが奇数のアドレスで始まっている場合はバッファを偶数のアドレスで始める。valueが0の場合はibwrtにおけるバイトスワッピングは不可能化される。

(次ページに続く)

IBCONFIG　(前ページより続く)　IBCONFIG

表5－5 デバイス構成における各オプション (前ページより続く)

オプション	内容
24	シリアルポーリングの時間制限を設定。IEEE-488.1規格はコントローラが応答バイトを待つ時間を規定していない。本ソフトウエアのデフォルト制限時間の1秒 (value 11)はたいていのデバイスにとって十分であると思われる。これより長い待ち時間が必要ならば、valueをibtmo の時間制限値の何れかにセットすればよい。合法値は0から15の範囲内である。
27	読み取り、書き込みオペレーションが終わる度にバスコマンドのuntalk (UNT) およびunlisten (UNL)を自動的に送ることの可能化／不可能化。valueが0(デフォルト) ならば、各入出力オペレーション毎にUNTとUNLは送られない。このモードはIEEE-488.2に合致している。valueが1ならば、IEEE-488.1バスコマンドのUNTとUNLが入出力オペレーションが終わる毎にバス上に送られる。このモードはIEEE-488.2 に合致していないが、488.2に従っていないデバイスとの通信の際に必要になる可能性がある。

IBCONFIG (前ページより続く) IBCONFIG

デバイス関数の例

デバイス読取の準備として各構成可能なパラメータをセットする。

BASICA/QuickBASIC/BASIC

```
100  dev$ = "dev1"
110  CALL ibfind (dev$, dev1%)
120  REM Enable repeat addressing
130  option% = 6 : value% = 1
140  CALL ibconfig (dev1%, option%, value%)
150  REM Set linefeed as the EOS character
160  option% = 15 : value% = 10
170  CALL ibconfig (dev1%, option%, value%)
180  REM Use 7-bit comparison for EOS character
190  option% = 14 : value% = 0
200  CALL ibconfig (dev1%, option%, value%)
210  REM Terminate reads on EOS
220  option% = 12 : value% = 1
230  CALL ibconfig (dev1%, option%, value%)
```

QuickBASIC/BASIC

```
' open device
CALL ibfind ("dev1", dev1%)
' Enable repeat addressing
CALL ibconfig (dev1%, 6, 1)
' Set linefeed as the EOS character
CALL ibconfig (dev1%, 15, 10)
' Use 7-bit comparison for EOS character
CALL ibconfig (dev1%, 14, 0)
' Terminate reads on EOS
CALL ibconfig (dev1%, 12, 1)
```

IBCONFIG　　(前ページより続く)　　IBCONFIG

C

```
/* open device */
dev1 = ibfind ("dev1");
/* Enable repeat addressing */
ibconfig (dev1, 6, 1)
/* Set linefeed as the EOS character */
ibconfig (dev1, 15, 10)
/* Use 7-bit comparison for EOS character */
ibconfig (dev1, 14, 0)
/* Terminate reads on EOS */
ibconfig (dev1, 12, 1)
```

ボード関数の例

1. ボード読取の準備として各構成可能なパラメータをセットする。

BASICA/QuickBASIC/BASIC

```
100  gpib$ = "gpib0"
110  CALL ibfind (gpib$, gpib0%)
120  REM Enable DMA transfers
130  option% = 18 : value% = 1
140  CALL ibconfig (gpib0%, option%, value%)
150  REM Turn off Autopolling
160  option% = 7 : value% = 0
170  CALL ibconfig (gpib0%, option%, value%)
180  REM Turn on interrupts
190  option% = 9 : value% = 1
200  CALL ibconfig (gpib0%, option%, value%)
```

IBCONFIG (前ページより続く) IBCONFIG

QuickBASIC/BASIC

```
' open GPIB interface board
CALL ibfind ("gpib0", gpib0%)
' Enable DMA transfers
CALL ibconfig (gpib0%, 18, 1)
' Turn off Autopolling
CALL ibconfig (gpib0%, 7, 0)
' Turn on interrupts
CALL ibconfig (gpib0%, 9, 1)
```

C

```
/* open GPIB interface board */
gpib0 = ibfind ("gpib0");
/* Enable DMA transfers */
ibconfig (gpib0, IBC_DMA, 1)
/* Turn off Autopolling */
ibconfig (gpib0, IBC_AUTOPOLL, 0)
/* Turn on interrupts */
ibconfig (gpib0, IBC_IRQ, 1)
```

IBCONFIG　(前ページより続く)　IBCONFIG

2. バイナリ整数データの自動バイトスワッピングを可能化する。

BASICA/QuickBASIC/BASIC

```
50   DIM array% (500)
100  REM Read in unswapped header data
101  REM (10 bytes).
102  header$ = space$(10)
103  CALL ibrd (ud%, header$)
104  REM Arrange for byte swapping.
105  option% = 19 : value% = 1
106  CALL ibconfig (ud%, option%, value%)
107  REM Read 1,000 bytes with automatic
108  REM swapping.
109  count% = 1000
110  CALL ibrdi (ud%, array% (0), count%)
111  REM Disable swapping for subsequent reads.
112  value% = 0
113  CALL ibconfig (ud%, option%, value%)
```

QuickBASIC/BASIC

```
DIM array% (500)
' Read in unswapped header data (10 bytes).
header$ = space$(10)
sta% = ilrd (ud%, header$, 10)
' Arrange for byte swapping.
sta% = ilconfig (ud%, 19, 1)
' Read 1,000 bytes with automatic swapping.
sta% = ilrdi (ud%, array% (), 1000)
' Disable swapping for subsequent reads.
sta% = ilconfig (ud%, 19, 0)
```

IBCONFIG (前ページより続く) IBCONFIG

C

```
short array [500]
char header [10]

/* Read in unswapped header data. */
ibrd (ud, header, 10)
/* Arrange for byte swapping. */
ibconfig (ud, 19, 1)
/* Read 1,000 bytes with automatic swapping. */
ibrdi (ud, array, 1000)
/* Disable swapping for subsequent reads. */
ibconfig (ud, 19, 0)
```

IBDEV　　IBDEV

目的　未使用で名の知られていないデバイスを開いて初期化する。

フォーマット

BASICA

```
CALL ibdev (board.index%,pad%,sad%,tmo%,eot%,eos%,ud%)
```

QuickBASIC/BASIC

```
CALL ibdev (board.index%,pad%,sad%,tmo%,eot%,eos%,ud%)
        あるいは
ud% = ildev (board.index%,pad%,sad%,tmo%,eot%,eos%)
```

C

```
ud = short ibdev (short boardindex, short pad, short sad,
                  short tmo, short eot, short eos)
```

boardindexはデバイス記述子が関係を設定すべきアクセスボードのインデックスで、その範囲は0から［(ボードの数) －1］です。引き数のpad, sad, tmo, eot 及び eosはNI-488の入出力関数のためのソフトウエア構成をダイナミックにセットします。pad, sad, tmo, eot 及び eosは各々1次アドレス、2次アドレス、入出力タイムアウト、入力されたデータの最後のバイトの時にEOIをアサートすること、及びEOS[ストリング終了］モードとバイトを構成します。[これらの引き数についてはおのおのIBPAD, IBSAD, IBTMO, IBEOT, 及び IBEOSの項を参照してください。]デバイス記述子は変数udで返されます。

ibdevコマンドはまだ開いていないデバイスを選択し、それを開き、初期化します。ibdevはibfindの代わりに使用することが出来ます。

IBDEV (前ページより続く) IBDEV

ibdevはそれが最初に見い出したまだ開かれていないユーザーにより構成可能なデバイスのデバイス記述子を返します。したがって、ibdevを使用する前にはibfindの呼び出しは全て終えてしまっている必要があります。そのようにしない限りibfindで使用するつもりのデバイスをibdevが使ってしまうということを完全に防ぐことは出来ません。ibdev関数はデバイスを開くことに関してibonl関数と等しい役割を果たします。

注) NI-488.2ドライバのデバイス記述子はアプリケーションを何回呼び出しても開いたままでいます。したがって、デバイスの使用が終わったら必ずibonlを v=0 で呼び出してデバイス記述子を利用可能なデバイスの集まりの中に返しておいてください。そうしないと、このデバイスはつぎの ibdev呼び出しの時に使用可能でなくなります。

ibdev呼び出しが失敗した場合は、デバイス記述子の代わりにある負数が返ってきます。ibdev呼び出しの時に起こりうるエラーのうちで目立つものとして次の2つがあります。

- 使用可能なデバイスが無い場合、あるいは指定したボードインデックスが現実に存在しないボードを指している場合にはEDVRかENEBが返ってきます。
- 最後の5つのパラメータのうちの1つがイリーガル値であると、正しいボードの記述子とEARGエラーが返ってきます。

デバイス関数の例

1. ibdevが使用可能なデバイスを開き、それをgpib0 (board = 0)にアクセスするようにする。1次アドレスは6 (pad = 6)、2次アドレスは0x67 (sad = 0x67)、タイムアウトは10 msec (tmo = 7)、ENDメッセージは可能化し(eot = 1)、EOSモードは不可能化とする(eos = 0)。

IBDEV (前ページより続く) IBDEV

BASICA/QuickBASIC/BASIC

```
100  REM  Get a device descriptor associated
105  REM  with board 0 (GPIB0).
110  board% = 0
115  pad% = 6
120  sad% = &H67
125  tmo% = 7
130  eot% = 1
135  eos% = 0
140  CALL ibdev(board%,pad%,sad%,tmo%,eot%,eos%,ud%)
150  REM  If ud% is less than 0, you have an
155  REM  error and you cannot continue with
156  REM  your program.
```

QuickBASIC/BASIC

```
ud% = ildev(0,6,&H67,7,1,0)
'  If ud% is less than 0, you have an error
'  and you cannot continue with your program.
```

C

```
if ((ud = ibdev(0,6,0x67,7,1,0)) < 0) {
    /* Handle GPIB error here */
    if (iberr == EDVR) {
        /* bad boardindex or no devices
         * available.
         */
    }
    else if (iberr == EARG) {
        /* the call succeeded, but at least one
         * of pad,sad,tmo,eos,eot is incorrect.
         */
    }
```

IBDMA　　IBDMA

目的　DMAを可能化あるいは不可能化する。

フォーマット

BASICA

```
CALL ibdma (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibdma (ud%, v%)
     あるいは
ildma (ud%, v%)
```

C

```
short ibdma (short ud, short v)
```

udは一つのインターフェースボードを指定します。vが0以外であればGPIBとメモリ間のDMA転送が読取・書込オペレーションに用いられます。vが0であればプログラム命令による入出力が行われます。

構成時にDMAの可能化が選択してあれば、この関数はプログラム命令による入出力と選択されたDMAチャネル間をスイッチするために使用できます。構成時にDMAが不可能化してあったり、コンピュータにDMAの能力が無かった場合は、この関数を　vが0でない値で呼び出すとECAPが返されます。

この関数が行った割り当てはibdmaがもう1度呼び出されるか、ibonlかibfindが呼び出されるか、あるいはシステムがリスタートされるまで効力を保ちます。

ibdmaが呼び出されてエラーが起きないときは、それ以前のvの値がiberrに記憶されます。

IBDMA (前ページより続く) IBDMA

ボード関数の例

1. 既に構成してあるチャネルを利用してDMA転送を可能化する。

BASICA/QuickBASIC/BASIC

```
100  v% = 1        ' Any non-zero value will do.
110  CALL ibdma (brd0%, v%)
```

QuickBASIC/BASIC

```
sta% = ildma (brd0%, 1)
```

C

```
ibdma(brd0, 1); /* Any non-zero value will do. */
```

2. DMAを不可能化し、プログラム命令による入出力のみ行う。

BASICA/QuickBASIC/BASIC

```
100  v% = 0
110  CALL ibdma (brd0%, v%)
```

QuickBASIC/BASIC

```
sta% = ildma (brd0%, 0)
```

C

```
ibdma (brd0, 0);
```

IBEOS　　IBEOS

目的　EOS停止モードを変更するか不可能化する。

フォーマット

BASICA

```
CALL ibeos (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibeos (ud%, v%)
     あるいは
ileos (ud%, v%)
```

C

```
short ibeos (short ud, short v)
```

udはデバイスかインターフェースボードを指定します。vは表5－6にしたがってEOS文字とデータ転送終了方法を指定します。ibeosは構成の設定値を変更する場合のみ必要になります。

この関数によりなされた割り当てはibeosがもう一度呼び出されるか、ibonlかibfind関数が呼び出されるか、システムがリスタートされるまで効力を保ちます。

ibeosが呼び出されエラーが起きないときは以前のvの値がiberr中に記憶されます。

IBEOS　　(前ページより続く)　　IBEOS

表5－6 各種データ転送停止方法

	vの値	
方法	高バイト	低バイト
A. EOSの検出で読取を停止。	00000100	EOS
B. 書込機能においてEOS とともにEOIを設定。	00001000	EOS
C. (全ての読取・書込機能について) EOSの下位7ビットでなくて8ビット全部を比較。	00010000	EOS

方法AとCは読取オペレーションの停止方法を決定します。方法Aのみが選択されると、読み取られたバイトの下位の7ビットがEOS文字の下位の7ビットと合致した時に読取は停止します。方法AとCが選択されると、8ビット全部の比較がなされます。

方法BとCをともに選択すると、いつ書込オペレーションがENDメッセージを送るかが決定されます。方法Bのみが選択されると、下位の7ビットがEOS文字の下位の7ビットと合致した時にENDメッセージが自動的に送られます。方法BとCが選択されると、8ビット全部の比較がなされます。

注) EOSバイトをデバイスやボードのために定義してもドライバは書込みにおいて自動的にバイトを送ったりはしません。御使用になるアプリケーションプログラムは、それが定義するデータストリングの中にEOSバイトを含んでいる必要があります。

デバイスのIBEOS関数

udがある一つのデバイスを指定している場合は、vにおいて符号化されたオプションがそのデバイスが指定された全てのデバイス読取・書込において使用されます。

IBEOS (前ページより続く) IBEOS

ボードIBEOS関数

udがある一つのボードを指定している場合は、vにおいて符号化されているオプションは全てのボード読取・書込に関係してきます。

*IBEOT*の項と表2－2も参照してください。

デバイス関数の例

デバイスdvmに改行文字が書き込まれているときはENDメッセージを送る。

BASICA/QuickBASIC/BASIC

```
10   EOSV% = &H0A          ' EOS info for ibeos.
:
:
100  v% = EOSV% + &H0800
110  CALL ibeos (dvm%, v%)
120  wrt$ = "123" + chr$(&H0A)
150                        ' EOS character is the
160                        ' last byte of string.
170  CALL ibwrt (dvm%, wrt$)
```

QuickBASIC/BASIC

```
EOSV% = &H0A
.
.
.
sta% = ileos (dvm%, EOSV% + &H0800)
ilwrt (dvm%, "123" + chr$(&H0A), 4)
```

C

```
v = XEOS | '¥n'; /* EOS information for ibeos. */
ibeos (dvm, v);
ibwrt (dvm, "123\n", 4);
```

IBEOS (前ページより続く) IBEOS

ボード関数の例

1. インターフェースボード brd0 をプログラムして読取の200バイト以内に改行文字 (hex 0A)が検出された時は読取を終了するようにする。

BASICA/QuickBASIC/BASIC

```
10   EOSV% = &H0A
:
:
100  v% = EOSV% + &H0400
110  CALL ibeos (brd0%, v%)
120  REM  Assume board has been addressed; do
130  REM  board read.
140  rd$ = space$(200)
150  CALL ibrd (brd0%, rd$)
160  REM  The END bit in ibsta% is set if the
170  REM  read terminated on the EOS.
```

QuickBASIC/BASIC

```
EOSV% = &H0A
.
.
.
sta% = ileos (brd0%, EOSV% + &H0400)
rd$ = space$(200)
sta% = ilrd (brd0%, rd$, 200)
```

C

```
char rd[200];

v = REOS | '\n';
ibeos (brd0, v);
ibrd (brd0, rd, 200);
```

IBEOS (前ページより続く) IBEOS

2. インターフェースボードbrd0をプログラムして7ビット文字hex 0Aでなく8ビット値hex 82で読取オペレーションを停止するようにするには第1例のライン10と100を以下の様に変更する。

BASICA/QuickBASIC/BASIC

```
10   EOSV% = &H82
:
:
100  v% = EOSV% + &H1400
:
:
```

QuickBASIC/BASIC

```
EOSV% = &H82
.
.
.
sta% = ileos(brd0%, EOSV% + &H1400)
```

C

```
v = BIN | REOS | 0x82;
ibeos (brd0, v);
```

3. EOS文字受信時の読取停止を不可能化するには第1例のライン100を変更する。

BASICA/QuickBASIC/BASIC

```
:
:
100  v% = EOSV%
```

IBEOS (前ページより続く) IBEOS

QuickBASIC/BASIC

```
.
.
.
sta% = ileos (brd0%, EOSV%)
```

C

```
v= '\n';
.
.
.
ibeos (brd0, v);
```

4. 改行文字が書かれたときにENDメッセージを送る。

BASICA/QuickBASIC/BASIC

```
10   EOSV% = &H0A     ' EOS info for IBEOS.
:
:
100  v% = EOSV% + &H0800
110  CALL ibeos (brd0%, v%)
120  REM  Assume the board has been
130  REM  addressed; do board write.
140  wrt$ = "123" + chr$(&H0A)
150  CALL ibwrt (brd0%, wrt$)
```

QuickBASIC/BASIC

```
EOSV% = &H0A
.
.
.
sta% = ileos (brd0%, EOSV% + &H0800)
sta% = ilwrt (brd0%, "123" + chr$(&H0A), 4)
```

IBEOS (前ページより続く) IBEOS

C

```
v = XEOS | '\n';
ibeos (brd0, v);
ibwrt (brd0, "123\n", 4);
```

5. 改行文字と一緒にENDを送り、改行文字で読取を停止するには、第4例のライン100を次のように変更する。

BASICA/QuickBASIC/BASIC

```
:
:
100  v% = EOSV% + &H0C00
```

QuickBASIC/BASIC

```
sta% = ileos (brd0%, EOSV% + &H0C00)
```

C

```
v = REOS | XEOS | 0x0A;
ibeos (brd0, v);
```

IBEOT　　IBEOT

目的　書込オペレーションにおいてENDメッセージを可能化／不可能化する。

フォーマット

BASICA

```
CALL ibeot (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibeot (ud%, v%)
     あるいは
ileot (ud%, v%)
```

C

```
short ibeot (short ud, short v)
```

udはデバイスあるいはインターフェースボードを指定します。vが0以外であると、一つ一つの書込オペレーションの最後のバイトとともにENDメッセージが送られます。vが0であると、ENDは自動的に送られません。ibeotは構成設定値を変更する場合のみ必要となります。[デフォルト構成ではこの機能は可能化されています。]

ENDメッセージはGPIBのEOI信号線のアサートです。自動END停止メッセージが可能化されていると、データストリングの最後のバイトを示すためにEOS文字を使用する必要が無くなります。ibeotは主として長さが不定のデータを送るために用いられます。

ENDメッセージをEOS文字と一緒に送ることはibeosに依存し、ibeotには依存しません。

ibeotによる割り当ては次にもう1度ibeotが呼び出されるか、ibonlかibfindが呼び出されるか、またはシステムがリスタートされるまで効力を持続します。

IBEOT (前ページより続く) IBEOT

ibeotが呼び出されてエラーが起きなかった場合、自動的ENDメッセージが可能化されていればiberrは1を戻します。自動的ENDメッセージが不可能化されていれば0を戻します。

デバイスIBEOT関数

udがある1つのデバイスを指定していれば、選択されたEND終了メッセージ法はそのデバイスへの全てのデバイス入出力書込オペレーションに使用されます。

ボードIBEOT関数

udがある1つのインターフェースボードを指定していれば、選択されたEND終了メッセージ法は、書込されるデバイスがいずれであるを問わず、全てのボード入出力書込オペレーションで使用されます。

*IBEOS*の項及び表2－2も参照してください。

デバイス関数の例

今後の全ての書込オペレーションにおいて、最後のバイトにはいつもENDメッセージを付ける。

BASICA/QuickBASIC/BASIC

```
100  plotter$ = "DEV5"
110  CALL ibfind (plotter$, plotter%)
120  v% = 1        ' Enable sending of EOI.
130  CALL ibeot (plotter%, v%)
140  REM  It is assumed that wrt$ contains
150  REM  the data to be written to the
160  REM  plotter.
170  CALL ibwrt (plotter%, wrt$)
```

IBEOT (前ページより続く) IBEOT

QuickBASIC/BASIC

```
plotter% = ilfind ("DEV5")
sta% = ileot (plotter%, 1)
sta% = ilwrt (plotter%, wrt$, cnt%)
```

C

```
plotter = ibfind ("DEV5");
ibeot (plotter, 1);
ibwrt (plotter, wrt, cnt);
```

ボード関数の例

1. 最後のバイトにENDを付けて送ることをやめる。

BASICA/QuickBASIC/BASIC

```
100  v% = 0     ' Disable sending of EOI.
110  CALL ibeot (brd0%, v%)
```

QuickBASIC/BASIC

```
sta% = ileot (brd0%, 0)
```

C

```
ibeot (brd0, 0);
```

2. 全ての書込オペレーションにおいて最後のバイトにENDメッセージを付けて送る。

BASICA/QuickBASIC/BASIC

```
100  v% = 1        ' Enable sending of EOI.
110  CALL ibeot (brd0%, v%)
120  REM  It is assumed that wrt$ contains
130  REM  the data to be written and all
140  REM  Listeners have been addressed.
150  CALL ibwrt (brd0%, wrt$)
```

IBEOT (前ページより続く) IBEOT

QuickBASIC/BASIC

```
sta% = ileot (brd0%, 1)
sta% = ilwrt (brd0%, wrt$, cnt%)
```

C

```
ibeot (brd0, 1);
ibwrt (brd0, wrt, cnt);
```

IBEVENT　　IBEVENT

目的　次の事象を返す。

フォーマット

BASICA

```
CALL ibevent (ud%, event%)
```

QuickBASIC/BASIC

```
CALL ibevent (ud%, event%)
    あるいは
ilevent (ud%, event%)
```

C

```
short ibevent (short ud, unsigned *event)
```

udはインターフェースボードを指定します。eventは事象コードを記憶します。

ibevent関数はデバイスクリアとデバイストリガという2つのGPIB事象のうちどちらが起こったかを知るために使用されます。通常この関数はEVENTビットがibsta中にセットされている時に呼び出されます。変数eventには次に示す値のうちの一つを用います。

0＝待ち行列中には事象無し。

1＝デバイスクリアメッセージを受信。

2＝デバイストリガメッセージを受信。

この関数から返った際、ibcntには事象待ち行列の中にある事象の数を含んでいます。

IBEVENT　　(前ページより続く)　　IBEVENT

この関数の代表的使用形態はトーカ／リスナ (T/L)アプリケーション中での使用であって、コントローラアプリケーションでは使用されません。T/Lアプリケーションでは、インターフェースボードがデバイスクリア(Device Clear)とデバイストリガ (Device Trigger) メッセージの受信順序を知ることがしばしば必要になります。ibstaのDCASビットとDTASビットはこれら事象の順番を知るには不十分です。ibconfig関数を使用してibstaのEVENTビットを可能化することにより待ち行列を可能化すると、ドライバがDCAS或いはDTASメッセージを受け取った際、そのメッセージはボードの事象待ち行列の中に記憶され、EVENTビットがibsta中にセットされます。もし入出力動作が進行中であったりすると、その動作はEABOエラーで停止されます。この様にして、アプリケーションプログラムはibeventを呼び出すことによりどの事象が起こったかを知ることが出来ます。この後で、次の入出力動作を行う場合には、事象待ち行列を空にして、EVENTがibsta中に留まらないようにしてください。

事象待ち行列が一杯になっている時にibeventを呼び出すと、ETABエラーが行列中の最も古い事象と共に返ってきます。

ボード関数例

BASICA/QuickBASIC/BASIC

```
100  mask% = &H4400       ' TIMO EVENT
110  CALL ibwait (brd0%, mask%)
120  IF ibsta% and EVENT then GOTO 140
130  STOP
140  ibevent (brd0%, eve t %)
150  REM event% contains the event code.
```

QuickBASIC/BASIC

```
sta% = ilwait (brd0%, &H4400)
IF sta% and EVENT THEN
   sta% = ilevent (brd%, event%)
   ' event% contains the event code.
END IF
```

IBEVENT (前ページより続く) IBEVENT

C

```
ibwait (brd0, TIMO | EVENT);
if (ibsta & EVENT) {
   ibevent (brd0, &event);
   /* event contains the event code */
```

IBFIND IBFIND

目的 デバイスを開き、与えられた名に関係するユニット記述子を返す。

フォーマット

BASICA

```
CALL ibfind (udname$, ud%)
```

QuickBASIC/BASIC

```
CALL ibfind (udname$, ud%)
        あるいは
ud% = ilfind (udname$)
```

C

```
short ibfind (char udname [])
```

udnameはデフォルトまたは構成されたデバイスあるいはボードの名を含むストリングです。udはibfindによって返されたユニット記述子を含む変数です。

ibfindは、一つ一つの関数について、その関数で使用されるデバイスなりボードを識別するために用いる一つの数を返します。ibfindの呼び出しにおいては、アプリケーションプログラム中の一つの変数の名と一つの特定のデバイスあるいはボードの名との関係を設定することが要求されます。udname引き数中で使用された名はデフォルトあるいは構成されたデバイスかボードの名とマッチしている必要があります。このマニュアル中を通じてユニット記述子と呼ばれる数はここでは変数udの中で返されます。

注) ボード呼び出しにおいては、ユニット記述子の代りに整数ボードインデックスの0か1を使うことが出来ます。この特長のおかげで、全てのNI-488ボード関数は第4章で説明されているNI-488.2の手続きと互換性を持つことが出来ます。

IBFIND　　(前ページより続く)　　IBFIND

ibfindは指定されたデバイスあるいはボードを開き、デフォルトと構成値にしたがってソフトウエアパラメータを初期化することに関し、ibonlと等しい働きをします。プログラミングを楽にするために変数の名を実際のデバイスやボードの名に近いものにすることをお勧めします。

ユニット記述子はibonlによってそのデバイスあるいはインターフェースボードがオフラインにされるまで有効性を保ちます。

ibfindの呼び出しが失敗に終わった場合はユニット記述子の代りに一つの負数が返されます。多くの場合失敗はibfindに引き渡されたストリング引き数がデフォルトあるいは構成されたデバイスまたはボードの名と完全にマッチしなかった時に起こります。

デバイス関数の例

DEV4 (デバイス No. 4) と言う名のデバイスのユニット記述子を dvm に割り当てる。

BASICA/QuickBASIC/BASIC

```
100  devname$ = "dEV4"    ' Device name
110                       ' assigned at
120                       ' configuration time.
130  CALL ibfind (devname$, dvm%)
140  IF dvm% < 0 GOTO 1000        ' ERROR ROUTINE
```

QuickBASIC/BASIC

```
dvm% = ilfind ("DEV4")
IF dvm% < 0 GOTO 1000     ' JUMP IF ERROR
```

C

```
if ((dvm = ibfind ("DEV4")) & ERR) error ();
```

IBFIND (前ページより続く) IBFIND

ボード関数の例

ボード"GPIB0"のユニット記述子をbrd0に割り当てる。

BASICA/QuickBASIC/BASIC

```
100  udname$ = "GPIB0"    ' Board name assigned
110                        ' at configuration
120                        ' time.
130  CALL ibfind (udname$, brd0%)
140  IF brd0% < 0 GOTO 1000      ' ERROR ROUTINE
```

QuickBASIC/BASIC

```
brd0% = ilfind ("GPIB0")
IF brd0% < 0 GOTO 1000    ' ERROR ROUTINE
```

C

```
int brd0;

   brd0 = ibfind ("GPIB0");
if (brd0 < 0) error ();
```

IBGTS　　IBGTS

目的　アクティブコントローラ状態から待機状態に移る。

フォーマット

BASICA

```
CALL ibgts (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibgts (ud%, v%)
     あるいは
ilgts (ud%, v%)
```

C

```
short ibgts (short ud, short v)
```

udはインターフェースボードを指定します。vが0以外であると、GPIBボードはアクセプタとしてデータ転送のシャドウハンドシェイクを行います。ENDメッセージが検出されると、GPIBボードはGPIBをNot Ready For Data (NRFD—現在データ受信不能) と言うハンドシェイク延期状態にします。vが0であると、シャドウハンドシェイクもそれの延期も行われません。

ibgts関数はGPIBボードをコントローラ待機の状態にします。もしボードがアクティブコントローラであった場合はATN信号線のアサートを解除します。そしてGPIBコントローラボードを待機状態になるようにします。その結果、GPIBデバイス間のデータの転送がGPIBボードの干渉を受けずに行われるようになります。

シャドウハンドシェイクのオプションが働いている間は、GPIBボードはアクセプタとしてデータのハンドシェイクに参与しています。但し実際にデータを読み取ることはしていません。ボードはデータ転送をモニターし、ENDメッセージがあると、それ以降の転送を一時中止します。このメカニズムにより、GPIBボードはその後 ibcmd

IBGTS (前ページより続く) IBGTS

や ibrpp 等のオペレーションがあった時に同期的にコントロールを得ることが出来ます。

ibgts でシャドウハンドシェイクを行う前には、ibeos 関数を呼び出して正しいEOS文字を成立させるかEOS検出を不可能化する手段をとる必要があります。

GPIBボードがCICでない場合はECICエラーが起こります。

*IBCAC*の項も参照してください。

以下の例では、GPIBコマンドとアドレスは印刷可能なASCII文字の形で符号化されています。

ボード関数例

すべてのリスナのアドレシングを解除（UNLかASCIIの?を使用）した後ATN信号線をオフにする。次にhex 46 (ASCIIのF)にあるトーカとhex 31 (ASCIIの1)にあるリスナにアドレスし、トーカがデータメッセージを送ることが出来るようにする。

BASICA/QuickBASIC/BASIC

```
100   cmd$ = "?F1"          ' UNL MTA1 MLA2
110   CALL ibcmd (brd0%,cmd$)
120   v% = 1        ' Listen in continuous mode.
130   CALL ibgts (brd0%,v%)
```

QuickBASIC/BASIC

```
ilcmd (brd0%,"?F1",3)
sta% = ilgts (brd0%,1)
```

C

```
ibcmd (brd0,"?F1",3);
ibgts (brd0,1);
```

IBIST　　IBIST

目的　パラレルポーリングのためにistをセットするかクリアする。

フォーマット

BASICA

```
CALL ibist (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibist (ud%, v%)
     あるいは
ilist (ud%, v%)
```

C

```
short ibist (short ud, short v)
```

udはインターフェースボードを指定します。v が 0 以外であれば ist はセットされ、v が 0 であれば ist はクリアされます。

ibist 関数は、GPIBボードがもう一つのデバイスがアクティブコントローラとして行うパラレルポーリングに参加する時に使用されます。アクティブコントローラはパラレルポーリングを行うときにEOI信号線をアサートしてIDY (Identify—識別せよ) のメッセージを送ります。このメッセージが働いている間、ポーリングに参加するよう構成されたデバイスの各々は、自身の ist ビット値にしたがって、あらかじめ論理の規定されたGPIBデータ線を真か偽のいずれかにアサートするように応答します。例えば、GPIBボードは、DIO3データ線を正の論理にしたがってist = 1 が真で ist = 0 が偽であるとするようにドライブするよう設定することが出来ますし、反対に、負の論理にしたがって ist = 0 が真で ist = 1 が偽であるようにドライブするようにも出来ます。

ist の値、ドライブされる信号線、及び信号線がドライブする際の意味(1 か 0) の間の関係は一つ一つのデバイス毎にPPE (Parallel Poll Enable—パラレルポーリング可能化) によって決定されます。

IBIST (前ページから続く) IBIST

GPIBボードはPPEメッセージを ibppc 関数を使用してそれ自身でローカルに受信することも出来ますし、キーボードから（リモートで）入力されたアクティブコントローラからのコマンドによって受信することも出来ます。PPE メッセージが実行されると、ibist 関数がパラレルポーリング中に信号線をドライブする際の意味（1か0）を変更します。このようなやり方でGPIBボードは単ビット、デバイス依存性のメッセージをコントローラに送ります。

ibist が呼び出されてエラーが起こらなければ、以前の ist の値が iberr 中に記憶されます。

IBPPC の項と表2－2も参照してください。

ボード関数例

1. ist をセットする。

BASICA/QuickBASIC/BASIC

```
100  v% = 1     ' Any non-zero value will do.
110  CALL ibist (brd0%,v%)
```

QuickBASIC/BASIC

```
sta% = ilist (brd0%,1)
```

C

```
ibist (brd0,1);
```

IBIST (前ページから続く) IBIST

2. ist をクリアする。

BASICA/QuickBASIC/BASIC

```
100  v% = 0
110  CALL ibist (brd0%,v%)
```

QuickBASIC/BASIC

```
sta% = ilist (brd0%,0)
```

C

```
ibist (brd0,0);
```

IBLINES　　IBLINES

目的　GPIBコントロールラインのステータスを返す。

フォーマット

BASICA

```
CALL iblines (ud%, clines%)
```

QuickBASIC/BASIC

```
CALL iblines (ud%, clines%)
        あるいは
illines (ud%, clines%)
```

C

```
short iblines (short ud, unsigned short *clines)
```

udはボード記述子です。有効な (valid) マスクはGPIBコントロールライン状態の情報とともにclinesで返されます。clinesの下位のバイト（ビット0からビット7までが示すマスクによりGPIBインターフェースボードは各GPIBコントロールラインのステータス情報を得ることが出来ます。上位のバイト（ビット8からビット15まで）はGPIBコントロールラインの状態の情報を含んでいます。各バイトのパターンは次の通りです。

7	6	5	4	3	2	1	0
EOI	ATN	SRQ	REN	IFC	NRFD	NDAC	DAV

あるGPIBコントロールラインがアサートされているか否かを知るには、まず下位バイトのうちのしかるべきビットをチェックしてそのラインがモニターできるか否かを確かめます。そのビットがモニターできる場合には（これはしかるべきビットポジションが1であることでわかります）、次に上位バイトでそれに相当するビットをチェックします。もしそのビットが1にセットされていれば、相当するコ

IBLINES (前ページから続く) IBLINES

ントロールラインはアサートされています。もしビットがクリア(0)ならば、コントロールラインはアサートされていません。

iblines が有効なデータを返すことが出来るためには正しく動作する(Well-behaved) IEEE-488バスの存在が必要です。正しく動作するIEEE-488バスとは、それに接続したデバイスが全てIEEE-488規格に合致しているバスのことです。

デバイス／ボード関数例

REN (リモートモード可能化) のテスト

BASICA/QuickBASIC/BASIC

```
100 CALL ibfind ("GPIB0", gpib0%)
110 CALL iblines (gpib0%, clines%)
120 REN% = clines% AND &H10
130 IF REN% <> &H10 THEN GOTO 800
140 REN% = clines% AND ((clines%/256) AND &H10)
150 IF REN% <> &H10 THEN GOTO 900
160 PRINT "REN is asserted!" : STOP
800 PRINT "GPIB board is unable to monitor REN."
810 STOP
900 PRINT "REN is not asserted!" : STOP
920 END
```

IBLINES (前ページから続く) IBLINES

QuickBASIC/BASIC

```
gpib0% = ilfind ("GPIB0")
IF gpib0% AND EERR THEN GOTO ERROR
CALL iblines (gpib0%, clines%)
IF ibsta% AND EERR THEN GOTO ERROR
REN% = clines% AND &H10
IF REN% <> &H10 THEN GOTO BRDERR
REN% = REN% AND (clines%/256)
IF REN% <> &H10 THEN GOTO UNASSERTED
PRINT "REN is asserted." : STOP
ERROR:
 PRINT "GPIB Handler error." : STOP
BRDERR:
 PRINT "GPIB board is unable to monitor REN!"
 STOP
UNASSERTED:
 PRINT "REN is not asserted." : STOP
END
```

C

```
unsigned short clines;

if ((brd0 = ibfind ("GPIB0")) < 0) error();
if ((ibsta = iblines (brd0, &clines)) < 0)
    error();
if (!(clines & 0x10)) {
    printf("GPIB board can't monitor REN!");
    exit();
}
if (clines & 0x1000 {
    printf("REN is asserted.");
    exit();
}
printf("REN is not asserted.");
```

IBLN　　IBLN

目的　バス上にあるデバイスが存在するか否かをチェックする。

フォーマット

BASICA

```
CALL ibln (ud%, pad%, sad%, listen%)
```

QuickBASIC/BASIC

```
CALL ibln (ud%, pad%, sad%, listen%)
        あるいは
illn (ud%, pad%, sad%, listen%)
```

C

```
short ibln (short ud, short pad, short sad, short *listen)
```

udはボードあるいはデバイスの記述子です。pad(リーガル値は0から30まで）はデバイスの1次GPIBアドレスを指定します。sad(リーガル値はhex 60から7eまで、あるいはNO_SADかALL_SAD）はデバイスの2次GPIBアドレスを指定します。

ibln関数は一つのリスナが指定されたGPIBアドレスにある場合に変数listenで0以外の値を返します。

sadパラメータはhex 60から7eまでの値、あるいは定数NO_SADか定数ALL_SADであることに注意してください。リスナの存在のテストには、sad=NO_SADによってGPIB1次アドレシングだけをするか、sad=ALL_SADをセットするときに一つの1次アドレスに関連した全ての2次アドレス（総計31デバイスアドレス）をテストすることが出来ます。後者の場合は、ibln関数は、NDACを待機する前に1次アドレスと2次アドレスを送ります。もしそのリスンフラグが真であれば、そのリスナを見つけるためには1つの1次アドレスに関連した31だけの2次アドレスを探す必要があります。

IBLN (前ページから続く) IBLN

2次アドレスの替りに次に示す2つの特殊な定数を使用することが出来ます。

```
NO_SAD = 0
ALL_SAD = -1
```

BASICA, QuickBASIC, 及びBASICでは、下線(_)の代りにピリオド(.)を用います。したがってこれらの定数は下に示すようになります。

```
NO.SAD
ALL.SAD
```

udがあるデバイスを指定していると、ibln はそのデバイスと関連のあるボード上のリスナを捜してテストします。

IBDEV と *IBFIND* の項も参照してください。

デバイス／ボード関数例

pad 2 と sad 0x60 におけるGPIBリスナを捜してテストする。

BASICA/QuickBASIC/BASIC

```
100  pad% = 2
105  sad% = &H60
110  CALL ibln (ud%,pad%,sad%,listen%)
115  if listen% = 0 THEN RETURN
120  REM  Error:  no device at this address
```

QuickBASIC/BASIC

```
sta% = illn (ud%, 2, &H60, listen%)
if listen% = 0 THEN RETURN
```

C

```
ibsta = ibln (ud, 2, 0x60, &listen);
if (!listen) {
/* No listener found at this address */
}
```

IBLOC　　IBLOC

目的　ローカルモードに移る。

フォーマット

BASICA

```
CALL ibloc (ud%)
```

QuickBASIC/BASIC

```
CALL ibloc (ud%)
     あるいは
illoc (ud%)
```

C

```
short ibloc (short ud)
```

udはデバイスあるいはインターフェースボードを指定します。

REN (Remote Enable) ラインのアサートが ibsre によって解除になっていない限り、全てのデバイス関数は指定されたデバイスを自動的にリモートプログラムモードにします。リモートプログラムモードになったデバイスを次のデバイス関数がそのデバイス上で実行されるまで一時的にローカルモードにするためにibloc が使用されます。

デバイスIBLOC関数

ibloc は ibcmd を呼び出して次のコマンドシーケンスを送り、指定されたデバイスをローカルモードにする。

1. アクセスボードのトークアドレス
2. もし必要ならばアクセスボードの2次アドレス
3. UNL コマンド

IBLOC (前ページから続く) IBLOC

4. デバイスのリスンアドレス

5. もし必要ならば、デバイスの2次アドレス

6. GTL (ローカルモードに移動) コマンド

必要に応じて他のコマンドバイトを送ることもできます。

ボードIBLOC関数

udがインターフェースボードを指定している場合は、ボードがリモートモードにロックイン（固定されている）されていない限り、ボードはローカルメッセージのRTL（ローカルモードに戻れ）によってローカル状態にされます。ステータスワードのLOKビットはボードがロックアウト状態にあるか否かを示します。ibloc 関数はコンピュータが計測器として使用されている時、計測器の前面パネルのRTLスイッチをシミュレートするために使われます。

デバイス関数例

デバイス dvmをローカル状態に戻す。

BASICA/QuickBASIC/BASIC

```
100  CALL ibloc (dvm%)
```

QuickBASIC/BASIC

```
sta% = illoc (dvm%)
```

C

```
ibloc (dvm);
```

IBLOC　　(前ページから続く)　　IBLOC

ボード関数例

インターフェースボード brd0% をローカル状態に戻す。

BASICA/QuickBASIC/BASIC

```
100  CALL ibloc (brd0%)
```

QuickBASIC/BASIC

```
sta% = illoc (brd0%)
```

C

```
ibloc (brd0);
```

IBONL IBONL

目的 デバイスあるいはインターフェースボードをオンライン／オフラインにする。

フォーマット

BASICA

```
CALL ibonl (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibonl (ud%, v%)
     あるいは
ilonl (ud%, v%)
```

C

```
short ibonl (short ud, short v)
```

udはデバイスあるいはインターフェースボードを指定します。vが0以外であるとデバイスあるいはインターフエースボードはオペレーションが出来るように可能化(つまり、オンラインに）されます。vが0であると、リセット(つまり、オフラインに）されます。

デバイスあるいはインターフェースボードがオフラインにされた後ではハンドル(ud)はもはや有効ではありません。再びボードなりデバイスなりにアクセスするには、もう一度ibfindかibdevの呼び出しを実行してボードなりデバイスなりを開かなければなりません。

ibonlをvが0以外の値で呼び出すと、デバイスあるいはインターフエースボードの構成がデフォルトの構成値に戻ります。

IBONL (前ページから続く) IBONL

デバイス関数例

1. デバイスとしてのプロッターを不可能化する。

BASICA/QuickBASIC/BASIC

```
100  v% = 0
110  CALL ibonl (plotter%,v%)
```

QuickBASIC/BASIC

```
sta% = ilonl (plotter%,0)
```

C

```
ibonl (plotter,0);
```

2. 一時的にオフラインになっているデバイスのプロッターを可能化する。

BASICA/QuickBASIC/BASIC

```
100  udname$ = "plotter"
110  CALL ibfind (udname$,plotter%)
120  REM  ibfind automatically places the
130  REM  device online.
```

QuickBASIC/BASIC

```
plotter% = ilfind ("PLOTTER")
```

C

```
plotter = ibfind ("PLOTTER");
```

IBONL (前ページから続く) IBONL

3. デバイス(プロッター) の構成をデフォルト構成値に戻す。

BASICA/QuickBASIC/BASIC

```
100  v% = 1
110  CALL ibonl (plotter%,v%)
```

QuickBASIC/BASIC

```
sta% = ilonl (plotter%,1)
```

C

```
ibonl (plotter,1);
```

ボード関数例

1. インターフェースボード brd0 を不可能化する。

BASICA/QuickBASIC/BASIC

```
100  v% = 0
110  CALL ibonl (brd0%,v%)
```

QuickBASIC/BASIC

```
sta% = ilonl (brd0%,0)
```

C

```
ibonl (brd0,0);
```

IBONL (前ページから続く) IBONL

2. インターフェースボード brd0 を可能化する。

BASICA/QuickBASIC/BASIC

```
100  udname$ = "GPIB0"
110  CALL ibfind (udname$,brd0%)
120  REM  ibfind automatically places board
130  REM  online.
```

QuickBASIC/BASIC

```
brd0% = ibfind ("GPIB0")
```

C

```
brd0 = ibfind ("GPIB0");
```

3. インターフェースボード brd0 の構成をデフォルト構成値に戻す。

BASICA/QuickBASIC/BASIC

```
100  v% = 1
110  CALL ibonl (brd0%,v%)
```

QuickBASIC/BASIC

```
sta% = ilonl (brd0%,1)
```

C

```
ibonl (brd0,1);
```

IBPAD　　IBPAD

目的　　1次アドレスを変更する。

フォーマット

BASICA

```
CALL ibpad (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibpad (ud%, v%)
     あるいは
ilpad (ud%, v%)
```

C

```
short ibpad (short ud, short v)
```

udはデバイスあるいはインターフェースボードを指定します。vは1次GPIBアドレスを指定します。ibpadは構成値を変更したい場合のみ必要となります。

0から1Eにわたる31の有効なGPIBアドレスがありますので、vの下位5ビットが有意です。但しこれらは全部1であってはなりません。vの値が上記の範囲外の場合はEARGエラーが返されます。

ibpad関数により割り当てられたアドレスは、ibpadがもう一度呼び出されるか、ibonlかibfindが呼び出されるか、あるいはシステムがリセットされるまで効力を保ちます。

ibpadが呼び出されてエラーが起きなかった場合は、以前の1次アドレスがiberrに記憶されます。

IBPAD　　(前ページから続く)　　IBPAD

デバイスIBPAD関数

udがデバイスを指定している場合、ibpadはvの値を基にしてトークアドレスとリスンアドレスを決定します。デバイスのリスンアドレスは1次アドレスにhex 20を加算して得られます。また、トークアドレスはhex 40を足して得られます。したがって、1次アドレスのhex 10はリスンアドレスのhex 30とトークアドレスのhex 50に相応します。あるデバイスのGPIBアドレスはそのデバイスの内部でハードウエアのスイッチかソフトウエアプログラムによって設定されます。実際の手続きについては個々のデバイスの説明文書を参照してください。

ボードIBPAD関数

udがボードを指定していると、ibpadはボードをプログラムしてvによって示されたアドレスに応答するようにします。

IBSAD と *IBONL* の項及び表2－2も参照してください。

デバイス関数例

プロッター(plotter)の1次GPIBアドレスをhex Aに変更する。

BASICA/QuickBASIC/BASIC

```
100  v% = &HA
110  CALL ibpad (plotter%,v%)
```

QuickBASIC/BASIC

```
sta% = ilpad (plotter%,&HA)
```

C

```
ibpad (plotter,0xA);
```

IBPAD (前ページから続く) IBPAD

ボード関数例

ボードbrd0の1次GPIBアドレスをhex 7に変更する。

BASICA/QuickBASIC/BASIC

```
100  V% = &H7
110  CALL ibpad (brd0%,V%)
```

QuickBASIC/BASIC

```
sta% = ilpad (brd0%,&H7)
```

C

```
ibpad (brd0,0x7);
```

IBPCT　　IBPCT

目的　コントロールを移す。

フォーマット

BASICA

```
CALL ibpct (ud%)
```

QuickBASIC/BASIC

```
CALL ibpct (ud%)
```
・　あるいは
```
ilpct (ud%)
```

C

```
short ibpct (short ud)
```

udはデバイスを指定します。

ibpct関数はCICの資格を指定されたデバイスへ、そのデバイスに割り当てられたアクセスボードから移します。ボードは自動的にCIDS (Controller Idle State—コントローラアイドル状態) になります。この関数の使用にはデバイスがコントローラの資格を持っていることが前提になります。

ibpct はボードの ibcmd 関数を呼び出して以下のコマンドを送ります。

- UNLコマンド
- アクセスボードのリスンアドレス
- デバイスのトークアドレス
- もしあれば、デバイスの2次アドレス
- TCT(Take Control) コマンド

IBPCT (前ページから続く) IBPCT

必要なら他のコマンドも送ることも出来ます。

IBCMD の項も参照してください。

デバイス関数例

デバイス(ibmxt) にコントロールを引き渡す。

BASICA/QuickBASIC/BASIC

```
100  CALL ibpct (ibmxt%)
```

QuickBASIC/BASIC

```
sta% = ilpct (ibmxt%)
```

C

```
ibpct (ibmxt);
```

IBPPC　　IBPPC

目的　パラレルポーリングのための構成

フォーマット

BASICA

```
CALL ibppc (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibppc (ud%, v%)
     あるいは
ilppc (ud%, v%)
```

C

```
short ibppc (short ud, short v)
```

ud はデバイスあるいはインターフェースボードを指定します。v は有効なパラレルポーリング可能化／不可能化コマンドであるか 0 であるかのいずれかでなければなりません。

ibppc はエラーが起きなかった場合は以前の値で iberr に記憶されたものを返します。

デバイスIBPPC関数

ud がデバイスを指定している場合は、ibppc 関数はデバイスのパラレルポーリングに対する応答を可能化するか不可能化します。

ibppc はボードの ibcmd 関数を呼び出して以下のコマンドを送るようにさせます。

- アクセスボードのトークアドレス
- UNL コマンド

IBPPC (前ページから続く) IBPPC

- デバイスのリスンアドレス
- もしあれば、デバイスの2次アドレス
- PPC (Parallel Poll Configure) コマンド
- PPE (Parallel Poll Enable) コマンドか PPD (Parallel Poll Disable) コマンド

必要ならば、他のコマンドバイトも送ることがあります。

16 の PPE メッセージの一つ一つがGPIBデータライン(DIO1から DIO8 の何れか) とセンス (1あるいは 0) を規定しています。これらはデバイスがパラレルポーリングに応答する際に必要となるものです。デバイスは ist (individual status bit) の現在の値にしたがって、選択されたラインが真でドライブされているか偽でドライブされているかを決定して割り当てられたメッセージを解釈します。例えば、PPE=hex 64 では、DIO5 は ist=0 で真でドライブされており、ist=1 では偽でドライブされています。また、PPE=hex 68 では、DIO1 は ist=1で真、ist=0 で偽でドライブされています。PPD メッセージや 0 値のいずれでもPPEメッセージをキャンセルします。ユーザーはPPEとPPDメッセージのどれが送られるかを知り、応答の意味を決定しなければなりません。

ボードIBPPC関数

ud がインターフェースボードを指定している場合、ボードはその Local Poll Enable (LPE) メッセージを v にセットしてパラレルポーリングに応答します。

IBCMD と *IBIST* の項及び表2－2も参照してください。

IBPPC　　(前ページから続く)　　IBPPC

デバイス機能例

1. データラインDIO5が真である (ist=0) 応答をするようdvmを構成する。

BASICA/QuickBASIC/BASIC

```
100  v% = &H64
110  CALL ibppc (dvm%,v%)
```

QuickBASIC/BASIC

```
sta% = ilppc (dvm%,&H64)
```

C

```
ibppc (dvm,0x64);
```

2. データラインDIO1が真である (ist=1) 応答をするようdvmを構成する。

BASICA/QuickBASIC/BASIC

```
100  v% = &H68
110  CALL ibppc (dvm%,v%)
```

QuickBASIC/BASIC

```
sta% = ilppc (dvm%,&H68)
```

C

```
ibppc (dvm,0x68);
```

IBPPC (前ページから続く) IBPPC

3. dvm のパラレルポーリングのための構成をキャンセルする。

BASICA/QuickBASIC/BASIC

```
100  v% = &H70
110  CALL ibppc (dvm%,v%)
```

QuickBASIC/BASIC

```
sta% = ilppc (dvm%,&H70)
```

C

```
ibppc (dvm,0x70);
```

ボード関数例

ボード brd0 を構成してデータライン DIO5 が真である (ist=0) 応答をするようにする。

BASICA/QuickBASIC/BASIC

```
100  v% = &H64
110  CALL ibppc (brd0%,v%)
```

QuickBASIC/BASIC

```
sta% = ilppc (brd0%,&H64)
```

C

```
ibppc (brd0,0x64);
```

IBRD　　IBRD

目的　デバイスからデータをストリングに読み込む。

フォーマット

BASICA

```
CALL ibrd (ud%, rd$)
```

QuickBASIC/BASIC

```
CALL ibrd (ud%, rd$)
     あるいは
ilrd (ud%, rd$, cnt&)
```

C

```
short ibrd (short ud, char rd [], unsigned long cnt)
```

ud はボードあるいはデバイスを指定します。rd はデータの記憶バッファです。BASICAでは rd$ は255バイト以下です。QuickBASIC と BASIC では、rd$ は32 キロバイトマイナス1 (2の15乗マイナス 1) です。Cでは rd は 4 ギガバイトマイナス1 (2の32乗マイナス1)まで記憶することが出来ます。

ibrd は次の事象のうちの 1 つが起こったときに終了します。

- 割り当てられたバッファが一杯になった時。
- エラーが検出された時。
- 制限時間を超過した時。
- ENDメッセージが検出された時。
- EOS 文字が検出された時 (このオプションが可能化されていることが条件) 。

IBRD (前ページから続く) IBRD

ibrdが完了した時、ibstaに最も時間的に近いデバイスステータスが含まれます。また、ibcntlが読み取られたバイトの数を示します。ibcntは読取バイト数を16ビットで示します。ibstaにエラービットがセットされていると、iberrは最初に検出されたエラーです。

デバイスIBRD関数

udがデバイスを指定している場合、そのデバイスはトーカとしてアドレスされ、アクセスボードはリスナとしてアドレスされます。そして次にデータがデバイスから読み取られます。

ボードIBRD関数

udがボードを指定している場合、ibrd関数はGPIBデバイスから読取を行いますが、この時そのデバイスは既にCICにより正しくアドレスされていることが前提となります。上に列挙したibrd終了条件のほかに、Device Clear (DCL) 及び Selected Device Clear (SDC) コマンドがCICから受信された時もibrdは終了します。

アクセスボードがアクティブコントローラの場合は、ボードは、オペレーションが完了した後も、ATNをオフにして待機コントローラの状態におかれます。アクセスボードがアクティブコントローラでなければ、ibrdは直ちに働き始めます。

ボードがCICの場合は、ibrdの使用前にibcmd関数を使用してトーカとしてのデバイスとリスナとしてのボードにアドレスする必要があります。

ボードがCICであるけれどもibcmd関数からリスナとしてアドレスされていなかった場合はEADRエラーが起こります。また、ibrdが何らかの理由で制限時間内に動作を完了できなかった場合にはEABOエラーが起こります。

IBRD　　(前ページから続く)　　IBRD

デバイス関数例

100バイトのデータをデバイスから読み取る。

BASICA/QuickBASIC/BASIC

```
100  brd% = 0 : pad% = 10 : sad% = 0
110  tmo% = 15 : eot% = 1 : eos% = 0
120  CALL ibdev(brd%,pad%,sad%,tmo%,eot%,eos%,dvm%)
130  rd$ = space$(100)
140  CALL ibrd (dvm%, rd$)
```

QuickBASIC/BASIC

```
dvm% = ildev(0, 10, 0, 15, 1, 0)
rd$ = space$ (100)
sta% = ilrd (dvm%, rd$, 100)
```

C

```
short dvm;
char rd [100];

dvm = ibdev(0, 10, 0, 15, 1, 0);
ibrd (dvm, rd, 100);
```

ボード関数例

1. 100バイトのデータをデバイスのトークアドレスの hex 4C (ASCIIのL) から読み取る。(ボードのリスンアドレスは hex 20 あるいは ASCII の <space> である。)

BASICA/QuickBASIC/BASIC

```
100  brd0$ = "GPIB0"
110  CALL ibfind (brd0$, brd0%)      'open board
120  cmd$ = "? L"                    ' UNL MLA TAD
130  CALL ibcmd (brd0%, cmd$)
140  rd$ = space$(100)
150  CALL ibrd (brd0%, rd$)
```

IBRD (前ページから続く) IBRD

QuickBASIC/BASIC

```
brd0% = ilfind ("gpib0")          'open board
sta% = ilcmd (brd0%, "? L", 3)    'UNL TAD MLA
rd$ = space$ (100)
sta% = ilrd (brd0%, rd$, 100)
```

C

```
short brd0;
char rd [100];

brd0 = ibfind ("gpib0");          /* open board */
ibcmd (brd0, "?L ", 3);           /* UNL TAD MLA */
ibrd (brd0, rd, 100);
```

2. EOS文字で読取を終了させる方法については、*IBEOS* の項のボード関数例 を見てください。

3. バイナリ整数データの自動バイトスワッピングを可能化する方法については、*IBCONFIG* の項のボード関数例を見てください。

IBRDA　　IBRDA

目的　データを非同期でストリングに読み込む。

フォーマット

BASICA

```
CALL ibrda (ud%, rd$)
```

QuickBASIC/BASIC

```
CALL ibrda (ud%, rd$)
     あるいは
ilrda (ud%, rd$, cnt&)
```

C

short ibrda (short ud, char rd [], unsigned long cnt)

udはデバイスかインターフェースボードを指定します。rdはデータ記憶のバッファを示します。BASICAでは rd$ はただの255バイトです。QuickBASIC と BASIC では rd$ は32キロバイト(2の15乗)マイナス1まで記憶することが出来ます。C では rd は4ギガバイト(2の232乗) マイナス1まで記憶することが出来ます。

ibrda は、GPIBの入出力オペレーションが進行中にアプリケーションプログラムが他の機能を実行する必要がある場合に、ibrd に代って使用されます。ibrda は入出力動作が始まるとすぐに戻ります。

ibcmda, ibrda, 及び ibwrta と言う3つの非同期入出力関数は入出力のプロセスが進行中にアプリケーションプログラムによってGPIB機能以外の機能を実行したいときのために用意されたものです。一度この種の非同期入出力関数の呼び出しが始まると、それ以降のデバイスあるいはアクセスボード関係のGPIB呼び出しはその入出力が完了してGPIBソフトウエアとアプリケーションプログラムが再び同期化されるまでは不可能となります。

IBRDA (前ページより続く) IBRDA

再同期化は次の3つの関数のうちの1つを使用して行います。

注) 再同期化は、返された ibsta がCMPLを含んでいなければ成功しません。

- ibwait (マスクにCMPLを含む) – ドライバとアプリケーションが同期化される。
- ibstop – 非同期的な入出力がキャンセルされ、ドライバとアプリケーションが同期される。
- ibonl – 非同期入出力のキャンセル、インターフェースのリセットさ、及びドライバとアプリケーションの同期が行われる。

デバイスあるいはアクセスボードが関係する他の全てのGPIB関数の呼び出しはEOIPエラーを起こします。

デバイスIBRDA関数

ud がデバイスを指定している場合、デバイスはトーカとしてアドレスされ、アクセスボードはリスナとしてアドレスされ、この状態でデータはデバイスから読み出されます。他のコマンドバイトも必要に応じて送ることが出来ます。

ボードIBRDA関数

ud がインターフェースボードを指定している場合、ibrda はGPIBデバイスからの読取を行おうとします。この場合、デバイスは正しい手続きで既にアドレスされていることが必要です。

ボードがCICの場合は、ibrda 呼び出し前にibcmdを呼び出してトーカとしてのデバイスとリスナとしてのボードにアドレスする必要があります。そうしない場合には、実際のCICがアドレスを行わなければなりません。

IBRDA　　(前ページより続く)　　IBRDA

ボードがアクティブコントローラである場合は、読取動作の完了後でも、ボードはひとまずATNをオフにしたコントローラ待機の状態におかれます。ボードがアクティブコントローラでない場合は、ただちに読取動作が始まります。

インターフェースボードがCICであるにかかわらずibcmdを使って自身にリスナとしてアドレスしていなかった場合EADRエラーが起こります。

デバイス関数例

他の仕事をしながら56バイトのデータをテープから読み取る。

BASICA/QuickBASIC/BASIC

```
100  REM  Perform device read.
110  rd$ = space$(56)
120  CALL ibrda (tape%, rd$)
130  REM  Perform other processing here, then
140  REM  wait for I/O completion or a
150  REM  timeout.
160  mask% = &H4100   ' TIMO CMPL
170  CALL ibwait (tape%, mask%)
180  REM  ibsta% indicates how the read
190  REM  terminated: CMPL, END, TIMO, or ERR.
```

QuickBASIC/BASIC

```
rd$ = space$(56)
sta% = ilrda(tape%, rd$, 56)
'perform other processing here
sta% = ilwait(tape%, &H4100)   'TIMO CMPL
```

C

```
char rd[56];
ibrda (tape, rd, 56);
/*  Perform other processing here.      */
ibwait (tape, TIMO | CMPL);
```

IBRDA (前ページより続く) IBRDA

ボード関数例

1. トークアドレスの hex 4C (ASCIIのL) のデバイスから56バイトのデータを読み出す。(ボードのリスンアドレスは hex 20 或いは ASCII <space>。)

BASICA/QuickBASIC/BASIC

```
100  REM  Perform addressing in preparation
110  REM  for board read.
120  cmd$ = "? L"          ' UNL MLA TAD
130  CALL ibcmd (brd0%, cmd$)
140  REM  Perform board read.
150  rd$ = space$(56)
160  CALL ibrda (brd0%, rd$)
170  REM  Perform other processing here, then
180  REM  wait for I/O completion or timeout.
190  mask% = &H4100        ' TIMO CMPL
200  CALL ibwait (brd0%, mask%)
210  REM  ibsta% indicates how the read
220  REM  terminated:  CMPL, END, TIMO, or ERR.
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, "? L", 3)
rd$ = space$(56)
sta% = ilrda (brd0%, rd$, 56)
'perform other processing here
sta% = ilwait (brd0%, &H4100)
```

C

```
char rd[56];
ibcmd (brd0, "? L", 3);    /*   UNL MLA TAD      */
ibrda (brd0, rd, 56);
/*  Perform other processing here.               */
ibwait (brd0, TIMO | CMPL);
```

2. 読取をEOS文字が出たところで終了する方法については、*IBEOS* の項のボード関数例を参照してください。

IBRDA (前ページより続く) IBRDA

3. バイナリ整数データの自動バイトスワッピングの可能化手続きについては、*IBCONFIG* の項のボード関数例を参照してください。

IBRDF　IBRDF

目的　GPIBから呼び出したデータをファイルに移す。

フォーマット

BASICA

```
CALL ibrdf (ud%, flname$)
```

QuickBASIC/BASIC

```
CALL ibrdf (ud%, flname$)
```
あるいは
```
ilrdf (ud%, flname$)
```

C

```
short ibrdf (short ud, char flname [])
```

udはデバイスあるいはインターフェースボードを指定します。flnameはデータがファイルとして記憶されるときのファイル名です。flnameはドライブとパス（path)の指定をいれて50文字までの長さです。

ibrdfはファイルをバイナリファイルとして（文字ファイルとしてでなく）開きます。ファイルが存在していない場合はibrdfがファイルを作成します。最後にはibrdfがファイルを閉じます。

ibrdfが指定されたファイルを開いたり、閉じたり、作成したり、捜したり、ファイルに書き込んだりすることが出来なかった場合はEFSOエラーが起こります。

ibrdfの終了は次の事象の何れかによって起こります。

- エラーの検出。
- 時間制限を超過した場合。

IBRDF　　(前ページより続く)　　IBRDF

- ENDメッセージが検出された時。
- EOS文字が検出された時(EOSオプションが可能化されている場合に限ります。)
- DCL (Device Clear) あるいはSDC (Selected Device Clear) コマンドがCICであるもう一つのデバイスから受信された時。

終了後のibcntlは読み取られたバイトの数です。また、ibcntは読み取られたバイト数を16ビットで表わしたものです。

デバイスibrdf関数が戻された時、ibstaは最新のデバイスステータスを含んでいます。ibcntlは読み取られたバイトの数であり、ibcntは読み取られたバイト数を16ビットで表わしたものです。ibstaにERRビットがセットされている場合は、iberr は最初に検出されたエラーです。

デバイスIBRDF関数

udがデバイスを指定する場合は、デバイスibrd関数の場合と同じボード関数が自動的に実行されます。ibrdf関数はibrdと同じ様な条件で終了されます。

ボードIBRDF関数

udがインターフェースボードを指定していると、ボードibrd関数がGPIBデバイスからの読取を行います。この場合GPIBデバイスはあらかじめ正しくアドレスされていることが前提となります。

ボードがCICであるけれども、あらかじめibcmd関数によりリスナとしてアドレスされていないと、EADRエラーが起こります。何らかの理由により、読取が制限時間内に完了しない場合にはEABOエラーが起こります。また、EABOエラーはトーカになるはずのデバイスがアドレスされていない場合及び／或いは何らかの理由で読取が制限時間内に完了しなかった場合にも起こります。

IBRDF (前ページより続く) IBRDF

デバイス関数例

rdr%なるデバイスからBドライブにあるRDGSなるファイルにデータを読み込む。

BASICA/QuickBASIC/BASIC

```
110  flname$ = "B:RDGS"
120  CALL ibrdf (rdr%, flname$)
130  REM  ibsta% and ibcnt% show the results of
140  REM  the read operation.
```

QuickBASIC/BASIC

```
sta% = ilrdf (rdr%, "B:RDGS")
```

C

```
ibrdf (rdr, "B:RDGS");
```

ボード関数例

1. トークアドレス &H4C (ASCII の L) なるデバイスからデータを現在使用中のディスクドライブにあるRDGSなるファイルに読込み、次に全てのアドレシングを解除する。(GPIBボードのリスンアドレスは hex 20またはASCIIの <space> です。)

BASICA/QuickBASIC/BASIC

```
100  REM  Perform addressing in preparation
110  REM  for board read.
120  cmd$ = "?L "     ' UNL TAD MLA
130  CALL ibcmd (brd0%, cmd$)
140  REM  Perform board read.
150  flname$ = "RDGS"
160  CALL ibrdf (brd0%, flname$)
170  REM  ibsta% and ibcnt% show the results of
180  REM  the read operation.
```

IBRDF (前ページより続く) IBRDF

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%, "?L ", 3)  'UNL TAD MLA
sta% = ilrdf (brd0%, "RDGS")
```

C

```
ibcmd (brd0, "?L ", 3);
ibrdf (brd0, "RDGS");
```

2. バイナリ整数データの自動バイトスワッピングを可能化する手順については*IBCONFIG*の項のボード関数例を参照してください。

IBRDI　　IBRDI

目的　データを整数配列に読み込む。

フォーマット

BASICA

```
CALL ibrdi (ud%, iarr%(0), cnt%)
```

QuickBASIC/BASIC

```
CALL ibrdi (ud%, iarr%(), cnt&)
        あるいは
ilrdi (ud%, iarr%(), cnt&)
```

ibrdiはCにおいては必要がないため供給されていません。Cにおいては、ibrdを使用すればあらゆる型のバッファにデータを記憶することが十分できます。一方、BASICA, QuickBASIC及びBASICにおいては型の規則がより厳格なため、配列バッファにはストリングの場合とは違った関数が必要となります。

ud%はデバイス或いはインターフェースボードを指定します。iarr%はデータが読み込まれる整数配列です。cntは読み込まれるバイト数の最大限度を指定します。

ibrdiは文字列変数にデータを読み込むibrdに似ています。ibrdiは64キロバイト（2の16乗バイト）マイナス1までのデータを読み込むことが出来ます。データが読まれる際は、各バイト対は一つの整数として扱われ、iarr%似記憶されます。

ibrdと違い、ibrdiはデータを直接整数配列中に記憶しますので、演算のためにデータを整数に変換する必要はありません。

QuickBASICと BASICでは、配列は single, double或いは long 型をとることも出来ます。デバイスがこれらの型の何れとも合うバイナリデータを返せば、データは直接配列中に読み込むことが出来ます。

IBRDI　　(前ページより続く)　　IBRDI

IBRD の項及び本章の初めの「BASICA/QuickBASIC/BASIC/Cによる NI-488 の入出力関数呼び出し」も参照してください。

デバイス関数例

512バイトのデータをテープから読み出して整数配列のrdに記憶する。

BASICA

```
100 brd% = 0 : pad% = 6 : sad% = 0
110 tmo% = 14 : eot% = 1 : eos% = 0
120 CALL ibdev(brd%,pad%,sad%,tmo%,eot%,eos%,tape%)
130 cnt% = 512
140 REM  Array size is equal to cnt% divided by 2
150 DIM rd% (256)
160 CALL ibrdi (tape%, rd%(0), cnt%)
```

QuickBASIC/BASIC

ライン160を

```
CALL ibrdi (tape%, rd%(), cnt%)
```

あるいは

```
DIM rd% (256) 'integer array size = byte cnt/2
CALL ibdev(0,6,0,14,1,0,tape%) 'open device
sta% = ilrdi (tape%, rd%(), 512)
```

IBRDI (前ページより続く) IBRDI

ボード関数例

1. トークアドレス hex 4C (ASCIIの L) のデバイスから56バイトのデータを整数配列の rd% に読み込む。(GPIBボードのリスンアドレスは hex 20 あるいは ASCII の<space>)。

BASICA

```
100  gpib$ = "GPIB0"
110  CALL ibfind (gpib0$, brd0%)
120  cmd$ = "? L"       ' UNL MLA TAD
130  CALL ibcmd (brd0%, cmd$)
140  cnt% = 56
150  REM  Array size = cnt% divided by 2.
160  DIM rd% (28)
170  CALL ibrdi (brd0%, rd%(0), cnt%)
180  REM  ibsta% shows how the read terminated:
190  REM  on CMPL, END, TIMO, or ERR.
200  REM  Data is stored in rd%().
```

QuickBASIC/BASIC

ライン170を

```
CALL ibrdi (brd0%, rd%(), cnt%)
```

あるいは

```
DIM rd%(28) 'byte count / 2
brd0% = ilfind ("gpib0")
sta% = ilcmd (brd0%, "? L", 3) 'UNL MLA TAD
sta% = ilrdi (brd0%, rd%(), 56)
```

2. EOS文字で読みを終了させる方法については *IBEOS* の項のボード関数例を参照してください。

3. バイナリ整数データの自動バイトスワッピングの可能化については *IBCONFIG* の項のボード関数例を参照してください。

IBRDIA　　IBRDIA

目的　整数配列に非同期にデータを読み込む。

フォーマット

BASICA

```
Call ibrdia (ud%, iarr%(0), cnt%)
```

QuickBASIC/BASIC

```
Call ibrdia (ud%, iarr%(), cnt&)
        あるいは
ilrdia (ud%, iarr%(), cnt&)
```

Cの ibrdia は必要でないので供給されていません。Cにおいては ibrda 呼び出しを用いれば如何なる型のバッファでも十分にデータを受け入れることが出来ます。BASICA, QuickBASICおよびBASICにおいては、型の規定はより厳しいので、配列バッファの場合には文字列の場合とは異なった関数が必要となります。

udはデバイスかインターフェースボードを指定します。iarr%はデータが読み込まれる整数配列です。cntは読み込まれるバイトの最大数を指定します。

ibrdia はデータを文字列変数に読み込むibrda関数に似ています。ibrdiaは最大64キロバイト((2の16乗) マイナス1のデータをudから非同期的に読出し、iarr中に記憶することが出来ます。データが読まれている間、バイト対は整数として扱われ、iarrの中に記憶されます。

ibrdaと違って、ibrdiaはデータを直接整数配列の中に記憶します。したがって演算処理のために整数変換を行う必要がありません。

QuickBASIC と BASICでは配列には single, double 及び long の型があります。したがって、デバイスがこれらの型の何れかと互換なバイナリデータを返した場合は、データを直接配列に読み込むことが可能です。

IBRDIA (前ページより続く) IBRDIA

注) QuickBASIC と BASICの場合、ダイナミック配列を非同期的関数のibrdia と ibwrtiaに引き渡すことは入出力動作中にそれらのロケーションが変わることがあるので避けてください。

IBRDA の項及び本章の初めの「BASICA/QuickBASIC/BASIC/Cによる NI-488 の入出力関数呼び出し」も参照してください。

デバイス関数例

一方で他の処理を行いながら、テープから56バイトのデータを読み出して整数配列 rdに移す。

BASICA

```
100  REM  Perform device read.
110  cnt% = 56
120  REM  Array size is equal to cnt% divided
130  REM  by 2.
140  DIM rd% (28)
150  CALL ibrdia (tape%,rd%(0),cnt%)
160  MASK% = &H4100  ' TIMO CMPL
170  REM  Perform other processing here, then
180  REM  wait for I/O completion or a
190  REM  timeout.
200  CALL ibwait (tape%, mask%)
210  REM  ibsta% shows how the read terminated:
220  REM  CMPL, END, TIMO, or ERR.
230  REM  If CMPL or ERR is not set, continue
240  REM  processing.
250  IF (ibsta% AND &H8100) = 0 GOTO 160
```

IBRDIA (前ページより続く) IBRDIA

QuickBASIC/BASIC

ライン150を

```
CALL ibrdia (tape%, rd%(), cnt%)
```

あるいは

```
DIM rd% (28) 'space to store 56 bytes
sta% = ilrdia (tape%, rd%(), 56)
again:
sta% = ilwait (tape%, &H4100)
IF (sta% AND &H8100) THEN GOTO again
```

IBRDIA (前ページより続く) IBRDIA

ボード関数例

1. トークアドレス hex 4C (ASCII の L) のデバイスから整数配列 rd%に56バイトのデータを読み込む。

BASICA

```
100  REM  Perform addressing in preparation
110  REM  for board read.
120  cmd$ = "? L"       ' UNL MLA TAD
140  CALL ibcmd (brd0%, cmd$)
150  REM  Perform board read.
160  cnt% = 56
170  REM  Array size is equal to cnt% divided
180  REM  by 2.
190  DIM rd% (28)
200  CALL ibrdia (brd0%, rd%(0), cnt%)
210  mask% = 0
220  REM  Perform other processing here, then
230  REM  wait for I/O completion or a
240  REM  timeout.
250  CALL ibwait (brd0%, mask%)
260  REM  ibsta% shows how the read terminated:
270  REM  CMPL, END, TIMO, or ERR.
280  REM  If CMPL or ERR are not set,
290  REM  continue processing.
300  IF (ibsta% AND &H8100) = 0 GOTO 200
310  REM  Properly terminate the asynchronous I/O
320  mask% = &H4100  '  TIMO CMPL
330  CALL ibwait (brd0%, mask%)
340  REM  Data is stored in RD%.
```

IBRDIA　　(前ページより続く)　　IBRDIA

QuickBASIC/BASIC

ライン200を

```
CALL ibrdia (brd0%, rd%(), cnt%)
```

あるいは

```
sta% = ilcmd (brd0%, "? L", 3)   'UNL MLA TAD
DIM rd% (28)
sta% = ilrdia (brd0%, rd%(), 56)
again:
  'do other processing here...
  sta% = ilwait (brd0%, 0)
  IF (ibsta% AND &H8100) = 0 GOTO again 'ERR CMPL
  'The I/O is now complete (check ibsta for any errors).
```

2. EOS文字で読みを終了する方法については、*IBEOS* の項のボード関数例を参照のこと。

3. 3. バイナリ整数データの自動バイトスワッピングの可能化については*IBCONFIG* の項のボード関数例を参照のこと。

IBRPP　IBRPP

目的　パラレルポーリングを行う。

フォーマット

BASICA

```
CALL ibrpp (ud%, ppr%)
```

QuickBASIC/BASIC

```
CALL ibrpp (ud%, ppr%)
     あるいは
ilrpp (ud%, ppr%)
```

C

```
short ibrpp (short ud, char *ppr)
```

udはデバイス或いはインターフェースボードを指定します。pprはパラレルポーリングの応答を記憶します。

デバイスIBRPP関数

udがある1つのデバイスを指定していると、そのGPIB上の全てのデバイスは上記のデバイスのアクセスボードを使用したパラレルポーリングを受けます。これは適当なアクセスボードを指定した上にボードibrpp関数を実行することにより可能となります。

ボードIBRPP関数

udがボードを指定していると、ibrpp関数は指定されたボードがあらかじめ構成されたデバイスに対してパラレルポーリングを行うようにします。この場合ボードはIDYメッセージを送り(この時ATNとEOIがともにアサートされています) GPIBのデータラインからの応答を読み取ります。

IBRPP　　(前ページより続く)　　IBRPP

GPIBボードがCICでなかった場合はECICエラーが起こります。GPIBボードが待機コントローラであった場合は、ボードはポーリングに先立ってコントロールを獲得し、ATNをアサートし、アクティブになります。それ以後ボードはアクティブコントローラの状態を保ちます。

バス拡張器を持つバスでパラレルポーリングを実行しているあいだは、拡張器はポーリングが終了するまでに離れたデバイスからの応答を返すことができません。バス拡張器がアプリケーションから見えない形で動作できるようにするために、ドライバが長いポーリング期間で動作するように構成することができます。ポーリング期間はIBCONFでセットすることもできるし、実行中にibconfig関数を呼び出して設定することもできます。

次に示す各例においていくつかのGPIBコマンドとアドレスは印刷可能のASCII文字として符号化されています。印刷可能のASCII文字の使用はこれらの値を指定する最も簡単な方法です。そこで、ASCII文字を出来るだけ使用することをお勧めします。数値のASCII文字への変換については付録Aを御覧ください。

パラレルポーリングに関するコマンドのいくつかを表5－7に示します。

表5－7 パラレルポーリング用コマンド

コマンド	16進数値	意味
PPC	05	パラレルポーリング構成
PPU	15	パラレルポーリング構成解除
PPE	60	パラレルポーリング可能化
PPD	70	パラレルポーリング不可能化

パラレルポーリング定数は適当な宣言ファイル中で定義されます。

IBRPP (前ページより続く) IBRPP

デバイス関数例

キーボードから[リモートモードで] デバイスlcrmtrを構成してDIO3においてistが1なら正の応答をするようにして、次に構成した全てのデバイスにパラレルポーリングを行う。

BASICA/QuickBASIC/BASIC

```
100  v% = &H6A
110  CALL ibppc (lcrmtr%, v%)
120  CALL ibrpp (lcrmtr%, ppr%)
```

QuickBASIC/BASIC

```
sta% = ilppc (lcrmtr%, &H6A)
sta% = ilrpp (lcrmtr%, ppr%)
```

C

```
ibppc (lcrmtr, 0x6A);
ibrpp (lcrmtr, &ppr);
```

ボード関数例

1. キーボードから[リモートで] リスンアドレス hex 23 (ASCII の #) のボードbrd0 を構成してDIO3 においてistが1の時に正の応答をするようにし、次に構成された全てのデバイスをパラレルポーリングする。

BASICA/QuickBASIC/BASIC

```
100  REM  Send UNL, TAD, LAD, PPC, PPE, and UNL.
110  cmd$ = "?@#" + chr$(&H05) + "j?"
120  CALL ibcmd (brd0%, cmd$)
130  CALL ibrpp (brd0%, ppr%)
```

IBRPP (前ページより続く) IBRPP

QuickBASIC/BASIC

```
cmd$ = "?@#" + chr$(&H05) + "j?"  'UNL TAD LAD PPC PPE
UNL
sta% = ilcmd (brd0%, cmd$, 6)
sta% = ilrpp (brd0%, ppr%)
```

C

```
cmd[0] = UNL;
cmd[1] = 0x40
cmd[2] = 0x23;
cmd[3] = PPC;
cmd[4] = PPE | S | 2;
cmd[5] = UNL;
ibcmd (brd0, cmd, 6);
ibrpp (brd0, &ppr);
```

2. PPU (hex 15) コマンドを使用して全てのGPIBデバイスをパラレルポーリング構成から解除し不可能化する。

BASICA/QuickBASIC/BASIC

```
100  cmd$ = chr$(&H15)          ' PPU
110  CALL ibcmd (gpib0%, cmd$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (gpib0%, chr$(&H15), 1)
```

C

```
ibcmd (gpib0, "\x15", 1);
```

IBRSC　　IBRSC

目的　システムコントロールの要求または解除

フォーマット

BASICA

```
CALL ibrsc (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibrsc (ud%, v%)
     あるいは
ilrsc (ud%, v%)
```

C

```
short ibrsc (short ud, short v)
```

udはインターフェースボードを指定します。vが0以外であれば、これから後にシステムコントローラの資格を要求する関数の使用が可能です。一方vが0であれば、今後のシステムコントローラの資格を要求する関数の使用は許されません。

GPIBボードは ibsicと ibsre の各関数を使用してIFC(インターフェースをクリア) と REN (リモートモードを可能化) メッセージを各々送ることが出来ますが、ibsre関数は、このGPIBボードの資格を可能化したり不可能化したりするために使用します。インターフェースボードは、もしそれがシステムコントローラであると、もう一つのコントローラから送ってきたIFC に応答することが出来ません。

たいていのアプリケーションプログラムではGPIBボードは初めから終までシステムコントローラであるのが普通です。しかし、GPIBが一度もシステムコントローラにならないようなアプリケーションプログラムもあります。何れにせよ、コンピュータがプログラムの実行中にシステムコントローラにならない場合に限ってibrsc関数が使用されます。IEEE-488規格ではシステムコントロールが一つのデ

IBRSC　　(前ページより続く)　　IBRSC

バイスからもう一つのデバイスに動的に引き渡されるという概念を規定していませんが、ibrsc関数を使用すると、この概念を実際に適用することが出来ます。

ibrscの呼び出しに対してエラーが起こらなかった時、インターフェースボードがそれまでシステムコントローラであった場合はiberrは1にセットされます。そうでなかった場合は0にセットされます。

ボード関数例

インターフェースボードbrd0が現在システムコントロールでないのであれば、そうなるように要求する。

BASICA/QuickBASIC/BASIC

```
100  v% = 1        ' Any non-zero value will do.
110  CALL ibrsc (brd0%, v%)
```

QuickBASIC/BASIC

```
sta% = ilrsc (brd0%, 1)
```

C

```
ibrsc (brd0, 1);
```

IBRSP　　IBRSP

目的　シリアルポールバイトを返す。

フォーマット

BASICA

```
CALL ibrsp (ud%, spr%)
```

QuickBASIC/BASIC

```
CALL ibrsp (ud%, spr%)
     あるいは
ilrsp (ud%, spr%)
```

C

```
short ibrsp (short ud, char *spr)
```

udはデバイスを指定します。sprはシリアルポールの応答を記憶します。

ibrsp関数は一つのデバイスをシリアルポーリングしてそのステータスバイトを得るため、或いは以前に記憶されたステータスバイトを得るために使用されます。もし応答ビットのうちビット6 (hex 40のビット) がセットされていれば、デバイスがサービス要求をしているということです。

自動シリアルポーリングが可能化されている場合には、指定されたデバイスは既にポーリングされている可能性があります。ポーリングが既になされていて、正の応答が得られた場合は、ibstaのRQSビットがそのデバイスについてセットされます。この状況では、ibrspは以前に得られたステータスビットを返します。ibstaのRQSビットが自動ポーリング中にセットされない場合は、ibrspはそのデバイスをシリアルポールします。

IBRSP　　(前ページより続く)　　IBRSP

ポーリングが実際に行われる際の手続きは次のステップからなっています。

1. UNLコマンド
2. コントローラのリスンアドレス
3. もしあれば、アクセスボードの2次アドレス
4. シリアルポーリングの可能化(SPE)
5. デバイスのトークアドレス
6. もしあれば、デバイスの2次アドレス
7. デバイスからのシリアルポーリング応答バイトの読み取り
8. シリアルポーリングの不可能化(SPD)
9. 必要ならば、他のコマンドを送ることも可

応答バイトのsprの各ビットはRQSビットを除いてデバイスにより変わることになっています。例えば、ポーリングされたデバイスは応答バイト中のある特定のビットを転送するデータの存在を示すためにセットし、もう一つのビットをプログラミングのやり直しの必要を示すためにセットすることがありえます。応答バイトの解釈については個々のデバイスの説明文書を参照してください。

IEEE-488.1規格はシリアルポーリング中にコントローラがデバイスからの応答バイトを待つ時間を規定していません。ドライバーのデフォルトの1秒は大抵のデバイスの場合は十分であるはずです。しかし、それを延長したいときは、IBCONFかibconfigを使用してください。

また、*IBCMD*と*IBRD*の項も参照してください。

IBRSP (前ページより続く) IBRSP

デバイス関数例

デバイスのテープからシリアルポーリング応答 (spr) バイトを得る。

BASICA/QuickBASIC/BASIC

```
100  CALL ibrsp (tape%, spr%)
110  REM  The application program would then
120  REM  analyze the response in spr%.
```

QuickBASIC/BASIC

```
sta% = ilrsp (tape%, spr%)
```

C

```
ibrsp (tape,&spr);
```

IBRSV　　IBRSV

目的　サービスの要求及び/あるいはシリアルポーリングステータスバイトのセット或いは変更

フォーマット

BASICA

```
CALL ibrsv (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibrsv (ud%, v%)
     あるいは
ilrsv (ud%, v%)
```

C

```
short ibrsv (short ud, short v)
```

udはインターフェースボードを指定します。vはGPIBボードがGPIBのCICであるもう一つのデバイスからシリアルポーリングを受けたとき与えたステータスバイトです。ビット6 (hex 40)がセットされると、GPIBボードはGPIB SRQ線をアサートしてさらにコントローラからのサービスを要求します。

ibrsv関数はSRQ信号によりコントローラからのサービスを要求し、コントローラがGPIBボードにシリアルポーリングを行ったときシステム依存性のステータスバイトを得られるようにします。

ibrsvの呼び出しにおいてエラーが起きなければ、それ以前のvの値がiberrに記憶されます。

ibstaのSPOLLビットがibconfig関数によって可能化されていれば、そのSPOLLビットはこの関数の呼び出しのあとでibstaによりクリアされます。ボードがコントローラによりシリアルポールされるとき、第6ビットがステータスバイト (v) にセットされていると、SPOLLビットはibsta中にセットされます。

IBRSV (前ページより続く) IBRSV

ボード関数例

1. シリアルポーリングのステータスバイトをhex 41 にセットすると、それは同時に外部のCICからのサービスを要求する。

BASICA/QuickBASIC/BASIC

```
100  stb% = 1
110  v% = &H40 OR stb%          'assert SRQ
120  CALL ibrsv (brd0%, v%)
```

QuickBASIC/BASIC

```
stb% = 1
sta% = ilrsv (brd0%, stb% OR &H40)
```

C

```
stb% = 1
ibrsv (brd0, stb | 0x41);
```

2. サービスを要求せずにステータスバイトを変更する。

BASICA/QuickBASIC/BASIC

```
100  stb% = &H23  ' New status byte value.
110  CALL ibrsv (brd0%, stb%)
```

QuickBASIC/BASIC

```
sta% = ilrsv (brd0%, &H23)
```

C

```
ibrsv (brd0, 0x23);
```

IBSAD　　IBSAD

目的　2次アドレスを変更するか不可能化する。

フォーマット

BASICA

```
CALL ibsad (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibsad (ud%, v%)
     あるいは
ilsad (ud%, v%)
```

C

```
short ibsad (short ud, short v)
```

udはデバイス或いはインターフェースボードを指定します。vがhex 60とhex 7Eの間の数であると、その数はデバイスかインターフェースボードの2次GPIBアドレスとなります。もしvがhex 0であると2次アドレシングは不可能化されます。ibsadは構成時に設定された2次アドレス値を変更する時だけ必要になります。

この関数により割り当てられた値はもう一度ibsadが呼び出されるか、ibonl或いはibfindが呼び出されるか、またはシステムがリスタートするまで有効です。

ibsadが呼び出されてエラーが起きない場合は、以前の2次アドレスがiberr中に記憶されます。

デバイスIBSAD関数

udがデバイスを指定していると、この関数はデバイスに対する拡張GPIBアドレシングを可能化または不可能化します。2次アドレシングが可能化されると、そのデバイスの指定された2次GPIBアドレ

IBSAD (前ページより続く) IBSAD

スはそれ以降のデバイス入出力機能において自動的に送られるようになります。

ボードIBSAD関数

udがインターフェースボードを指定している場合は、ibsadは拡張GPIBアドレシングを可能化ないしは不可能化し、可能化の場合はGPIBボードの2次アドレスを割り当てます。

IBPAD と *IBONL* の項、及び表2－2も参照してください。

デバイス関数例

1. プロッターの2次GPIBアドレスを現在の値から hex 6Aに変更する。

   BASICA/QuickBASIC/BASIC

   ```
   100  v% = &H6A
   110  CALL ibsad (plotter%, v%)
   ```

   QuickBASIC/BASIC

   ```
   sta% = ilsad (plotter%, &H6A)
   ```

   C

   ```
   ibsad (dvm, 0x6A);
   ```

2. デバイスdvmへの2次アドレシングを不可能化する。

   BASICA/QuickBASIC/BASIC

   ```
   100  v% = 0
   110  CALL ibsad (dvm%,v%)
   ```

IBSAD (前ページより続く) IBSAD

QuickBASIC/BASIC

```
sta% = ilsad (dvm%,0)
```

C

```
ibsad (dvm,0);
```

ボード関数例

1. インターフェースボードbrd0 の2次GPIBアドレスを現在の値からhex 6A に変更する。

BASICA/QuickBASIC/BASIC

```
100  v% = &H6A
110  CALL ibsad (brd0%,v%)
```

QuickBASIC/BASIC

```
sta% = ilsad (brd0%,&H6A)
```

C

```
ibsad (brd0,0x6A);
```

2. インターフェースボードbrd0への2次アドレシングを不可能化する。

BASICA/QuickBASIC/BASIC

```
100  v% = 0
110  CALL ibsad (brd0%,v%)
```

IBSAD (前ページより続く) IBSAD

QuickBASIC/BASIC

```
sta% = ilsad (brd0%,0)
```

C

```
ibsad (brd0,0);
```

IBSIC　　IBSIC

目的　IFC信号を100 µsecの間 送信する。

フォーマット

BASICA

```
CALL ibsic (ud%)
```

QuickBASIC/BASIC

```
CALL ibsic (ud%)
     あるいは
ilsic (ud%)
```

C

```
short ibsic (short ud)
```

udはインターフェースボードを指定します。ボード関数が使用されるプログラムではibsicをプログラムの初めで用いる必要があります。
GPIBボードがシステムコントローラである場合には、ibsic関数は少なくとも 100 µsecの間 IFC信号線をアサートします。この動作はGPIBを初期化し、ATNをアサートしてインターフェースボードをCICとアクティブコントローラとします。また、ibsicは一般にバスの故障の疑いのある場合にも使用されます。

IFC信号はバスデバイスのGPIBインターフェース機能のみをリセットします。それは内部のデバイス機能はリセットしません。デバイス機能はDCL(デバイスクリア) と SDC (選択されたデバイスをクリア) のコマンドによってリセットされます。これらのメッセージの効果を確かめるには、各デバイスの説明文書を参照してください。

GPIBボードがシステムコントローラの資格を持っていない場合は、ESACエラーが起こります。

IBSIC (前ページより続く) IBSIC

*IBRSC*の項も参照してください。

ボード関数例

ボードは、プログラムの初めにGPIBを初期化し、CICとアクティブコントローラとなる。

BASICA/QuickBASIC/BASIC

```
100  CALL ibsic (brd0%)
```

QuickBASIC/BASIC

```
sta% = ilsic (brd0%)
```

C

```
ibsic (brd0);
```

IBSRE　　IBSRE

目的　REN信号線をセット或いはクリアする。

フォーマット

BASICA

```
CALL ibsre (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibsre (ud%, v%)
     あるいは
ilsre (ud%, v%)
```

C

```
short ibsre (short ud, short v)
```

udはインターフェースボードを指定します。vが0以外であるとRENがアサートされ、0であると信号のアサートが解除されます。

ibsre関数はREN信号をオンにしたりオフにしたりします。RENはデバイスがローカルモードの動作とリモートモードの動作を選択する場合に使用されます。デバイスはそれがリスンアドレスを得るまでは実際にリモートモードに入ることは出来ません。

GPIBボードがシステムコントローラでないとESACエラーが起きます。

ibsreが呼び出されてエラーが起きなかった場合は、それ以前のREN状態がiberr中に記憶されます。

*IBRSC*の項と表2－2も参照してください。

IBSRE (前ページより続く) IBSRE

ボード関数例

1. リスンアドレスの hex 23 (ASCII の #) にあるデバイスをリモートモードにする。

   BASICA/QuickBASIC/BASIC

```
100  v% = 1      ' Any non-zero value will do.
110  CALL ibsre (brd0%,v%)
120  cmd$ = "#" ' LAD
130  CALL ibcmd (brd0%,cmd$)
```

   QuickBASIC/BASIC

```
sta% = ilsre (brd0%,1)
sta% = ilcmd (brd0%,"#",1)
```

   C

```
ibsre (brd0,1);
ibcmd (brd0,"#",1);
```

2. デバイスがローカルモードに戻る可能性をなくすために、ローカルロックアウト(LLOあるいは hex 11) コマンドを送るか、それを例1の120のストリング中に含まれるようにする。

   BASICA/QuickBASIC/BASIC

```
140  cmd$ = chr$(&H11)
150  CALL ibcmd (brd0%,cmd$)
```

   あるいは

```
120  cmd$ = "#" + chr$(&H11)
130  CALL ibcmd (brd0%,cmd$)
```

IBSRE (前ページより続く) IBSRE

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%,chr$ (&H11),1)
```

あるいは

```
sta% = ilcmd (brd0%,"#" + chr$ (&H11),2)
```

C

```
ibcmd (brd0,"\x11",1);
```

あるいは

```
ibcmd (brd0,"#\x11",2);
```

3. 全てのデバイスをローカルモードに戻す。

BASICA/QuickBASIC/BASIC

```
100  v% = 0     ' Set REN to false.
110  CALL ibsre (brd0%, v%)
```

QuickBASIC/BASIC

```
sta% = ilsre (brd0%, 0)
```

C

```
ibsre (brd0, 0);
```

IBSRQ　IBSRQ

目的　SRQの「割込ルーチン」を登録する。

フォーマット

C

short ibsrq (void (far *func) (void))

ibsrq関数はGPIBインターフエースボードのステータスワード(ibsta)の中のSRQIビットが1にセットされているのをドライバが見い出した時は何時でも呼び出すべき手続きとしてC言語のルーチンfuncを制定するものです。ドライバが呼び出されるたびにSRQIはチェックされます。SRQIがセットされていると、funcはコントロールがアプリケーションプログラムに返される前に呼び出されます。funcがNULLに等しいときibsrqが呼び出されると、SRQサービシングがオフにされます。

注)　ibsrq使用の場合は自動シリアルポーリングを不可能化してください。

BASICA, QuickBASIC 及び BASIC の場合のSRQのサービシングについては、第3章に説明されているON PEN のメカニズムの項を参照してください。

注)　ON PEN メカニズムがインストールされている場合、マウスのドライバを不可能化する必要があるかもしれません。マウスは往々にしてライトペン割込を使用している場合があります。

IBSRQ (前ページより続く) IBSRQ

例

SRQサービシング要求の機能として srqservice ()を制定する。

C

```
short dvm;

void far srqservice() {

char spr;

ibrsp (dvm, &spr);

/* analyze the response here */
}

main () {

short gpib0 = ibfind ("gpib0");

/* disable autopolling */
ibconfig (gpib0, IbcAUTOPOLL, 0);

dvm = ibfind ("DEV4");
ibsrq (srqservice);

}
```

IBSTOP　IBSTOP

目的　非同期のオペレーションを中止する。

フォーマット

BASICA

```
CALL ibstop (ud%)
```

QuickBASIC/BASIC

```
CALL ibstop (ud%)
     あるいは
ilstop (ud%)
```

C

```
short ibstop (short ud)
```

udはデバイス或いはインターフェースボードを指定します。

ibstopは非同期の読取、書込、及びコマンドオペレーションは何でも終了させ、次にそのアプリケーションを再びドライバと同期させます。

非同期の入出力オペレーションが進行中の場合は、ステータスワードのERRビットがセットされ、EABOエラーが返されます。

デバイスIBSTOP関数

udがデバイスを指定している時は、ibstopはそのデバイスに対する未完了の非同期入出力関数は何でも終了させようとします。

ボードIBSTOP関数

udがボードを指定している場合、ibstopはそのボードから始まった非同期の入出力オペレーションで未完了のものは何でも終了させようとします。

IBSTOP　　(前ページより続く)　　IBSTOP

デバイス関数例

ドライブのrdrと関連する非同期オペレーションは何でも止める。

BASICA/QuickBASIC/BASIC

```
100  CALL ibstop (rdr%)
```

QuickBASIC/BASIC

```
sta% = ilstop (rdr%)
```

C

```
ibstop (rdr);
```

ボード関数例

ボードbrd0に関連する非同期オペレーションは何でも止める。

BASICA/QuickBASIC/BASIC

```
100  CALL ibstop (brd0%)
```

QuickBASIC/BASIC

```
sta% = ilstop (brd0%)
```

C

```
ibstop (brd0);
```

IBTMO　　IBTMO

目的　タイムリミットを変更するが不可能化する。

フォーマット

BASICA

```
CALL ibtmo (ud%, v%)
```

QuickBASIC/BASIC

```
CALL ibtmo (ud%, v%)
     あるいは
iltmo (ud%, v%)
```

C

```
short ibtmo (short ud, short v)
```

udはデバイス或いはインターフェースボードを指定します。vは以下に示すように時間制限を規定します。

表5－8 各タイムアウトコード値

vの値	最小タイムアウト値
0	不可能化されている
1	10 µsec
2	30 µsec
3	100 µsec
4	300 µsec

(次ページに続く)

IBTMO　　(前ページより続く)　　IBTMO

表5－8 各タイムアウトコード値 (前ページより続く)

vの値	最小タイムアウト値
5	1 µsec
6	3 µsec
7	10 µsec
8	30 µsec
9	100 µsec
10	300 µsec
11	1 sec
12	3 sec
13	10 sec
14	30 sec
15	100 sec

注)　vが0ならば時間制限はありません。

ibtmoは構成時に設定された値を変更する場合にのみ必要となります。

ibtmoによる割り当てはこの関数が再び呼び出されるか、ibonlまたは ibfind関数が呼び出されるか、或いはシステムがリスタートされるまで有効です。

IBTMO (前ページより続く) IBTMO

ibtmo関数は多くの関数が入出力が終了するのを待つ時間の長さを変更します。GPIBバスにアクセスするほとんどの関数がこれらの関数のうちに入ります。幾つかの例を下に示します。

- ibcmd
- ibrd
- ibrdi
- ibwrt
- ibwrti

ibtmo関数はデバイス関数がコマンドの受け入れを待つ時間の長さも変化させます。デバイスが制限時間内にコマンドを受け入れないと、EBUSエラーが返されます。

ibtmo呼び出しにおいてエラーが起こらないと、それ以前のタイムアウトコード値がiberr中に記憶されます。

デバイスIBTMO関数

udがデバイスを指定すると、それ以後そのデバイスに対して使用されるデバイス関数では新しい制限時間が用いられます。

ボードIBTMO関数

udがボードを指定すると、それ以後そのボードに対して使用されるボード関数では新しい制限時間が用いられます。

*IBWAIT*の項及び表2－2も参照してください。

IBTMO　　(前ページより続く)　　IBTMO

デバイス関数例

デバイス(テープ)が関係する関数呼び出しの制限時間を約300 μsecに変更する。

BASICA/QuickBASIC/BASIC

```
100  tape$ = "DEV9"
110  CALL ibfind(tape$, tape%)
120  v% = 10
130  CALL ibtmo (tape%, v%)
```

QuickBASIC/BASIC

```
tape% = ilfind ("DEV9")
sta% = iltmo (tape%,10)
```

C

```
tape = ibfind ("dev9");
ibtmo (tape, 10);
```

ボード関数例

brd0を使用するボード関数のための制限時間を10 μsecにする。

BASICA/QuickBASIC/BASIC

```
100  v% = 7
110  CALL ibtmo (brd0%, v%)
```

QuickBASIC/BASIC

```
sta% = iltmo (brd0%, 7)
```

C

```
ibtmo (brd0,7);
```

IBTRAP IBTRAP

目的 Applications Monitorのトラップとディスプレイモードを変更する。

フォーマット

BASICA

```
CALL ibtrap (mask%, v%)
```

QuickBASIC/BASIC

```
CALL ibtrap (mask%, v%)
     あるいは
iltrap (mask%, v%)
```

C

```
short ibtrap (short mask, short mode)
```

maskはibstaと同じビット割り当てでビットマスクを指定します。個々のビットはセットされ、GPIB呼び出し後相応するビットがステータスワードの中に現われた時、モードの値にしたがって、トラップ及び/或いはレコードされます。全てのビットがセットされると、それ以後はibfind以外の全てのGPIB呼び出しはトラップされるようになります。

modeはレコーディングとトラッピングの起こる条件を決定します。決定に有効な値は表5－9に示されています。

IBTRAP　　(前ページより続く)　　IBTRAP

表5－9 IBTRAPの各モード

値	効果
1	Monitorをオフにする。レコーディングもトラッピングも行われない。
2	レコードがオンになる。全ての呼び出しはレコードされるが、トラッピングは起きない。
3	レコードとトラップをオンにする。全ての呼び出しはレコードされ、トラップ条件が存在するときはいつでもMonitorが表示される。

ibtrap呼び出し中にエラーが起きると、ibstaのERRビットがセットされ、iberrは表5-10中の値の一つをとります。

表5－10 IBTRAPの各エラー

値	説明
ECAP	Applications Monitorがインストールされていません。
EARG	Monitorモードが無効です。

エラーが起きなかった場合は、iberrは以前のマスク値を保持しています。

第7章 *Applications Monitor* も参照してください。

IBTRAP (前ページより続く) IBTRAP

デバイス関数例

Applications MonitorをSRQ 或いは CMPLにおいてレコード及びトラップを行うように構成する。

BASICA/QuickBASIC/BASIC

```
100  mask% = &H1100          'SRQ or CMPL
110  mode% = 3               'Record and trap on
120  CALL ibtrap (mask%, mode%)
```

QuickBASIC/BASIC

```
sta% = iltrap (&H1100, 3)
```

C

```
ibtrap (0x1100, 3);
```

IBTRG　IBTRG

目的　選択されたデバイスをトリガする。

フォーマット

BASICA

```
CALL ibtrg (ud%)
```

QuickBASIC/BASIC

```
CALL ibtrg (ud%)
     あるいは
iltrg (ud%)
```

C

```
short ibtrg (short ud)
```

udはデバイスを指定します。

ibtrgは指定されたデバイスにアドレスしトリガします。

ibtrgは次に示すコマンドを送ります。

- アクセスボードのトークアドレス
- 必要な場合は、アクセスボードの2次アドレス
- UNLコマンド
- デバイスのリスンアドレス
- 必要な場合は、デバイスの2次アドレス
- GET(Group Execute Trigger) コマンド

IBTRG (前ページより続く) IBTRG

他のコマンドバイトも必要に応じて送ることが出来ます。

IBCMD の項も参照してください。

デバイス関数例

デバイス(analyz)をトリガする。

BASICA/QuickBASIC/BASIC

```
100  CALL ibtrg (analyz%)
```

QuickBASIC/BASIC

```
sta% = iltrg (analyz%)
```

C

```
ibtrg (analyz);
```

IBWAIT　　IBWAIT

目的　選択された事象の生起を待つ。

フォーマット

BASICA

```
CALL ibwait (ud%, mask%)
```

QuickBASIC/BASIC

```
CALL ibwait (ud%, mask%)
     あるいは
ilwait (ud%, mask%)
```

C

```
short ibwait (short ud, short mask)
```

udはデバイス或いはインターフエースボードを指定します。maskはビットマスクでステータスワードibstaと同じビットを割り当てられています。ibwaitはmaskの中のビットにより選択された事象をモニターし、これらの事象の何れかが起こるまでは処理を遅らせるために使われます。これらの事象とビット割り当てを表5-11に示します。

BASICA/QuickBASIC/BASIC/Cの宣言ファイルはステータスバイトibstaとiberrの個々のビットの簡略記号を定義します。例えば、QuickBASICでプログラミングしている場合、次の2つの呼び出しは等価です。

- `IF IBSTA% AND TACS THEN PRINT "TALK ADDRESS"`
- `IF IBSTA% AND &H0008 THEN PRINT "TALK ADDRESS"`

IBWAIT　　(前ページより続く)　　IBWAIT

表5－11 Wait [待機]マスクのレイアウト

簡略記号	ビット位置	16進数値	内容
ERR	15	8000	GPIBエラー
TIMO	14	4000	制限時間超過
END	13	2000	GPIBボードがENDかEOSを検出
SRQI	12	1000	SRQがオン
RQS	11	800	デバイスがサービス要求中
EVENT	10	400	
SPOLL	9	200	
CMPL	8	100	非同期入出力が完了
LOK	7	80	GPIBボードがロックアウト状態
REM	6	40	GPIBボードがリモートモード状態
CIC	5	20	GPIBがCIC
ATN	4	10	ATNがアサート中
TACS	3	8	GPIBボードがトーカ
LACS	2	4	GPIBボードがリスナ
DTAS	1	2	GPIBボードがデバイストリガのコマンドを検出
DCAS	0	1	GPIBボードがデバイスクリアのコマンドを検出

ibwaitはibstaも更新します。ibwaitはmask=0或いはmask=hex 8000 (ERRビット) で直ちに返ります。

IBWAIT　　(前ページより続く)　　IBWAIT

TIMOビットを0にするか、ibtmo関数で制限時間を0にセットすると、タイムアウトは不可能化されます。タイムアウトを不可能化するのはmask=0にセットするとき、或いは選択された事象が生起することがはっきり解っている時のみに行ってください。さもないと、プロセッサがその事象の起こるのをいつまでも待つということになりかねません。

デバイスIBWAIT関数

udがデバイスを指定している時は、待機マスクとステータスワードの各ビットのうちERR, TIMO, END, RQS, 及びCMPLのみが適用可能になっています。自動ポーリングが可能化されていると、ibwaitのRQSにおいて、GPIB SRQ信号線がアサートされる度に、指定されたデバイスのアクセスボードがGPIB上の全てのデバイスをシリアルポーリングし、応答を保存します。これは指定されたデバイスから返された待っていたデバイスのステータスバイトが、hex 40のビットをその中にセットして、サービス要求をしているのはそのデバイスであることを示すまで続きます。この場合、TIMOビットがセットしてあると、そのデバイスのタイムアウト時間中にその事象が起こらなければ、ibwaitは返されます。

ボードIBWAIT関数

udがボードを指定していると、待機マスクとステータスワードの全てのビットはRQSを除いて適用可能です。

デバイス関数例

デバイス(logger)サービスを要求するまでいつまでも待つ。

BASICA/QuickBASIC/BASIC

```
100  mask% = &H800      ' RQS
110  CALL ibwait (logger%,mask%)
```

IBWAIT (前ページより続く) IBWAIT

QuickBASIC/BASIC

```
sta% = ilwait (logger%,&H800)
```

C

```
mask = RQS;      /*   mask = 0x800;   */
ibwait (logger,mask);
```

ボード関数例

1. サービス要求或いはタイムアウトを待つ。

BASICA/QuickBASIC/BASIC

```
100  mask% = &H5000     ' TIMO SRQI
110  CALL ibwait (brd0%, mask%)
120  REM  ibsta% indicates what occurred.
```

QuickBASIC/BASIC

```
sta% = ilwait (brd0%, &H5000)
```

C

```
mask = SRQI | TIMO;      /*   mask = 0x5000;   */
ibwait (brd0,mask);
```

2. ibstaの現在のステータスを更新する。

BASICA/QuickBASIC/BASIC

```
100  mask% = 0
110  CALL ibwait (brd0%,mask%)
```

QuickBASIC/BASIC

```
sta% = ilwait (brd0%,0)
```

IBWAIT (前ページより続く) IBWAIT

C

```
ibwait (ud,0);
```

3. 他のCICからコントロールが引き渡されるまではいつまでも待つ。

BASICA/QuickBASIC/BASIC

```
100  mask% = &H20 ' CIC
110  CALL ibwait (brd0%,mask%)
```

QuickBASIC/BASIC

```
sta% = ilwait (brd0%,&H20)
```

C

```
mask = CIC;      /*   CIC = 0x20;   */
ibwait (ud,mask);
```

4. 他のCICからトーカ或いはリスナとしてアドレスされるまでいつまでも待つ。

BASICA/QuickBASIC/BASIC

```
100  mask% = &H0C ' TACS LACS
110  CALL ibwait (brd0%,mask%)
```

QuickBASIC/BASIC

```
sta% = ilwait (brd0%,&H0C)
```

C

```
mask = TACS | LACS;  /*  TACS | LACS = 0x0C;   */
ibwait (ud,mask);
```

IBWRT　IBWRT

目的　ストリングからのデータの書き込み。

フォーマット

BASICA

```
CALL ibwrt (ud%, wrt$)
```

QuickBASIC/BASIC

```
CALL ibwrt (ud%, wrt$)
     あるいは
ilwrt (ud%, wrt$, cnt&)
```

C

```
int ibwrt (int ud, char wrt [], unsigned long cnt)
```

udはデバイス或いはインターフェースボードを指定します。wrtはGPIBを通して送るデータのバッファです。

BASICAではwrtは255バイトに過ぎません。QuickBASICと BASICでは、wrtは32キロバイト(2の15乗)マイナス1バイトです。Cでは、wrtは4ギガバイト(2の32乗)マイナス1バイトまで記憶することが出来ます。

ibwrtは次の事象の何れかが起こったときに終了します。

- 全てのバイトが転送を完了した時。
- エラーが検出された時。
- 制限時間が超過した時。
- DCL(デバイスクリア) あるいは SDC(選択されたデバイスをクリア) のコマンドがCICである他のデバイスから受信した時。

IBWRT　　(前ページより続く)　　IBWRT

終了後のibcntlは読み取られたバイトの数を示します。 ibcntは読み取られたバイトの数を16ビットで示したものです。上記のibwrtを終了する事象のうち、最初の事象（全てのバイトの転送が完了した場合）を除く事象によりibwrtが終了した場合は、ショートカウントが起きている可能性があります。

デバイスibwrt関数が返されたとき、ibstaは最新のデバイスステータスを示します。ibcntlはデバイスに書き込まれた実際のデータバイト数を示します。また、ibcntは書き込まれたデータバイト数を16ビットで表わしています。ERRビットがibsta中にセットしてある場合は、iberrは最初に検出されたエラーです。

デバイスIBWRT関数

udがデバイスを指定していると、デバイスはリスナとしてアドレスされ、アクセスボードはトーカとしてアドレスされます。

そして次にデータがデバイスに書き込まれます。

ボードIBWRT関数

udがインターフェースボードを指定していると、ibwrt関数は既にCICによりアドレスされているはずのGPIBデバイスに書込をしようとします。

アクセスボードがCICの場合は、ibwrtに先立ってibcmdを呼び出してデバイスにリスナとして、またボードにトーカとしてアドレスしておく必要があります。

アクセスボードがアクティブコントローラであると、ボードは、書込が完了した後といえども、まずATNをオフにした待機コントローラの状態にされます。もしアクセスボードがアクティブコントローラでなければ、ibwrt は直ちに動作を始めます。

IBWRT (前ページより続く) IBWRT

ボードがCICであるが、まだ ibcmdによりトーカとしてアドレスされていないと、EADRエラーが起きます。また、ある事情によりibwrtが制限時間内に完了していないとEABOエラーが起きます。データバイトが送られたのにバス上にリスナが無い時にはENOLエラーが起きます。

注) 送っているデータストリングの終にEOS文字を付けたい時は命令を使用して正式にEOS文字を付けてください。これについてはデバイス例の2を御覧ください。

デバイス関数例

1. デバイスのdvmに10バイトの命令を書き込む。

BASICA/QuickBASIC/BASIC

```
100  wrt$ = "F3R1X5P2G0"
110  CALL ibwrt (dvm%, wrt$)
```

QuickBASIC/BASIC

```
sta% = ilwrt (dvm%, "F3R1X5P2G0", 10)
```

C

```
ibwrt (dvm, "F3R1X5P2G0", 10);
```

2. デバイス (ptr) に5命令バイト書き込み、復帰文字と改行文字をもって終える。改行文字はこのデバイスのEOS文字である。

BASICA/QuickBASIC/BASIC

```
100  wrt$ = "IP2X5" + chr$(&H0D) + chr$(&H0A)
110  CALL ibwrt (ptr%, wrt$)
```

IBWRT　　(前ページより続く)　　IBWRT

QuickBASIC/BASIC

```
wrt$ = "IP2X5" + chr$ (&H0D) + chr$ (&H0A)
sta% = ilwrt (ptr%, wrt$, 7)
```

C

```
ibwrt (ptr, "IP2X5\r\n", 7);
```

ボード関数例

1. リスンアドレス hex 2F (ASCIIの /) のデバイスに10命令バイトを書き込む。(GPIBボードのトークアドレスは hex 40 (ASCIIの @) である。)

BASICA/QuickBASIC/BASIC

```
100  REM  Perform addressing.
110  cmd$ = "?@/" ' UNL MTA LAD
120  CALL ibcmd (brd0%,cmd$)
130  REM  Perform board write.
140  wrt$ = "F3R1X5P2G0"
150  CALL ibwrt (brd0%,wrt$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%"?@/",3)
sta% = ilwrt (brd0%,"F3R1X5P2G0",10)
```

C

```
ibcmd (brd0,"?@/",3);
ibwrt (brd0,"F3R1X5P2G0",10);
```

2. バイナリ整数データの自動バイトスワッピングを可能化する手順については、*IBCONFIG* の項のボード関数例を参照のこと。

IBWRTA　IBWRTA

目的　データをストリングより非同期に書き出す。

フォーマット

BASICA

```
CALL ibwrta (ud%,wrt$)
```

QuickBASIC/BASIC

```
CALL ibwrta (ud%, wrt$)
     あるいは
ilwrta (ud%, wrt$, cnt&)
```

C

```
int ibwrta (int ud, char wrt [], unsigned long cnt)
```

udはデバイス或いはインターフェースボードを指定します。wrtはGPIBを介して送られるデータを含みます。

BASICA ではwrtは255バイトに過ぎません。QuickBASICと BASICではwrtは32キロバイト(2の15乗)マイナス1バイトまで含むことができます。Cではwrtは4ギガバイト(2の32乗)マイナス1バイトまで含むことが出来ます。

ibwrtaはGPIB入出力処理がなされている間にアプリケーションプログラムでほかの機能を実行する必要がある時にibwrtに代って使用されます。ibwrtaは入出力動作が始まると直ちに返されます。

ibcmda, ibrda及び ibwrta と言う3つの非同期入出力関数が入出力処理中に他の機能（非GPIB機能)を実行できるように与えられています。一度非同期入出力関数がスタートすると、その後のGPIB呼び出しでそのデバイスやアクセスボードに関連するものは、その入出力動作が完了してGPIBドライバとアプリケーションが再び同期されるまでは、行うことが出来ません。

IBWRTA　　(前ページより続く)　　IBWRTA

再同期化は次の3つの関数のうちの1つを使用して行うことが出来ます。

注)　再同期化は返されたibstaがCMPLを含む場合のみ実現します。

- ibwait (マスクにCMPLを含む) – ドライバとアプリケーションが同期化される。
- ibstop – 非同期的な入出力がキャンセルされ、ドライバとアプリケーションが同期される。
- ibonl – 非同期入出力のキャンセル、インターフェースのリセットさ、及びドライバとアプリケーションの同期が行われる。

非同期入出力中に許されるGPIB呼び出しとしてはこのほかにibwaitだけです（マスクは任意）。その他のデバイス及びアクセスボードが関係したGPIB呼び出しではいつでもEOIPエラーが返されます。

デバイスIBWRTA関数

udがデバイスを指定していると、そのデバイスはリスナとしてアドレスされ、アクセスボードはトーカとしてアドレスされ、次にデータがデバイスに書き込まれます。

ボードIBWRTA関数

udがインターフェースボードを指定している場合、ibwrta関数は既に正しい手続きで初期化され、実際のCICによりアドレスされている筈のGPIBデバイスに対して書き込みを試みます。

ボードがCICである場合は、ibwrtaに先立ってibcmdを呼び出し、デバイスにリスナとして、またボードにトーカとしてアドレスしておく必要があります。

IBWRTA (前ページより続く) IBWRTA

ボードがアクティブコントローラの場合、ボードは、書込動作が完了した後でも、まずATNをオフにした待機コントローラの状態におかれます。そうでない場合には直ちに書込動作が始まります。

ボードがCICであるがibcmd関数によってあらかじめトーカとしてアドレスされていない場合にはEADRエラーが起こります。リスナが存在しない場合にはENOLエラーは起こりません。

注) 送るデータストリングのおわりにEOS文字を付けたい場合には、命令を使用して正式に付けなければなりません。

デバイスibwrt関数が返された場合、ibstaは最も新しいデバイスのステータスを保持しています。もしibsta内のERRビットがセットされている場合、iberrは最初に検出されたエラーです。

デバイス関数例

一方でほかの処理を行いながらdvmに10命令バイトを書き込む。

BASICA/QuickBASIC/BASIC

```
100  wrt$ = "F3R1X5P2G0"
110  CALL ibwrta (dvm%,wrt$)
120  mask% = &H4100     'TIMO CMPL
130  REM  Perform other processing here, then
140  REM  wait for I/O completion or a
150  REM  timeout.
160  CALL ibwait (dvm%,mask%)
170  REM  ibsta% indicates how the write
180  REM  terminated:CMPL, END, TIMO, or ERR.
```

QuickBASIC/BASIC

```
ilwrta (dvm%,"F3RIX5P2G0",10)
sta% = ilwait (dvm%,&H4100)
```

IBWRTA　(前ページより続く)　IBWRTA

C

```
ibwrta (dvm, "F3R1X5P2G0", 10);
/*   Perform other processing here   */
ibwait (dvm, TIMO | CMPL);
```

ボード関数例

1. ある優先事象の生起をテストする一方、リスンアドレスが hex 2F (ASCII の /) のデバイスに10命令バイトを書き込み (GPIBボードのトークアドレスは hex 40 或いはASCII の @)、その後アドレシングを解除する。

BASICA/QuickBASIC/BASIC

```
100  REM  Perform addressing in preparation
110  REM  for board write.
120  cmd$ = "?@/"      ' UNL MTA LAD
130  CALL ibcmd (brd0%,cmd$)
140  REM  Perform board asynchronous write.
150  wrt$ = "F3R1X5P2G0"
160  CALL ibwrta (brd0%,wrt$)
170  REM  Perform other processing here, then
180  REM  wait for I/O completion or timeout.
190  mask% = &H4100                ' TIMO CMPL
200  CALL ibwait (brd0%,mask%)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%,"?@/",3)
sta% = ilwrta (brd0%,"F3RIX5P2G0",10)
sta% = ilwait (brd0%,&H4100)
```

C

```
ibcmd (brd0,"?@/",3);     /*   UNL MTA LAD   */
ibwrta (brd0,"F3R1X5P2G0",10);
/* Perform other processing here.           */
ibwait (brd0,TIMO | CMPL);
```

2. バイナリ整数データの自動バイトスワッピングの可能化については、*IBCONFIG* の項のボード関数例を参照してください。

IBWRTF IBWRTF

目的　ファイルからのデータの書き込み。

フォーマット

BASICA

```
CALL ibwrtf (ud%, flname$)
```

QuickBASIC/BASIC

```
CALL ibwrtf (ud%, flname$)
     あるいは
ilwrtf (ud%, flname$)
```

C

```
int ibwrtf (int ud, char flname [])
```

udはデバイス或いはインターフェースを指定します。flnameはそこからデータが書き取られるファイルの名です。flnameはドライブ名とパス名こみで50文字までの長さをとることが出来ます。

ibwrtfは自動的にファイルを開きます。終了の際は ibwrtfはファイルを閉じます。

ibwrtfが指定されたファイルを開いたり、捜したり、読み取ったり、或いは閉じたり出来なかった時には、EFSOエラーメッセージが返されます。

ibwrtf関数のオペレーションは以下に示す事象の何れかに遭ったときに終了します。

- 全てのバイトを送ってしまった時。
- エラーが検出された時。
- 制限時間を超過した時。

IBWRTF　　(前ページより続く)　　IBWRTF

- CICであるもう一つのデバイスが送出したDCL(デバイスクリア)コマンドあるいはSDC(選択したデバイスをクリア)のコマンドを受信した時

ibwrtf終了後に得られるibcntlは書き込まれたバイト数を示し、ibcntは書き込まれたバイト数を16ビットで表わします。

デバイスIBWRTF関数

udがデバイスを指定している場合、デバイスibwrt関数の場合と同じボード関数が自動的に行われます。この関数はibwrtの場合と似た条件で終了されます。

ibwrtf関数が返された時にはibstaはデバイスの最新のステータスを示します。ibcntlは書き込まれたデータのバイト数を示し、ibcntは書き込まれたバイトの数を16ビットで表わします。ibsta中でERRビットがセットされると、iberrは最初に検出されたエラーを示します。

ボードIBWRTF関数

udがインターフェースボードを指定している場合は、ボードibwrt関数は既に正しい手続きにしたがってアドレスされている筈のGPIBデバイスに書き込みを行います。

ボードがCICであるがibcmd関数によりトーカとしてアドレスされていないとEADRエラーが起こります。また、何らかの理由で、書込が制限時間内に完了しないとEABOエラーが起こります。データバイトが送られるときにリスナがバス上に存在しないとENOLエラーが返されます。

デバイス関数例

現在のドライブ中にあるファイルのY.DATからのデータをデバイスのrdrに書き込む。

IBWRTF (前ページより続く) IBWRTF

BASICA/QuickBASIC/BASIC

```
100  flname$ = "Y.DAT"
110  CALL ibwrtf (rdr%,flname$)
```

QuickBASIC/BASIC

```
sta% = ilwrtf (rdr%,"Y.DAT")
```

C

```
ibwrtf (rdr,"Y.DAT");
```

ボード関数例

1. 現在のドライブにあるファイルのY.DATからのデータをリスンアドレスが hex 2C (ASCIIの ,) のデバイスに書き込み、次にbrd0 に対するアドレシングを解除する。

BASICA/QuickBASIC/BASIC

```
100  REM  Perform addressing in preparation
110  REM  for board write.
120  cmd$ = "?@,"       ' UNL MTA LAD
130  CALL ibcmd (brd0%,cmd$)
140  REM  Perform board write.
150  flname$ = "Y.DAT"
160  CALL ibwrtf (brd0%,flname$)
```

QuickBASIC/BASIC

```
sta% = ilcmd (brd0%,"?@,",3)
sta% = ilwrtf (brd0%,"Y.DAT")
```

C

```
ibcmd (brd0,"?@,",3);
ibwrtf (brd0,"Y.DAT");
```

2. バイナリ整数データの自動バイトスワッピングの可能化については、*IBCONFIG* の項のボード関数例を参照してください。

IBWRTI　　IBWRTI

目的　整数配列からデータを書き込む。

フォーマット

BASICA

```
Call ibwrti (ud%, iarr%(0), cnt%)
```

QuickBASIC/BASIC

```
Call ibwrti (ud%, iarr%(), cnt%)
        あるいは
ilwrti (ud%, iarr%(), cnt%)
```

ibwrtiはCにおいては必要がないので供給されていません。Cにおいてはibwrtを使用するだけであらゆる型のバッファにデータを送ることが出来ます。BASICA, QuickBASIC及びBASICでは型の指定の規則がより厳格なので、配列バッファにはストリングと異なった関数の呼び出しが必要となります。

ud%はデバイス或いはインターフェースボードを指定します。
iarr%はそこからデータが書き出されるところの整数配列です。
cnt%は書き出すことの出来る最大バイト数を指定します。データはiarr%の中に2バイト整数として記憶され、下位バイトー上位バイトの順でGPIBに送られます。

ibwrtiは文字列変数からデータを書き出すibwrtに似ています。ibwrtiは最大64キロバイトマイナス1バイト（2の16乗マイナス1バイト）のデータを書き込むことが出来ます。

*IBWRT*の項とこの章の初めに出ている「BASICA/QuickBASIC/BASIC/CによるNI-488の入出力関数呼び出し」を参照してください。また、*IBWRTIA*の項も参照してください。

IBWRTI (前ページより続く) IBWRTI

デバイス関数例

1. 整数配列wrtからの10命令バイトをdvmに書き込む。

BASICA

```
100  DIM wrt%(4)
110  wrt%(0) = ASC("F") + ASC("3") * 256
120  wrt%(1) = ASC("R") + ASC("1") * 256
130  wrt%(2) = ASC("X") + ASC("5") * 256
140  wrt%(3) = ASC("P") + ASC("2") * 256
150  wrt%(4) = ASC("G") + ASC("0") * 256
160  cnt% = 10
170  CALL ibwrti (dvm%,wrt%(0),cnt%)
```

QuickBASIC/BASIC

QuickBASICにおいては、ライン170を

```
CALL ibwrti (ud%, wrt%(), cnt%)
        あるいは
sta% = ilwrti (dvm%, wrt%(), 10)
```

によって置き換える。

2. 整数配列wrt%からの5命令バイトをデバイスptr%に書き込む。命令バイトは復帰文字と改行文字で終了されるものとする。改行文字はこのデバイスのEOS文字である。

BASICA

```
100  DIM wrt%(3)
110  wrt%(0) = ASC("I") + ASC("P") * 256
120  wrt%(1) = ASC("2") + ASC("X") * 256
130  wrt%(2) = ASC("5") + &H0D * 256
140  wrt%(3) = &H0A
150  cnt% = 7
160  CALL ibwrti (ptr%, wrt%(0), cnt%)
```

IBWRTI　　(前ページより続く)　　IBWRTI

QuickBASIC/BASIC

ライン160を

```
CALL ibwrti (ptr%, wrt%(), cnt%)
```

あるいは

```
DIM wrt% (3)
wrt% (0) = ASC ("I") + (ASC ("P") * 256)
wrt% (1) = ASC ("2") + (ASC ("X") * 256)
wrt% (2) = ASC ("5") + (&H0D * 256)
wrt% (3) = &H0A
sta% = ilwrti (ptr%, wrt%(), 7)
```

ボード関数例

1. 整数配列wrt%から10命令バイトをリスンアドレスがhex 2F (ASCIIの/)のデバイスに書き込む。(GPIBボードのトークアドレスはhex 40あるいはASCIIの@である。)

BASICA

```
100  REM  Perform addressing.
120  cmd$ = "?@/" ' UNL MTA LAD
130  CALL ibcmd (brd0%,cmd$)
140  REM  Perform board write.
150  DIM wrt%(4)
160  wrt%(0) = ASC("F") + ASC("3") * 256
170  wrt%(1) = ASC("R") + ASC("1") * 256
180  wrt%(2) = ASC("X") + ASC("5") * 256
190  wrt%(3) = ASC("P") + ASC("2") * 256
200  wrt%(4) = ASC("G") + ASC("0") * 256
210  cnt% = 10
220  CALL ibwrti (brd0%,wrt%(0),cnt%)
```

IBWRTI (前ページより続く) IBWRTI

QuickBASIC/BASIC

QuickBASICの場合は、ライン220を

```
CALL ibwrti (brd0%, wrt%(), cnt%)
```

あるいは

```
sta% = ilwrti (brd0%, wrt%(), 10)
```

で置き換える。

2. バイナリ整数データの自動バイトスワッピングの可能化については、*IBCONFIG* の項のボード関数例を参照してください。

IBWRTIA　　IBWRTIA

目的　整数配列からのデータを非同期で書き込む。

フォーマット

BASICA

```
Call ibwrtia (ud%, iarr%(0), cnt%)
```

QuickBASIC/BASIC

```
Call ibwrtia (ud%, iarr%(), cnt%)
        あるいは
ilwrtia (ud%, iarr%(), cnt%)
```

ibwrtiaはCにおいては必要がないので供給されていません。Cにおいてはibwrtaを使用するだけであらゆる型のバッファにデータを送ることが出来ます。BASICA, QuickBASIC及びBASICは型の規則がより厳格なので、配列バッファにはストリングの場合とは異なった関数の呼び出しが必要となります。

ud%はデバイス或いはインターフェースボードを指定します。
iarr%はそこからデータが書き出されるところの整数配列です。
cnt%は書き出すことの出来る最大バイト数を指定します。データはiarr%の中に2バイト整数として記憶され、下位バイトー上位バイトの順でGPIBに送られます。

ibwrtiaは文字列変数からデータを書き出すibwrtaに似ています。ibwrtiaは最大64キロバイトマイナス1バイト（2の16乗マイナス1バイト）のデータを書き込むことが出来ます。

IBWRTA の項とこの章の初めに出ている「BASICA/QuickBASIC/BASIC/CによるNI-488の入出力関数呼び出し」を参照してください。また、*IBWRTIA* の項も参照してください。

IBWRTIA (前ページより続く) IBWRTIA

注) QuickBASICと BASICの場合、ダイナミックな配列を非同期的関数である ibrdia や ibwrtiaに引き渡さないでください。それらのロケーションが入出力動作中に変わる恐れがあります。

デバイス関数例

1. 一方で別の処理を行いながら、整数配列からの10命令バイトをデバイスdvm%に書き込む。

BASICA

```
100  DIM wrt%(4)
110  wrt%(0) = ASC("F") + ASC("3") * 256
120  wrt%(1) = ASC("R") + ASC("1") * 256
130  wrt%(2) = ASC("X") + ASC("5") * 256
140  wrt%(3) = ASC("P") + ASC("2") * 256
150  wrt%(4) = ASC("G") + ASC("0") * 256
160  cnt% = 10
170  CALL ibwrtia (dvm%,wrt%(0),cnt%)
180  mask% = &H4100      ' TIMO CMPL
190  REM  Perform other processing here, then
200  REM  wait for I/O completion or timeout.
210  CALL ibwait (dvm%,mask%)
220  REM  ibsta% shows how the write terminated:
230  REM  CMPL, END, TIMO, or ERR.
250  REM  If CMPL is not set, continue
260  REM  processing.
270  IF (ibsta% AND &H100) = 0 GOTO 190
```

QuickBASIC/BASIC

ライン170を

```
CALL ibwrtia (dvm%, wrt%(), cnt%)
```

あるいは

```
sta% = ilwrtia (brd0%,wrt% (),10)
```

IBWRTIA　(前ページより続く)　IBWRTIA

ボード関数例

整数配列 wrtからの10命令バイトをリスンアドレスが hex 2F (ASCII の /) のデバイスに書き込み、次にアドレシングを解除する。(GPIBボードのトークアドレスは hex 40 または ASCIIの @ である。)

BASICA

```
100  REM  Perform addressing.
110  cmd$ = "?@/" ' UNL MTA LAD
120  CALL ibcmd (brd0%,cmd$)
130  REM  Perform board write.
140  DIM wrt%(4)
150  wrt%(0) = ASC("F") + ASC("3") * 256
160  wrt%(1) = ASC("R") + ASC("1") * 256
170  wrt%(2) = ASC("X") + ASC("5") * 256
180  wrt%(3) = ASC("P") + ASC("2") * 256
190  wrt%(4) = ASC("G") + ASC("0") * 256
200  cnt% = 10
210  CALL ibwrtia (brd0%,wrt%(0),cnt%)
220  REM  Perform other processing here then
230  REM  wait for I/O completion or timeout.
240  mask% = &H4100      ' TIMO CMPL
250  CALL ibwait (brd0%,mask%)
```

QuickBASIC/BASIC

ライン210を

```
CALL ibwrtia (brd%, wrt%(), cnt%)
```

あるいは

```
sta% = ilwrtia (brd0%,wrt%(),10)
```

2. バイナリ整数データの自動バイトスワッピングの可能化については、*IBCONFIG* の項のボード関数例を参照してください。

BASICA/QuickBASIC/BASIC/C GPIB プログラミングの各例

以下に示す各種のプログラム例は、ユーザー各位が、各々御自分のパーソナルコンピュータ上でNI-488の関数を使用して、ある代表的なIEEE-488用の計測器をプログラムする際に利用出来る手順を示したものです。これらのアプリケーションプログラムはBASICA, QuickBASIC及びCによって書かれています。ここで対象となる計測器はディジタル電圧計 (DVM) で、それ以上詳細な指定はしていません。言い替えれば、メーカー名その他は指示されていません。これらの例の目的とするところは、どの様にドライバを使用してあるプログラミングを遂行し、シーケンスを制御するかを説明することであって、シーケンスそのものを決定するための方法を説明することではありません。

デバイスをプログラムするために送られる命令も、そのデバイスから返される筈のデータもデバイス依存性メッセージと呼ばれる性質のものです。したがってこれらのプログラム例の中で使用されるメッセージのフォーマットもシンタクスも、デバイスによって変わるはずのものです。それだけでなく、個々のデバイスに送らなければならないインターフェースメッセージまたはバスコマンドと言われるものも、程度は少ないけれども、やはり多少は変わってきます。ある特定のデバイスをプログラムしコントロールするためのメッセージの正確なシーケンスを見出すには、そのデバイスの説明文書を御覧ください。

例えば、次に示す動作シーケンスは先に述べたDVMをプログラムして、オートレンジモードで高周波AC電圧信号の測定値を返すようにさせるために必要とされるものです。

1. DVMのGPIBインターフェース回路を初期化してメッセージに応答出来るようにする。

2. DVMをリモートプログラミングモードにし、計器盤のコントロールをオフにする。

3. 内部の測定回路を初期化する。

4. メータに、オートレンジ(AUTO)を使用して交流電圧 (VAC) を測定すること、測定を始める前にコントローラからのトリガを待つこと (TRIGGER 2), 及び、測定が完了してメータが結果を送り出す準備が出来ると IEEE-488のサービス要求信号線 SRQ をアサートすること (*SRE 16)を命令する。

5. 個々の測定の度に、

   a. TRIGGER(トリガ) コマンドをマルチメータに送る。ibwrt コマンドの "VAL1?"がメータに次のトリガされた読取値をIEEE-488出力バッファに送るよう命令する。

   b. DVM が SRQをアサートして測定値の読出の準備完了を告げるのを待つ。

   c. DVMに対してシリアルポーリングを行って測定データが有効であるか、或いは故障状態が存在しているかを決定する。これはメッセージアベイラブル(MAV)ビット(ステータスバイトの第4ビット) をチェックすることで行うことが出来る。

   d. データが有効であれば、DVMから10バイトを読み取る。

6. セッションを終える。

後に示す各プログラムは下記の前提をもととして書かれたものです。

- GPIBボードは GPIBの 指定されたシステムアクティブコントローラです。
- GPIBボードのハードウエアのデフォルト設定値には何ら変更を加えてありません。
- ソフトウエアのパラメータは、デバイスDVMを1次アドレス1で定義するに必要な場合を除いて変更していません。
- GPIBボードは一つしか使用していません。このボードはGPIB0として指定されています。
- GPIB0の1次のリスンとトークアドレスは各々 hex 20 (ASCIIの <space>)と hex 40 (ASCIIの@) です。

BASICAプログラム例—デバイス関数

```
100  REM
110  REM You must merge this code with DECL.BAS.
120       CLS
130       print "READ MEASUREMENT FROM FLUKE 45..." : print
140  REM
150  REM Assign an unique identifier to the FLUKE 45 that you
160  REM configured using IBCONF.EXE.
170  REM
180       bdname$ = "DVM"
190       call ibfind(bdname$, dvm%)
200       msg$ = "IBFIND ERROR"
210       if (dvm% < 0) then GOSUB 6000 : STOP
220  REM
230  REM Clear the device.
240  REM
250       call ibclr (dvm%)
260       msg$ = "ibclr ERROR"
270       if (ibsta% and EERR) then GOSUB 6000 : STOP
280  REM
290  REM Write the function, range, and trigger source
300  REM instructions to the Fluke 45.
310  REM
320       wrt$ = "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
330       call ibwrt (dvm%,wrt$)
340       msg$ = "IBWRT ERROR"
350       if (ibsta% and EERR) then GOSUB 6000 : STOP
360  REM
370  REM Trigger the device and request measurement.
380  REM
390       call ibtrg (dvm%)
400       msg$ = "IBTRG ERROR"
410       if (ibsta% and EERR) then GOSUB 6000 : STOP
420  REM
430       wrt$ = "VAL1?"
440       call ibwrt(dvm%, wrt$)
450       msg$ = "ibwrt error"
460       if (ibsta% and EERR) then GOSUB 6000 : STOP
470  REM
480  REM Wait for the DVM to set RQS (hex 800) or for a timeout
490  REM (hex 4000).  These status bits are listed in Chapter 3
500  REM of the Software Reference Manual.  If the current time
510  REM limit is too short, use ibtmo to change it.
520  REM
530       print "Waiting for RQS..."
540       print
550       mask% = &H4800                    ' rqs + timo
560       call ibwait(dvm%, mask%)
570       msg$ = "ibwait error"
580       if (ibsta% and (EERR or TIMO)) then GOSUB 6000 : STOP
590  REM
```

```
600  REM If neither a timeout nor an error occurred, IBWAIT must
610  REM have returned on RQS.  Now read the status byte.  If it
620  REM is &H50, the Fluke 45 has valid data to send; otherwise,
630  REM it has a fault condition to report.
640  REM
650       call ibrsp(dvm%,spr%)
660       msg$ = "IBRSP ERROR"
670       if (ibsta% and EERR) then GOSUB 6000 : STOP
680  REM
690       msg$ = "FLUKE 45 ERROR"
700       if spr% <> &H50 then GOSUB 7000 : STOP
710  REM
720  REM Read the measurement.
730  REM
740       rd$ = space$(10)
750       call ibrd (dvm%,rd$)
760       msg$ = "IBRD ERROR"
770       if (ibsta% and EERR) then GOSUB 6000 : STOP
780  REM
790       reading$ = left$(rd$, ibcnt%)
800       print "reading: "; reading$
810  REM
820  REM Call the ibonl function to disable the device DVM.
830  REM
840       v% = 0 : call ibonl (dvm%, v%) : STOP
850       END


6000 REM This routine would notify you that an IB call failed
6010 REM and print the status variables.
6020 REM
6030      Print msg$
6040 REM
6050      Print "ibsta= &H"; hex$(ibsta%); " <";
6060      If ibsta% and EERR  then print " ERR";
6070      If ibsta% and TIMO then print " TIMO";
6080      If ibsta% and EEND then print " END";
6090      If ibsta% and SRQI then print " SRQI";
6100      If ibsta% and RQS  then print " RQS";
6110      If ibsta% and CMPL then print " CMPL";
6120      If ibsta% and LOK  then print " LOK";
6130      If ibsta% and RREM then print " REM";
6140      If ibsta% and CIC  then print " CIC";
6150      If ibsta% and AATN then print " ATN";
6160      If ibsta% and TACS then print " TACS";
6170      If ibsta% and LACS then print " LACS";
6180      If ibsta% and DTAS then print " DTAS";
6190      If ibsta% and DCAS then print " DCAS";
6200      Print ">"
```

```
6210 REM
6220     Print "iberr= "; iberr%;
6230     If iberr% = EDVR then print " EDVR <DOS Error>"
6240     If iberr% = ECIC then print " ECIC <Not CIC>"
6250     If iberr% = ENOL then print " ENOL <No Listener>"
6260     If iberr% = EADR then print " EADR <Address error>"
6270     If iberr% = EARG then print " EARG <Invalid argument>"
6280     If iberr% = ESAC then print " ESAC <Not Sys Ctrlr>"
6290     If iberr% = EABO then print " EABO <Op. aborted>"
6300     If iberr% = ENEB then print " ENEB <No GPIB board>"
6310     If iberr% = EOIP then print " EOIP <Async I/O in prg>"
6320     If iberr% = ECAP then print " ECAP <No capability>"
6330     If iberr% = EFSO then print " EFSO <File sys. error>"
6340     If iberr% = EBUS then print " EBUS <Command error>"
6350     If iberr% = ESTB then print " ESTB <Status byte lost>"
6360     If iberr% = ESRQ then print " ESRQ <SRQ stuck on>"
6370     If iberr% = ETAB then print " ETAB <Table Overflow>"
6380 REM
6390     Print "ibcnt = "; ibcnt%
6400 REM
6410 REM Call the ibonl function to disable the device DVM.
6420 REM
6430     v% = 0 : call ibonl (dvm%, v%) : RETURN


7000 REM This routine would notify you that the DVM returned an
7010 REM invalid serial poll response byte.
7020 REM
7030     print msg$
7040     print "Status Byte = "; spr%
7050 REM
7060 REM Call the ibonl function to disable the device DVM.
7070 REM
7080     v% = 0 : call ibonl (dvm%, v%) : RETURN
```

BASICAプログラム例—ボード関数

```
150  REM
160  REM You must merge this code with DECL.BAS.
170       CLS
180       print "READ MEASUREMENT FROM FLUKE 45..." : print
190  REM
200  REM Assign a unique identifier to board 0 and store in
205  REM variable BRD0%.
210  REM
220       bdname$ = "GPIB0"
230       call ibfind(bdname$, brd0%)
240       msg$ = "IBFIND ERROR"
250       if brd0% < 0 then GOSUB 6000 : STOP
260  REM
270  REM Send the Interface Clear (IFC) message to all devices.
280  REM
290       call ibsic(brd0%)
300       msg$ = "IBSIC ERROR"
310       if (ibsta% and EERR) then GOSUB 6000 : STOP
320  REM
330  REM Turn on the Remote Enable (REN) signal.
340  REM
350       v% = 1 : call ibsre(brd0%, v%)
360       msg$ = "IBSRE ERROR"
370       if (ibsta% and EERR) then GOSUB 6000 : STOP
380  REM
390  REM Inhibit front panel control with the Local Lockout
400  REM (LLO) command (hex 11).  Place the Fluke 45 in remote
410  REM mode by addressing it to listen. The listen address of
420  REM device 1 (MLA1) is its primary address, 1, plus hex 20.
430  REM This is an ASCII "!".  Send the Device Clear (DCL)
440  REM message (hex 14) to clear internal device functions.
450  REM Finally, address the GPIB interface board to talk (MTA0)
460  REM by sending its talk address, primary address 0 plus hex
465  REM 40, or ASCII "@".  These commands can be found in
465  REM Appendix A of the Software Reference Manual.
470  REM
480       cmd$ = CHR$(&H11) + "!" + CHR$(&H14) + "@"
490       call ibcmd(brd0%, cmd$)
500       msg$ = "IBCMD ERROR"
510       if (ibsta% and EERR) then GOSUB 6000 : STOP
520  REM
530  REM Write the function, range, and trigger source
540  REM instructions to the FLUKE 45.
550  REM
560       wrt$ = "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
570       call ibwrt(brd0%, wrt$)
580       msg$ = "IBWRT ERROR"
590       if (ibsta% and EERR) then GOSUB 6000 : STOP
```

```
600  REM
610  REM Send the GET message (hex 8) to trigger a
615  REM measurement reading.
620  REM
630       cmd$ = CHR$(&H8)                    ' GET
640       call ibcmd(brd0%, cmd$)
650       msg$ = "IBCMD ERROR"
660       if (ibsta% and EERR) then GOSUB 6000 : STOP
670  REM
680  REM Request triggered measurement reading.
690  REM
700       wrt$ = "VAL1?"
710       call ibwrt(brd0%, wrt$)
720       msg$ = "IBWRT ERROR"
730       if (ibsta% and EERR) then GOSUB 6000 : STOP
740  REM
750  REM Wait for the DVM to set SRQ (hex 1000) or for a timeout
760  REM (hex 4000).  These status bits are listed in Chapter 3
770  REM of the Software Reference Manual.  If the current time
780  REM limit is too short, use ibtmo to change it.
790  REM
800       print "Waiting for SRQ..."
810       print
820       mask% = &H5000                      ' SRQI + TIMO
830       call ibwait(brd0%, mask%)
840       msg$ = "IBWAIT ERROR"
850       if (ibsta% and (EERR and TIMO)) then GOSUB 6000 : STOP
860  REM
870  REM If neither a timeout nor an error occurred, ibwait must
880  REM have returned on SRQ.  Next, do a serial poll.  First
890  REM unaddress bus devices by sending the untalk (UNT)
900  REM command (ASCII "_", or hex 5F) and the unlisten (UNL)
910  REM command (ASCII "?", or hex 3F).  Then send the Serial
920  REM Poll Enable (SPE) command (hex 18) and the DVM's talk
930  REM address (MTA1) (device 1's primary address, 1, plus
940  REM hex 40, or ASCII "A") and the GPIB interface board
950  REM listen address (MLA0) (primary address 0 plus hex 20,
960  REM or ASCII space).  These commands can be found in
970  REM Appendix A of the Software Reference Manual.
980  REM
990       cmd$ = "_?" + CHR$(&H18) + "A "
1000      call ibcmd(brd0%, cmd$)
1010      msg$ = "IBCMD ERROR"
1020      if (ibsta% and EERR) then GOSUB 6000 : STOP
1030 REM
1040 REM Now read the status byte.  If it is &H50, the Fluke 45
1050 REM has valid data to send; otherwise, it has a fault
1060 REM condition to report.
1070 REM
1080      rd$ = space$(1)
1090      call ibrd(brd0%, rd$)
1100      msg$ = "ibrd error"
```

```
1110      if (ibsta% and EERR) then GOSUB 6000 : STOP
1120 REM
1130      msg$ = "Fluke 45 error"
1140      if asc(rd$) <> &H50 then GOSUB 7000 : STOP
1150 REM
1160 REM Complete the serial poll by sending the Serial Poll
1170 REM Disable (SPD) message, hex 19.  This command can be
1180 REM found in Appendix A of the Software Reference Manual.
1190 REM
1200      cmd$ = CHR$(&H19)                    ' SPD
1210      call ibcmd(brd0%, cmd$)
1220      msg$ = "IBCMD ERROR"
1230      if (ibsta% and EERR) then GOSUB 6000 : STOP
1240 REM
1250 REM Read the measurement.
1260 REM
1270      rd$ = space$(10)
1280      call ibrd(brd0%, rd$)
1290      msg$ = "IBRD ERROR"
1300      if (ibsta% and EERR) then GOSUB 6000 : STOP
1310 REM
1320      Reading$ = left$(Rd$, IBCNT%)
1330      Print "Reading: "; Reading$
1340 REM
1350 REM Call the ibonl function to disable the hardware and
1360 REM software.
1370 REM
1380      v% = 0 : call ibonl(brd0%, v%) : STOP
1390    END

6000 REM This routine would notify you that an ib call failed
6010 REM and print the status variables.
6020 REM
6030      Print msg$
6040 REM
6050      Print "ibsta= &H"; hex$(ibsta%); " <";
6060      If ibsta% and EERR  then print " ERR";
6070      If ibsta% and TIMO then print " TIMO";
6080      If ibsta% and EEND then print " END";
6090      If ibsta% and SRQI then print " SRQI";
6100      If ibsta% and RQS  then print " RQS";
6110      If ibsta% and CMPL then print " CMPL";
6120      If ibsta% and LOK  then print " LOK";
6130      If ibsta% and RREM then print " REM";
6140      If ibsta% and CIC  then print " CIC";
6150      If ibsta% and AATN then print " ATN";
6160      If ibsta% and TACS then print " TACS";
6170      If ibsta% and LACS then print " LACS";
6180      If ibsta% and DTAS then print " DTAS";
6190      If ibsta% and DCAS then print " DCAS";
6200      Print ">"
```

```
6210 REM
6220     Print "iberr= "; iberr%;
6230     If iberr% = EDVR then print " EDVR <DOS Error>"
6240     If iberr% = ECIC then print " ECIC <Not CIC>"
6250     If iberr% = ENOL then print " ENOL <No Listener>"
6260     If iberr% = EADR then print " EADR <Address error>"
6270     If iberr% = EARG then print " EARG <Invalid argument>"
6280     If iberr% = ESAC then print " ESAC <Not Sys Ctrlr>"
6290     If iberr% = EABO then print " EABO <Op. aborted>"
6300     If iberr% = ENEB then print " ENEB <No GPIB board>"
6310     If iberr% = EOIP then print " EOIP <Async I/O in prg>"
6320     If iberr% = ECAP then print " ECAP <No capability>"
6330     If iberr% = EFSO then print " EFSO <File sys. error>"
6340     If iberr% = EBUS then print " EBUS <Command error>"
6350     If iberr% = ESTB then print " ESTB <Status byte lost>"
6360     If iberr% = ESRQ then print " ESRQ <SRQ stuck on>"
6370     If iberr% = ETAB then print " ETAB <Table Overflow>"
6380 REM
6390     Print "ibcnt = "; ibcnt%
6400 REM
6410 REM Call the ibonl function to disable the hardware and
6410 REM software.
6420 REM
6430     v% = 0 : call ibonl(brd0%, v%) : RETURN

7000 REM This routine would notify you that the device DVM
7010 REM returned an invalid serial poll response byte.
7020 REM
7030     print msg$
7040     print "Status Byte = ";rd$
7050 REM
7060 REM Call the ibonl function to disable the hardware and
7065 REM software.
7070 REM
7080     v% = 0 : call ibonl(brd0%, v%) : RETURN
```

QuickBASICプログラム例—デバイス関数

```
REM $include: 'qbdecl.bas'

declare sub gpiberr(msg$)
declare sub dvmerror(msg$)

CLS
PRINT "READ MEASUREMENT FROM FLUKE 45..."
PRINT

'  Assign a unique identifier to the FLUKE 45 that you
'  configured using IBCONF.EXE.

BDNAME$ = "DVM"
CALL IBFIND(BDNAME$, DVM%)
IF (DVM% < 0) THEN CALL GPIBERR("IBFIND ERROR")

'  Clear the device.

CALL IBCLR (DVM%)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCLR ERROR")

'  Write the function, range, and trigger source instructions
'  to the FLUKE 45.

WRT$ = "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
CALL IBWRT (DVM%,WRT$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

'  Trigger the device and request measurement.

CALL IBTRG (DVM%)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBTRG ERROR")

WRT$ = "VAL1?"
CALL IBWRT(DVM%, WRT$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

'  Wait for the DVM to set RQS (hex 800) or for a timeout
'  (hex 4000).  These status bits are listed in Chapter 3 of
'  the Software Reference manual.  If the current time limit
'  is too short, use ibtmo to change it.

print "Waiting for RQS..."
print
MASK% = &H4800                    ' RQS + TIMO
CALL IBWAIT(DVM%, MASK%)
IF (IBSTA% AND (EERR or TIMO)) THEN CALL GPIBERR("IBWAIT ERROR")
```

```
'  If neither a timeout nor an error occurred, ibwait must have
'  returned on RQS.  Now read the status byte.  If it is &H50,
'  the Fluke 45 has valid data to send; otherwise, it has a
'  fault condition to report.

   CALL IBRSP(DVM%,SPR%)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRSP ERROR")

   IF SPR% <> &H50 THEN CALL DVMERR("FLUKE 45 ERROR")

'  Read the measurement.

   RD$ = SPACE$(10)
   CALL IBRD (DVM%,RD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRD ERROR")

   Reading$ = left$(Rd$, IBCNT%)
   Print "Reading: "; Reading$

'  Call the ibonl function to disable the device DVM.

   CALL IBONL (DVM%,0)
END


'  This routine prints the result of the status variables.

SUB GPIBERR(MSG$) STATIC

   PRINT MSG$

   PRINT "IBSTA=&H"; HEX$(IBSTA%); " <";
   IF IBSTA% AND EERR THEN PRINT " ERR";
   IF IBSTA% AND TIMO THEN PRINT " TIMO";
   IF IBSTA% AND EEND THEN PRINT " END";
   IF IBSTA% AND SRQI THEN PRINT " SRQI";
   IF IBSTA% AND RQS  THEN PRINT " RQS";
   IF IBSTA% AND CMPL THEN PRINT " CMPL";
   IF IBSTA% AND LOK  THEN PRINT " LOK";
   IF IBSTA% AND RREM THEN PRINT " REM";
   IF IBSTA% AND CIC  THEN PRINT " CIC";
   IF IBSTA% AND AATN THEN PRINT " ATN";
   IF IBSTA% AND TACS THEN PRINT " TACS";
   IF IBSTA% AND LACS THEN PRINT " LACS";
   IF IBSTA% AND DTAS THEN PRINT " DTAS";
   IF IBSTA% AND DCAS THEN PRINT " DCAS";
   PRINT " >"
```

```
   PRINT "IBERR="; IBERR%;
   IF IBERR% = EDVR THEN PRINT " EDVR <DOS Error>"
   IF IBERR% = ECIC THEN PRINT " ECIC <Not CIC>"
   IF IBERR% = ENOL THEN PRINT " ENOL <No Listener>"
   IF IBERR% = EADR THEN PRINT " EADR <Address error>"
   IF IBERR% = EARG THEN PRINT " EARG <Invalid argument>"
   IF IBERR% = ESAC THEN PRINT " ESAC <Not Sys Ctrlr>"
   IF IBERR% = EABO THEN PRINT " EABO <Op. aborted>"
   IF IBERR% = ENEB THEN PRINT " ENEB <No GPIB board>"
   IF IBERR% = EOIP THEN PRINT " EOIP <Async I/O in prg>"
   IF IBERR% = ECAP THEN PRINT " ECAP <No capability>"
   IF IBERR% = EFSO THEN PRINT " EFSO <File sys. error>"
   IF IBERR% = EBUS THEN PRINT " EBUS <Command error>"
   IF IBERR% = ESTB THEN PRINT " ESTB <Status byte lost>"
   IF IBERR% = ESRQ THEN PRINT " ESRQ <SRQ stuck on>"
   IF IBERR% = ETAB THEN PRINT " ETAB <Table Overflow>"

   PRINT "IBCNT="; IBCNT%

'  Call the ibonl function to disable the device DVM.

   CALL IBONL(dvm%, 0)
   STOP

END SUB


'  This routine would notify you that the DVM returned an
'  invalid serial poll response byte.

SUB DVMERR(MSG$) STATIC
   PRINT MSG$
   PRINT "Status Byte = ";SPR%

'  Call the ibonl function to disable the device DVM.

   CALL IBONL(dvm%, 0)
   STOP
END SUB
```

QuickBASICプログラム例—ボード関数

```
REM $include: 'qbdecl.bas'

declare sub gpiberr(msg$)
declare sub dvmerr(msg$)

CLS
PRINT "READ MEASUREMENT FROM FLUKE 45..."
PRINT

' Assign an unique identifier to board 0 and store in variable
' BRD0%.

BDNAME$ = "GPIB0"
CALL IBFIND(BDNAME$, BRD0%)
IF BRD0% < 0 THEN CALL GPIBERR("IBFIND ERROR")

' Send the Interface Clear (IFC) message to all devices.

CALL IBSIC(BRD0%)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBSIC ERROR")

' Turn on the Remote Enable (REN) signal.

CALL IBSRE(BRD0%, 1)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBSRE ERROR")

' Inhibit front panel control with the Local Lockout (LLO)
' command (hex 11). Place the Fluke 45 in remote mode by
' addressing it to listen. The listen address of device 1
' (MLA1) is its primary address, 1, plus hex 20. This is
' an ASCII "!". Send the Device Clear (DCL) message (hex
' 14) to clear internal device functions. Finally, address
' the GPIB interface board to talk by sending its talk
' address (MTA0), primary address plus hex 40, or ASCII "@".
' These commands can be found in Appendix A of the
' Software Reference Manual.

CMD$ = CHR$(&H11) + "!" + CHR$(&H14) + "@"
CALL IBCMD(BRD0%, CMD$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

' Write the function, range, and trigger source instructions to
' the FLUKE 45.

WRT$ = "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
CALL IBWRT(BRD0%, WRT$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")
```

```
' Send the GET message (hex 8) to trigger a measurement reading.

   CMD$ = CHR$(&H8)                    ' GET
   CALL IBCMD(BRD0%, CMD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

' Request triggered measurement reading.

   WRT$ = "VAL1?"
   CALL IBWRT(BRD0%, WRT$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

' Wait for the DVM to set SRQ (hex 1000) or for a timeout
' (hex 4000).  These status bits are listed in Chapter 3 of
' the Software Reference manual.  If the current time limit
' is too short, use IBTMO to change it.

   print "Waiting for SRQ..."
   print
   MASK% = &H5000                      ' SRQI + TIMO
   CALL IBWAIT(BRD0%, MASK%)
   IF (IBSTA% AND (EERR or TIMO) THEN
      CALL GPIBERR("IBWAIT ERROR")
   END IF

' If neither a timeout nor an error occurred, IBWAIT must have
' returned on SRQ.  Next do a serial poll.  First unaddress
' bus devices by sending the untalk (UNT) command (ASCII "_",
' or hex 5F) and the unlisten (UNL) command (ASCII "?", or hex
' 3F).  Then send the Serial Poll Enable (SPE) command (hex 18)
' and the DVM's talk address (device 1's primary address, 1,
' plus hex 40, or ASCII "A") and the GPIB interface board
' listen address (primary address 0 plus hex 20, or ASCII space).
' These commands can be found in Appendix A of the Software
' Reference Manual.

   CMD$ = "_?" + CHR$(&H18) + "A "    ' UNT, UNL, SPE, MTA1, MLA0
   CALL IBCMD(BRD0%, CMD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

' Now read the status byte.  If it is &H50, the Fluke 45 has
' valid data to send; otherwise, it has a fault condition to
' report.

   RD$ = SPACE$(1)
   CALL IBRD(BRD0%, RD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRD ERROR")

   IF ASC(RD$) <> &H50 THEN
     CALL DVMERR("FLUKE 45 ERROR")
   END IF
```

```
'  Complete the serial poll by sending the Serial Poll Disable
'  (SPD) command, hex 19.  This command can be found in Appendix
'  A of the Software Reference Manual.

   CMD$ = CHR$(&H19)                    ' SPD
   CALL IBCMD(BRD0%, CMD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

'  Read the measurement.

   RD$ = SPACE$(10)
   CALL IBRD(BRD0%, RD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRD ERROR")

   Reading$ = left$(Rd$, IBCNT%)
   Print "Reading: "; Reading$

'  Call the ibonl function to disable the hardware and software.

   CALL IBONL(BRD0%, 0)
END


'  This routine prints the result of the status variables.

SUB GPIBERR(MSG$) STATIC

   PRINT MSG$

   PRINT "IBSTA=&H"; HEX$(IBSTA%); " <";
   IF IBSTA% AND EERR THEN PRINT " ERR";
   IF IBSTA% AND TIMO THEN PRINT " TIMO";
   IF IBSTA% AND EEND THEN PRINT " END";
   IF IBSTA% AND SRQI THEN PRINT " SRQI";
   IF IBSTA% AND RQS  THEN PRINT " RQS";
   IF IBSTA% AND CMPL THEN PRINT " CMPL";
   IF IBSTA% AND LOK  THEN PRINT " LOK";
   IF IBSTA% AND RREM THEN PRINT " REM";
   IF IBSTA% AND CIC  THEN PRINT " CIC";
   IF IBSTA% AND AATN THEN PRINT " ATN";
   IF IBSTA% AND TACS THEN PRINT " TACS";
   IF IBSTA% AND LACS THEN PRINT " LACS";
   IF IBSTA% AND DTAS THEN PRINT " DTAS";
   IF IBSTA% AND DCAS THEN PRINT " DCAS";
   PRINT " >"

   PRINT "IBERR="; IBERR%;
   IF IBERR% = EDVR THEN PRINT " EDVR <DOS Error>"
   IF IBERR% = ECIC THEN PRINT " ECIC <Not CIC>"
   IF IBERR% = ENOL THEN PRINT " ENOL <No Listener>"
   IF IBERR% = EADR THEN PRINT " EADR <Address error>"
   IF IBERR% = EARG THEN PRINT " EARG <Invalid argument>"
```

```
   IF IBERR% = ESAC THEN PRINT " ESAC <Not Sys Ctrlr>"
   IF IBERR% = EABO THEN PRINT " EABO <Op. aborted>"
   IF IBERR% = ENEB THEN PRINT " ENEB <No GPIB board>"
   IF IBERR% = EOIP THEN PRINT " EOIP <Async I/O in prg>"
   IF IBERR% = ECAP THEN PRINT " ECAP <No capability>"
   IF IBERR% = EFSO THEN PRINT " EFSO <File sys. error>"
   IF IBERR% = EBUS THEN PRINT " EBUS <Command error>"
   IF IBERR% = ESTB THEN PRINT " ESTB <Status byte lost>"
   IF IBERR% = ESRQ THEN PRINT " ESRQ <SRQ stuck on>"
   IF IBERR% = ETAB THEN PRINT " ETAB <Table Overflow>"

   PRINT "IBCNT="; IBCNT%

'  Call the ibonl function to disable the hardware and software.

   CALL IBONL(BRD0%, 0)
   STOP

END SUB


'  This routine would notify you that the DVM returned an
'  invalid serial poll response byte.

SUB DVMERR(MSG$) STATIC
   PRINT MSG$
   PRINT "Status Byte = ";RD$

'  Call the ibonl function to disable the hardware and software.

   CALL IBONL(BRD0%, 0)
   STOP

END SUB
```

Microsoft BASICプログラム例—デバイス関数

```
REM $include: 'mbdecl.bas'

declare sub gpiberr(msg$)
declare sub dvmerr(msg$)

CLS
PRINT "READ MEASUREMENT FROM FLUKE 45..."
PRINT

' Assign an unique identifier to the FLUKE 45 that you
' configured using ibconf.exe.

BDNAME$ = "DVM"
CALL IBFIND(BDNAME$, DVM%)
IF (DVM% < 0) THEN CALL GPIBERR("IBFIND ERROR")

' Clear the device.

CALL IBCLR (DVM%)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCLR ERROR")

' Write the function, range, and trigger source instructions
' to the FLUKE 45.

WRT$ = "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
CALL IBWRT (DVM%,WRT$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

' Trigger the device and request measurement.

CALL IBTRG (DVM%)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBTRG ERROR")

WRT$ = "VAL1?"
CALL IBWRT(DVM%, WRT$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

' Wait for the DVM to set RQS (hex 800) or for a timeout (hex
' 4000). These status bits are listed in Chapter 3 of the
' Software Reference manual. If the current time limit is too
' short, use IBTMO to change it.

PRINT "Waiting for RQS..."
PRINT
MASK% = &H4800                    ' RQS + TIMO
CALL IBWAIT(DVM%, MASK%)
IF (IBSTA% AND (EERR or TIMO)) THEN CALL GPIBERR("IBWAIT ERROR")
```

```
'  If neither a timeout nor an error occurred, ibwait must have
'  returned on RQS.  Now read the status byte.  If it is &H50,
'  the Fluke 45 has valid data to send; otherwise, it has a
'  fault condition to report.

   CALL IBRSP(DVM%,SPR%)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRSP ERROR")

   IF SPR% <> &H50 THEN CALL DVMERR("FLUKE 45 ERROR")

'  Read the measurement.

   RD$ = SPACE$(10)
   CALL IBRD (DVM%,RD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRD ERROR")

   READING$ = LEFT$(RD$, IBCNT%)
   PRINT "READING: "; READING$

'  Call the ibonl function to disable the device DVM.

   CALL IBONL (DVM%,0)

END


'  This routine prints the result of the status variables.

SUB GPIBERR(MSG$) STATIC

   PRINT MSG$

   PRINT "IBSTA=&H"; HEX$(IBSTA%); " <";
   IF IBSTA% AND EERR THEN PRINT " ERR";
   IF IBSTA% AND TIMO THEN PRINT " TIMO";
   IF IBSTA% AND EEND THEN PRINT " END";
   IF IBSTA% AND SRQI THEN PRINT " SRQI";
   IF IBSTA% AND RQS  THEN PRINT " RQS";
   IF IBSTA% AND CMPL THEN PRINT " CMPL";
   IF IBSTA% AND LOK  THEN PRINT " LOK";
   IF IBSTA% AND RREM THEN PRINT " REM";
   IF IBSTA% AND CIC  THEN PRINT " CIC";
   IF IBSTA% AND AATN THEN PRINT " ATN";
   IF IBSTA% AND TACS THEN PRINT " TACS";
   IF IBSTA% AND LACS THEN PRINT " LACS";
   IF IBSTA% AND DTAS THEN PRINT " DTAS";
   IF IBSTA% AND DCAS THEN PRINT " DCAS";
   PRINT " >"
```

```
   PRINT "IBERR="; IBERR%;
   IF IBERR% = EDVR THEN PRINT " EDVR <DOS Error>"
   IF IBERR% = ECIC THEN PRINT " ECIC <Not CIC>"
   IF IBERR% = ENOL THEN PRINT " ENOL <No Listener>"
   IF IBERR% = EADR THEN PRINT " EADR <Address error>"
   IF IBERR% = EARG THEN PRINT " EARG <Invalid argument>"
   IF IBERR% = ESAC THEN PRINT " ESAC <Not Sys Ctrlr>"
   IF IBERR% = EABO THEN PRINT " EABO <Op. aborted>"
   IF IBERR% = ENEB THEN PRINT " ENEB <No GPIB board>"
   IF IBERR% = EOIP THEN PRINT " EOIP <Async I/O in prg>"
   IF IBERR% = ECAP THEN PRINT " ECAP <No capability>"
   IF IBERR% = EFSO THEN PRINT " EFSO <File sys. error>"
   IF IBERR% = EBUS THEN PRINT " EBUS <Command error>"
   IF IBERR% = ESTB THEN PRINT " ESTB <Status byte lost>"
   IF IBERR% = ESRQ THEN PRINT " ESRQ <SRQ stuck on>"
   IF IBERR% = ETAB THEN PRINT " ETAB <Table Overflow>"

   PRINT "IBCNT="; IBCNT%

'  Call the ibonl function to disable the device DVM.

   CALL IBONL (DVM%,0)
   STOP

END SUB


'  This routine would notify you that the DVM returned an
'  invalid serial poll response byte.

SUB DVMERR(MSG$) STATIC
   PRINT MSG$
   PRINT "Status byte = ";SPR%

'  Call the ibonl function to disable the device DVM.

   CALL IBONL (DVM%,0)
   STOP

END SUB
```

Microsoft BASICプログラム例—ボード関数

```
REM $include: 'mbdecl.bas'

declare sub gpiberr(msg$)
declare sub dvmerr(msg$)

CLS
PRINT "READ MEASUREMENT FROM FLUKE 45..."
PRINT

' Assign a unique identifier to board 0 and store in variable
' BRD0%.

BDNAME$ = "GPIB0"
CALL IBFIND(BDNAME$, BRD0%)
IF BRD0% < 0 THEN CALL GPIBERR("IBFIND ERROR")

' Send the Interface Clear (IFC) message to all devices.

CALL IBSIC(BRD0%)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBSIC ERROR")

' Turn on the Remote Enable (REN) signal.

CALL IBSRE(BRD0%, 1)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBSRE ERROR")

' Inhibit front panel control with the Local Lockout (LLO)
' command (hex 11). Place the Fluke 45 in remote mode by
' addressing it to listen. The listen address of device 1
' (MLA1) is its primary address, 1, plus hex 20. This is an
' ASCII "!". Send the Device Clear (DCL) message (hex 14) to
' clear internal device functions. Finally, address the GPIB
' interface board to talk by sending its talk address (MTA0),
' the primary address 0 plus hex 40, or ASCII "@". These
' commands can be found in Appendix A of the Software
' Reference Manual.

CMD$ = CHR$(&H11) + "!" + CHR$(&H14) + "@"
CALL IBCMD(BRD0%, CMD$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

' Write the function, range, and trigger source instructions
' to the FLUKE 45.

WRT$ = "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
CALL IBWRT(BRD0%, WRT$)
IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

' Send the GET message (hex 8) to trigger a measurement
```

```
'  reading.
   CMD$ = CHR$(&H8)                    ' GET
   CALL IBCMD(BRD0%, CMD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

'  Request triggered measurement reading.

   WRT$ = "VAL1?"
   CALL IBWRT(BRD0%, WRT$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBWRT ERROR")

'  Wait for the DVM to set SRQ (hex 1000) or for a timeout
'  (hex 4000).  These status bits are listed in Chapter 3 of
'  the Software Reference Manual.  If the current time limit
'  is too short, use IBTMO to change it.

   PRINT "Waiting for SRQ..."
   PRINT
   MASK% = &H5000                      ' SRQI + TIMO
   CALL IBWAIT(BRD0%, MASK%)
   IF (IBSTA% AND (EERR or TIMO)) THEN
      CALL GPIBERR("IBWAIT ERROR")
   END IF

'  If neither a timeout nor an error occurred, IBWAIT must have
'  returned on SRQ.  Next, do a serial poll.  First unaddress
'  bus devices by sending the untalk (UNT) command (ASCII "_",
'  or hex 5F) and the unlisten (UNL) command (ASCII "?", or hex
'  3F).  Then send the Serial Poll Enable (SPE) command (hex 18)
'  and the DVM's talk address (device 1's primary address, 1,
'  plus hex 40, or ASCII "A") and the GPIB interface board
'  listen address (primary address 0 plus hex 20, or ASCII space).
'  These commands can be found in Appendix A of the Software
'  Reference Manual.

    CMD$ = "_?" + CHR$(&H18) + "A "   ' UNT, UNL, SPE, MTA1, MLA0
    CALL IBCMD(BRD0%, CMD$)
    IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

'  Now read the status byte.  If it is &H50, the Fluke 45 has
'  valid data to send; otherwise, it has a fault condition to
'  report.

   RD$ = SPACE$(1)
   CALL IBRD(BRD0%, RD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRD ERROR")

   IF ASC(RD$) <> &H50 THEN
     CALL DVMERR("FLUKE 45 ERROR")
   END IF

'  Complete the serial poll by sending the Serial Poll Disable
'  (SPD) command, hex 19.  This command can be found in Appendix
```

```
'  A of the Software Reference Manual.
   CMD$ = CHR$(&H19)                    ' SPD
   CALL IBCMD(BRD0%, CMD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBCMD ERROR")

'  Read the measurement.

   RD$ = SPACE$(10)
   CALL IBRD(BRD0%, RD$)
   IF (IBSTA% AND EERR) THEN CALL GPIBERR("IBRD ERROR")

   READING$ = LEFT$(RD$, IBCNT%)
   PRINT "READING: "; READING$

'  Call the ibonl function to disable the hardware and software.

   CALL IBONL(BRD0%, 0)
END


'  This routine prints the result of the status variables.

SUB GPIBERR(MSG$) STATIC

   PRINT MSG$

   PRINT "IBSTA=&H"; HEX$(IBSTA%); " <";
   IF IBSTA% AND EERR THEN PRINT " ERR";
   IF IBSTA% AND TIMO THEN PRINT " TIMO";
   IF IBSTA% AND EEND THEN PRINT " END";
   IF IBSTA% AND SRQI THEN PRINT " SRQI";
   IF IBSTA% AND RQS  THEN PRINT " RQS";
   IF IBSTA% AND CMPL THEN PRINT " CMPL";
   IF IBSTA% AND LOK  THEN PRINT " LOK";
   IF IBSTA% AND RREM THEN PRINT " REM";
   IF IBSTA% AND CIC  THEN PRINT " CIC";
   IF IBSTA% AND AATN THEN PRINT " ATN";
   IF IBSTA% AND TACS THEN PRINT " TACS";
   IF IBSTA% AND LACS THEN PRINT " LACS";
   IF IBSTA% AND DTAS THEN PRINT " DTAS";
   IF IBSTA% AND DCAS THEN PRINT " DCAS";
   PRINT " >"

   PRINT "IBERR="; IBERR%;
   IF IBERR% = EDVR THEN PRINT " EDVR <DOS Error>"
   IF IBERR% = ECIC THEN PRINT " ECIC <Not CIC>"
   IF IBERR% = ENOL THEN PRINT " ENOL <No Listener>"
   IF IBERR% = EADR THEN PRINT " EADR <Address error>"
   IF IBERR% = EARG THEN PRINT " EARG <Invalid argument>"
   IF IBERR% = ESAC THEN PRINT " ESAC <Not Sys Ctrlr>"
   IF IBERR% = EABO THEN PRINT " EABO <Op. aborted>"
   IF IBERR% = ENEB THEN PRINT " ENEB <No GPIB board>"
```

```
   IF IBERR% = EOIP THEN PRINT " EOIP <Async I/O in prg>"
   IF IBERR% = ECAP THEN PRINT " ECAP <No capability>"
   IF IBERR% = EFSO THEN PRINT " EFSO <File sys. error>"
   IF IBERR% = EBUS THEN PRINT " EBUS <Command error>"
   IF IBERR% = ESTB THEN PRINT " ESTB <Status byte lost>"
   IF IBERR% = ESRQ THEN PRINT " ESRQ <SRQ stuck on>"
   IF IBERR% = ETAB THEN PRINT " ETAB <Table Overflow>"

   PRINT "IBCNT="; IBCNT%

'  Call the ibonl function to disable the hardware and software.

   CALL IBONL(BRD0%, 0)
   STOP
END SUB


'  This routine would notify you that the DVM returned an
'  invalid serial poll response byte.

SUB DVMERR(MSG$) STATIC
   PRINT MSG$
   PRINT "Status Byte = ";RD$

'  Call the ibonl function to disable the hardware and software.

   CALL IBONL(BRD0%, 0)
   STOP
END SUB
```

Cプログラム例—デバイス関数

```
#include <stdio.h>
#include <stdlib.h>
#include "decl.h"

void dvmerr(char *msg, char code);  /* device error function     */
void gpiberr(char *msg);            /* gpib error function      */


/* Application program variables passed to GPIB functions       */

   char rd[512];                /* read data buffer      */
   int  dvm;                    /* device number         */
   char spr;                    /* serial poll response byte     */


main() {
    system("cls");

    printf("READ MEASUREMENT FROM FLUKE 45...\n");
    printf("\n");

 /* Assign a unique identifier to the Fluke 45 that you
    configured using ibconf.exe.        */

    if ((dvm = ibfind ("DVM")) < 0) {
       gpiberr("ibfind Error");
       exit(1);
       }

 /* Clear the device.           */

    if (ibclr (dvm) & ERR) {
       gpiberr("ibclr Error");
       exit(1);
       }

 /* Write the function, range, and trigger source
    instructions to the DVM.            */

    ibwrt (dvm,"*RST; VAC; AUTO; TRIGGER 2; *SRE 16", 35L);
    if (ibsta & ERR) {
       gpiberr("ibwrt Error");
       exit(1);
       }

 /* Trigger the device and request measurement.  */

    if (ibtrg (dvm) & ERR) {
       gpiberr("ibtrg Error");
       exit(1);
```

```
      }
   ibwrt (dvm,"VAL1?", 5L);
   if (ibsta & ERR) {
      gpiberr("ibwrt Error");
      exit(1);
      }

/* Wait for the DVM to set RQS or for a timeout; if the
   current time limit is too short, use ibtmo to change it.     */

   printf("Waiting for RQS...\n");
   printf("\n");
   if (ibwait (dvm,TIMO|RQS) & (ERR|TIMO)) {
      gpiberr("ibwait Error");
      exit(1);
      }

/* Because neither a timeout nor an error occurred, ibwait
   must have returned on RQS.  Next, serial poll the device.    */

   if (ibrsp (dvm, &spr) & ERR) {
      gpiberr("ibrsp Error");
      exit(1);
      }

/* Now test the status byte.  If spr is 0x50, the Fluke 45
   has valid data to send; otherwise, it has a fault
   condition to report.                  */

   if (spr != 0x50) {
      dvmerr("Fluke 45 Error", spr);
      exit(1);
      }

/* If the data is valid, read the measurement.          */

   if (ibrd (dvm,rd,10L) & ERR) {
      gpiberr("ibrd Error");
      exit(1);
      }

   rd[ibcnt] = '\0';

   printf("Reading :  %s\n", rd);

/* Call the ibonl function to disable device DVM.       */

   ibonl (dvm,0);
   }


void gpiberr(char *msg) {
```

```
/* This routine would notify you that an ib call failed. */

   printf ("%s\n", msg);

   printf ( "ibsta=&H%x  <", ibsta);
   if (ibsta & ERR ) printf (" ERR");
   if (ibsta & TIMO) printf (" TIMO");
   if (ibsta & END ) printf (" END");
   if (ibsta & SRQI) printf (" SRQI");
   if (ibsta & RQS ) printf (" RQS");
   if (ibsta & CMPL) printf (" CMPL");
   if (ibsta & LOK ) printf (" LOK");
   if (ibsta & REM ) printf (" REM");
   if (ibsta & CIC ) printf (" CIC");
   if (ibsta & ATN ) printf (" ATN");
   if (ibsta & TACS) printf (" TACS");
   if (ibsta & LACS) printf (" LACS");
   if (ibsta & DTAS) printf (" DTAS");
   if (ibsta & DCAS) printf (" DCAS");
   printf (" >\n");

   printf ("iberr= %d", iberr);
   if (iberr == EDVR) printf (" EDVR <DOS Error>\n");
   if (iberr == ECIC) printf (" ECIC <Not CIC>\n");
   if (iberr == ENOL) printf (" ENOL <No Listener>\n");
   if (iberr == EADR) printf (" EADR <Address error>\n");
   if (iberr == EARG) printf (" EARG <Invalid argument>\n");
   if (iberr == ESAC) printf (" ESAC <Not Sys Ctrlr>\n");
   if (iberr == EABO) printf (" EABO <Op. aborted>\n");

   if (iberr == ENEB) printf (" ENEB <No GPIB board>\n");
   if (iberr == EOIP) printf (" EOIP <Async I/O in prg>\n");
   if (iberr == ECAP) printf (" ECAP <No capability>\n");
   if (iberr == EFSO) printf (" EFSO <File sys. error>\n");
   if (iberr == EBUS) printf (" EBUS <Command error>\n");
   if (iberr == ESTB) printf (" ESTB <Status byte lost>\n");
   if (iberr == ESRQ) printf (" ESRQ <SRQ stuck on>\n");
   if (iberr == ETAB) printf (" ETAB <Table Overflow>\n");

   printf ("ibcnt= %d\n", ibcntl);
   printf ("\n");

 /* Call the ibonl function to disable device DVM.       */

    ibonl (dvm,0);
    }


void dvmerr(char *msg,  char spr) {
```

```
/* This routine would notify you that the DVM returned an
   invalid serial poll response byte.    */

   printf ("%s\n", msg);
   printf("Status Byte = %x\n", spr);

 /* Call the ibonl function to disable device DVM.       */

   ibonl (dvm,0);
   }
```

Cプログラム例—ボード関数

```
#include <stdio.h>
#include "decl.h"

/* Application program variables passed to GPIB functions        */

  char rd[512];                  /* read data buffer      */
  int  bd;                       /* board or device number        */

void dvmerr(char *msg, char *code);  /* device error function    */
void gpiberr(char *msg);             /* gpib error function      */


main() {
   system("cls");

   printf("READ MEASUREMENT FROM FLUKE 45...\n");
   printf("\n");

/* Assign an unique identifier to board 0 and store in
/* variable bd.  */

   if ((bd = ibfind ("GPIB0")) < 0) {
      gpiberr("ibfind Error");
      exit(1);
      }

/* Send the Interface Clear (IFC) message to all devices.        */

   if (ibsic (bd) & ERR) {
      gpiberr("ibsic Error");
      exit(1);
      }

/* Turn on the Remote Enable (REN) signal.       */

   if (ibsre (bd,1) & ERR) {
      gpiberr("ibsre Error");
      exit(1);
      }

/* Inhibit front panel control with the Local Lockout (LLO)
   command, place the Fluke 45 in remote mode by addressing it
   to listen, send the Device Clear (DCL) message to clear
   internal device functions, and address the GPIB board to
   talk.         */

   ibcmd (bd,"\021!\024@",4L);
   if (ibsta & ERR) {
      gpiberr("ibcmd Error");
      exit(1);
      }
```

```
/* Write the function, range, and trigger source
   instructions to Fluke 45.     */

   ibwrt (bd,"*RST; VAC; AUTO; TRIGGER 2; *SRE 16", 35L);
   if (ibsta & ERR) {
      gpiberr("ibwrt Error");
      exit(1);
      }

/* Send the GET message to trigger a measurement reading.        */

   ibcmd (bd,"\010",1L);
   if (ibsta & ERR) {
      gpiberr("ibcmd Error");
      exit(1);
      }

/* Request the triggered measurement reading.    */

   ibwrt (bd,"VAL1?", 5L);
   if (ibsta & ERR) {
      gpiberr("ibwrt Error");
      exit(1);
      }

/* Wait for the DVM to set SRQ or for a timeout; if the current
   time limit is too short, use ibtmo to change it.      */

   printf("Waiting for SRQ...\n");
   printf("\n");
   if (ibwait (bd,TIMO|SRQI) & (ERR|TIMO)) {
      gpiberr("ibwait Error");
      exit(1);
      }

/* Because neither a timeout nor an error occurred, ibwait must
   have returned on SRQ.  Next, do a serial poll.  First,
   unaddress bus devices and send the Serial Poll Enable (SPE)
   command, followed the talk address of the Fluke 45, and the
   listen address of the GPIB board.     */

   ibcmd (bd,"?_\030A ",5L);      /* UNL UNT SPE TAD MLA  */
   if (ibsta & ERR) {
      gpiberr("ibcmd Error");
      exit(1);
      }

/* Now read the status byte.  If it is 0x50, the Fluke 45 has
   valid data to send; otherwise, it has a fault condition to
   report.       */
```

```
   if (ibrd (bd,rd,1L) & ERR) {
      gpiberr("ibrd Error");
      exit(1);
      }

   if (rd[0] != 0x50) {
      dvmerr("Fluke 45 Error", rd);
      exit(1);
      }

/* Complete the serial poll by sending the Serial Poll Disable
   (SPD) message.        */

   if (ibcmd (bd,"\031",1L) & ERR) {
      gpiberr("ibcmd Error");
      exit(1);
      }

/* Because the DVM and GPIB board are still addressed to talk
   and listen, the measurement can be read as follows:   */

   if (ibrd (bd,rd,10L) & ERR) {
      gpiberr("ibrd Error");
      exit(1);
      }

   rd[ibcnt] = '\0';

   printf("Reading :  %s\n", rd);

/* Call the ibonl function to disable the hardware and
   software.     */

   ibonl (bd,0);
   }


void gpiberr(char *msg) {

/* This routine would notify you that an ib call failed. */

   printf ("%s\n", msg);

   printf ( "ibsta=&H%x  <", ibsta);
   if (ibsta & ERR ) printf (" ERR");
   if (ibsta & TIMO) printf (" TIMO");
   if (ibsta & END ) printf (" END");
   if (ibsta & SRQI) printf (" SRQI");
   if (ibsta & RQS ) printf (" RQS");
   if (ibsta & CMPL) printf (" CMPL");
   if (ibsta & LOK ) printf (" LOK");
   if (ibsta & REM ) printf (" REM");
```

```
   if (ibsta & CIC ) printf (" CIC");
   if (ibsta & ATN ) printf (" ATN");
   if (ibsta & TACS) printf (" TACS");
   if (ibsta & LACS) printf (" LACS");
   if (ibsta & DTAS) printf (" DTAS");
   if (ibsta & DCAS) printf (" DCAS");
   printf (" >\n");

   printf ("iberr= %d", iberr);
   if (iberr == EDVR) printf (" EDVR <DOS Error>\n");
   if (iberr == ECIC) printf (" ECIC <Not CIC>\n");
   if (iberr == ENOL) printf (" ENOL <No Listener>\n");
   if (iberr == EADR) printf (" EADR <Address error>\n");
   if (iberr == EARG) printf (" EARG <Invalid argument>\n");
   if (iberr == ESAC) printf (" ESAC <Not Sys Ctrlr>\n");
   if (iberr == EABO) printf (" EABO <Op. aborted>\n");
   if (iberr == ENEB) printf (" ENEB <No GPIB board>\n");
   if (iberr == EOIP) printf (" EOIP <Async I/O in prg>\n");
   if (iberr == ECAP) printf (" ECAP <No capability>\n");
   if (iberr == EFSO) printf (" EFSO <File sys. error>\n");
   if (iberr == EBUS) printf (" EBUS <Command error>\n");
   if (iberr == ESTB) printf (" ESTB <Status byte lost>\n");
   if (iberr == ESRQ) printf (" ESRQ <SRQ stuck on>\n");
   if (iberr == ETAB) printf (" ETAB <Table Overflow>\n");

   printf ("ibcnt= %d\n", ibcntl);
   printf ("\n");

/* Call the ibonl function to disable the hardware
   and software. */

   ibonl (bd,0);
   }


void dvmerr(char *msg,char *code) {

/* This routine would notify you that the DVM returned an
   invalid serial poll response byte.    */

   printf ("%s\n", msg);
   printf("Status byte = %x\n", code[0]);

/* Call the ibonl function to disable the hardware
   and software. */

   ibonl (bd,0);
   }
```

第 6 章
IBIC

この章ではNI-488.2ルーチンとNI-488関数を利用してInterface Bus Interactive Control (IBIC)プログラムを実行する手続きを説明しています。また、これらのルーチンや関数のシンタクスと重要な特性を述べ、IBICのプログラム例を示しています。

インターフエース・バス・インタラクティブ・コントロール即ちIBICは、ユーザーが関数をキーボードから入力することによってGPIBデバイスと交信することが出来るようになっているプログラムです。ユーザーの皆様は、IBICの使用を通してデバイスとの通信、故障の発見と解決、及びアプリケーションプログラムの開発の方法を学ぶことが出来ます。

IBICで使用する関数はこのマニュアルの第4章と第5章に説明されている NI-488.2ルーチンとNI-488関数のほとんど、およびIBICに特有の補助関数を含みます。IBICではIL (QuickBASICの関数フォーマット) はサポートしません。

IBICを使用すれば、GPIBコマンドをキーボードからデバイスに入力し、デバイスから受け取ったデータをスクリーンに表示することが出来ます。コマンドが実行される度に、数値とステータスワードibstaを簡略記号で表わしたものが表示されます。バイトカウントのibcntとエラーコードのiberrもまた必要に応じて表示されます。

IBICのインタラクティブなデータ入力法とデータ／ステータスの出力法はユーザーの方々にNI-488関数とNI-488.2ルーチンを使用して各々のデバイスをプログラムするための方法を覚えていただく手段として開発されたものです。ユーザー各位がひとたび自分のシステムに合ったステップのシーケンスを作り出せば、適当な言語と第4—6章に説明されたシンタクスを使用してそのシーケンスをアプリケーションプログラムに組み込むことは容易です。

IBIC を走らせる

INSTALL を走らせて、ディストリビューションディスクからIBICプログラムの IBIC.EXE を適当なサブディレクトリ(「適当なサブディレクトリ」は御使用のインターフェースボードによって変わります) にコピーすると、実行可能なファイルが得られます。以下の例では GPIB-PC を「適当なサブディレクトリ」として使っています。

IBIC を走らせるには、ディレクトリをこのサブディレクトリに替え、現われたプロンプトで *ibic* と入力してください。

注) 入力後 <Enter> キーを押してください。以下の例では、ユーザーの入力はイタリックで示されています。

```
C:\GPIB-PC>          ibic


National Instruments
IEE-488 Interface Bus Interactive Control Program (IBIC)
Copyright © 1990 National Instruments, Inc., Version 1.0
All Rights Reserved

Type "help" for help.
```

スクリーン上には、HELP, ibfind 及び SET についての情報を与えるメッセージが表示されます。

IBIC プログラムで最初に入力を促すプロンプトはコロン (:) です。

NI-488.2 ルーチンの使用

NI-488.2 ルーチンモードを選択したい場合は SET を使います。この場合の SET コマンドの書式シンタクスは次の通りです。

```
set 488.2  n
```

ここで n はボード番号を示します。(例えば gpib1の場合は n=1です。)

注) n のデフォルト値は 0 (gpib0)です。ここで示された以外のボードインデックスを使用することも出来ます。

上記の書式で SET コマンドを入力しますと、IBIC は 488.2 のプロンプトを示し、ボード n の NI-488.2 モードに入ったことを告げます。

```
set 488.2 1
488.2    (1):
```

この set 488.2 コマンドの入力後は、488.2 ルーチンは何でも使用出来ます。NI-488.2 ルーチンのシンタクスについては表 6 － 2 を御覧ください。

Send の使用

Send ルーチンはデータを単一の GPIB デバイスに送ります。SendList コマンドはデータを復数のGPIB デバイスに送るために使用出来ます。例えば、コンピュータから 1 次アドレスが 2 と 17 にある 2 つのデバイスに 5 文字の 文字列 *IDN? とそれに続くNL(改行)文字及び EOI を送るには、488.2 (0):なるプロンプトに対し次の如くコマンドを入力します。

```
488.2   (0):     SendList 2, 17 "*IDN?" NLend
[0128]    (cmpl  cic  tacs)
count:  6
```

すると、返されたステータスワードの [0128] が入出力がうまく完了したことを示します。バイトカウントの 6 は 6 文字全部がコンピュータから 2 つのデバイスに送られたことを告げています。

Receive の使用

Receive ルーチンを使用すると、GPIB ボードが他の GPIB デバイスからデータを受け取ります。次に示す例は Receive の使用法の説明になっています。

```
488.2   (0):     Receive 5 10 STOPend
[2124]    (end  cmpl  cic  tacs)
count:  5
48  65  6c  6c  6f                H  e  l  l  o
```

上に示す例のコマンドにより1次アドレスが5であるデバイスからデータを得ることが出来ます。このコマンドでは、10文字が受信され、ENDメッセージを受け取ったところで動作が終ります。受け取ったデータは16進数とそれに相当するASCII文字の両方法で表示されます。また、ステータスワードとバイト数も表示されます。

NI-488 関数の使用

IBIC中で NI-488関数を使用する場合、最も重要なコマンドは HELP, ibfind か ibdev, ibwrt, 及び ibrd です。これらの関数は次の数節にわたって説明されています。

HELP の使用

HELP によって IBICの情報および IBIC環境下で使用出来る 各関数のオンライン情報を得ることが出来ます。これによってGPIB呼び出しの機能やシンタクスをチェックするための参考事項を手早く知ることが出来ます。

IBFIND の使用

NI-488 GPIB関数を実行するには、先ず ibfind によってこれから使用するデバイスなりボードを開きます。デバイス或いはボードがオープンすると、プロンプトにはそのデバイスなりボードの符号名が現われます。NI-488.2 のルーチンを使用の場合には ibfind を呼び出す必要はありません。直ちに SET コマンドを使って NI-488.2 モードに入ってください。(既出の「NI-488.2 ルーチンの使用」の項を参照してください。)

次の２つの例は ibfind により dev1(例１）と gpib0 (例２）を開く場合を示しています。

例１

```
: ibfind dev1
id = 32005

dev1:
```

例２

```
: ibfind gpib0
id = 32005

gpib0:
```

ibfind において使用されるデバイスやボードの名は、IBCONF で説明されていて (第２章の *IBCONF* の項参照) ドライバが認識できる有効な符号名でなければなりません。dev1 と gpib0 はドライバにおけるデフォルトの名です。IBIC では、大文字と小文字の間の区別はしていません。

IBDEV の使用

あるGPIB関数を実行するためには、先ずibfindかibdevを使ってまだ使っていないデバイスを開いて初期化する必要があります。ibdevコマンドはまだ開いていないデバイスを選択し、それを開き、アクセスボードを割り当て、以下のフィールドの入力値にしたがってそのデバイスを初期化します。

- 1次アドレス
- 2次アドレス
- タイムアウト設定値
- EOT
- EOS

例1ではibdevは使用可能なデバイスを開き、それを1次アドレス6 (pad = 6), 2次アドレスhex 67 (sad = 0x67)のgpib0 (board = 0)に、,タイムアウト10 msec (tmo = 7), ENDメッセージ可能化 (eot = 1)およびEOSモード不可能化(eos = 0)で割り当てます。

例1

```
: ibdev 0 6 0x67 7 1 0
id = 32006

ud0:
```

もし上の例の入力の代りに、ただ

ibdev <Enter>

と入力すると、下の例2に示すように各パラメータの値を入力するよう促すスクリーンが現われます。

例2

```
: ibdev
     enter board index:     0
     enter primary address:       6
     enter secondary address:       0x67
     enter timeout:     7
     enter 'EOI on last byte' flag:       1
     enter end-of-string mode/byte:       0
id = 32006

ud0
```

ibdev呼び出しの時に起こるエラーのうち以下の3種類が目立っています。

- デバイスが見当たらなかった場合にはEDVRエラーが返ってきます。また、EDVRエラーは存在しないボードのボードインデックスを指定した場合（0か1以外が入力された場合）にも起こります。例3に示すのはこの場合のエラーです。

例3

```
:  ibdev 4 6 0x67 7 1 0
[8000]    ( err )
error:  EDVR  (-1)

:
```

- 入力したボードインデックスが存在する筈のボード(つまり0か1）を指定していても、ドライバがそのボードを見つけられないとENEBエラーが返されます。この場合には、IBCONFを走らせて、そのボードの基底アドレスが間違っていないか否かを確かめてください。

- 最後の5つのパラメータの何れかが違法値であると、ibdevの呼び出しはudxのプロンプトとEARGエラー(無効な関数引き数）を返します。例4に示すのはこの誤りの例です。

例4

```
: ibdev 0 66 0x67 7 1 0
[8000]   ( err cmpl )
error:  EARG

ud0:    ibpad 6
previous  value:  16
```

ibdevがEARGエラーを返した場合、どのパラメータが誤っているかを見出し、それをしかるべきコマンドを使用して使用せねばなりません。例4ではpadが誤っているのでibpadを使用して訂正します。

注) ibdevの呼び出しはすべてのデバイスに対するibfindの呼び出しが終わってからにしてください。こうすることによって後にibfind呼び出しで用いるかもしれないデバイスをibdevが選択してしまうおそれがなくなります。また、すべてのデバイス識別子はibonl 0呼び出しによってオフラインにされるまでは有効なので、一つのアプリケーションが終わったところですべてのデバイスをオフラインにすることを忘れないでください。こうしておけば、次のアプリケーションプログラムの実行の際にibdevが正しく動作するようになります。

IBWRT の使用

ibwrtコマンドはデータをGPIBデバイスに送る時に使用します。下に示すのは6文字のデータストリングF3R5T1をコンピュータからdev1と言うデバイスに送る時の例です。dev1:と言うプロンプトでデータストリングを入力します。

例

```
dev1:  ibwrt "F3R5T1"
[0100]  (cmpl)
count:   6
```

ステータスワードの [0100] は入出力動作が問題なく完了したことを示しています。また、バイトカウント6は6文字がコンピュータから送られ、デバイスにより受け取られたことを示しています。

IBRD の使用

ibrd コマンドはある GPIB デバイスが他の GPIB デバイスからデータを受け取るようにします。下の例は ibrd 関数の使用法を示しています。

例

```
dev1:  ibrd 20
[2100]  (end  cmpl)
count:   18
4E  44  43  56  28  30  30  30  N D C V ( 0 0 0
2E  30  30  34  37  45  2B  30  . 0 0 4 7 E + 0
0D  0A                          • •
```

このコマンドはデバイスからデータを受取り、それをスクリーン上に16進法フォーマットとそれに相当する ASCII の形で示します。さ

らに、スクリーン上には、ステータスワードやバイト数の如きデータ転送上の情報も表示されます。

IBICからのExit (イグジット)

e 或いは q を入力するとDOS オペレーティングシステムに戻ることが出来ます。

一般に用いられる EOS 文字の付け加え

GPIB 計測器のなかには終止文字とか列終止 (EOS) 文字といわれる特殊な文字によって伝送の終了を知らせる必要があるものがあります。もし御使用のデバイスが EOS 文字を必要としている場合は、ibwrt ステートメントにより伝送するデータストリングの最後に EOS 文字を付け加えてください。

下に示す例では、最もよく使われる EOS 文字である復帰文字と改行文字が付け加えられています。

例

```
dev1:  ibwrt "F3R5T1\r\n
[0100]  (cmpl)
count:   6
```

ここで \r と \n は各々復帰文字と改行文字を表わしています。このようなプリント不可能な文字の表わし方については表 6 − 4 の注に詳細な説明があります。

SET の使用

NI-488 関数の使用においては、デバイスやボードは ibfind を使用して開きます。デバイスやボードを開いたら、補助関数の SET を使用して、開かれたデバイスかボードのうちどれを選択するかを指定します。すると SET は プロンプトにおけるデバイス名を IBCONF で指定した符号名に変えます。なお、この機能の他に、SET はNI-488 モードと NI-488.2 モードの間の変換を行う際にも使用されます。

例

```
dev1:  set plotter
id:  32006

plotter:
```

上の例では、デバイスはIBCONF によって plotter と名付けられています。

次にibfind と SET の使用の代表的な例をまとめて示します。

例

```
: ibfind dev1
id = 32006
dev1:    ibfind plotter

plotter:      ibwrt "F3T7G0"
[0100]    (cmpl)
count:  6

plotter:     set dev1

dev1:    ibwrt "X7Y39G0"
[0100]    (cmpl)
count:  7

dev1:
```

IBIC 関数とシンタクス

IBIC は関数の呼び出しの直後にそれぞれの関数について次のような情報を表示します。

- ibrd と ibrda のデータメッセージはスクリーン上に 16 進数とASCII のフォーマットの両方で表示されます。
- グローバル変数の ibsta, ibcnt 及び iberr はスクリーン上に表示されます。

IBIC と第 5 章のプログラミング言語では関数呼び出しのシンタクスが違っています。主な相違は IBIC では ibwrt, ibwrta, ibcmd 及び ibcmda のメッセージがキーボードからストリングとして入力

されるという点です。IBICのシンタクスは表6－1と6－2に示されています。

IBICではユニット記述子(ud)は関数シンタクスの一部としては指定されていません。デバイスやボードを使用する前には、ibfindを呼び出してそのユニットを開き、ユニット記述子をIBICに引き渡すようにします。すると、スクリーン上のプロンプトが、これらの開かれたユニットのうちの何れがその後の関数動作においてIBICにより使用されるかを示します。これらのユニットの一つからもう一つに変換するには、SETを使用してください[ページ6-13を参照]。

表6－1と表6－2はIBICから呼び出される場合のNI-488関数及びNI-488.2ルーチンのシンタクスをそれぞれまとめたものです。IBIC用のシンタクスのルールは表に付属した注に説明されています。ルーチンと関数の詳細な説明と各プログラミング言語シンタクスのルールについては第4章と第5章をおのおの参照してください。

表6－1 IBICにおけるGPIB関数のシンタクス

関数シンタクス	内容	関数タイプ	注
ibbna brdname	デバイスのアクセスボードを変更する	dev	1
ibcac [v]	アクティブコントローラとなる	brd	2,3
ibclr	指定されたデバイスをクリアする	dev	
ibcmd string	ストリングからコマンドを送る	brd	4
ibcmda string	ストリングからコマンドを非同期に送る	brd	4
ibconfig mn v	構成可能のパラメータ値の変更	dev,brd	15,3
ibdev v v v v v v	未使用で名の知られていないデバイスを開く	dev	9
ibdma [v]	DMAの可能化／不可能化	brd	2,3

(次ページに続く)

表6－1 IBICにおけるGPIB関数のシンタクス(前ページより続く)

関数シンタクス	内容	関数タイプ	注
ibeos v	EOSメッセージの変更／不可能化	dev, brd	2,3
ibeot [v]	ENDメッセージの可能化／不可能化	dev, brd	2,3
ibevent	次の事象を返す	brd	
ibfind udname	ユニット記述子を返す。	dev, brd	5
ibgts [v]	アクティブコントローラの状態から待機状態に移る	brd	2,3
ibist [v]	istをセット／クリアする	brd	2,3
iblines	全てのGPIB線の状態を読む	dev, brd	
ibln v v	バス上にデバイスが存在するか否かをチェック	dev, brd	10
ibloc	ローカルモードに移る	dev, brd	
ibonl [v]	デバイスをオンライン／オフラインにする	dev, brd	2,3
ibpad v	1次アドレスを変更	dev, brd	3
ibpct	コントロールを引き渡す	dev	
ibppc v	パラレルポーリング用に構成する	dev, brd	3
ibrd v	データを読み取る	dev, brd	6
ibrda v	データを非同期に読み取る	dev, brd	6
ibrdf flname	データをファイルに読み込む	dev, brd	7
ibrpp	パラレルポーリングを行う	dev, brd	
ibrsc [v]	システムのコントロールを要求／解除	brd	2,3
ibrsp	シリアルポールのバイトを返す	dev	

(次ページに続く)

表6－1 IBICにおけるGPIB関数のシンタクス(前ページより続く)

関数シンタクス	内容	関数タイプ	注
ibrsv v	サービスを要求する	dev	3
ibsad v	2次アドレスを変更	dev, brd	3
ibsic	IFC信号を送る	brd	3
ibsre [v]	REN信号線をセット／クリア	brd	2,3
ibstop	非同期動作を中止	dev, brd	
ibtmo v	タイムアウトの変更／不可能化	dev, brd	3
ibtrap mask v	アプリケーションズモニターの構成	dev, brd	3,8
ibtrg	選択したデバイスをトリガ	dev	
ibwait [mask]	選択した事象が起きるのを待機	dev, brd	2,8
ibwrt string	データを書き込む	brd	4
ibwrta string	データを非同期に書き込む	dev, brd	4
ibwrtf flname	ファイルからデータを書き込む	dev, brd	7

表6－2 IBICにおけるNI-488.2ルーチンのシンタクス

ルーチンシンタクス	内容	注
AllSpoll list	全てのデバイスをシリアルポーリングする	11
DevClear address	一つのデバイスをクリアする	13
DevClearList list	幾つかのデバイスをクリアする	11

(次ページに続く)

表6－2 IBICにおけるNI-488.2ルーチンのシンタクス
(前ページより続く)

ルーチンシンタクス	内容	注
`EnableLocal list`	ローカルコントロールを可能化する	11
`EnableRemote list`	リモートコントロールを可能化する	11
`FindLstn list limit`	全てのリスナを見出す	11
`FindRQS list`	サービス要求中のデバイスを見出す	11
`PassControl address`	コントロールをあるデバイスに引き渡す	13
`PPoll`	パラレルポーリングを行う	13
`PPollConfig addr. line sense`	パラレルポーリング用に構成を行う	13,14
`PPollUnconfig address`	パラレルポーリング用の構成を解除する	13
`RcvRespMsg address data mode`	応答メッセージを受け取る	4,12,13
`ReadStatusByte address`	シリアルポーリングを行う	13
`Receive address data mode`	データを受信する	4,12,13
`ReceiveSetup address`	データ受信用の設定を行う	13
`ResetSys list`	デバイスを3レベルにリセットする	11
`Send address data mode`	一つのデバイスにデータを送る	4,12,13

(次ページに続く)

表6－2 IBICにおけるNI-488.2ルーチンのシンタクス
(前ページより続く)

ルーチンシンタクス	内容	注
SendCmds data	コマンドバイトを送る	4
SendDataBytes list data mode	既にアドレスされたデバイスにデータバイトを送る	4,11,12
SendIFC	IFCを送る	
SendList list data mode	複数のデバイスにデータを送る	4,11,12
SendLLO	すべてのデバイスをLLO状態にする	11
SendSetup list	特定のデバイスをデータ受信可能にする	11
SetRWLS list	特定のデバイスをリモートにロックアウトする	11
TestSys list	デバイスの自己テスト	11
TestSRQ	SRQ線のテスト	
Trigger address	一つのデバイスをトリガ	13
TriggerList list	複数のデバイスをトリガ	11
WaitSRQ	SRQを待つ	

表6－1と6－2の注

1. brdnameは新しいボードの符号名を指します(例えば、ibbna gpib1)。

2. 角括弧 [ ] 中の値はオプションとして与えるものです。デフォルト値はibwaitの場合は0、その他の関数の場合は1です。

3. v は16進数(hex)、8進数(octal)、及び10進数(decimal)の整数です。16進数はその前に0xを伴います(たとえば0xD)。8進数はその前に0を伴います(たとえば015)。そのほかの数はすべて10進数と考えてください。

4. string は ASCII 文字、8進数バイト、16進数バイト、及び特殊符号からなっています。ストリングを構成する文字の全シーケンスは引用符 " " で囲われていなければなりません。8進数バイトはバックスラッシュ\とそれに続く8進数値とからなっています。たとえば、8進数の40は\40として表わされます。16進数バイトはバックスラッシュ\ とx のつぎに16進数値が来る形で表わされます。たとえば、16進数の40は\x40として表わされます。特殊符号には、復帰文字を表わす \r と改行文字を表わす\n があります。これらの符号はストリングの中に復帰文字や改行文字を挿入したい場合に便利です。次の例はこれれら特殊符号を挿入したストリングの例です。"F3RtT1\r\n".但し、改行文字は16進数でも表わすことが出来ます。したがって、\xD と \r は改行文字を表わす等価のストリングです。

5. udname は新しいデバイス或いはボードの符号名です(例えば、ibfind dev1 とか set gpib0等です。

6. v は読むべきバイトの数です。

7. flname は、読み込まれる、或いは書き出されるファイルのMS-DOS パス名です(たとえば、\test\meter とか printr.buf 等です)。

8. mask は一つの16進、8進あるいは10進整数(注3を参照してください)或いはマスクビット簡略記号です。

9. ibdev のパラメータは board の id, pad, sad, tmo, eos 及び eot です。

10. ibln のパラメータは pad と sad です。

11. list はコンマでお互いに区切られたアドレス整数のリストで、オプションとして括弧( )で括られている場合もあります。括弧内に何も無い場合はリストが空であることを示します。

12. mode は終止モード用の簡略記号か整数です。Send型のオペレーションでは簡略記号は NLend か NULLend です。Receive型のオペレーションでは簡略記号は STOPend です。

13. address は GPIB アドレスを表わす整数です。GPIB の1次アドレスのみが必要の場合はその整数を入力するだけで結構です。2次アドレスも必要の場合には、1次アドレスを下位のバイトに置き、2次アドレスを上位のバイトに置いて1つの整数を作ってください。たとえば、pad の3と sadの 61_{16} は0x6103として表わすことが出来ます。

14. line と sense は各々応答すべきデータ線を表わす整数と、応答の内容を示す整数(1か0) です。

15. mn は構成パラメータ用簡略記号またはそれに相当する整数値です。この場合に使用出来る簡略記号及びそれらに相当する値については第5章の ibconfig の項を参照してください。

ステータスワード

全てのIBIC関数はステータスワードの ibsta を2種の形式、すなわち角括弧内に入った16進数と括弧内に入った簡略記号のつながりの形で返します。

例

```
dev1:   ibwrt "f2t3x"
[900]  (rqs  cmpl)
count:  5

dev1:
```

この例では、ステータスワードは書込関数 ibwrt のオペレーションが問題無く完了したことと dev1 がサービスを要求していることを示しています。

表6－3はステータスワード用の簡略記号のリストを示します。このリストは表3－1に示されたものと同じです。

表6－3 ステータスワードのレイアウト

簡略記号	ビット位置	Hex 値	関数型	内容
ERR	15	8000	dev, brd	GPIBエラー
TIMO	14	4000	dev, brd	時間制限をオーバー
END	13	2000	dev, brd	END あるいは EOS を検知
SRQI	12	1000	brd	SRQ割込命令を受信
RQS	11	800	dev	デバイスがサービスを要求中
EVENT	10	400	brd	DTASまたは DCAS 事象が起こっている
SPOLL	9	200	brd	ボードがコントローラによりシリアルポーリングを受けている
CMPL	8	100	dev, brd	入出力動作が完了
LOK	7	80	brd	ロックアウト状態
REM	6	40	brd	リモートモード
CIC	5	20	brd	コントローラ・イン・チャージ
ATN	4	10	brd	ATN アサート状態
TACS	3	8	brd	GPIB ボードがトーカ
LACS	2	4	brd	GPIB ボードがリスナ

(次ページに続く)

表6－3 ステータスワードのレイアウト(前ページより続く)

簡略記号	ビット位置	Hex 値	関数型	内容
DTAS	1	2	brd	デバイストリガ状態
DCAS	0	1	brd	デバイスクリア状態

エラーコード

NI-488 関数なり NI-488.2ルーチン がエラーで完了すると、IBIC は エラーを表わす簡略記号を表示します。以下に示すのは、データ転送中にエラー状態が起こった場合の例です。

例

```
dev1:   ibwrt "f2t3x"
[8100]  (err  cmpl)
ERROR:  ENOL
COUNT:  1

dev1:
```

この例は多分 dev1 のスイッチが切れたためリスナが検出できなかった場合です。

バイト数

IBICは、一つの入出力関数の動作が終了すると、実際に送受したバイトの数を表示します。これはエラー状態の存在の有無に関係無く行われます。

補助関数

表6－4にIBICがサポートする補助関数をまとめて示します。

表6－4 IBICがサポートする補助関数

内容	関数シンタクス	注
デバイス或いはボードを選択する	set udname or set 488.2 n	1,2 7
ヘルプ情報を表示する	help [option]	3
前に行った関数を繰り返す	!	
表示をしない	-	
表示をする	+	
関数をn回繰り返す	n* function	4
前の関数をn回繰り返す	n* !	
間接ファイルを実行する	$ filename	5
ストリングをスクリーン上に表示	print string	6
(イグジットまたは終止)	e	
(イグジットまたは終止)	q	

表6－4の注

1. udnameは新しいデバイス或いはボードの符号名(例えばibfind dev1とか set gpib0 など)。

2. 初めにibfindを呼び出してデバイス或いはボードを開いてください。

3. optionの入力を省略すると、代わりにオプションのメニューが表示されます。

4. functionをこれから使用する正しいIBIC関数のシンタクスで置き換えてください。

5. filenameとしてこれから実行しようとするIBIC関数を含むファイルのMS-DOSにおけるパス名を入力してください。

6. ここでstringは一連のASCII文字、8進数バイト、16進数バイト、或いは特殊符号です。この一連の文字の全ては引用符""の中に収めておく必要があります。8進数バイトはバックスラッシュ\の後に8進数値を置いてください。例えば、8進数の40は\040として表わされます。16進数バイトの場合は、バックスラッシュ\のあとにxを置き、それに16進数を続けます。例えば、16進数の40は\x40となります。特殊符号は2種類あり、\rは復帰文字を表わし、\nは改行文字を表わします。これらの特殊符号はストリング中に復帰文字や改行文字を挿入する際に用いれば便利です。例えば次の如くです。"F3R5T1\r\n" 復帰文字は16進数を用いて表わすことも出来ます。したがって \xD と \r は等価のストリングです。

7. nはボード番号を示します(例えば、gpib1の場合はn=1です)。

HELP（ヘルプ情報の表示）

HELP補助関数によってIBIC及びその各機能の情報を即座にスクリーン上に表示することが出来ます。

!(この前に実行した関数を繰り返す)

!助関数は時間的に最も近くに行われた関数機能を再び実行させます。

例

スクリーン上の表示	内容
`gpib0: ibsic` `[130]  ( cmpl cic atn )`	IFCを送る
`gpib0:  !` `[130]  ( cmpl cic atn )`	ibsic を繰り返す
`gpib0:  !` `[130]  ( cmpl cic atn )`	またibsic を繰り返す

-(表示をオフにする)

-補助関数はNI-488関数がスクリーン上の表示を行わないようにします。この機能は、あるNI-488の入出力関数をそれが一々スクリーンに表示を出すのを待たずに即座に繰り返したいときに使用すれば便利です。

+ (表示をオンにする)

+ 補助関数は、オフ (消去状態) のスクリーンに表示が再び現れるようにします。

以下に− 補助関数と + 補助関数を使用した例を示します。ここでは24の連続したアルファベット文字を3回のibrd呼び出しを使って読み取っています。

例

```
dev1:   ibrd 8
[4100]  (end  cmpl)
COUNT:  8
61  62  63  64  65  66  67  68   a  b  c  d  e  f  g  h

dev1:  -

dev1:  ibrd 8

dev1:  +

dev1:   ibrd 8
[4100]  (end  cmpl)
COUNT:  8
71  72  73  74  75  76  77  78   q  r  s  t  u  v  w  x
```

n* (関数を n 回繰り返す)

n* 補助関数は指定された関数を n 回繰り返します。ここで、n は整数です。下に示した例では、Hello と言うメッセージの書込を 5 回繰り返すことになります。

例

```
printer:  5*ibwrt "Hello"
```

その後で、同じ関数の名は！で置き換えることが出来ます。下に示した例では、Hello の送信をもう 20 回繰り返した後、さらに10 回繰り返して送信しています。

例

```
printer:  20* !
printer:  10* !
```

上の例で、乗算符 (*) が関数命令の 1 部として扱われていないことに注意してください。即ち、20 回繰り返されるのは ibwrt "Hello" であって、5* ibwrt "Hello" ではありません。

$ (間接ファイルを実行する)

$ 補助関数では、間接ファイルとはIBIC 関数を含むテキストファイルを意味します。これは MS-DOS バッチファイルと似ており、同じ方法で作成されます。$ は指定された間接ファイルを読み、まるでキーボードから入力した関数を次々と実行するかのようにIBIC 関数を順序にしたがって実行します。例えば、次のように入力すると、

```
gpib0:  $ usrfile
```

usrfile 中の IBIC 関数が実行されます。また、下の例のように入力すると、

```
gpib0:  3*$ usrfile
```

初めの例の動作が3回繰り返されます。

この関数が実行される前の表示モードは実行後に元の状態に戻されます。但し、間接ファイル中の関数がそれをコマンドにより変更した場合は別です。

PRINT (ASCII文字列の表示)

PRINT 補助関数はストリングをスクリーン上に表示するために使用されます。

例

```
dev1: print "hello"
hello
dev1: print
      "and\r\n\x67\x6f\x6f\x64\x62\x79\x65"
and
goodbye
```

PRINT は間接ファイルからのコメントを表示するために使用することが出来ます。この場合の文字列は - 補助関数を使って表示をオフにした場合もスクリーン上に現われます。

上の例で2 回目の PRINT は IBIC 文字列に16進数値を使用した例です。

E 或いは Q (exit)

MS-DOS のexitコマンド或いは IBIC 補助関数E か Q の使用で MS-DOS に戻ることができます。

IBIC の各種サンプルプログラム

NI-488.2ソフトウエアとNI-488関数をパーソナルコンピュータで使用して典型的なIEEE-488計測器をプログラムするために、第4章と第5章に各々示されているプログラミングの手順を使用することができます。ここに示されるアプリケーションはIBICコマンドを使用して書かれています。

NI-488.2 ルーチン

IBIC において NI-488.2 ルーチンを呼び出すには、set コマンドから始めます。

例

```
: set 488.2

488.2  (0):
```

次にインターフェースクリアのメッセージ ((IFC) を全てのデバイスに送り、バスをクリアします。GPIB 関数呼び出しの後では何時でも ERR をチェックしてエラーの無いことを確認してください。

例

```
488.2  (0):  SendIFC
[0120]  (cmpl  cic)
```

今度はデバイスをクリアします。このデバイスは GPIB バス上の 1 次アドレス 2 にあります。

例

```
488.2  (0):  DevClear 2
[0138]  (cmpl  cic  atn  tacs)
count:  1
```

更に、ファンクション、範囲及びトリガソースのデータをデバイス（ディジタル電圧計）に書き込みます。

例

```
488.2  (0):  Send 2 "F3R7T3" DABend
[0128]  (cmpl  cic  tacs)
count:  6
```

デバイスをトリガします。

例

```
488.2  (0):  Trigger 2
[0138]  (cmpl  cic  atn  tacs)
count:  1
```

電圧計がSRQバス線をアサートしてサービスを要求するのを待ち、その後に電圧計のステータスバイトを読み取ります。

例

```
488.2  (0):  WaitSRQ
[1138]  (srqi  cmpl  cic  atn  tacs)
SRQ  line  is  asserted


488.2  (0):   ReadStatusByte 2
[0174]  (cmpl  rem  cic  atn  lacs)
Poll:   2=>  0x0040   (decimal  :  32)
```

電圧計のデータを読み取ります。

例

```
488.2  (0):  Receive 2 20 STOPend
[2164]  (end  cmpl  rem  cic  lacs)
count:  20
0d  0a  4e  44  43  56  2d  30   .  .  N  D  C  V  -  0
30  30  2e  30  30  34  37  45   0  0  .  0  0  4  7  E
2b  30  0d  0a                   +  0  .  .
```

DOSに戻ります。

例

```
488.2  (0):  e
```

デバイス関数

あるデバイスとの通信を行うには、先ずIBCONFプログラムでそのデバイスに与えられたデバイス名を見出します。

例

```
:  ibfind dvm
id  =  32005

DVM:
```

デバイスをクリアします。GPIB 関数の呼び出しを行う度にERR をチェックしてエラーの無いことを確認してください。

例

```
DVM:  ibclr
[0100]  (cmpl)
```

ファンクション、範囲及びトリガソース命令を DVM に書き込みます。

例

```
DVM:  ibwrt "F3R7T3"
[0100]  (cmpl)
count:  6
```

デバイスをトリガします。

例

```
DVM:  ibtrg
[0100]  (cmpl)
```

DVMがサービスを要求するか、或いはタイムアウトになるまで待ちます。もし現在の制限時間が短すぎる場合は、ibtmoを使ってそれを変更してください。

例

```
DVM:  ibwait (TIMO RQS)
[800]  (rqs)
```

シリアルポールのステータスバイトを読みます。シリアルポールのステータスバイトは使用のデバイスによって変わってきます。

例

```
DVM:  ibrsp
[0100]  (cmpl)
Poll:  0x40  (decimal  :  32)
```

読み取りコマンドによりデータをスクリーン上に表示します。データは16進数とそれに相当するASCII文字で表示されています。

例

```
DVM:  ibrd 18
[0100]  (cmpl)
count:  18

4E  44  43  56  20  30  30  30   N D C V    0 0 0
2E  30  30  34  37  45  28  30   . 0 0 4 7 E + 0
0A  0A
```

DOS に戻ります。

例

```
DVM:  e
```

ボード関数

次の数ページでは対象をデバイス関数でなく、ボード関数に限って、実行法を説明します。ほとんどのアプリケーションでは、ボード関数は不必要です。

先ずインターフェースボードをカレント(アクティブ) ボードにします。

例

```
: ibfind gpib0
id = 32006

GPIB0:
```

次にインターフェースクリアのメッセージ (IFC) を全てのデバイスに送ります。これによりバスがクリアされ、ATNがアサートされます。いつものことですが、GPIB 関数の呼び出しの後はERR をチェックしてエラーの無いことを確認してください。

例

```
GPIB0:   ibsic
[0130]  (cmpl  cic  atn)
```

リモートモード可能化の信号線 REN をオンにします。

例

```
GPIB0:   ibsre 1
[0130]  (cmpl  cic  atn)
previous  value:  0
```

デバイスをリスナ、コンピュータをトーカとしてアドレスするよう設定します。下の例で、疑問符文字（?）はアンリスン (UNL) コマンドを表わします。@ 文字は GPIB のトーカとしてのアドレスであって、付録 A に有るマルチライン・インターフェースメッセージ表を基に計算して得られたものです。 GPIB ボードは GPIB 1 次アドレスの 0 にあります。メッセージ表のトークアドレスのコラムを見ると、1 次のトークアドレス0は ASCII 文字の @で示されることがわかります。同じようにして、！文字はデバイスのリスンアドレスであるとわかります。この場合のデバイスの GPIB 1 次アドレスは 1 です。付録 A のマルチライン・インターフェース表を見れば、1 次アドレスが 1 であるデバイスのリスンアドレスは！であるとわかります。

例

```
GPIB0:   ibcmd "@?!"
[0138]  (cmpl  cic  atn  tacs)
count:  3
```

ファンクション、範囲、及びトリガソース命令を DVM に書き込みます。サンプルプログラムに進む前に、エラーが起こっていないことを確認してください。

例

```
GPIB0:   ibwrt "F3R7T3"
[0128]  (cmpl  cic  tacs)
count:  6
```

GET (Group Execute Trigger) メッセージを送って測定値の読み取りをトリガします。GET メッセージは16進数の 8 で表わします。

例

```
GPIB0:   ibcmd "\x08"
[0138]  (cmpl  cic  atn  tacs)
count:  1
```

DVM が SRQ をセットするかタイムアウトになるまで待ちます。現在の制限時間が短すぎるようでしたら ibtmo を使ってそれを変更して長くします。

例

```
GPIB0:   ibwait (TIMO SRQI)
[1138]  (srqi  cmpl  cic  atn  tacs)
```

デバイスをシリアルポーリング用に設定します。下の例で (?) はアンリスン(UNL) コマンドを表わしています。1文字分の空白 ( ) はコントローラのリスンアドレスを表わします。16進数の18はシリアルポーリング可能化機能を示します。A はデバイスのトークアドレスです。

例

```
GPIB0:   ibcmd  "? \x18A"
[1174]  (srqi  cmpl  rem  cic  atn  lacs)
count:  4
```

ステータスバイトを読みます。返されるステータスバイトは使用デバイスにより変わります。

例

```
GPIB0:   ibrd 1
[0164]  (cmpl  rem  cic  lacs)
count:  1
50                    P
```

SPD (シリアルポーリング不可能化) メッセージを送ってシリアルポーリングを終了させます。16進数の19は SPD (シリアルポーリングの不可能化) をあらわしています。

例

```
GPIB0:   ibcmd  "\x19"
[0174]  (cmpl  rem  cic  atn  lacs)
count:  1
```

DVM と NI-488 はまだトーク及びリスンするようアドレスされているので、測定値の読み取りが可能です。

例

```
GPIB0:   ibrd 20
[2164]  (end  cmpl  rem  cic  lacs)
0D  0A  4E  44  43  56  2D  30   •  •  N  D  C  V  -  0
30  30  2E  30  30  34  37  45   0  0  .  0  0  4  7  E
2B  30  0D  0A                   +  0  •  •
```

DOS に戻ります。

例

```
GPIB0:  e
```

第７章
Applications Monitor

この章では、applications monitorと言うプログラムのインストール、構成、使用方法が説明されています。applications monitorはGPIB呼び出し手順のデバッグをユーザー各自のアプリケーション内から行うことが出来る便利な常駐プログラムです。applications monitorはある関数呼び出しが終了した時点で、もしその呼び出しがユーザーがあらかじめ指定した条件を満たすならば、プログラムの実行を一時中止する（トラップする）ようになっています。そのあとで、ユーザーは関数の引き数、バッファ、返された各種の値、GPIBのグローバル変数、及びその他の関係データを検討することが出来ます。プログラムを一時停止させる条件はユーザーが選択して指定します。選択できる条件として、NI-488.2ルーチンやNI-488関数が終了した時は何時でもプログラムを一時中止する、ルーチンか関数がエラーの状態を返した時にプログラムを一時中止する、或いはルーチンまたは関数がGPIBステータスワード中にある種のビットパターンを返した時に一時中止する、などです。

選択した条件の何れかが満たされると、ポップアップスクリーン表示[図７－１]が現われます。この中には、トラップされる関数の詳細な内容が示されています。また、過去に逆上って255の関数が示されますので、それによって今までになされた一連の関数の呼び出しと引き数が最初に意図した通りか否かを確かめることが出来ます。

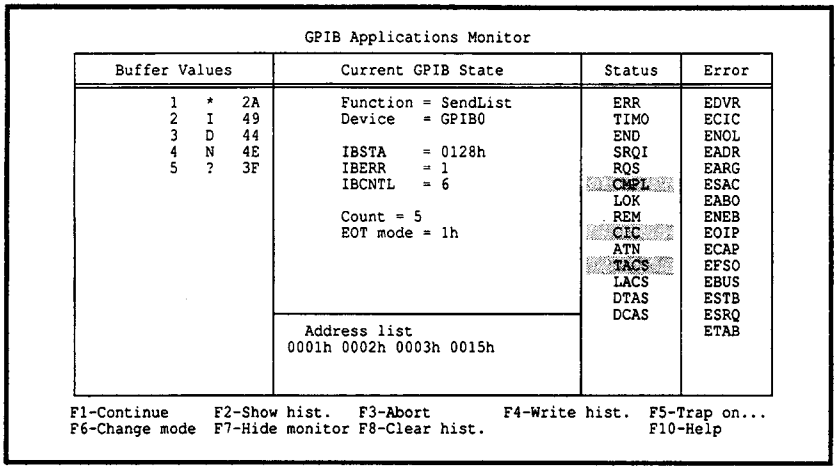


図7－1 Applications Monitorのポップアップスクリーン

applications monitorの使用中には、多くの場合わざわざエラーチェッキングをしないで済みます。エラーなしで行われるはずのプログラムが、GPIB呼び出し中にエラーが起こると、プログラムがトラップされ、applications monitorが動作するようになります。そこでユーザーは問題解決のために必要な手段をとることが出来ます。

現在NI-488.2 MS-DOSドライバの改正バージョンの全てに対してapplications monitorの各バージョンが準備されています。

Applications Monitorのインストレーション

applications monitorは、ディストリビューションディスクのなかにAPPMON.EXEと言うファイルとして収められています。それをインストールするには、DOSプロンプトが出たところで、次のコマンドをタイプします。

```
APPMON
```

GPIBドライバが存在していなかったり、applications monitorがすでにインストールされている場合は、このプログラムはロードされず、エラーメッセージが表示されます。

インストール後のapplications monitorは、システムがリスタートされるまではメモリに留まっています。後になって、applications monitorをメモリ中にとどめておく必要はないと判断したときは、システムをリスタートすれば、applications monitorはロードされた状態でなくなります。

IBTRAP

インストールされたapplications monitorは`ibtrap`関数で構成されます。applications monitorはGPIBステータスワード中にある種のビットの組み合わせを持つGPIBドライバ呼び出し関数をトラップすることが出来ます。トラップのオプションは特別なGPIBドライバ関数である`ibtrap`により設定することが出来ます。この関数はアプリケーションプログラムから呼び出すことも出来ますし、DOSプロンプトが出た時に`IBTRAP.EXE`と言うユーティリティプログラムを使って呼び出すことも出来ます。

ユーザーは関数呼び出しやDOSユーティリティによってマスクを選択します。マスクはトラップされる事象やトラップが起こったときに何が表示されるかを選択する*monitor mode*を決定します。

正確に言えば、関数呼び出しのシンタクスは使用されるプログラミング言語により多少相違があります。ユーザー各位は第5章で`ibtrap`の説明を読み、御自分のアプリケーション中での`ibtrap`呼び出しの詳細を知るようにしてください。

ユーティリティプログラムのibtrapを使用すれば、DOSからトラップモードの設定をすることが出来ます。DOSのプロンプトが出たらibtrapとタイプし、以下の数ページに列挙されるフラグを好きなように組み合わせて指示すればよいのです。

以下のマスクフラグは一つ或いはそれ以上を選択してください。

-all	GPIB呼び出しの全て
-err	GPIBエラー
-timo	タイムアウト
-end	GPIBボードがENDかEOSを検出
-srqi	サービス要求線(SRQ)がアサート中
-rqs	デバイスがサービスを要求中
-event	DTASまたはTCASメッセージを受信
-spoll	GPIBボードのシリアルポーリングが行われた
-cmpl	入出力が完了
-lok	GPIBボードがロックアウト状態
-rem	GPIBボードがリモートモード
-cic	GPIBボードがコントローラインチャージ(CIC)
-atn	ATN線がアサート中
-tacs	GPIBボードがトーカ
-lacs	GPIBボードがリスナ
-dtas	GPIBボードがデバイストリガ状態
-dcas	GPIBボードがデバイスクリア状態

以下のモニターフラグは一つだけ選択してください。

`-off` applications monitorをオフにする。記録やトラッピングは行わない。

`-rec` applications monitorに、全てのGPIBドライバの呼び出しは記録するがトラッピングは行わないよう命令する。

`-dis` applications monitorに、全てのGPIBドライバの呼び出しを記録し、トラップ条件が存在するときはそれを表示するよう命令する。

マスクフラグ或いはモニターフラグのいずれかを省略した場合は、その時の構成は変化しないまま残ります。フラグを選択しないで`ibtrap`を呼び出すと、その時有効なフラグと、その時の状態が表示されます。これはapplications monitor の構成には何の影響も与えません。

`-dis`を選択して、PCのモニタをグラフィックスモードにするようなアプリケーションプログラムを使用した場合は、APPMONはスクリーン上に表示されません。但し、それは`-rec`を選択した場合と同じように事象の記録を続けています。PCのモニタがグラフィックスモードからキャラクタ(文字) モードに戻ると、APPMONは次の事象から表示されるようになります。

種々のマスクフラグとモニタフラグを選択することにより、多くの種類のトラップ条件を構成することが出来ます。次に幾つかの例を示します。

- `IBTRAP -cic -atn -dis` 全てのGPIBドライバの呼び出しを記録し、ATNがアサートされているかGPIBボードがCICである場合は何時でもapplications monitorを表示します。

- `IBTRAP -srq -rec` 全てのGPIBドライバの呼び出しが記録され、SRQがアサートされているとトラップが起こる設定です。トラップ条件が存在しているとapplications monitorは表示されません。

- `IBTRAP -dis` 全てのGPIBドライバの呼び出しを記録し、トラップ条件が存在するときは何時でもapplications monitorを表示します。トラップのマスクは変化しません。

- `IBTRAP -off` applications monitorを不可能化します。記録もトラップも行われません。

ユーザー各位がそれぞれのアプリケーションプロプログラムにおいて使用すべき正しいシンタクスについては、第5章を参照してください。

Applications Monitorのオプション

applications monitorが表示されていると、現在のGPIB呼び出しのパラメータを見ること、表示モードとトラップモードの変更、およびGPIBの履歴を見ることができます。applications monitorは現在のGPIB呼び出しに関して、次のような情報を表示します。

- `Device` デバイスの符号名。

- `Function` NI-488呼び出しまたは関数の簡略記号と説明。

- `ibsta` GPIBのステータス情報を含みます。

- `iberr` GPIBのエラー情報、または、エラーがない時は、valueパラメータのこの前の値を含んでいます。

- `ibcntl` 転送されたバイトの数を32ビットで表わしたものを含んでいます。

- `Address list` パラメータとしてアドレスリストを持つ関数について、アドレスリストの内容を表示します。

- `Buffer Value` パラメータとしてバッファを持つ関数について、バッファの内容を表示します。バッファの一つ一つのバイトは、インデックス、文字像、ASCII値で示されます。

- `Status` ibstaの一つ一つのビット状態を示します。全てのアサートされたビットはハイライトで示されます。

- `Error` iberrの状態を示します。エラーが起きた場合は、そのエラーの簡略記号がハイライトで示されます。

- `Other` GPIB呼び出しの種類によっては、以上のほかのパラメータも示されます。

注) 表示された数字は、そのあとに16進数(hex)であることを示すhが附いていない限り10進数です。但し、ibstaとバッファ値を示すASCIIコードの2つの場合は例外として何時もhexで表わされます。また、applications monitorは`ibfind`と`ibtrap`の呼び出しは記録することが出来ません。

メインコマンド

applications monitorのメインスクリーンが表示されていると、次のコマンドキーの使用が可能です。

- <F1> アプリケーションプログラムを継続して実行します

- <F2> セションの要約を表示します

- <F3> アプリケーションプログラムの実行を中止し、DOSに戻ります

- <F4> ファイルに履歴を書き込みます

- <F5> トラップマスクを構成します

- <F6> applications monitorモードを構成します
- <F7> applications monitorの動作のスクリーン表示/消去の交互変換を行います
- <F8> GPIB履歴バッファをクリアします
- <F10> コマンドキーの種類を表示します
- <Cursor Up> バッファを上に1文字だけスクロールします
- <Cursor Down> バッファを下に1文字だけスクロールします
- <Page Up> バッファを上に1ページだけスクロールします
- <Page Down> バッファを下に1ページだけスクロールします
- <Home> バッファの始めまでスクロールします
- <End> バッファの終までスクロールします
- <Cursor Right> アドレスリストを右に2文字だけスクロールします
- <Cursor Left> アドレスリストの始めまでスクロールしま。

GPIB履歴スクリーン

GPIBの履歴はF2を押せば表示されます。表示部分は次に示すキー使用して変えることが出来ます。

- <Cursor Up> 履歴表示を1行だけ上にスクロールする
- <Cursor Down> 履歴表示を1行だけ下にスクロールする
- <Cursor Right> 履歴表示を右に2文字分だけスクロールする
- <Cursor Left> 履歴表示を左に2文字分だけスクロールする

- <Page Up> 履歴表示を上に1ページだけスクロールする
- <Page Down> 履歴表示を下に1ページだけスクロールする
- <Home> 履歴表示の始めまでスクロールする
- <End> 履歴表示の終までスクロールする
- <Escape>か<F2> GPIB履歴表示を終了し、applications monitorのメインスクリーンに戻る

トラップマスクの構成手順

現在使用中のトラップマスクの構成を変更するには先ずF5キーを押します。ポップアップメニューが現われ、各ステータスビットとその現在の状態（ONかOFF）が示されます。矢印キーを利用して所望のビットをハイライトの状態にした後<Spacebar>を押しますと状態がONからOFF或いはOFFからONに変わります。次に<Enter>を押すことによって変更を記録します。この場合、<Escape>を押しますと手順がキャンセルされ、マスクは変更されない元の状態で残ります。全てのビットを選択すると（つまり全部をONにすると）、あらゆる呼び出しがトラップされるようになります。反対に全てのビットをOFFにすると、トラップは全然起こらなくなります。

Applications Monitorのモード構成の手順

現在使用中'のapplications monitorのモードを変更するには先ずF6キーを押します。これによって現われたポップアップメニューには現在使用中のモードが示されています。矢印キーを操作して新しいモードを選択し、<Enter>を押すと変更が記録されます。この場合、<Escape>を押すと変更手順はキャンセルされ、現在のモードが未変更のまま残ります。

Applications Monitorの動作のスクリーン表示／消去

applications monitorの動作を表示するスクリーンを消去したりもとのスクリーン表示を回復したりする交互変換にはF7キーを押します。applications monitorのスクリーンが表示されているときにF7を押しますと、applications monitor中で実行中のアプリケーションプログラムの出力をスクリーン上に見ることが出来るようになります。もう一度F7を押しますと、applications monitorのスクリーンが再び現われます。

直接DOSに戻る

F3キーを押しますと、アプリケーションプログラムから直接DOSに戻ることが出来ます。この場合アプリケーションプログラムは終了され、DOSのプロンプトが現われますので、そこから仕事を継続することが出来ます。

付録A
マルチライン・インターフェース・メッセージ

本付録にはマルチラインインターフェースメッセージの参照表が含まれています。参照表はインターフェース機能、相当する簡略記号とそれらのメッセージの内容を示します。これらマルチライン・インターフェース・メッセージはATN線が真である時に送受されます。

これらのメッセージの詳細な説明については、ANSI/IEEE Std. 488.1-1987 [アメリカ国家規格協会／アメリカ電気電子通信学会488規格1987版]、*IEEE Standard Digital Interface for Programmable Instrumentation* [「プログラム可能計装についてのIEEE規格ディジタルインターフェース」]を御覧ください。

マルチライン・インターフェース・メッセージ

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US		3F	077	63	?	UNL

メッセージの内容

DCL　デバイスクリア
GET　グループトリガ
GTL　ローカルモードに移る
LLO　ローカルロックアウト
MLA　マイリスンアドレス
MSA　マイ2次アドレス
MTA　マイトークアドレス
PPC　パラレルポーリング用に構成
PPD　パラレルポーリングを不可能化

マルチライン・インターフェース・メッセージ

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

メッセージの内容

PPE	パラレルポーリンを可能化	SPE	シリアルポーリングを可能化
PPU	パラレルポーリング用構成を解除	TCT	コントロールを委譲する
		UNL	リスナへのアドレシングを停止
SDC	選択したデバイスをクリア		
SPD	シリアルポーリングを不可能化	UNT	トーカへのアドレシングを停止

付録B
起こりやすいエラーと解決法

この付録では起こりやすいエラーとその解決法を示しています。これらのエラーは関数から返されてiberrによって示されるエラーコードの順になっています。また、この付録ではエラーコードとして返ってこないエラー状況も説明されています。ここで説明されるエラーだけでなく、起こりうるあらゆるエラーについて知りたい場合には、第3章の*Error Variable-iberr* [エラー変数—iberr]の項を参照してください。

EDVR(0)

エラーの内容　DOSエラー(IBCNTがDOSのエラーコード値を示しています。)

解決法

- CONFIG.SYSが該当するラインを含んでいるか否かをチェックします。

  ```
  device=y:\dir\gpib.com
  ```

  ここで*y*は始動ドライブのことであり、*dir*はINSTALLによってコピーしたNI-488.2のソフトウエアファイルが収められたディレクトリの名です。例えば、AT-GPIBを御使用の場合は、*dir*は多分AT-GPIBです。

  始動ドライブにおいて、TYPE \CONFIG.SYSを入力し、ラインがそこに存在していることを確認します。また、GPIB.COMが上記のディレクトリ中に存在することも確認します。

- ソフトウエアをインストールしたあとでシステムをリスタートします。

- IBCONFを利用して、GPIBボードとデバイスの名が御自分の思っていたとおりに設定されていることを確認します。

ECIC(1)

エラーの内容　GPIBボードがコントローラインチャージ(CIC)でないとこの関数は使用出来ないと言う場合。

解決法

- IBCONFを走らせて使用中のボードがシステムコントローラとして構成されていることを確認します。
- ボード関数を実行する場合であれば、ibsicを呼び出してボードがCICになるようにする。この処置の後でGPIBボードがコントローラインチャージであることを要求する関数の使用が可能になります。
- コントロールがibpctの使用により他のデバイスに引き渡されていた場合は、ibwaitを使用してコントロールが戻されるのを待ちます。

ENOL(2)

エラーの内容　関数がリスナを見つけることが出来ない。

解決法

- そのデバイスにスイッチが入っていること、それからGPIBと接続した全デバイスの少なくとも3分の2にスイッチが入っていることを確認します。
- 接続ケーブルを点検してデバイスが接続され、コネクタが正しく挿入固定されていることを確かめます。

- デバイスのスイッチや操作盤を調べ、そのGPIBアドレスが思った通りであることを確認します。また、デバイスが拡張アドレスを使用していて1次アドレスのほかに2次アドレスも必要としていないかをチェックします。(デバイスの中には種々の内部機能を行えるように複数の2次アドレスを持ったものがあります）。

- デバイス書込関数の場合は、IBCONFを走らせてデバイスのアドレスが、2次アドレスも含んで、正しいか否かをチェックしてください。もしその結果必要な変更を行った場合はシステムのリスタートを忘れないように。次に、ibfindを使用して、以下の手順によりアドレスが正しいことを確認してください。まずibfindを使用して書込をしたいデバイスを開き、ibpadとibsadを呼び出して正しいと思うアドレス値を引き渡します。2次アドレシングを行わない場合は、0の値をibsad関数に引き渡します。これらの呼び出しでエラーが返されていなければ、IBICは以前のアドレス値を返します。以前のアドレス値は、デバイスの構成が正しくなされていれば、新しいアドレス値と同じであるはずです。

- ボード書込関数の場合は、書込関数を呼び出す前にibcmd関数を使用してデバイスが正しくアドレスされていることを確認します。ibcmd呼び出しで使用されるリスンアドレス(及び必要な時は2次アドレス）の下位5ビットがデバイスのGPIBアドレスとマッチしていること、また、リスンアドレスは16進数の20から3E(すなわち10進数の32から62)の範囲内であり、2次アドレスは16進数の60から7E（すなわち10進数の92から126)の範囲内であることを確認します。

EADR(3)

エラーの内容　GPIBボード（GPIB0かGPIB1）正しくアドレスされていない。

解決法

- ibwrtやibrdを実行する前にibcmdを使用して適当なトークアドレスやリスンアドレスを送ります。
- シャドウハンドシェイクを利用してibgtsを呼び出す時は、ibcacを使ってGPIBのATN線がアサートされるようにします。

EARG(4)

エラーの内容　関数呼び出しに無効な引き数が存在。

解決法

- IBICからエラーを表示された場合。
  - 本マニュアルの第6章を参照してシンタクスを確かめます。
  - IBCONFにおけるボードのGPIBアドレスがデバイスのアドレスとかち合っていないことを確かめます。
- アプリケーションプログラムを実行中にエラーが示された場合。
  - 本マニュアルの第4章と第5章を参照してシンタクスを確かめます。
  - IBCONFにおけるボードのGPIBアドレスがデバイスのアドレスとかち合っていないことを確かめます。

ESAC(5)

エラーの内容　GPIBボードがシステムコントローラである必要があるのにそうなっていない場合。

解決法

- IBCONFを走らせてボード(GPIB0かGPIB1）がシステムコントローラとして構成されていることを確かめます。
- ibrsc呼び出しを１の値で行ってシステムコントロールを要求します。

EABO(6)

エラーの内容　入出力動作が中止された場合。

解決法

- デバイスにスイッチが入っていることを確認します。
- ケーブルが正しく接続されていることを確かめます。
- ibrdからエラーが返された場合。
  - デバイスのなかには何を送るかをあらかじめ告げられていないとデータの送出をしないものがあります。数種類のデータを送り出すことが出来るデバイスでこういうことが起きます。この様な場合には、まずibwrtによってデバイスをセットし、つぎにibrdにより情報を受け取るようにします。
  - IBCONFのEOSかEOIのデフォルト設定を変えていないと、バッファが一杯になったりEOIがセットされると読み取り動作の終了が起こります。もしデバイスがEOIでなく復帰文字の様なEOS終了文字を送る場合には、IBCONFを使ってデバイスの特性を変えてください。

ENEB(7)

エラーの内容　GPIBボードが存在しない。

解決法

- IBCONFを走らせてボードの入出力基底アドレスがハードウエアのアドレススイッチの設定とマッチしていることを確認します。IBCONFは選択された基底アドレス用の正しいスイッチの設定を表示します。

EDMA (8)

エラーの内容　Windows 3 のみ。仮想DMAデバイスエラー。

解決法

- Windowsのバージョン3.0をDMAを可能化して走らせている方々は、置き換えDMAデバイスの使用法の詳細について*Using Your NI-488.2 Software with Microsoft Windows* を参照してください。Windowsのバージョン3.1か3.0を`nivdmad.386`と共に使用している方々の場合は、Windows をリスタートしてDMAデバイスを既知の状態に戻してください。

EOIP(10)

エラーの内容　非同期入出力動作が進行中には関数使用は許されない。

解決法

- 非同期関数を使用中に次の呼び出しをしたい時は、まず`ibwait`を呼び出してCMPL(或いはTIMO CMPL)を待つようにさせます。非同期入出力関数の詳細な説明については第5章を参照してください。

ECAP(11)

エラーの内容　オペレーションを行う資格なし。

解決法

- IBCONFを走らせてその呼び出しの資格が可能化されていることを確かめます(例えば、ibsre関数を実行するには、インターフェースボードはシステムコントローラとして構成されている必要があります)。デバイスとボードの両方の資格をチェックしてください。そのあとで変更を行った場合は、IBCONFを終了してからリスタートを行うことを忘れないように。

EFSO(12)

エラーの内容　ファイルシステムのエラー

解決法

- ディスクのファイルをチエックして、ファイル名が正しく指示されているか、また、ファイルが存在しているかを確かめます。
- ディスク上のスペースが不足していると解った時は、幾つかのファイルを削除してください。
- ファイルのなかでIBCONF中のデバイス名と同じ名を持つものがありましたら、ファイル或いはデバイスの何れかを改名してください。

EBUS(14)

エラーの内容　コマンドバイトの転送エラー。

解決法

- コマンドを受けるのが異常に遅いデバイスを見出してそのデバイスについて問題を解決します。
- コマンドを送る時間が短すぎる場合は、IBCONF或いはibtmoによって制限時間を延長します。

ESTB(15)

エラーの内容　シリアルポーリングのステータスバイトが失われる。

解決法

- ibrspをより頻繁に呼び出してステータスバイトを読む。
- ESTBを無視する。

ESRQ(16)

エラーの内容　SRQがONの位置でつかえて動かなくなっている。

解決法

- 全てのデバイスが見出されるまでESRQは無視する。ESRQはSRQをアサートしているデバイスがibfindで開かれなかったために起こる。自動シリアルポーリング関数は開かれたデバイスのみポーリングするからです。
- ibfindを使ってGPIB上のデバイスででSRQをアサートできるものは全て開いてあるかをチェックします。そして、アクセスされていないデバイスはバスから除くようにします。

- IBICを使用してデバイスを一回に一つずつ附け、ポールされた後でそれがSRQのアサートをやめるか否かを見ます。

- 接続ケーブルを検査してデバイスが接続されており、コネクタが正しく挿入固定されていることを確かめます。

ETAB(20)

エラーの内容　表の問題。

解決法

- FindLstnルーチンにおいてはこれは警告であってエラー状態ではありません。このメッセージは、無視するか、或いはバッファの大きさを増加するかします。

- FindRQSルーチンの場合、このエラーは指示されたデバイスの何れもがサービスを要求していないことを示しています。デバイスリストをチェックして全てのオンラインデバイスのアドレスが含まれていることを確かめます。また、FindRQSは通常SRQ線がアサートされている時だけ呼び出すことになっています。SRQの状態を知るにはTestSRQかWaitSRQを使用します。

- ibevent関数の場合にETABが返されると、それは受信されたデバイスクリアやGET (group execute trigger) メッセージが多すぎたということを意味します。この場合最近送られたメッセージは失われてしまいます。この様な事態を回避するためには、ibeventをより頻繁に呼び出して待ち行列が一杯になるのを防ぎます。

他のエラー状態

次に列挙したのはGPIBハードウエアとソフトウエアを使用中に起こりうる一般的なエラーの例です。

エラーの内容　GPIBのユーティリティプログラムのibdiag、ibtest、或いはIBICを実行しようとするとDOSからBad command or Filenameのエラーコードが返ってくる。また、INSTALLを走らせた後でもディストリビューションファイルが始動ディスクに現われない。

解決法

- INSTALLは始動ディスクのあらかじめ定義されたサブディレクトリにファイルをコピーします。このディレクトリはデフォルトでは次の通りです。

  y:*dir*

  ここで*y*は、アルファベットの1字で表わされる始動ディスクの名です。*dir*はGPIBボードの名（例えば、c:\AT-GPIB)です。

  他のプログラムを実行するときは、DOSコマンドのcdを使ってあらかじめ定義されたディレクトリに変わります。

エラーの内容　呼び出した関数が返ってこない。また、プログラムが不動化してしまったようである。

解決法

- IBCONFを走らせてボードの割込線とDMAチャネルがハードウエアの設定とマッチしているかを確かめます。次に、リスタートを行ってからibtestを走らせます。更に、DMAチャネルと割込線がコンピュータの他のデバイスとかち合っていないかチェックします。

- タイムリミットをチェックしてそれが0になっていないかを確かめます。タイムリミット0は時間に制限がないことを意味します。手順としてはIBCONFを走らせてデバイスとボードのタイムリミット値を見ます。タイムリミットのチェックの代りに、IBICからibtmoを呼び出し、それに正しいと思われるタイムアウト引き数値を引き渡すことも出来ます(デフォルトは13で、これは10秒に当たります）。ibtmo呼び出しがエラーなく行われると、その前の値が返されてきます。この返された値は前に正しいタイムアウト値と思って引き渡した値と等しいはずです。

- 起こらない可能性のある事象をibwaitを使って待つ場合は、TIMOをセットすることを忘れないでください。例えば、あるデバイスからのRQSを待つときにTIMOをセットすることを忘れると、何らかの理由でデバイスがSRQをアサートしなかったときに、ibwaitは何も返して来ません。

エラーの内容　コンピュータが動かなくなる。

解決法

- IBCONFを走らせてボードの割込線とDMAチャネルがハードウエアの設定とマッチしているかを確かめます。更に、DMAチャネルと割込線がコンピュータの他のデバイスとかち合っていないかチェックします。

- ハードウエアとソフトウエアを、DMA及び／或いは割込を行わないように構成します。IBM PC互換機といっても100パーセント互換でないために起こった可能性もあります。

- デバイス名を調べて、それらがファイル或いはディレクトリの名と同じものがないことを確かめます。この場合、ファイルやディレクトリの名のサフィックス（拡張部）は考慮に入れません。

エラーの内容　ibdiagがDMA或いは割込に問題ありと報告している。

解決法

- 外のデバイスとかち合っている可能性がありますので、ハードウエアとソフトウエアを別のDMAチャネル及び／または割込線用に再構成します。
- 同じライン上にあるもう一つのデバイスが本当にラインを共用するための割込資格を有するか否かを調べます。多くのデバイスはこの様な資格を持っていません。
- それでもまだ割込やDMAの問題が解決できない場合は、ハードウエアとソフトウエアをDMAなし、或いは割込なしのオペレーション用に構成します。もしかすると、御使用中のIBM PC互換機が100パーセント互換でないのかもしれません。

エラーの内容　IBICがibfindについてエラーを報告するか ibfindが負のユニット記述子を返す。

解決法

- EDVR、ENEB、或いはEARGエラーコードの解決法を試みる。
- IBCONFを走らせてデバイス或いはボードの名が間違い無いかをチェックする。

エラーの内容　プログラムされたGPIBデバイスが命令を受け入れたようなのにそれを正しく実行しない。

解決法

- 書込関数とともに送られた命令が正しい区切り文字とメッセージ終止文字を含んでいるかをチェックします。例えば、ある種のデバイスは復帰文字及び改行文字を受け取ってからでないと命令を実行しません。また、多数の命令メッセージを分離するために特殊文字を必要とするデバイスもあります。そのデバイ

スのマニュアルが不完全であったり曖昧であったりする場合もありますので、幾つかこれはと思われる組み合わせを試して見てください。

- プログラムを実行中、NI-488関数或いはNI-488.2ルーチンが一つ終わるごとにエラーをチェックしてください。もし先に行った呼び出しが失敗したのにIBSTAでそれをチェックしないでそのままにしておくと、その後の関数は正しく実行されず、間違いにつながるステータスデータを返して来ることになります。

- 使用中のボードとプログラム中のデバイスが同じGPIBアドレスにないかどうかチェックします。デバイス関数を使用中にこのチェックを行うにはIBCONFを走らせます。

付録C
シリアル／パラレルポートの方向変換—GPIBデバイスへ

この付録においてはMS-DOSドライバ用のNI-488.2ソフトウエアを通じて標準のDOSプリンタ・コマンド（LPRINT, PRINT等)を使用してシリアル或いはパラレルポートからのデータをGPIB プリンタ、プロッタ、その他のデバイスに向け換える方法が説明されています。

データの方向を変換する方法

NI-488.2ソフトウエアを第２章に示す手順にしたがってインストールする際、構成プログラムIBCONFを使用してGPIBプリンターとプロッターのデバイス名をDOSプリンターコマンドに合わせて変えます。この処置をとることにより、DOSは標準のプリンターデバイスを無視し、代わりにNI-488.2ドライバを使ってGPIBデバイスに書込を行うようになります。

注）方向を変えられたデータは、DOSの割り込みをインターセプトして一方向性で送られます。ソフトウエアパッケージで、GPIBデバイスに対して２方向性通信を行うようにデザインされたもの、或いはシリアル或いはパラレルポートにデータを出力する時にDOS割り込みを発生しないものはこの種のデータ送出方向変換を行いません。

次に示すのはパラレルポート方向変換 (LPT1)の手続きと使用例です。これらは他のポート(COM1,LPT2,LPT3)への方向変換にも応用できます。

ソフトウエアをインストールするには次の手順を踏んでください。

1. 第２章に説明されている手順にしたがってINSTALLを実行します。

2. IBCONFを走らせてそこに示される手順を全て実行します。

3. IBCONF中で。

   a. dev1という名をLPT1に変えます。

   b. LPT1の1次アドレスを御使用になるプリンターのGPIBアドレスに変えます。

4. IBCONFを終了させます。

5. システムをリスタートさせます。

6. ソフトウエアが正しくインストールされたことを確認するためibtestを走らせます。

例

システム

```
PRINT FILE
COPY FILE LPT1
```

BASIC

```
10  LPRINT "hello"

10  OPEN "LPT1" FOR OUTPUT AS #1
20  PRINT #1, "it works!"
```

付録D
GPIBのオペレーション

この付録ではGPIBのオペレーションを行う際に必要な基本的な事項が説明されています。またこの付録ではGPIBの物理的、電気的特性と構成に必要な条件が述べられています。

メッセージの型

GPIBはデバイス依存性のメッセージとインターフェースメッセージを運びます。

- デバイス依存性メッセージはしばしばデータ或いはデータメッセージとも呼ばれ、そのデバイスについての情報（例えばプログラミングの命令、測定結果、機械のステータス、及びデータのファイル）を含んでいます。

- インターフェースメッセージはバス自体を管理します。この種のメッセージは通常コマンドとかコマンドメッセージと呼ばれます。インターフェースメッセージは、バスの初期化、デバイスに対するアドレシング或いはアドレシングの解除、デバイスのリモート或いはローカルプログラミングモードの設定等を行います。

ここで使用されたコマンドという語をデバイス用命令(それらもまたコマンドと言われることがあります）と混同しないようにしてください。ここで言うコマンド、すなわちデバイス特異性の命令は実際にはデータメッセージなのです。

トーカ、リスナ、及びコントローラ

トーカはデータメッセージを一つ或いは複数のリスナに送ります。コントローラはコマンドを全てのデバイスに送りGPIB上の情報の流れを管理します。

デバイスはリスナ、トーカ及び／或いはコントローラになることが出来ます。例えば、ディジタル電圧計はトーカであると共にリスナであることがあります。

GPIBは通常のコンピュータのバスと似ていますが、コンピュータの場合はプリント回路板がバックプレーンバスを介して相互に接続されているのに対して、GPIBでは、各々独立のデバイスがケーブルバスを介して相互に接続されています。

GPIBコントローラの役割は、コンピュータにおけるCPUの役割に似ていますが、例えとしてはある都市における電話回線の切り替えセンターを考えたほうが解りやすいでしょう。

切り替えセンター（コントローラ）では通信の回線網(GPIB)を監視（モニター）します。センター（コントローラ）では、ある電話加入者（デバイス）が電話をかける（データメッセージを送る）時に、電話をかける側（トーカ）とかけられる側（リスナ）との間の配線を接続します。

コントローラがまずトーカとリスナにアドレスした後で始めてトーカがそのメッセージをリスナに送ることが出来るようになります。メッセージの転送が終わると、コントローラはリスナへのアドレシングを解除することがあり得ます。

バスの構成によってはコントローラを必要としないものがあります。例えば、何時でもトーカの役しかしないデバイスがあり、「トークオンリーデバイス」と呼ばれています。それに対して、一つ或いは復数の「リスンオンリーデバイス」があります。

コントローラは活状態、或いはアドレスされた状態のトーカ或いはリスナを変えるときに必要になります。通常コンピュータがコントローラの機能をはたします。

ナショナルインスツルメンツのGPIBインターフェースボードとそのソフトウエアを備えたコンピュータは次の３つの役割の全てをはたします。

- コントローラとしてGPIBを管理する
- トーカとしてデータを送る
- リスナとしてデータを受け取る

コントローラ・イン・チャージとシステムコントローラ

GPIB上に複数のコントローラの存在することは可能ですが、一つの時点における活コントローラすなわちコントローラ・イン・チャージ(CIC)は唯一つだけです。活コントロールの資格は現在のCICから休コントローラに引き渡されることが出来ます。コントローラの中で自分の力でCICになることが出来るのはシステムコントローラだけです。通常GPIBインターフェースがシステムコントローラです。

GPIB信号と信号線

インターフェースシステムは16の信号線と８つの接地リード或いはシールド・ドレイン線からなっています。

16の信号線は次の３つのグループに分けられます。

- ８つのデータ線
- ３つのハンドシェイク線
- ５つのインターフェース管理線

データ線

DI01からDI08までの８つのデータ線はデータとコマンドメッセージを共に搬送します。全てのコマンドとほとんどのデータでは７ビットのASCIIかISOコードのセットを使用します。これらの中では８番目のビットのDI08は使用されないかパリティービットとして使用されます。

ハンドシェイク線

これら３つの線はデバイス間のメッセージバイトの転送を非同期的に制御します。このプロセスは３線インタロックド・ハンドシェイクと呼ばれ、データ線上のメッセージバイトの送受が転送エラー無しに行われることを保証しています。

NRFD (not ready for data)線[データ受信準備未了線]

NRFDはデバイスがメッセージバイトを受信する準備が完了したか否かを示す線です。デバイスがコマンドを受信する場合には全てのデバイスがこの線をドライブします。また、リスナがデータメッセージを受信する時はリスナがこの線をドライブします。

NDAC (not data accepted)線[データ受け入れ未了線]

NDACはデバイスがメッセージバイトを受け入れたか否かを示します。デバイスがコマンドを受信する場合には全てのデバイスがこの線をドライブします。また、リスナがメッセージを受信する時はリスナがこの線をドライブします。

DAV (data valid)線[データ有効線]

DAVはデータ線上の信号がその時点において安定（有効）であり、デバイスによる受信が安全に行われ得るか否かを示します。コマンドを送るときのコントローラがこの線をドライブしますし、トーカがメッセージを送るときにはトーカがこの線をドライブします。

インターフェース管理線

以下に示す5つの線がインターフェース上のインフォーメーションの流れを管理するために使用されています。

ATN(attention)線[アテンション線]

コントローラはコマンドを送るためにデータ線を使用する時はATNを「真に」ドライブします。また、コントローラはトーカがデータメッセージを送ることを許す場合にはATNを「偽に」ドライブします。

IFC(interface clear)線[インターフェースクリア線]

システムコントローラはIFC線をドライブしてバスを初期化することにより自身がCICになります。

REN (remote enable)線 [リモートモード可能化線]

REN線はデバイスをリモートかローカルプログラムモードにするために使用されます。システムコントローラがREN線をドライブします。

SRQ (service request)線 [サービス要求線]

どのデバイスもSRQ線をドライブしてコントローラから非同期的にサービスを要求することが出来ます。

EOI (end or identify) 線 [終止或いは明示線]

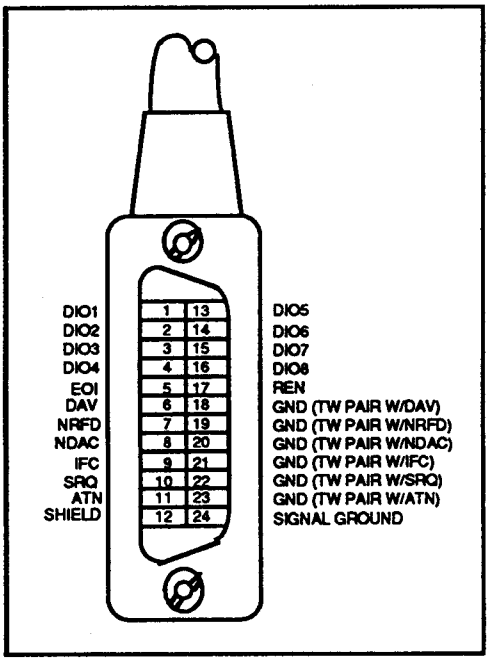
EOI線には2つの目的があります。トーカはEOI線を使ってメッセージストリングの終を示します。コントローラはEOIを通して、デバイスがパラレルポーリングでの応答を明示するように命令します。

物理的・電気的特性

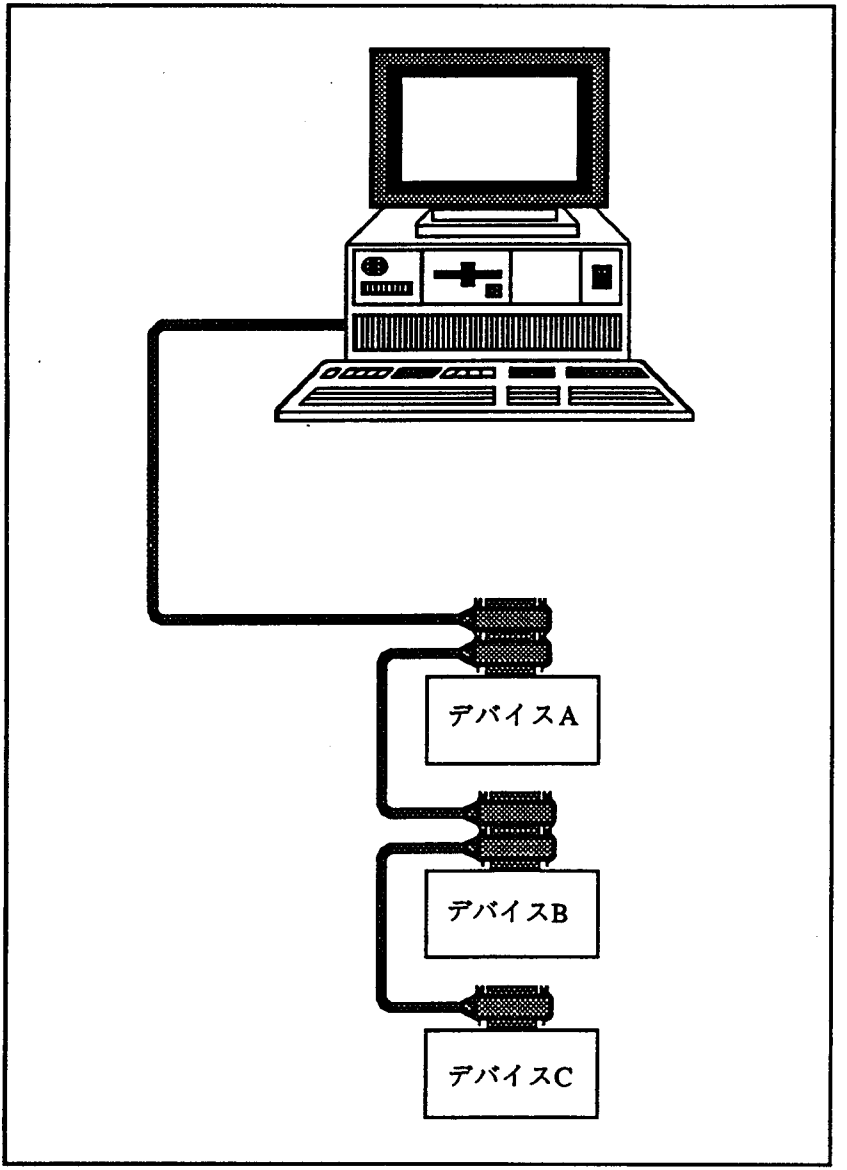
デバイスは通常ケーブルアセンブリで接続されています。ケーブルアセンブリは24の導電線と両端にあるプラグとレセプタクル式コネクタからなっています。このデザインでは、直列接続、星型接続、及び両者の混合型接続が可能になります。図D－１、D－２、D－３を御覧ください。

標準型コネクタとしてはAmphenolシリーズ57型、Cinch マイクロリボン、またはAMP Champ型コネクタが使用されます。特殊な相互接続の応用を図る時には非標準型ケーブル及び／またはコネクタが使用されます。

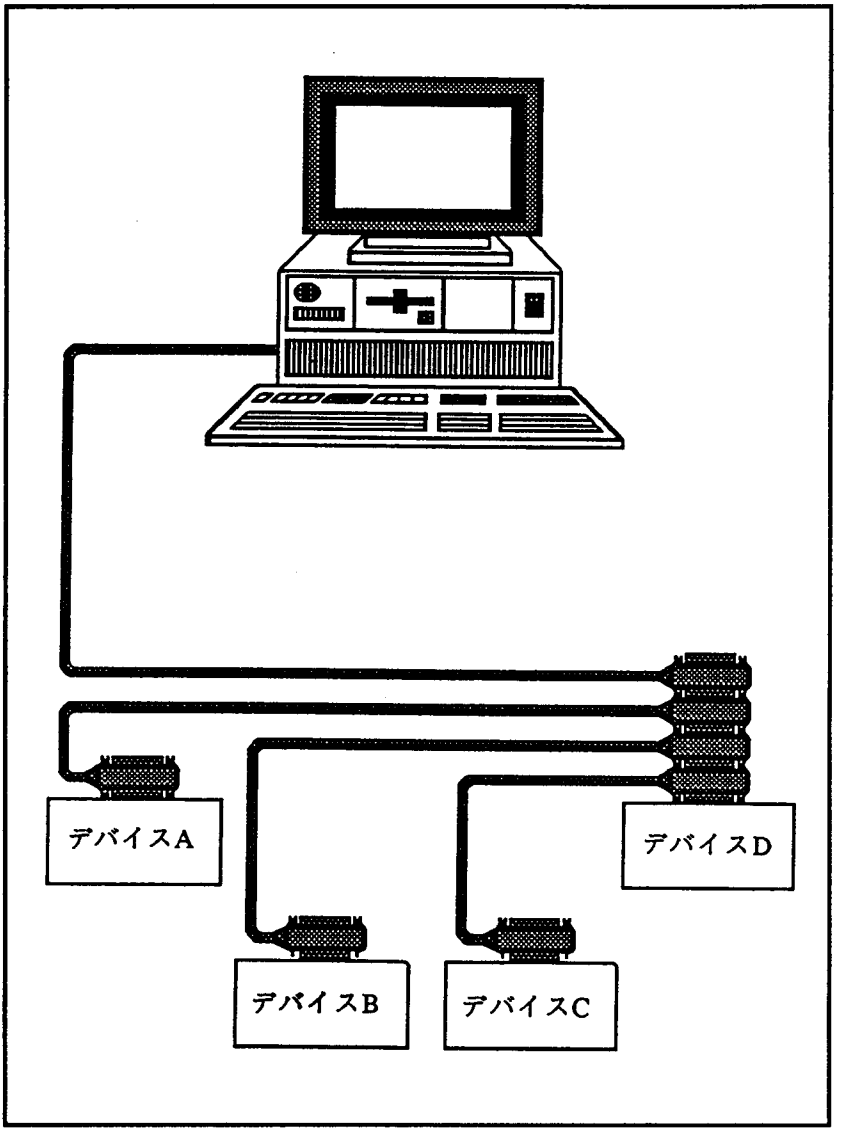
GPIBは標準のTTL論理レベルの負論理を使用します。例えば、DAVが「真」であるとTTLの低レベル(≦0.8 V)であり、DAVが「偽」であるとTTL高レベル(≧2.0 V)となります。



図D－１ GPIBコネクタと信号割り当て



図D－２ 直列接続



図D－3 星型接続

接続構成上の要項

GPIBは高速データ転送が出来る用に設計されていますが、その能力を十分に発揮するためには、デバイス間の距離、及びバスに接続するデバイスの数を一定の範囲内にとどめる必要があります。

典型的な範囲制限の例を以下に示します。

- ２つのデバイスの間は如何なる場合でも４メートルを超えないこと。また、バス全体を通じて平均間隔が２メートルを超えないこと。
- ケーブルの全長は20メートルを超えないこと。
- 一つのバスに接続されたデバイスの数は、少なくともその３分の２にスイッチが入っているものとして、15を超えないこと。

以上に述べた範囲を超えた使用が必要な場合は、ナショナルインスツルメンツから入手可能のバス拡張機を使用してください。

関連文書

本付録の内容につき、より詳細な情報が必要な場合は、*IEEE Standard Digital Interface for Programmable Instrumentation* [「プログラム可能計装についてのIEEE規格ディジタルインターフェース」], ANSI/IEEE 488.1-1987 [アメリカ国家規格協会／アメリカ電気電子通信学会488.1規格1987版]、]を御覧ください。

付録E
使用者各位との連絡

私どもは当社製品を御使用下さるユーザー各位との連絡を保ちたいと望んでおります。また、皆様が当社の製品を使用して開発したアプリケーションについても多大な興味を持っておりますので、ぜひそれらについてお聞かせ願いたいと存じます。皆様から気軽に連絡をしていただくため、必要事項を書き込めばよいようになっているフォーム（書式)をこの付録内に加えて置きますので御利用ください。あらかじめ空白部に必要事項を御記入の上当社にご連絡下されば私どもとのコミュニケーションをスムースに進行させることが出来ます。この付録中のフォームを原本として何枚かのコピーをとって保存され、必要がある毎に書き込んで当社に連絡くだされればよいかと存じます

FAXによるテクニカルサポート

技術的な問題がある場合は、FAXによりサポートを受けることが可能です。下記の番号にFAXでご連絡ください。

０３（３７８８）１９２３

FAXによりテクニカルサポートを受けるには、テクニカルサポートFAXフォームとハードウエア／ソフトウエア構成フォームにあらかじめ必要事項を御記入ください。

電話によるテクニカルサポート

電話によるテクニカルサポートもいたしております。午前9時から午後5時までの間に下記の番号にお電話ください。

03（3788）1921

私どもが手早くテクニカルサービスを行えるようハードウエア／ソフトウエア構成フォームにあらかじめ記入したものをお手元に御準備ください。エラーメッセージがありましたら記録を忘れないように。また、コンピュータはすぐ御使用になれるようにしてからお電話下さる様お願い申し上げます。

マニュアル等についての御意見

当社のマニュアル等についての御意見をお寄せ下さる際は、当付録中の「マニュアル等についての御意見」フォームを御使用になり、同フォームの終に示す宛先にご送付願います。

テクニカルサポートFAXフォーム

FAXによるテクニカルサポートの御用は24時間中いつでも承っています。FAXの番号は０３（３７８８）１９２３です。十分なサポートを受けるためには出来るだけ詳しい情報をお書きください。

ふりがな
氏名 _____

会社名 _____

住所 _____

_____

FAX番号 ___(____)_____ 電話番号___(___)_____

コンピュータメーカー名 _____

モデル(型)_____ プロセッサ_____

オペレーティングシステム _____

CPUクロック _____ MHz RAM _____ Mバイト

表示装置(ディスプレイ)アダプタ _____

マウス _____有 _____ 無

このコンピュータにインストールされている他のアダプタ _____

_____

ハードディスク容量____M ハードディスクメーカー_____

使用計測器 _____

ナショナルインスツルメンツ製品(ハードウエア)モデル _____

_____

Rev.番号 _____

構成 _____

(次ページに続く)

National Instruments software product _____

Rev. 番号_____

構成 _____

問題は _____

_____

_____

_____

_____

_____

エラーメッセージは _____

_____

_____

_____

_____

_____

次の手順を踏んだ場合にこの問題が起こります _____

_____

_____

_____

_____

_____

_____

[ハードウエア／ソフトウエア構成フォーム中の情報もここに書いてください。用紙が足りなければ別紙に書き足してください]

NI-488.2 ハードウエアとソフトウエアの構成フォーム

もしハードウエアとソフトウエアをインストールし構成する手続きをすべて完了したにもかかわらずこれらのハードウエアとソフトウエアのいずれかかあるいは両方がうまく動作しない場合にはこの付録中のフォームを利用してご連絡ください。ご連絡の際は、フォームに必要事項をすべて書込んだ後にナショナルインスツルメンツのテクニカルサポート係まで電話してください。

このフォームには、サポートの御相談に預かるナショナルインスツルメンツのアプリケーションズエンジニアが問題解決のため必要とする事項が含まれています。お電話をくださる前にこのフォームに必要事項を書き込んでおかれますとサポートを受ける際の時間の節約になります。

フォーム記入に際しては、各事項の右横の線上に必要な情報を簡潔正確に書き込んでください。これにより、当社のシステムズエンジニアが御質問に正確・迅速に要領を得たお答えをすることが可能となります。

ナショナルインスツルメンツ製品の情報

- NI-488.2ソフトウエアのレビジョン番号(ディスク上に示されています) _____
- アプリケーションプログラミング言語(BASICA, QuickBASIC, C, Pascal, 等) _____
- プログラミング言語インターフェースのレビジョン番号 _____

_____

- インストールされたナショナルインスツルメンツの各GPIBボードの形式と各々のハードウエア設定。

ボード形式	インタラプトレベル	DMAチャネル	入出力基底アドレス
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

他社製品に関する情報

- コンピュータメーカー及びモデル名_____
- マイクロプロセッサ_____
- クロック周波数_____
- インストールされたモニターカードのタイプ_____
- DOSのバージョン_____
- プログラミング言語及びそのバージョン_____
- コンピュータシステム中にある他のボード_____
- 他のボードの入出力基底アドレス_____
- 他のボードのDMAチャネル_____
- 他のボードのインタラプトレベル_____

マニュアルについての御意見

ナショナルインスツルメンツでは製品のマニュアル類についてユーザーの皆様の御意見をお伺いし、今後の当社製品の品質向上に役立てたいと存じております。ぜひ御意見をお寄せ下さい。

タイトル　NI-488.2ソフトウエア参考マニュアルーMS-DOS版ー

この版の出版年月　1992年12月 版

部品番号　370914A-01

マニュアルの内容が行き届いているか、解りやすいか、構成はどうか等について皆様の御意見をお知らせください。

(次ページに続く)

マニュアルに誤りがあることに気付かれた場合はページ数を書いた上、誤りの内容を記してください。

ご協力ありがとうございました。

お名前＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

役職名＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

会社名＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住所＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

電話＿＿＿＿（＿＿＿）＿＿＿＿＿＿

以下の宛先に御送付願います。

〒142 東京都品川区戸越5-14-23池田ビル3階
日本ナショナルインスツルメンツ株式会社
営業技術部
TEL 03(3788)1921　FAX 03(3788)1923

用語解説

A

applications monitor
メモリレジデントプログラムの一つ。アプリケーションプログラムGPIB関数呼び出しのシーケンスのデバッグに使用すると便利です。

appmon.exe
ディストリビューションディスクの中にあり、applications monitorを含んでいるファイル。

ATN (attention)
GPIB線の一つで、この線によりコマンドとデータメッセージ間の区別がなされます。例えば、ATNがアサートされている時はGPIBのDIO線で運ばれるバイトはコマンドバイトです。

C

CIC
「コントローラ」の項をご覧ください。

config.sys
DOSファイルで、DOSがスタートする時にロード可能なデバイスドライバの名を含んでいるもの。

D

DAV (あるいはデータ有効線)
3本のGPIBハンドシェイク線の一つ。「ハンドシェイク」の項も参照してください。

DCL
全てのデバイスの内部機能のリセット

(またはデバイスクリア)	に使用するGPIBコマンド。「IFC」の項と「SDC」の項も参照してください。
DMA (ディレクトメモリアクセス)	GPIBとメモリ間の高速データ転送であって直接CPUにより処理されないもの。システムによってはDMAが使用出来ないものもあります。「プログラムによる入出力」の項も参照のこと。

E

END (またはエンドメッセージ)	データストリングの終止を示すメッセージ。ENDの送出はデータバイトの最後において「終了、あるいは識別コード要求(end or identify)」を意味するGPIBのEOI線をアサートすることにより行います。
EOI	GPIB線の一つであって、データメッセージの最終バイトを示す信号(END)として、あるいは、パラレルポーリングにおいて応答を要求する(IDYまたはIdentify)信号として使用されます。
EOS (EOS バイト)	7ビットあるいは8ビットでストリング終了(End-Of-String)文字を示すバイト。データメッセージの末尾に附けて送られる。

G

GET (group execute trigger)	GPIBコマンドでアドレスしたリスナのデバイスあるいは内部機能をトリガする時に使用するもの。
go to local	GTLの項を見てください。
GPIB	IEEE規格488で定義されている通信用

インターフェースシステム General Purpose Interface Bus (汎用インターフェースバス) の一般名。このバスを発明したヒューレットパッカード社ではこれをHP-IBと呼んでいます。

GPIBアドレス

GPIB上のデバイスのアドレス。1次アドレス（MLAとMTA)と、時によってはさらに2次アドレス(MSA)を加えてなっています。GPIBボードは一つのGPIBアドレスと一つの入出力アドレスを持っています。

GPIBボード

ナショナルインスツルメンツの一連のGPIBインターフェースボードの呼び名。

gpib.com

NI-488.2ドライバのファイル名。

group execute trigger

GETの項を見てください。

GTL (go to local)

GPIBコマンドの一つ。アドレスしたリスナをデバイス操作盤より制御するため　ローカルコントロールモードにする時に用います。

I

ibcntおよびibcntl

入出力関数の呼び出し毎に更新される実際に送信あるいは受信されたバイトの数を示すグローバル変数。ibcntは16ビットで転送されたバイト数を表わし、ibcntlは32ビットを全て使用してバイト数を表わしています。

ibconf.exe

NI-488.2ドライバの構成プログラム。「構成」の項も参照してください。

iberr

グローバル変数の一つ。関数呼び出し

	が失敗した場合に、その失敗に関するエラーコードを示します。
ibic	対話式インターフェース制御(Interface Bus Interactive Control)プログラム。GPIBデバイスとの通信、問題の原因発見と解決、及びアプリケーションプログラムの開発の目的で使用されます。
ibsta	グローバル変数の一つ。関数呼び出しの動作の後に返され、エラーの発生その他の重要なステータス情報を含んでいます。
install.exe	NI-488.2ソフトウエアのインストレーションプログラム。
ibtest.exe	NI-488.2ソフトウエアの診断プログラム。
IFC (インターフェースクリア)	GPIB線の一つ。システムコントローラはこの線によってバスを初期化します。「DCL」と「SDC」の項も参照のこと。
I/O	このマニュアルの中では、入出力とは、GPIBボードを介し、コンピュータとGPIB上のデバイス間に行われるコマンドないしはメッセージの伝送を意味します。
ist	パラレルポーリング構成関数により使用されるステータスバイト中の個々のステータスビット。

L

LAD(リスンアドレス)	MLAの項を見てください。

LLO
(ローカルロックアウト)

全てのデバイスに対するGPIBコマンド。これらのデバイスがローカルモードにある場合はこのコマンドはキーボードからのリモート(GPIB) データメッセージを無視せよと言うことを意味します。

M

MLA (my listen address)

リスナとなるデバイスにアドレスする時に使用するGPIBコマンド。31のMLA 1 次アドレスが存在する。

MSA(my secondary address)

2バイトによる拡張アドレスを使用してリスナあるいはトーカになるデバイスにアドレスする場合のGPIBコマンド。MLAまたはMTAの次にMSAを置くことにより完全なアドレスが形成される。31のこれら 2 次アドレスがあるので、961の明確に区別されたアドレスをデバイスに使用出来る。

MTA (my talk address)

トーカになるデバイスにアドレスするために使用するGPIBコマンド。31の1次アドレスがMTAとして使用出来る。

N

NDAC (not data accepted)

3本のGPIBハンドシェイク線の一つ。「ハンドシェイク」'の項を見てください。

NRFD (not ready for data)

3本のGPIBハンドシェイク線の一つ。「ハンドシェイク」の項を見てください。

O

on pen

ナショナルインスツルメンツではこのステートメントを使ってSRQ割り込みをインタセプトし、それらがユーザーのプログラムで使用出来る様にします。

P

PPC (parallel poll configure)

アドレスされたリスナがパラレルポーリングに参加できるよう構成する時に使用するGPIBコマンド。

PPD (parallel poll disable)

ポーリングに参加するように構成されたデバイスのポーリング参加を不可能化する場合に使用するGPIBコマンド。

PPE (parallel poll enable)

ポーリングに参加するように構成されたデバイスのポーリング参加を可能化し、１つのDIO応答線を割り当てる場合に使用するGPIBコマンド。

PPU (parallel poll unconfigure)

デバイスは何でもポーリングに参加出来ないようにする場合に使用するGPIBコマンド。

R

REN
(リモートモードの可能化)

システムコントローラにより制御されるGPIB線の１つ。但し、これはデバイスをリモートプログラムモードにする目的でCICが使用する線です。

S

SDC (選択したデバイスクリア)
アドレスしたリスナの内部機能あるいはデバイス機能をリセットする目的で使用するGPIBコマンド。「DCL」及び「IFC」の項も参照のこと。

SPD(シリアルポーリングの不可能化)
SPEコマンドをキャンセルするために使用するGPIBコマンド。

SPE (シリアルポーリングの可能化)
ある特定のデバイスをポーリングすることを可能化する目的で使用するGPIBコマンド。デバイスはトーカとしてアドレスされていなければなりません。「SPD」の項を参照のこと。

SRQ (サービス要求) 線
デバイスがCICにサービスを要求する時にアサートするGPIB線。

T

T1
一つのGPIBタイミングパラメータであって、主としてデータ整定時間—DIO線における新バイトがDAV信号線がアサートされる前に指定の裕度以内に達するまでの時間—と関連している。T1は350 nsecから2 µsec以上の範囲内に入る。

TAD (トークアドレス)
MTAを見てください。

TCT (take control)
バスの制御資格を現在のコントローラからアドレスしたトーカに引き渡す時に使用するGPIBコマンド。

TLC
GPIBのトーカ、リスナ、及びコントローラの機能のほとんどをハードウエアで行うための集積回路。

U

ud

一つの変数であって、関数の呼び出しがその対象としてGPIBインターフェースボードあるいはGPIBデバイスのユニット記述子を含む場合には何時も第一引き数となるもの。「ユニット記述子」の項を参照のこと。

UNL (unlisten)

アクティブなリスナに対するアドレスを解除する時に使用するGPIBコマンド。

UNT (untalk)

アクティブなトーカに対するアドレスを解除する時に使用するGPIBコマンド。

ア

アクセプタハンドシェイク

GPIBインターフェースのデータあるいはコマンド受信機能の一つ。これによりリスナはデータを受け取ることが出来、どのデバイスもこの機能によりコマンドを受け取ります。「ソースハンドシェイク」の項及び「ハンドシェイク」の項も参照してください。

アクセスボード

GPIBボードであって、バスに接続されたデバイスと相互に通信し、それらをコントロールするものをそれらのデバイスのアクセスボードと言います。

イ

IFC (インターフェースクリア)
GPIB線の一つ。システムコントローラはこの線によってバスを初期化します。「DCL」と「SDC」の項も参照のこと。

インターフェースメッセージ
コントローラから全てのデバイスに同時に送られる、GPIBの管理を目的としたメッセージ。よく使用されるインターフェースメッセージにはインターフェースクリア、リスンアドレス、トークアドレス、シリアルポーリング可能化／不可能化等があります。

ウ

ウォーム始動
「始動」の項を見てください。

エ

エンドメッセージ
データストリングの終止を示すメッセージ。ENDの送出はデータバイトの最後において「終了、あるいは識別コード要求 (end or identify)」を意味するGPIBのEOI線をアサートすることにより行います。

ケ

言語インターフェース
ある特定の言語で書かれたアプリケーションプログラムがソフトウエア関数を呼び出せる様にするためのコード。インタプリティブBASICの言語インターフェースはBIB.Mです。

コ

構成

ドライバ中のソフトウエアのパラメータを変更するプロセスを言います。これらのパラメータはドライバにより動かされるデバイス及びボードの主要特性をコントロールする情報です。この様な情報(例えばGPIBアドレス）をドライバの中にとっておくと、アプリケーションプログラムの度にアドレスを定義する手間が省けます。ibconf.exeがNI-488構成用プログラムです。

コマンド（またはコマンドメッセージ）

インターフエースメッセージを意味する一般的用語です。

コントローラ（あるいはコントローラインチャージまたはCIC)

インターフェースメッセージを他のデバイスに送ることによりGPIBを管理するデバイス。

サ

サービス要求

SRQの項を見てください。

シ

システムコントローラ

IFC (インターフェースクリア）メッセージを送ることによりシステム制御の資格を（と言うことはGPIBのCICになること）を要求出来る様に決められた唯一つのコントローラ。システムコントローラ以外のデバイスは制御の資格を引き渡されることによってのみCICになることが出来る。

始動

フロッピーあるいはハードディスクにあるオペレーティングシステムプログラムをメモリにロードしてコードの実行を開始すること。ハード始動はコンピュータの電力スイッチをいれることにより行うのに対し、ウォーム始動（ソフト始動とも言う）は特定のキーを押すことにより行われる。IBM PCの場合ウォーム始動のキーは<Ctr-Shift-Del>である。

自動シリアルポーリング

NI-488ソフトウエアの機能の一つ。デバイスがGPIBのSRQ線をアサートすると何時でもドライバが自動的にシリアルポーリングを行います。

始動ドライブ

コンピュータを始動する時のハードディスクドライブあるいはフロッピーディスクドライブ。

シリアルポーリング

デバイスを一つ一つポーリングし、ステータスバイトを読んでゆくプロセス。

シリアルポーリングの可能化

SPEの項を見てください。

シリアルポーリングの不可能化

SPDの項を見てください。

ス

ステータスバイト

デバイスがシリアルポーリングを受けたときに応答として送り出すデータバイト。

ステータスワード

ibstaと同じ。ibstaの項を見てください。

セ

宣言ファイル

NI-488ファイルの一つ。その中にはアプリケーションプログラムの始めの部分に置かれて、それが正しくドライバにアクセスできるようにするコードを含んでいます。DECL.BASはインタプリティブBASICA言語で書かれたプログラム用の宣言ファイルです。「言語インターフェース」の項も参照してください。

選択デバイスクリア (SDC)

アドレスしたリスナの内部あるいはデバイスファンクションをリセットする目的で使用するGPIBコマンド。「DCL」及び「IFC」の項も参照のこと。

ソ

ソースハンドシェイク

データとコマンドを伝送するGPIBインターフェース機能。トーカはこの機能を利用してデータを送り、コントローラはこの機能を利用してコマンドを送る。「アクセプタハンドシェイク」と「ハンドシェイク」の項を参照してください。

ソフト始動

「始動」の項を見てください。

タ

タイムアウト

NI-488.2ドライバの特長の一つ。入出力関数の使用中に、もしGPIBに問題があると、関数の動作が何時までも終了しないことが起こりうるが、タイムアウトはこれを防ぐために設定される。

テ

DMA
(ディレクトメモリアクセス)

GPIBとメモリ間の高速データ転送であって直接CPUにより処理されないもの。システムによってはDMAが使用出来ないものもあります。「プログラムによる入出力」の項も参照のこと。

データ
(あるいはデータメッセージ)

デバイス依存性メッセージを意味する一般的な用語。

DAV
(あるいはデータ有効線)

3本のGPIBハンドシェイク線の一つ。「ハンドシェイク」の項も参照してください。

デバイス

計測器、周辺機器、コンピュータ、あるいは他の電子機器であって、GPIBを介してプログラムできるもの。「ボード」の項も参照してください。

デバイス依存性メッセージ

プログラムの命令、データ、あるいはデバイスのステータスのように一つのデバイスから他のデバイスに送られるメッセージ。「コマンド」あるいは「インターフェースメッセージ」の項を参照してください。

デバイス関数

コンピュータ中のGPIBインターフェースボードでなく、GPIBデバイスに対して働いたり、あるいは別の意味で関係する関数のこと。「ボード関数」の項も参照してください。

デバイスクリア

「DCL」の項を見てください。

ト

トーカ

GPIBデバイスであって、データメッセージをリスナに送るものを言う。

TAD(トークアドレス)

MTAを見てください。

ドライバ

デバイスやインターフェースボードを動かすために使用するソフトウエアを意味する一般的な用語。

ニ

入出力 (I/O)

このマニュアルの中では、入出力とは、GPIBボードを介し、コンピュータとGPIB上のデバイス間に行われるコマンドないしはメッセージの伝送を意味します。

入出力アドレス

CPUから見た場合のGPIBボードのアドレスであって、GPIBボードのGPIBアドレスと区別される。入出力アドレスはポートアドレスまたはボードアドレスとも呼ばれます。

ハ

ハード始動

「始動」の項を見ください。

ハイレベル関数

基礎的なボード動作の幾つかを一緒にして一つの関数の動作として行うデバイス関数。ハイレベル関数を使用すれば、ユーザーはバスの管理やその他のGPIBプロトコルなどに煩わされずにオペレーションを実行できます。ローレベル関数の項も参照してください。

パラレルポーリング	全ての構成されたデバイスを同時にポーリングして合同のポーリング応答を読み取ること。シリアルポーリングの項を参照のこと。
パラレルポーリング構成	「PPC」を見てください。
パラレルポーリング構成解除	「PPU」を見てください。
パラレルポーリング可能化	「PPE」を見てください。
パラレルポーリング不可能化	「PPD」を見てください。
ハンドシェイク	一つのデバイスのソースハンドシェイク機能からもう一つのデバイスのアクセプタハンドシェイク機能にバイトを転送するメカニズム。DAV, NRFDおよびNDACの3 GPIB線をインターロック的に使用して転送の各状況を伝え、遅いほうのデバイスのスピードでバイトを非同期に (例えばクロックなしで) 送るようになっています。
汎用インターフェースバス (General Purpose Interface Bus)	「GPIB」の項を見てください。

ヒ

開かれたデバイス (あるいはボード)	ibfindによって開かれてオンラインになったデバイス (あるいはボード)。

フ

プログラム命令による入出力

GPIBボードとメモリの間で行われる低速のデータ転送。転送中は、CPUが一つ一つのデータバイトをプログラムの命令にしたがって動かします。DMAの項を参照してください。

ホ

ボード

コンピュータ中のGPIBインターフェースボードのことを単にボードと呼びます。「デバイス」の項も参照してください。

ボード関数

コンピュータ中のインターフェースボードの一つに対して、または別の意味でそれに関連して働く関数。これらインターフェースボードはGPIB0, GPIB1,...等として使用されます。「デバイス関数」の項も参照してください。

ボートアドレス

「入出力アドレス」の項を見てください。

ユ

ユニット記述子

ドライバがibfind関数で開いたデバイスなりボードを暫定的に識別するために使用する数。記述子はユニットのGPIBアドレスとは関係が無い。

リ

リスナ

トーカから送られたデータメッセージを受け取るGPIBデバイスを言います。

リスンアドレス　　MLAの項を見てください。

REN
(リモートモードの可能化)　　システムコントローラにより制御されるGPIB線の1つ。但し、これはデバイスをリモートプログラムモードにする目的でCICが使用する線です。

ル

ルートディレクトリ　　フロッピーまたはハードディスクにおけるトップレベルのディレクトリ。

ロ

LLO
(ローカルロックアウト)　　全てのデバイスに対するGPIBコマンド。これらのデバイスがローカルモードにある場合はこのコマンドの意味はキーボードからのリモート(GPIB)データメッセージは無視してよい（あるいは無視すべきである）と言うことであり、デバイスがリモートプログラムモードにある場合は、このコマンド意味はローカルの（デバイスの操作盤による）コントロールは無視してもよい（あるいは無視すべきである）と言うことです。

ローレベル関数　　単一のオペレーションを行う基礎的なボードまたはデバイス関数。「ハイレベル関数」の項を参照のこと。

索引

記号

A

B

C

D

E

F

G

I

L

M

N

P

Q

R

S

T

U

W

あ

い

え

お

か

く

こ

さ

し

す

せ

そ

た

ち

て

へ

ほ

ま

め

も

ゆ

よ

り

る

れ

わ