ZBK開発セット操作説明書

（有）中日電工

ZBK開発セット操作説明書　目次

2000.12.24 REV.1.1
2003.02.15 REV.1.2
2003.03.02 REV.1.3
2005.04.01 REV.1.4
2006.02.21 REV.2.1

1章　準備

1.　開発セットの内容

ZBKボード（ZB20K～ZB28K、ND80K）のプログラム開発を行うにはDOS／Vパソコンと接続する必要があります。開発セットはそのためのものです。
［注記］ZBK開発セット単独では使用できません。ZB20K～ZB28K、ND80Kのいずれかと組み合わせて使用します。

ZBKボードをDOS／Vパソコンに接続するには、USB接続とプリンタポート接続の2通りの接続方法があり、開発セットにはどちらの方法でも接続できるよう必要なボード、説明書が付属しています。
以下がセットの内容です。UはUSB用、Pはプリンタポート用、＆は両方で使うものです。

DOS／V接続用プリンタコネクタ基板（P）（プリンタケーブルはセットには含まれていません）
プリンタ基板接続用10pフラットケーブル（P）
USB接続ケーブル（U）
ZBKボード接続用フラットケーブル26p（U）
USB／ROM　WRITER基板（＆）
BS62LV1027（RAM628128互換）（＆）
27C1001（未使用品。ROM27C010互換）（＆）
フロッピーディスク（システムプログラム、SBASICコンパイラ、Z80クロスアセンブラ、逆アセンブラ）（P）
CDROM（USBドライバ、システムプログラム、SBASICコンパイラ、Z80クロスアセンブラ、逆アセンブラ）（U）
ZBK開発セット操作説明書（本書）（＆）
ZBK－V3BASIC操作説明書（＆）
ZBKボードハードウェア説明書（＆）
SBASIC説明書（＆）
Z80命令説明書（＆）

2.　用意するもの

①DOS／Vパソコン。
　プリンタポート接続ではMSDOS、Windows3．1、95、98、98SE、Me）のいずれかのOSが必要です。WindowsNT、2000、XPは使用できません。PC98シリーズには接続できません。
　USB接続ではWindows98、98SE、Me、2000、XPのいずれかのOSが必要です。WindowsNTは使用できません。PC98シリーズには接続できません。
②プリンタ接続ではプリンタケーブル（DOS／Vとプリンタを接続する汎用ケーブル）が必要です。
　USB接続の場合には接続ケーブルはセットに含まれています。
③＋5V1A程度以上の安定化電源（ACアダプタは使用できません）
［注意（重要）］
　可変出力型の電源は絶対に使用しないでください。5Vに合わせていても、電源ON時に高いサージ電圧が発生するらしく、ICが破損する事故がおきています。

3.　システムプログラムの準備

付属のCDROMまたはフロッピィディスクからZBKボードのプログラム開発に必要なプログラムをハードディスクにコピーします。接続方法（USBかプリンタポートか）やOSによって準備の方法が異なります。

（A）USB接続の場合

付属のCDROMをCDROMドライブにセットします。CDROMのドライブ番号はパソコンのハードウェア構成によって異なりますが、以下の説明ではDドライブにCDROMが割り当てられているものとします。

同じUSB接続でもOSがWindows98、SE、Meの場合とWindows2000、WindowsXPの場合では設定の仕方が少しずつ違います。

付属CDROMには以下の2つのフォルダがあります。

usbdriver

zbk

ここでは「zbk」フォルダをCドライブにコピーします。以下の説明で「クリックする」とは「マウスの左ボタンをクリックする」、「ダブルクリック」とは「マウスの左ボタンをダブルクリック（2回続けてクリック）する」、「右クリック」とは「マウスの右ボタンをクリックする」の意味です。Windowsの標準的なマウスの使い方ですが、これと異なるマウス操作の設定をしている人はそれぞれの機能に従って読み替えて操作してください。

（ア）Windows98、SE、Meの場合

「ZBK」フォルダをCドライブ（C：）にコピーします。コピーの方法はいろいろあります。慣れている方法でコピーしてください。

操作例：「マイコンピュータ」→「Dドライブ」の順にダブルクリックして開き、「ZBK」フォルダを右クリックしてメニューを開き、「コピー」をクリックします。次にマイコンピュータをダブルクリックして開きCドライブを右クリックしてメニューを開き、「貼り付け」をクリックします。

Cドライブを開いてZBKフォルダができたことを確認してください。

次にDOSプロンプトをすぐに使えるようにします。ZBKシステムは一般的なWindowsプログラムと違い、DOSプロンプト（DOS窓）で使います。

「スタート」→「プログラム」→「MSDOSプロンプト」の順にマウスで選択し、「MSDOSプロンプト」を右クリックします。メニューが開くのでその中の「コピー」をクリックします。次にデスクトップ（起動後のマイコンピュータなどのショートカットアイコンが並んでいる画面）のアイコンなどがない地の部分にマウスを持っていって、そこで右クリックします。ここでもメニューが開くので「貼り付け」をクリックします。

デスクトップに「MSDOSプロンプト」のアイコンができたことを確認してください。

デスクトップにできたアイコンをマウスで右クリックします。メニューの中の「プロパティ」をクリックします。「MSDOSプロンプトのプロパティ」の「プログラム」タブをクリックします。「作業ディレクトリ」を書き換えます。初期状態では「C：￥WINDOWS」になっているはずです。ここを「C：￥ZBK」に書き換えてください（かならず半角で入れてください。半角なら大文字でも小文字でもよいのですが、全角では正しく実行されません）。書き換えたら「OK」をクリックします。

最後にMSDOSプロンプトアイコンの下に表示されている名前も変更しておきます。さきほどと同じようにMSDOSプロンプトアイコンを右クリックしてメニューを開き、「名前の変更」をクリックします。名前（MSDOSプロンプト）が白抜き文字で表示されるので、てきとうな名前に変更します。「ZBK」でよいでしょう（ここは全角でも漢字でも構いません）。このあとUSBドライバのインストール作業があるのでCDROMは入れたままにしておいてください。

（イ）Windows2000の場合

「ZBK」フォルダ」をローカルディスク（C：）にコピーします。コピーの方法はいろいろあります。慣れている方法でコピーしてください。

操作例：「マイコンピュータ」→「Dドライブ」の順にダブルクリックして開き「ZBK」フォルダを右クリックしてメニューを開き、「コピー」をクリックします。次に「マイコンピュータ」をダブルクリックして開きローカルディスク（C：）を右クリックしてメニューを開き、「貼り付け」をクリックします。

ローカルディスク（C：）を開いて「ZBK」フォルダができたことを確認してください。

次にコマンドプロンプトをすぐに使えるようにします。ZBKシステムは一般的なWindowsプログラムと違い、コマンドプロンプトで使います。

「スタート」→「プログラム」→「アクセサリ」→「コマンドプロンプト」の順にマウスで選択し、「コマンドプロンプト」を右クリックします。メニューが開くのでその中の「コピー」をクリックします。次に「マイコンピュータ」→「ローカルディスク（C：）」の順にダブルクリックして開き、さきほどコピーして作った「ZBK」フォルダを右クリックします。メニューが開くので「貼り付け」をクリックします。「ZBK」フォルダを開いて「コマンドプロンプト」がコピーされたこと

を確認してください。
「ZBK」フォルダにできた「コマンドプロンプト」アイコンを右クリックします。メニューの中の「プロパティ」をクリックします。「コマンドプロンプトのプロパティ」の「プログラム」タブをクリックします。「作業フォルダ」を書き換えます。初期状態では「%HOMEDRIVE%%HOMEPATH%」になっているはずです。ここを「%HOMEDRIVE%￥ZBK」に書き換えてください（かならず半角で入れてください。半角なら大文字でも小文字でもよいのですが、全角では正しく実行されません）。書き換えたら「OK」を左クリックします。
Windows98と違いWindows2000の場合にはコマンドプロンプトのショートカットをデスクトップに貼り付けると指定した作業フォルダが規定値に書き換えられてしまいます。そこでコマンドプロンプトのショートカットをデスクトップに作る代わりに、ZBKフォルダのショートカットをデスクトップに貼り付けます。
「マイコンピュータ」→「ローカルディスク（C:）」の順にダブルクリックして開き、「ZBK」フォルダを右クリックします。メニューが開くので「コピー」をクリックします。次にデスクトップ（起動後のマイコンピュータなどのショートカットアイコンが並んでいる画面）のアイコンなどがない地の部分にマウスを持っていって、そこで右クリックします。ここでもメニューが開くので「貼り付け」をクリックします。
デスクトップに「ZBKへのショートカット」のアイコンができたことを確認してください。このあとUSBドライバのインストール作業があるのでCDROMは入れたままにしておいてください。

（ウ）WindowsXPの場合
「ZBK」フォルダをローカルディスク（C:）にコピーします。コピーの方法はいろいろあります。慣れている方法でコピーしてください。
操作例：「マイコンピュータ」→「Dドライブ」の順にダブルクリックして開き「ZBK」フォルダを右クリックしてメニューを開き、「コピー」をクリックします。次に「マイコンピュータ」をダブルクリックして開き「ローカルディスク」（C:）を右クリックしてメニューを開き、「貼り付け」をクリックします。
「ローカルディスク」（C:）を開いて「ZBK」フォルダができたことを確認してください。
次にコマンドプロンプトをすぐに使えるようにします。ZBKシステムは一般的なWindowsプログラムと違い、コマンドプロンプトで使います。
「スタート」→「プログラム」→「すべてのプログラム」→「アクセサリ」→「コマンドプロンプト」の順にマウスで選択し、「コマンドプロンプト」を右クリックします。メニューが開くのでその中の「コピー」をクリックします。次にデスクトップ（起動後のマイコンピュータなどのショートカットアイコンが並んでいる画面）のアイコンなどがない地の部分にマウスを持っていって、そこで右クリックします。ここでもメニューが開くので「貼り付け」をクリックします。
デスクトップに「コマンドプロンプト」のアイコンができたことを確認してください。
デスクトップにできたアイコンを右クリックします。メニューの中の「プロパティ」をクリックします。「コマンドプロンプトのプロパティ」の「プログラム」タブをクリックします。「作業ディレクトリ」を書き換えます。初期状態では「%HOMEDRIVE%%HOMEPATH%」になっているはずです。ここを「%HOMEDRIVE%￥ZBK」に書き換えてください（かならず半角で入れてください。半角なら大文字でも小文字でもよいのですが、全角では正しく実行されません）。書き換えたら「OK」を左クリックします。
最後にコマンドプロンプトアイコンの下に表示されている名前も変更しておきます。さきほどと同じようにコマンドプロンプトアイコンを右クリックしてメニューを開き、「名前の変更」をクリックします。名前（コマンドプロンプト）が白抜き文字で表示されるので、てきとうな名前に変更します。「ZBK」でよいでしょう（ここは全角でも漢字でも構いません）。このあとUSBドライバのインストール作業があるのでCDROMは入れたままにしておいてください。
［注意］Windows98、SE、Me、Windows2000はMSDOSプロンプト（コマンドプロンプト）の作業フォルダを複数のショートカットでそれぞれ個別に指定できますが、WindowsXPでは複数のショートカットで別々に作業フォルダを指定してもすべてのショートカットが最後に書き換えた作業フォルダの内容に書き換えられてしまいます。ZBKシステムプログラム以外にコマンドプロンプトで実行するプログラムがあるときには、そのことに注意してください。

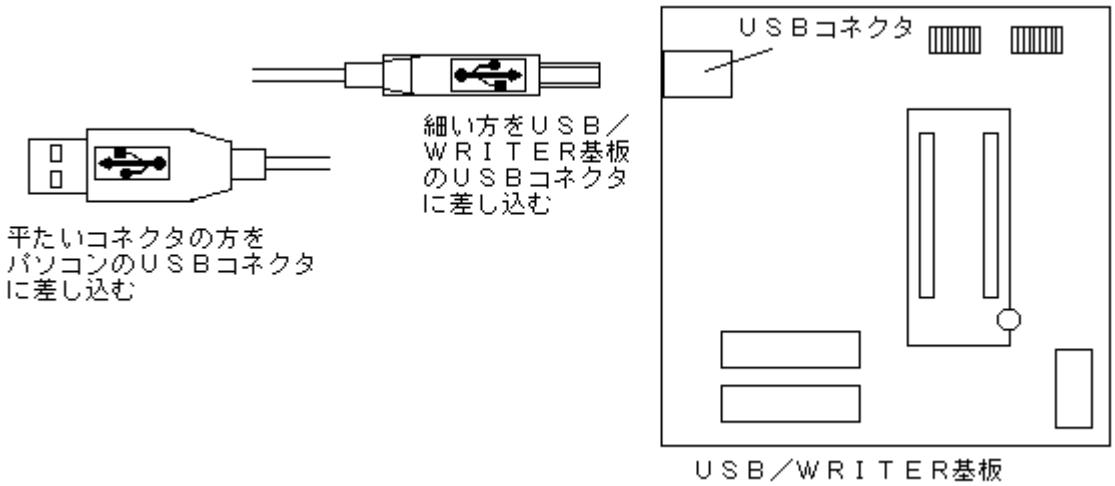
（B）プリンタポート接続の場合（Windows2000、XPでは使えません）
付属のフロッピーディスクをフロッピーディスクドライブに入れてください。
「マイコンピュータ」→「3．5インチFD」（A:）の順にダブルクリックして開きます。フロッピーディスクの中身が表示されます。フロッピーディスクには「ZBK」フォルダがあります。この「ZBK」フォルダをCドライブ（C:）にコピ

ーします。コピーの方法はいろいろあります。慣れている方法でコピーしてください。
　操作例：「ZBK」フォルダを右クリックしてメニューを開き、「コピー」をクリックします。次にツールバーの「上へ」をクリックするか、「マイコンピュータ」をダブルクリックして開き「Cドライブ」を右クリックしてメニューを開き、「貼り付け」をクリックします。
「Cドライブ」を開いて「ZBK」フォルダができたことを確認してください。
　次にDOSプロンプトをすぐに使えるようにします。ZBKシステムは一般的なWindowsプログラムと違い、DOSプロンプト（DOS窓）で使います。
「スタート」→「プログラム」→「MSDOSプロンプト」の順にマウスで選択し、「MSDOSプロンプト」を右クリックします。メニューが開くのでその中の「コピー」をクリックします。次にデスクトップ（起動後のマイコンピュータなどのショートカットアイコンが並んでいる画面）のアイコンなどがない地の部分にマウスを持っていって、そこで右クリックします。ここでもメニューが開くので「貼り付け」をクリックします。
　デスクトップに「MSDOSプロンプト」のアイコンができたことを確認してください。
　デスクトップにできたアイコンを右クリックします。メニューの中の「プロパティ」をクリックします。「MSDOSプロンプトのプロパティ」の「プログラム」タブをクリックします。「作業ディレクトリ」を書き換えます。初期状態では「C:￥WINDOWS」になっているはずです。ここを「C:￥ZBK」に書き換えてください（かならず半角で入れてください。半角なら大文字でも小文字でもよいのですが、全角では正しく実行されません）。書き換えたら「OK」をクリックします。
　最後にMSDOSプロンプトアイコンの下に表示されている名前も変更しておきます。さきほどと同じようにMSDOSプロンプトアイコンを右クリックしてメニューを開き、「名前の変更」をクリックします。名前（MSDOSプロンプト）が白抜き文字で表示されるので、てきとうな名前に変更します。「ZBK」でよいでしょう（ここは全角でも漢字でも構いません）。これでソフトウェアの準備は完了です。フロッピーディスクドライブからフロッピーディスクを取り出してください。

プリンタポート接続の場合には以上でソフトウェアの準備ができましたが、USB接続の場合にはUSBドライバを組込む作業が残っています。USBドライバの組込みは購入後最初に1回だけ必要な作業です。

3．2　USBドライバのインストール（USB接続の場合のみ最初に1回だけ必要）

　付属のUSBケーブルを使ってUSB／ROMWRIER基板とDOS／Vパソコンを接続します。この時にはZBKボードと組み合わせる必要はありません。
　USB／ROMWRITER基板に接続したUSBケーブルをDOS／VパソコンのUSBコネクタに接続します。



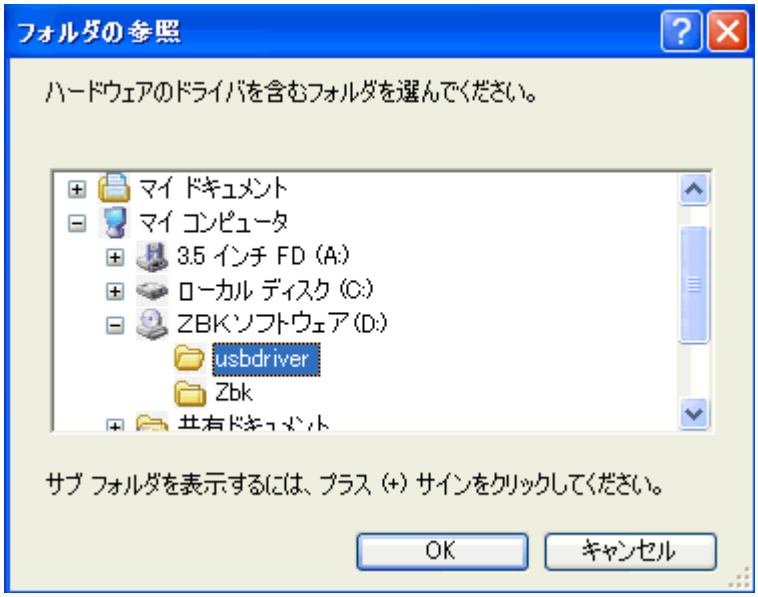
［Windows98，SE，Meの場合］
「新しいハードウェアの追加ウィザード」が表示されます。

（付属のCDROMが入っていない場合には付属のCDROMをCDROMドライバにセットしてください）
「次の新しいドライバを検索しています：
USB＜－＞Serial 」
の表示画面では「次へ」をクリックします。
「検索方法を選択してください。」の表示では「使用中のデバイスに最適なドライバを検索する」を選択して「次へ」をクリックします。
次の画面では
「検索場所を指定」のみを選択して（その他にもチェックがついていたら、他はすべてクリックしてチェックを外す）、「参照」をクリックします。「フォルダの参照」の窓にハードウェア構成のツリーが表示されるので「ZBKソフトウェア」（CDROM、D：）の前の「＋」をクリックします。CDROMの中にあるフォルダが表示されるので「usbdriver」をクリックして選択してから「OK」をクリックします。

「検索場所の指定」窓に「D：￥usbdriver」と表示されるので「次へ」をクリックします。
「ドライバのある場所」に「D：￥USBDRI~7￥FTDIBUS．INF」と表示されるので「次へ」をクリックします。
「新しいハードウェアデバイスに必要なソフトウェアがインストールされました」と表示されるので「完了」をクリックします。
CDROMからドライバが読み込まれてインストールが完了しますが、このとき少し時間がかかる場合があります（1～2分）。砂時計のままで1～2分待っても普通のマウスの矢印マークに戻らないときはマウスを少し動かしてみてください。なお上の「完了」のクリックのあと、「新しいハードウェアの追加ウィザード」がまた開始されることがありますが、そのときは同じ操作をもう一度繰り返してください。

［Windows2000の場合］
「新しいハードウェアの検索ウィザードの開始」が表示されます。「次へ」をクリックします。
（付属のCDROMが入っていない場合には付属のCDROMをCDROMドライバにセットしてください）
「ハードウェアデバイスドライバのインストール」の表示では「デバイスに最適なドライバを検索する」を選択して「次へ」をクリックします。
「ドライバファイルの特定」では「場所を指定」を選択して「次へ」をクリックします。
「製造元のファイルのコピー元」では「参照」をクリックします。
表示される「ファイルの場所」では左窓の「マイコンピュータ」を選び、右窓に表示されるハードウェア構成のなかから、CDROM（ZBKソフトウェア、D：）を選び、ダブルクリックします。
「Usbdriver」をダブルクリックして開き、「ファイル名」のところに「Ftdibus．inf」が表示されていることを確認して（表示されていなければ上の窓の中のFtdibus．inf」を選択します）、「開く」をクリックします。
「製造元のファイルのコピー元」に「D：￥usbdriver」と表示されるので、「OK」をクリックします。
「このデバイスのドライバがみつかりました」の表示には「次へ」をクリックします。
インストールが終了したら「完了」をクリックします。
再び「新しいハードウェアの検出ウィザード」が表示されます。
さきほどと同じように「次へ」をクリックし、「デバイスに最適なドライバを検索する」を選択して「次へ」をクリックします。
「場所を指定」のみを選択して「次へ」をクリックします。
「製造元のファイルのコピー元」に「D：￥usbdriver」と表示されているので、「OK」をクリックします。
「このデバイスのドライバがみつかりました」の表示には「次へ」をクリックします。
インストールが終了したら「完了」をクリックします。

［WindowsXPの場合］
「新しいハードウェアの検索ウィザードの開始」が表示されます。「いいえ、今回は接続しません」を選択して「次へ」をクリックします。
（付属のCDROMが入っていない場合には付属のCDROMをCDROMドライバにセットしてください）
「一覧または特定の場所からインストールする」を選択して「次へ」をクリックします。
「次の場所で最適のドライバを検索する」を選択し、「次の場所を含める」を選択して 「参照」をクリックします。
表示されるハードウェアツリーの「マイコンピュータ」→「ZBKソフトウェア（CDROM）」（D：）の順に前の「＋」をクリックして開き、「usbdriver」フォルダを選択して「OK」をクリックします。



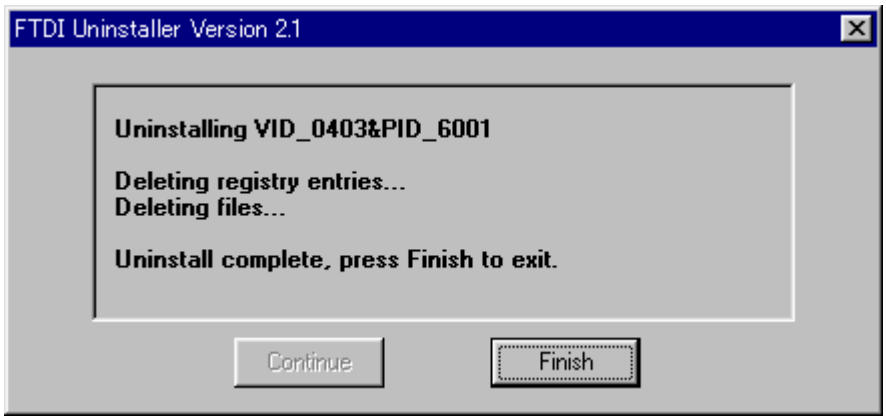
「次の場所を含める」の表示窓に「D：￥usbdriver」と表示されるので、「次へ」をクリックするとドライバのインストールが開始されます。
しばらくすると
「次のハードウェアのソフトウェアのインストールが完了しました
USB Serial Converter」
と表示されるので「完了」をクリックします。
再び「新しいハードウェアの検索ウィザードの開始」が表示されます。
先ほどと同じように、「いいえ、今回は接続しません」を選択して、「次へ」をクリックします。
今回は「ソフトウェアを自動的にインストールする」を選んで「次へ」をクリックします。
最終的にドライバの組込みが完了するので、「完了」をクリックします。

3．3 ドライバのアンインストール

何らかの理由でドライバがうまく機能しないなどの理由でドライバをアンインストールしたいときがあります。
付属のCDROMをセットして、「usbdriver」フォルダを開きます。
「Ftdiunin. exe」をダブルクリックします。


Ftdiunin.exe

次頁の表示が出るので、もしUSB／WRITER基板がパソコンに接続されていたらUSBケーブルを外します。
「Continue」をクリックします。
「Uninstall complete」の表示が出るので「Finish」をクリックします。



3．4　USB接続時の注意

USBドライバを組み込み後に、パソコンを起動させてUSB／WRITER基板をつないだUSBケーブルを接続すると、しばらくの間（30秒～1分程度）ハングアップしたようにマウスもキーボードも受け付けないことがありますが、これはOSがハート゜ウェアの検出を行っているためでトラブルではありません。

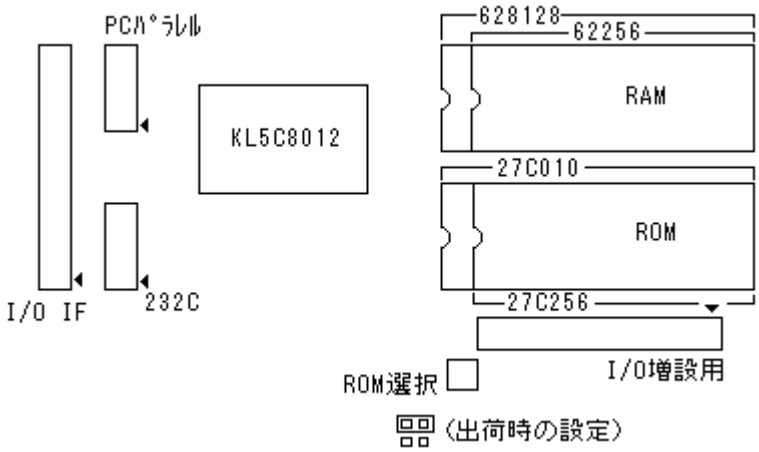
4．　接続

ZBKボードのプログラム開発をするためにZBKボードをUSBまたはプリンタケーブルでDOS／Vパソコンに接続します。購入直後のZBKボードはそのままでは開発のためのRAM容量が不足します。まずRAMを交換します。

4．1　ZBKボードのRAMを交換する

開発する場合にはRAMがBS62LV1027（628128互換）でなければいけません。標準ではZB10K～28K、ND80KにはBS62LV256（RAM62256互換）が実装されています。

基板を傷つけたり、RAMのピンを曲げたりしないように注意しながらBS62LV256をICソケットから外します（小型のマイナスドライバなどを使います。ドライバを深い角度で差し込むとプリント基板を傷つけてしまいます。パターンが切れると動作しなくなってしまいます。慎重に作業してください）。



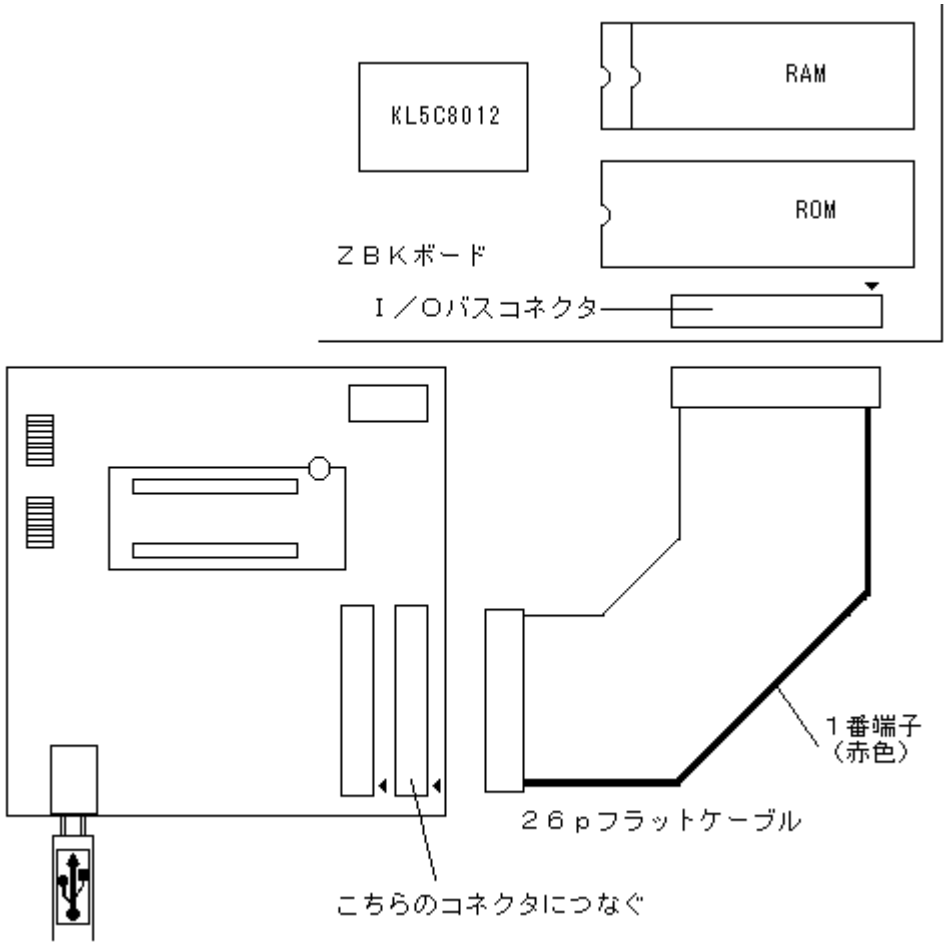
次に開発セットに入っている62LV1027（628128）をそのソケット取り付けます。向きに注意してください。ICソケットとRAMの丸い切り欠きの向きを同じにします。62256は28ピンなので1、2ピン側を空けて取り付け

ますが、628128はICソケットにすべてのピンが挿入されるように取り付けます（前ページ図）。
ND80KはシステムROMも交換します。標準では27C256が実装されています。27C256を外して別売のZBK－V3BASICROM（27C010）を取り付けます。27C256は28ピンなので1、2ピン側を空けて取り付けますが、27C010はICソケットにすべてのピンが挿入されるように取り付けます。
ZB10K～28K、ND80Kはすべて共通の接続コネクタになっています。CPU近くの配置は上図の通りです。

4．2　ZBKボードをDOS／Vパソコンに接続する

ここから先はUSB接続とプリンタポート接続とで方法が異なります。以下それぞれ分けて説明します。

（A）USB接続
ZBKボードにはUSBコネクタがないので直接パソコンと接続することはできません。USB／WRITER基板とZBKボードを接続し、USB／WRITER基板とパソコンとをUSBケーブルで接続します。
3．2で説明したUSB／WRITER基板とパソコンとの接続は、購入後はじめてパソコンに接続してドライバをインストールするときのみの方法です。その後はいつもここで説明するように必ずUSB／WRITER基板とZBKボード（ZB10K～ZB28K、ND80K）とを接続し、ZBKボードには＋5Vの電源を供給するようにしてください。
USB／WRITER基板とZBKボードとは26pフラットケーブルを使って接続します（下図）。

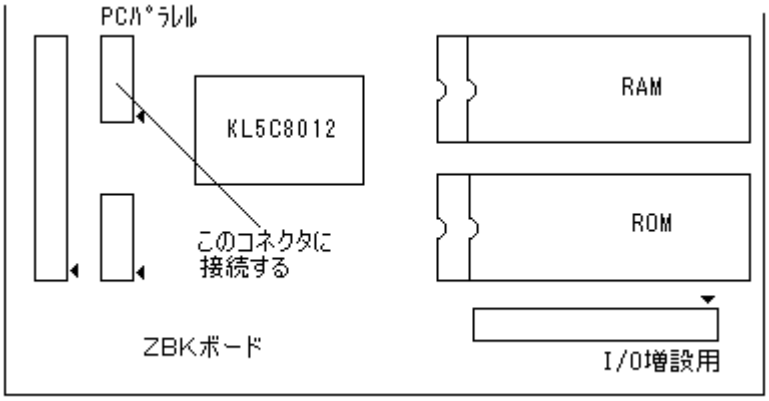


ZBKボードには26pコネクタが複数個あります。USB／WRITER基板と接続するのはROMのすぐ下にある26pコネクタです。USB／WRITERボードは前ページ図のように26pコネクタが2つあります。このどちらでも接続できますが図のように基板端側のコネクタの方が接続しやすいので通常はこちらに接続してください。残ったもうひとつの26pコネクタはADコンバータボードなどの増設ボードを接続するときに使います。
ZB10K～ZB28K、ND80Kに＋5V電源を接続します。

（B）プリンタポート接続

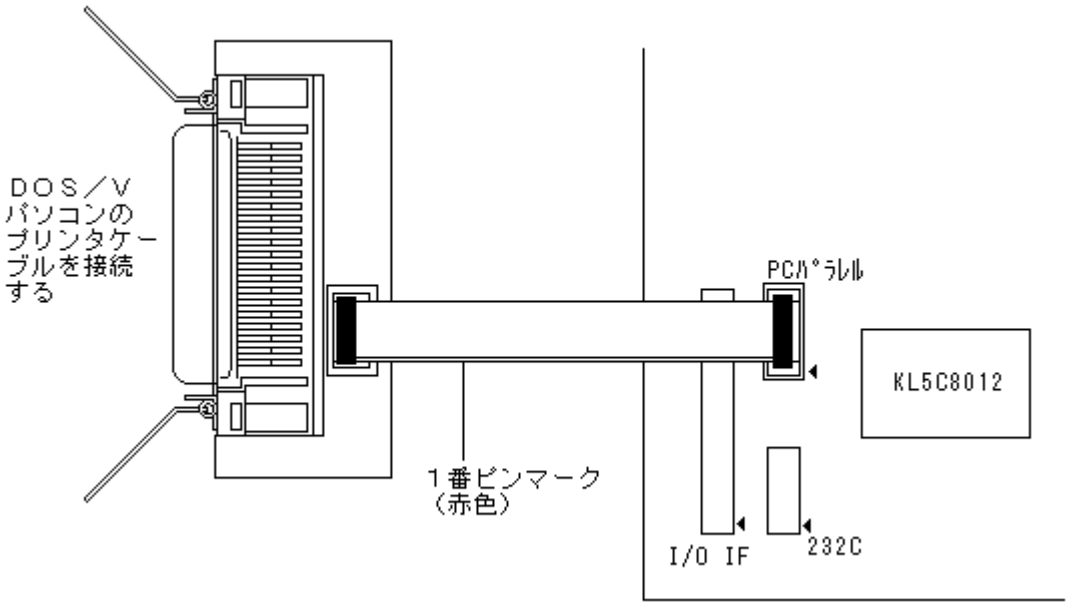
ZBK開発セットはUSB接続のほかにプリンタポートでの接続もできます。ただしWindows2000およびWindowsXPではOSの違いのためプリンタポート接続はできません。

次の図で「PCパラレル」と表示してある10PコネクタがDOS／Vパソコンのプリンタポートと接続するコネクタです。



このコネクタとプリンタコネクタ基板を次図のように接続します。向きをまちがえないように、1列だけ挿入したりしないように注意して接続します。

その後プリンタコネクタとDOS／Vパソコンを汎用のプリンタケーブルで接続します。（プリンタケーブルはセットには含まれていません）



ZB10K～ZB28K、ND80Kに＋5V電源を接続します。

5.　スタート

ZBKプログラムはMSDOSプロンプト（コマンドプロンプト）上で実行します。

3．でデスクトップに作ったZBKアイコンをダブルクリックします。

黒いDOS窓が開いてカーソルマーク（ _ ）が表示されます（Windows2000ではデスクトップのZBKアイコンをダブルクリックすると、ZBKフォルダが開くのでその中にあるコマンドプロンプトのアイコンをダブルクリック

します）。
C：￥ZBK＞_
と表示が出て、キーボードから入力できるようになります。以下のZBKプログラム作業ではマウスは使いません。キーボード入力のみになります。マウスを使うのは、ZBKでの作業を終了するか、一時中断してDOSプロンプト（コマンドプロンプト）を離れて他のWindowsアプリケーションを使うときだけです。一時的にDOSプロンプト（コマンドプロンプト）を離れたあと、ふたたびZBKプログラム作業に戻るには、いちどDOSプロンプト（コマンドプロンプト）の黒い画面をマウスで左クリックします。再びDOSプロンプト（コマンドプロンプト）でキーボードから入力ができるようになります。
上の表示のとき、キーボードから、ZBK［Enter］と入力すると、ZBKプログラムが起動するのですが、USB接続の場合には、先にZBKボードの電源をONにして＋5Vを供給してから、ZBK［Enter］と入力します。
プリンタポート接続では逆にZBK［Enter］と入力してから、ZBKボードの電源をONにします。

［注意］キーボードからの入力は半角英数で行います（IMEの入力モードは「直接入力」にしてください）。またUSB接続では英大文字でも小文字でも同じ動作をしますが、プリンタポート接続では大文字しか正しく動作しません（［Shift］を押しながら［CapsLock］を押すと大文字入力で固定します。再び同じ操作をするとCapsLockが解除されます）。

ZBK［Enter］と入力するとシステムプログラムが起動します。
C：￥ZBK＞ZBK［Enter］
DEBUG TOOL FOR KL5C8012
（C）Copyright CHUNICHIDENKO 2005
と表示されます。

（A）プリンタポート接続の場合
ここで表示がとまるのでZB10K～ZB28Kの電源をONにします。接続が正常ならば
t／0302のような表示（数字は機種によって異なる）が出て、プロンプトマーク ） 右括弧 が表示されます。
ND80Kは電源ONではDOS／Vと接続しないときと同じようにLEDがオール0表示になってキー入力待ちになります。ND80Kのキーボードから次のように操作してください。
0200［ADRSSET］［RUN］

［注記］
t／4507のように上位に0が無いような大きな数字の時は、接続がうまくいっていない可能性があります。［Ctrl］キーを押しながら［C］キーを押してブレイクし、ZBKボードの電源も切ってから、もう一度やり直してみてください。
またt／0302のような表示が出なくてハングUPしているときは、ZBK側の電源を入れ直してみてください。反応が無い場合には電源を切って、一度DOSプロンプトを終了し（タイトルバーの右上の［×］を左クリックし強制終了します）、もう一度最初からやり直してみてください。
タイミングやハードウェアによっては、USB接続の場合と同じように先にZBKボードの電源をONにしておいてから、ZBK［Enter］を入力した方がよい場合もあります。どうしてもだめな場合には当社までご連絡ください。

（B）USB接続の場合
先にZBKボードの電源をONにしておかないと、
ZBKボードが接続されていないかボードの電源が入っていません。
のメッセージが表示されます。
ZBKボードの電源を入れてから、ZBK［Enter］を入力してください。
USBポートを通じてZBKボードと接続が開始され、
now connecting to ZBK, wait a while. . .
の表示が出ます。数秒で接続が完了し、
connect ok
）

と表示されますが、もし数秒待っても、connect ok の表示が出ないときは、一度[Ctrl]キーを押しながら[C]キーを押してブレイクしてから、もう一度ZBK[Enter]と入力してみてください。それでも駄目な場合には、もう一度ブレイクし、ZBKの電源を切って、再度ONしてから、ZBK[Enter]を入力してください。どうしてもだめな場合には当社までご連絡ください。

これ以後コマンドの入力が可能になります。プリンタポート接続ではコマンドやBASICの命令はA～Zの大文字と数字です。小文字やカナはデータとしては扱えますが、コマンドやパラメータとしては使えません。USBポート接続ではコマンドもBASIC命令も小文字で入力ができます。

これ以後の説明は基本的にはUSB接続でもプリンタ接続でも同じです（一部プリンタ接続では対応していないUSB接続のみの機能もあります。そのような機能については「USB接続のみ」というように注記して説明を行います）。

)Z[Enter]

と入力します。

プロンプトマークが＞になって文字表示がグリーンになります。プロンプトマークがMSDOSと同じなのでBASICモードでは表示をグリーンにしてあります。DOS／Vのキーボードから入力する文字は白文字で、ZBKボードから送られてくる文字は緑文字になります。

[注記]

＞にならずにレジスタダンプが表示されることがありますが、もういちどZ[Enter]と入力してください。

ZBK開発セット操作説明書（本書）でとりあえず必要なのはここまでです。これ以後はZBK－V3BASIC操作説明書を参照してください。作成したプログラムをディスクに保存したり、ROMに書込んだりするときには、また本書が必要になります。

BASICプログラムについて一通り理解できたら、マシン語モニタコマンドやアセンブラも使ってみてください。

別冊のZBKボードハードウェア説明書は組込みボードを使ってI／O制御を行うときに必要なハードウェアの情報について説明しています。必要に応じて参照してください。

参考までにZBK－V3BASICを使う上での基本的な事柄の概要を下に記します。下記概要とその次の「6. システムの終了」「7. ログファイル」を読んでから、ZBK－V3BASIC操作説明書に移ってください。

[ZBK－V3BASIC基本操作の概要]

行番号つきのプログラムを入力作成することができます。

スクリーンエディタ機能が働いています。

マウスは使えません。

カーソルの移動は[←][↓][↑][→]と[Backspace]、[Delete]および[Insert]を使います。[Enter]を押すことでその行が入力確定されます。

プリンタポート接続ではPageUp、PageDown機能はありません。

USB接続では[PageUp][PageDown][Home][End]キーによってPageUp、PageDownができます。どんどん入力して画面の上から消えてしまった行も[PageUp][PageDown][Home][End]キーによって再表示できます。

AUTOコマンドで行番号を自動表示できます。

アセンブラ、逆アセンブラ、SBASICはここでは使用しません。ZBKを起動しないで、MSDOSプロンプト（コマンドプロンプト）上でZASM. COM、ZDAS. COM、SBASIC. COMを使用します（なおラインアセンブラ、ライン逆アセンブラはZBK－V3BASIC上で使用します）。

6. システムの終了

／EXITコマンドを使います。システムがハングUPしたときは、[Ctrl]CでMSDOSに戻ります。

DOS／VとZBKボード間での通信がくいちがってしまいハングUPした場合には[Ctrl]Cが受けつけられない

ことがあります。そのときはDOSプロンプト（コマンドプロンプト）画面の右上の×ボタンをクリックして強制終了してください。[！]メッセージが出ますが[はい]をクリックすれば終了できます。
終了後にZB10K～28Kの電源をOFFにしてください。ND80Kはリセットするか電源を切ってください。

7. ログファイル

開発システムが起動すると、その時の月日時分をファイル名とするログファイルがOPENします。システム起動中に入力、表示された内容はすべてログファイルに記録されます。
／EXITコマンドで終了したときは完全なログが保存されます。それ以外の方法で終了した場合には最後の1ページは保存されません（USB接続では[Ctrl]＋[C]で終了した場合でも正しくログファイルが保存されます）。
ログファイルはZBK. COM（ZBK. EXE）が置かれたフォルダ上のZBKLOGフォルダ（存在しなければ自動的に作成される）に12031453. TXTのように名前がつけられて作成されます（システムが起動した日時を名前にします。この例では12月3日14時53分）。ログファイルはテキストエディタで開いて参照、またはプリンタ出力したり、さかのぼって表示を確認するなどしてデバッグの助けとすることができます。
ログファイルはサイズが大きくなるので適宜削除してください。

7．1 ／CLOSE（USB接続のみ）

ZBKが起動中に作成されつつあるログファイルはそのZBKを終了するまではテキストエディタなどで参照することができません。
USB接続の場合には／CLOSEコマンドが用意されており、任意の時点で／CLOSE[Enter]と入力すると一旦ログファイルがクローズされ、その日時をファイルネームとするログファイルが新たに作られます。クローズされたログファイルはZBKを終了しなくてもテキストエディタなどで参照することができます。

ZBK開発セットのメリットは8ビットCPUボードの制御をBASICプログラムで行うことができる点にあります。
しかし処理によってはマシン語サブルーチンと併用したいときも出て来ます。ZBK開発セットはマシン語プログラムの開発にも適しています。

1．Z80アセンブラ

マシン語プログラムの作成にはアセンブラが便利です。ZBK開発セットにはDOS／Vパソコン上で使えるZ80アセンブラ（ZASM. COM）とZ80逆アセンブラ（ZDAS. COM）が含まれています。マシン語サブルーチンの作成にはZASM. COMを使えば簡単にZ80マシン語プログラムを作成することができます。ZASM. COMはZBK－V3BASICを起動して使うのではなく、MSDOSプロンプト（コマンドプロンプト）上で単独に使います。
ZBK－V3BASICを起動させた状態で別のMSDOSプロンプト（コマンドプロンプト）を開いてそこでZASM. COMを実行し、マシン語プログラムを作成してからZBK－V3BASICのMSDOSプロンプト（コマンドプロンプト）に戻って、いま作成したマシン語プログラムのバイナリファイルをロードする、といった使い方もできます。Z80アセンブラの使い方については9章を参照してください。

2．ラインアセンブラ

小さなマシン語プログラムで今ちょっと一回だけ使うだけで、特に保存しておく必要はない、という場合に便利なのがラインアセンブラです。
ZBK－V3BASICにエントリした状態で適当なユーザー用アドレス（テキストエリアのアドレス）を指定して、ASコマンドを入力するだけですぐに使うことができます。

```
＞AS　8000［Enter］
8000　_　……ここでZ80ニモニックを入力すればただちにマシン語コードに翻訳される
```

手軽に使えるアセンブラとして重宝します。詳細は7章を参照してください。

3．マシン語モニタコマンド

アセンブラ、ラインアセンブラが使えるシステムですからマシン語コードをそのまま入力するCMコマンドはプログラム作成の手段としては、すでに使命を終えた感があります。しかしメモリの中身を確認したり、少し値を変更したりするには、CMコマンドは非常に有効な手段になります。また数バイトではなくて数十バイトのメモリの内容を確認するにはDMコマンドが便利です。
アセンブラを利用したプログラムをデバッグするにはマシン語モニタコマンドの助けが必要です。BP／RT（ブレークポイント／リターンコマンド）はマシン語プログラムのデバッグには欠かせない機能です。
マシン語モニタコマンドの詳細については4章を参照してください。

4．逆アセンブラ

これも有りがたい機能です。マシン語プログラムのソースプログラムがどこかへいってしまって見つからない、とかあるいはもともと昔に作ったプログラムでROMだけしか残っていない、という場合に威力を発揮します。
ROMならばROM　WRITERにセットしてR64～R256コマンドでRAMに読みこみます。CMコマンドやDMコマンドを使えば書いてある内容を16進コードで確認することはできます。しかしその一部を変更するとか、別のシステムに移植するとかになると、マシン語コマンドでそれを行うのは至難の技です。
内容を判りやすいZ80ニーモニックに直して読みたいというのでしたら、即席で使えるライン逆アセンブラが便利です。DAコマンドを使って、解読したいメモリアドレスを指定すれば1ステップずつZ80ニーモニックに翻訳して表示します。ライン逆アセンブラについては8章を参照してください。

マシン語のバイナリファイルを読みこんで、それを再アセンブル可能なZ80アセンブラソースプログラムにまで作り上げてしまう、本格的な逆アセンブラも利用できます。MSDOS上で使用できるZ80逆アセンブラについては10章を参照してください。

5. KL5C8012(Z80A)のレジスタ

5.1 レジスタ

KL5C8012(Z80A)は内部に下記のレジスタを持っていて、これらのレジスタはプログラムの中で色々な処理に利用されます。

← 8ビット→ ← 8ビット→

A	F
B	C
D	E
H	L

← 8ビット→ ← 8ビット→

A'	F'
B'	C'
D'	E'
H'	L'

← 16ビット →

IX	
IY	
SP	
PC	
I	R

(裏レジスタ)
A～Lは全く同じレジスタがもう1組ある。
通常裏レジスタと呼んでいる。EXX' やEX AF, AF' 命令で表裏を切り換えるが厳密な意味での表裏の区別は無い(裏も表に呼び出せばその時点から表レジスタになってしまう)

[A]
一般にアキュムレータ(加算器)と呼ばれているように、演算命令はこのレジスタを中心に行われる。
[F]
フラグレジスタ。命令の実行により現れる色々な状態を1ビットずつに記録して保持する。各ビットの意味は5.2で説明する。
[B][C]
共に8ビットのレジスタとして、独立して使うこともできるが、つないで16ビットのレジスタ[BC]として使うこともできる。その場合には[B]が上位8ビット、[C]が下位8ビットになる。[B][BC]をカウンタとして使っている命令がいくつかある。
[D][E]
B、Cと同じだがカウンタとして使う命令はない。
[H][L]
B、Cと同じだがカウンタとして使う命令はない。16ビットレジスタ[HL]はメモリアドレスを入れて使うことが多い。また[HL]は16ビットの加減算命令で加算器(アキュムレータ)としても使われる。
[IX][IY]
インデックスレジスタとして使うが、16ビットのワークレジスタとして使うこともできる。[HL]と同様に16ビットの加減算命令で加算器(アキュムレータ)としても使われる。
[SP]
スタックポインタ。現在のスタックのトップ・アドレスを示している。スタックについては、5.3で説明する。
[PC]
プログラムカウンタ。現在実行中のアドレスを管理している。(正しくは、次のアドレスを示している)
[I]
インタラプト・ベクタ。モード2の割り込みで使用される。Iレジスタはシステムモニタでも使っているのでユーザーはIレジスタの値を変更してはならない。
[R]
リフレッシュレジスタ。ダイナミックRAMのリフレッシュアドレス出力用カウンタ。プログラムでは使わない。

5．2 KL5C8012（Z80A）のフラグ

フラグは8ビットのフラグレジスタに、下のように割りつけられています。

S	Z		H		P／V	N	C
7	6	5	4	3	2	1	0

（ビット位置）

各記号の意味は下の通りです。（ビット3とビット5は使用されない）

なお、フラグがセットされたときは、そのビットが1になり、リセットされたときは0になります。

C

キャリ・フラグ。計算結果がオーバーフローしたときなどにセットされる。ローテイト命令でもセット、リセットされる。

N

加減算フラグ。加算命令のときリセット、減算命令のときセットされる。

P／V

パリティ・オーバーフロー・フラグ。論理演算のときはパリティ・フラグとして用いられ、演算の結果1のビットが偶数個あるときにセットされ、奇数個のときはリセットされる。算術演算のときはオーバーフロー・フラグとして用いられ結果がオーバーフローしたときセットされる。

H

ハーフ・キャリ・フラグ。算術演算でビット3からビット4へのキャリーや、ビット4からビット3へのボローがあったときセットされる。これはNフラグとともに、BCD演算後のDAA命令で利用される。

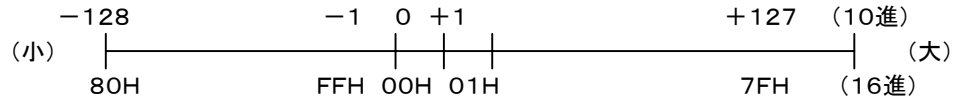
Z

ゼロ・フラグ。結果がゼロのときセットされる。

S

サイン・フラグ。結果が負のときセット、正またはゼロのときリセットされる。8ビットの数00H～FFHは符号なしでは10進の0～255として扱われるが、符号付の数として扱ったときには、－128～＋127の数になり、これは16進では80H～7FHになる。（ビット7が0のときはその数は正で、ビット7が1のときは負になる）

●符号付8ビットの数の大小



5．3 スタック

大きなプログラムになると、レジスタもたくさん必要で、とても5．1で説明した数では足りません。そこでレジスタの値をひとまずメモリのワークエリアにしまっておいて、そのレジスタを次の用途に使う、ということが簡単にできると便利になります。

ところがレジスタの値をしまうときに、一々異なるメモリアドレスに割りつけていくのでは大変です。

そんなときにこのスタックを使えば、一々メモリアドレスを指定しなくても簡単な操作でレジスタの値を保存することができます。

スタックとは積み重ねるという意味です。

ちょうど本などを積み重ねるように、メモリの中にレジスタの値を順番にしまうことができます。（PUSH命令を使います）

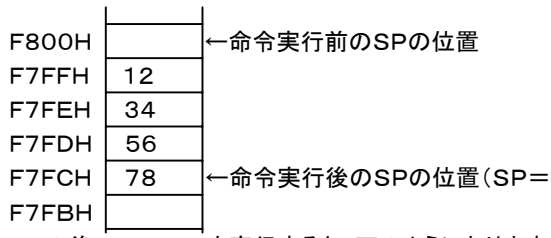
取り出すときは、入れたときと逆の順番で取り出します。（POP命令を使います）

そしてそのスタックの現在の位置を管理しているのがSP（スタックポインタ）です。

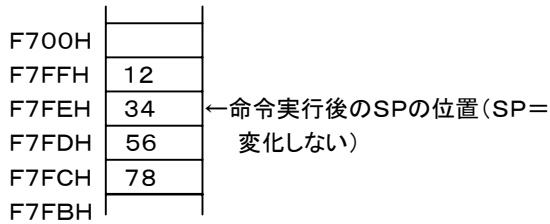
（例）SP＝F800H、BC＝1234H、DE＝5678Hのとき、

```
PUSH  BC
PUSH  DE
```

を実行すると、メモリ内容は下のようになります。



この後で POP HL を実行すると、下のようになります。



こうなってしまった後で、POP　DEを実行してもDEにはもとの値は戻りません（1234Hが入る）。

PUSH、POPは常に順番を覚えておいて、間違わないように使う必要があります。

〔注意1〕

スタックの操作はPUSH、POPだけではなくCALL、RET命令や割り込み処理でも使用されます（アドレスがスタックに入れられる）。

ZBK－V3BASICにエントリして作成したマシン語のプログラム、データやBASICのプログラムはZBK－V3BASICを終了すれば消えてしまいます。実際にはZB10Kを除いてはニッカドバッテリでRAMのバックアップをしているので消えてしまうことはありません。しかしその上から別のプログラムで上書きすれば無くなってしまいます。

マシン語プログラム、データやBASICプログラムはファイル名をつけてディスクに保存することができます。作成されるファイルはMSDOSの規則にしたがってつくられますから、MSDOSやWindowsのツールで参照したり編集することが可能です。マシン語プログラム、データはバイナリ形式のファイルになります。テキストエディタで見ることはできませんが、MSDOSのDEBUGコマンドで編集することができます。

BASICプログラムはテキスト形式で保存されますから、Notepadなどのテキストエディタで参照、修正、プリンタへの出力などが行えます。

各コマンドはZBK－V3BASICにエントリした状態で使用します。

1. マシン語プログラム（およびデータ）の保存

／SVコマンドを使います。

[書式]／SV ファイルネーム, aaaa, bbbb

指定アドレス範囲（aaaa～bbbb）のマシン語プログラム（データ）をファイル名をつけてディスクに保存します。aaaa, bbbbは4桁の16進数でaaaa＜＝bbbbです。ファイル名はMSDOSの規則に従います。マシン語プログラムはZ80アセンブラによってバイナリファイルを作成することができますから、このコマンドを使うことは少ないかもしれません。CMコマンドなどで一部を変更したり、あるメモリ範囲の内容をそのまま保存する場合に有効な手段になります。

／SVコマンドで作成したバイナリファイルを逆アセンブラ（ZDAS. COM）にかけてアセンブラソースプログラムを作成することができます。

[使用例]
＞／SV TEST__SUB. BIN, 4004, 4365[Enter]

[注記]
ファイル名は名前部が8桁以内の英数字および－＿で拡張子は3桁以内です。拡張子は任意です。付けなくても構いません。ファイル名にはドライブ名やディレクトリ名も付加することができます。
例: ／SV A:￥BASIC￥TESTPRO. ROM, 4000, 7FFF

2. マシン語プログラム（およびデータ）のファイルからの読出し

1. で作成したバイナリファイルをRAMの指定アドレスに読みこみます。／LDコマンドを使います。

[書式]／LD ファイルネーム, aaaa

ファイル名にはドライブ名やディレクトリ名も付加することができます。aaaaは4桁の16進数でLOAD先のRAMアドレスを示します。aaaaを省略することはできません。ファイルがみつからないときは FILE NOT FOUND と表示されます。

[使用例1]
＞／LD TEST__SUB. BIN, 4004[Enter]

3.　BASICプログラムの保存

ZBK－V3BASICにエントリして作成、編集したBASICプログラムは／SAVEコマンドでディスクに保存することができます。／SAVEコマンドについてはZBK－V3BASIC操作説明書でBASICコマンドとして説明していますのでそちらも参照してください。

[書式]／SAVE　ファイルネーム

ファイルネームがファイル名のみの場合にはZBK－V3BASICが存在するディレクトリにSAVEされます。同じファイル名があると上書きされます。
ファイル名にドライブ名やディレクトリ名を含めて記述することで、別のドライブやディレクトリにSAVEすることができます（使用例③）。
現在のテキストエリアのアドレス情報は保存されません。／LOAD時に新しく決定されます。
／SAVEコマンドではBASICシステムによって1行ずつBASIC内部コードをテキスト形式に変換していきます。このときプログラムは1行ずつ画面にリスト表示されます。

[使用例①]
＞／SAVE　TEST. TXT[Enter]
[使用例②]
＞／SAVE　MIHON[Enter]　　……拡張子はなくてもよい。
[使用例③]
＞／SAVE　A:￥BASIC￥TESTPRO. BAS[Enter]
[注記1]
使用例③のように別のドライブや別のディレクトリにSAVEすることもできます（上の①②例ではZBK－V3BASICの存在するディレクトリにSAVEされます）。

4.　BASICプログラムのファイルからの読出し

テキスト形式で保存されたファイルをBASICプログラムとしてZBKボードのRAMエリアに読みこむことができます。／LOADコマンドを使います。

[書式]
／LOAD　ファイルネーム, aaaa

ファイルネームがファイル名のみの場合にはZBK－V3BASICが存在するディレクトリからLOADします。ファイル名にドライブ名やディレクトリ名を含めて記述することで、別のドライブやディレクトリにあるファイルをLOADすることができます（使用例③）。ファイルがみつからないときは　FILE　NOT　FOUND　と表示されます。
aaaaは4桁の16進数で、省略することもできます。省略した場合には現在のテキストエリアの先頭からLOADされます。aaaaをつけるとaaaa番地からLOADされます。
／SAVEでファイルを作成したときのテキストエリアのアドレス情報は保存されません。／LOAD時に新しく決定されます。
／LOADコマンドでテキスト形式のファイルが読み込まれるとき、BASICシステムによって1行ずつBASIC内部コードに変換してRAMに格納していきます。このときプログラムは1行ずつ画面にリスト表示されます。内部コードに変換するときに文法エラーが見つかるとそこでエラーコードが表示されLOAD作業は打ち切られます。
メモ帳などのテキストエディタで作成したプログラムも文法的に正しければ／SAVEで保存されたファイルと同じようにLOADすることができます。

[注意1]
LOADコマンドの動作は、NEW　＋　／LOAD、またはNEW　aaaa　＋　／LOAD、というように必ずNEWコマンドの動作を伴っています（LOAD前にRAMに存在したプログラムは失われます）。

［注意2］
　メモ帳（Notepad）は問題ありませんが、WriteやWordでは通常はそれぞれの形式でファイルが作成されます。保存するときにテキスト形式を指定してもゴミが混じる場合があります。Writeなどで作成したファイルがうまくLOADできないときは、一度メモ帳（Notepad）で開いてゴミを削除して、メモ帳で保存してからLOADしてみてください。また全角英数モードで作成されたファイルも読みこめません（異常な表示になります）。必ず半角英数モードで作成してください。

［使用例①］
＞／LOAD　TEST．TXT［Enter］
［使用例②］
＞／LOAD　MIHON［Enter］　　……テキスト形式のファイルなら拡張子のついていないファイルでもよい
［使用例③］
＞／LOAD　A：￥BASIC￥TESTPRO．BAS，5000［Enter］
　この例のように別のドライブや別のディレクトリにあるファイルをLOADすることもできます。上の①②例ではプログラムは現在のテキストエリアの先頭からLOADされます。③では5000番地からLOADされます。

［注意3］
ファイル名は名前部分が半角英数8文字以内で拡張子は3文字以内に限られます。
［注意4］
テキスト形式（拡張子の種類に関わらず内容がテキスト形式になっている）以外のファイルをLOADするとZBKーV3BASICシステムが暴走してハングアップすることがあります。テキスト形式のファイルでも半角英数字以外の文字が使用してあると、やはり暴走してしまいます。

5．マシン語サブルーチンとBASICプログラムの一括保存

　マシン語プログラムはアセンブラで作成して、ソースプログラムは□□□．TXT、マシン語コードは□□□．BINなどというファイル名で保存します。BASICプログラムは△△△．TXTのようにそれらとはまた別に保存します。テキストエディタで編集を行うためにはこの形でなければいけませんが、プログラムとして完成した時点で保存を考えると、BASICプログラムだけではなくてマシン語サブルーチンと一緒に働くプログラムの場合には、別々に保存するのは面倒でもあります。
　この場合、BASICブログラムとマシン語プログラムを一括してバイナリ形式で保存できると便利です。
　マシン語プログラムだけの場合と同じように／SVコマントを使いますが、そのままではあとでBASICプログラムをLIST表示可能な状態に戻すことはできません。／SVコマンドに先だってBROMコマンドでBASICテキストの情報を作成します。

［書式］／SV　ファイルネーム，4000，bbbb

［使用例］
＞BROM［Enter］
4500－5732
＞／SV　TESTPRO．ROM，4000，5732［Enter］

［注記］
　ファイル名にはドライブ名やディレクトリ名も付加することができます。
例：　／SV　A：￥BASIC￥TESTPRO．ROM，4000，7FFF
［注意1］
　／SVの先頭アドレスは必ず4000にします。
　先にBROMを実行しておなかいと、／LD作業でBASICプログラムを再生させることができません。
　bbbbはBROMの実行により表示される2番目のアドレス値にします。使用例で／SVのアドレス指定は4500，5732ではなくて4000，5732になる点に注意してください。

［注意2］
　このようにして保存したファイルにはBASICプログラムも含まれていますがバイナリイメージで保存しているため、Notepadなどのテキストエディタで参照することはできません。

6．マシン語サブルーチンとBASICプログラムを一括保存したファイルからの読みこみ

　この場合もマシン語プログラムだけの場合と同様に／LDコマントを使いますが、LOAD後にBASICプログラムをLIST表示可能にするため、BSSETコマンドの実行が必要です。ファイルにBASICプログラムが含まれていない場合や／LDで4000以外のアドレスを指定した場合、または／SVでBROMを実行しなかったか、SAVE開始アドレスを4000にしなかった場合には、BSSETコマンド入力時にHOW？が表示されます。

［書式］／LD　ファイルネーム，4000

［使用例］
＞／LD　A：￥BASIC￥TESTPRO．ROM，4000［Enter］
2543BYTES　LOAD
＞BSSET［Enter］
TEXT　4500－6543
ﾍﾝｽｳ　DBC9－DFFF

メモリの内容をチェックしたり、部分的にマシン語のデータを変更したり、簡単なマシン語プログラムを書いて、すぐに実行してみたい、というようなことがよくあります。
BASICの命令をダイレクトモードで実行してもそのようなことはできるのですが、ZBK－V3BASICに含まれている、マシン語モニタコマンドを利用すると、BASICとはまた異なった細かい操作が、簡単に行えます。
またマシン語モニタコマンドのほかに、ZBK．COM（ZBK．EXE）にはファイル操作や終了処理などのシステムコマンドも用意されています。これらのコマンドはZBKボード単独で（ROM化して）使用することはできません。DOS／Vパソコンと接続しているときのみ、パソコンのキーボードから入力して実行することができます。
［注意］BASICのコマンドでもLISTはROM化することはできません。

●コマンドとパラメータの区切りについて

旧システムではコマンドとパラメータの区切りとして ,（カンマ）を使っていましたがBASICでは命令の後ろの区切り（セパレータ）としてスペースを使っており、またアセンブラニーモニックも命令とオペランドとの区切りはスペースを使っている点を考えてZBK－V3BASICではコマンドの後ろの区切りマークとしてスペースを使うように改めました。 しかしカンマでの使用に慣れているユーザーのためも考えて区切りマークとしてカンマも使えるようにしてあります。この説明書の書式としては、セパレータとしてスペースを使ってあります。

／BOOT ［省略形］無し

ブートプログラムに戻ります。通常の使い方では必要ありません。ブートプログラムについては12章を参照してください。

BP／RT／CR ［省略形］B.（RT、CRの省略形は無し）

［書式1］BP　aaaa
［書式2］BP　0
［書式3］BP　D

ZBKシステムではマシン語プログラムの開発、デバッグに役立つようRAM上に書かれたプログラムなら、どこでも実行中にブレイクできるようブレイクポイントが1カ所設定可能です。ただしROM上のプログラムには使用できません。（BPはbreak　pointの、またRTはreturnの略です）
aaaaは4桁の16進数で、ブレイクしたいアドレスを指定します。
例えば、下のようなプログラムを実行する場合を考えます。

```
C000  2100C1          LD  HL,＄C100
C003  75        LOOP:LD  (HL),L
C004  2C              INC  L
C005  C203C0          JP  NZ,LOOP
C008  C33310          JP  ＄1033
```

このプログラムは＄C100～＄C1FFにデータ＄00～＄FFを書きこむものです。
まず、CMコマンドでC000～C00Aまでに上のマシン語プログラムを1バイトずつ入力します。
これを実行するにはJP，C000でよいのですが、この動作を確認したいと仮定します。
下のように入力してみます。

＞BP　C003[Enter]

＞JP　C000[Enter]

　すると下のような出力が得られます。
```
A F　B C　D E　H L　A'F' B'C' D'E' H'L'　PC　 SP　 IX　 IY　I　SZ H PNC
xx44 xxxx xxxx C100 xxxx xxxx xxxx xxxx C003 F800 xxxx xxxx xx 01000100
```

　BPコマンドでアドレスを指定したうえで、マシン語のプログラムを実行させると、そのアドレスまで実行が進んだところで、そのアドレスの命令を実行する直前でブレイクしてその時の全レジスタの内容を表示してコマンド待ちになります。
　このプログラム例ではフラグレジスタ（F）、HLレジスタ、プログラムカウンタ（PC）、スタックポインタ（SP）以外は使用していないため、その他のレジスタの内容には意味はありません（誤解を避けるため、xxxx にしてあります）。
　一番最後の8桁はフラグレジスタ（F）の内容をビット毎に表示したものです。
　この状態は普通のコマンドモードと変わりないので、他のコマンドを受け付けることができます。例えばDMコマンドでメモリの内容を確認したりすることもできます。
　必要な確認が全て済んで、以後の実行を継続したい時は

＞RT[Enter]

　と入力します。以後は普通に処理が終了します。
　なおBPは続けてセットすることもできます。
　上の状態でブレイクしているときに、下のように入力してみて下さい。

＞BP　C005[Enter]
＞RT[Enter]

　再び、ブレイクしますが今度は下のようにF、HL、PCが変化していることが確認できるはずです。
```
A F　B C　D E　H L　A'F' B'C' D'E' H'L'　PC　 SP　 IX　 IY　I　SZ H PNC
xx00 xxxx xxxx C101 xxxx xxxx xxxx xxxx C005 F800 xxxx xxxx xx 00000000
```

　＄C005ではフラグの内容がチェックされ、Zフラグが0FFならば再び＄C003に戻って処理が続けられます。フラグの内容を見てみると、右のフラグのビット表示は全て0になっていて、どのフラグもOFFになっていることがわかります。
　そこで再びBP　C003[Enter]とセットしたうえで、RT[Enter]を入力すると、今度はC003でブレイクします。
　なおブレイクポイントを設定するときは、必ず命令の1バイト目のアドレスを指定する必要があります。
　上の例で、C000、C003、C004、C005、C008は指定できますが、C001、C002、C006、C007、C009、C00Aを指定すると正しく実行されません。
　ところが正しく指定しているにもかかわらず、指定アドレスによってはブレイクがかからないときがあります。
　上のプログラム例では全部の命令が少なくとも一回は実行されますが、プログラムによっては、条件付の分岐命令があって、条件によっては全く実行されない部分がでてきます。
　そのような場合にはブレイクポイントを設定したにもかかわらず、ブレイクしないで処理が終了してしまいます。
　このような場合には次のことに注意して下さい。
　このブレイク動作は指定アドレスの命令を、FF（RST　7）で置きかえることで実現しています。CPUはFFコードを見つけると＄0038からの命令を実行しますが、このシステムでは、その＄0038にブレイク処理ルーチンが置いてあります。
　ブレイクすると、ブレイク処理が行われると共に、ブレイクアドレスに、FFのかわりにもとの命令コードをもどします。ところがブレイクしないとこのFFが残ってしまいます。
　例えば上の例で、BP　C003[Enter]を入力したあと、すぐにCMコマンドかDMコマンドでC000～C00Aの内容を確認してみて下さい。C003にFFが入っているのが確認できます。

ブレイクしないために残ってしまったFFコードをもとの命令コードに戻すには、

＞BP 0[Enter] ……（書式2）

と入力します（もう一度C000～C00Aの内容を確認してみて下さい。C003にFFが入っていたのが元の75に戻っているはずです）。

ブレイクポイントがセットされたままになっているかどうかはっきりしない時は、下のようにBP D[Enter]と入力すればわかります。セットされているときはそのアドレスが表示されます。

＞BP D[Enter] ……（書式3）
C003 ……BPアドレスが表示される。BPがセットされていない時は何も表示されない

[注意]
リセットすると、BPがセットされたままになっていても、FFコードがプログラム内に残されたままで、BP関係の制御情報はクリアされてしまいます。そうなった後で、BP 0[Enter]を実行してもFFはもとに戻りませんし、BP D[Enter]を実行しても何も表示されません。

[**CR**コマンド]
ブレイクしたアドレスから続きを実行するときにレジスタの値を変更すると都合がよい場合があります。ループしているプログラムの動作をデバッグしていて、ループカウンタの途中のある値の近くを詳しく調べたいときなどがあります。1000回のループの終りの20回くらいをチェックしたいときに980回もBP／RTを繰り返すのでは疲れてしまいます。このときループカウンタとしてメモリのワークエリアを使っていれば、ループで一旦ブレイクしループカウンタの値をCMコマンドで変更しておいて（1000を20にしてしまいます。同時にループ回数に伴って変化するワークエリアがあれば、それも合理的な値に直します。）、ループ内の次のブレイクポイントを設定してRTコマンドを実行すれば、980回を一度で済ますことができます。
CRコマンドはA、B、CなどのCPUレジスタもメモリのワークエリアと同じように変更できる機能です。
ブレイクしているときに、CR[Enter]と入力します。
ブレイク直後と同じレジスタ表示が行われます。

```
>CR[Enter]
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
3785 0004 0000 C6BE 502C 0000 0000 0000 25C5 F7EA 204B 0182 FE 10000101

_
```

↑カーソルが表示されます。[↑][→]で変更したい値を書き換えます。スクリーンエディタが働いていますからレジスタ全部でも1レジスタでもフラグのみでも変更できます。変更する必要のないレジスタはそのままにしておきます。例としてA＝50、HL＝1234、キャリーフラグOFFに書き換えてみます。

```
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
5085 0004 0000 1234 502C 0000 0000 0000 25C5 F7EA 204B 0182 FE 10000100_
```

必要な変更が済んだら、[Enter]を入力します。カーソルはこの行の中ならどこにあっても構いません。[Enter]入力によってレジスタが更新されます。そして更新結果がすぐ下に表示されます。

```
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
5084 0004 0000 1234 502C 0000 0000 0000 25C5 F7EA 204B 0182 FE 10000100
>
```

[注意1]
フラグレジスタの変更は右のビット表示の部分で行います。左のAFの部分を書き換えても更新はできません。ビット部分を書きかえることで、結果のAFレジスタ部分が更新されます（上例でFレジスタが85から84に変化していることを確認してください）。
[注意2]
カーソルを他の行に移動してはいけません。通常の＞表示によるコマンド入力待ちのスクリーンエディタとは違うルーチンで入力しますから、他の行で[Enter]を入力するとエラーになります。
[注意3]

CRはブレイク中にレジスタの値を変更し、その後にRTで中断中のプログラムを再開することを前提にしています。その他のタイミングで使用しても無意味です（システムには影響を与えません）。

［注意4］

CRは繰り返して実行できますが、値の更新はCRコマンド入力直後のレジスタ表示の時のみ有効です。普通のブレイク表示の時やCRでの値の更新後の［Enter］入力により表示されたレジスタ表示行に対して更新作業はできません。もう一度更新したいときはあらためて、CRコマンドを入力します。

●以上のコマンドを下にまとめて整理しておきます。

①BP aaaa　ブレイクしたいアドレスをセットする。命令コードの1バイト目のアドレスを指定する。先の例でBP C003はよいが、BP C002やBP C001は不可。また、ROM上のプログラムにはセットできない
②BP 0　ブレイクポイントを解除する。
③BP D　ブレイクポイントがセットされているときはそのアドレスを表示する
④RT　ブレイクしていたアドレスから以後を続けて実行する。
⑤CR　ブレイク後にレジスタの値を更新してRTによる実行再開に反映させる

［注記1］

すでに説明したように、このブレイクの機能は、RST 7（FF）を利用しているため、ブレイクポイントがセットされているときに、マシン語プログラムのミスなどで暴走した結果、別のアドレスでたまたまFFコードに行きつくと、ブレイクポイント以外の部分でもブレイクしてしまうことがあります。

［注記2］

ユーザープログラム内でSPに新しい値を設定しない場合には、システムスタック（＄F490～＄F7FF）がそのまま使用されることになります。

BPの設定により、実行途中でブレイクするといままでユーザーが使用していたスタックをシステムが使用することになります。そのままではスタック内容が変化してしまうため、RTコマンドでユーザープログラムの実行を再開した場合に、正しく実行されなくなります。

そこでこのシステムでは、通常使われるスタックを768バイトと仮定し、＄F500～＄F7FFの内容を、ブレイク時に別のエリア（＄E500～＄E7FF）に一時待避します。そしてRTコマンドが入力されると、この待避データをもとの＄F500～に戻した上で、ユーザープログラムにリターンします。

ブレイク時のスタックの状態を確認したい場合に、もしシステムスタックをそのまま利用している場合には、この＄E500～＄E7FFを見るようにします（DM E500，E7FF[Enter]を入力する）。

BROM　［省略形］BR.　BRO.

BASICプログラムの開始アドレスと終了アドレスを＄4000～＄4003の制御エリアに書込みます。

＄4000～＄4003はBASICプログラムのROM化に必要なエリアです。ここに正しい情報が書き込まれていないと、ROM化してもシステムはBASICプログラムROMと判断しないでエラーメッセージを表示します。

W256 B、W1M Bコマンドは BROMコマンドの機能を含むため、あらためてBROMコマンドを実行する必要はありません。

／SVコマンドでBASICプログラムとマシン語プログラムをバイナリファイルとして一括保存することができますが、その場合には、／SVにさきだってBROMの実行が必要です。

BSSET［省略形］BS.　BSS.　BSSE.

バイナリイメージでロードしたBASICプログラムをLIST表示可能な形にします。

／SVコマンドでBASICプログラムとマシン語プログラムを一括保存したバイナリファイルもマシン語のみのバイナリファイルと同様に／LDコマンドでRAMに読み込むことができますが、そのままではBASICプログラムもバイナリのままなのでLIST表示させることはできません。BSSETコマンドによってLIST表示が可能になり、編集や追加、削除などができるようになります。

[使用例]
＞／LD TESTPRO. ROM, 4000[Enter]
＞BSSET[Enter]
TEXT 4500－6543
ﾍﾝｽｳ DBC9－DFFF

／CLOSE（USB接続のみ）[省略形]なし

ZBKが起動中に作成されつつあるログファイルはそのZBKを終了するまではテキストエディタなどで参照することができません。
USB接続の場合には／CLOSEコマンドが用意されており、任意の時点で／CLOSE[Enter]と入力すると一旦ログファイルがクローズされ、その日時をファイルネームとするログファイルが新たに作られます。クローズされたログファイルはZBKを終了しなくてもテキストエディタなどで参照することができます。

CM [省略形]C.

[書式]CM aaaa

指定アドレス（aaaa）のメモリの内容を1バイトずつ16進数で表示し、キーボードから入力する16進数と置き換えます。（aaaaは4桁の16進数）
マシン語で簡単なプログラムを書きたい時や、数バイト～数十バイト程度のメモリ内容を書き換えたい時などに使うと便利です。（CMはchange memoryの略です）
なおこの機能はRAMに対して有効です。指定したアドレスがROMであった場合でも、エラーメッセージは出ずに一応は正常に実行されますが、メモリの中味は書き換えられません。

[使用例]
①内容を確認する
＞CM 8100[Enter]
8100 2A－[Space] 指定したアドレスとそのメモリの内容が表示される。このままの内容でよい場合には
8101 87－ スペースキーを押すと、次のアドレスが同じように続けて下に表示される

②内容を更新する
8100 2A－31 下線部のように入力する。[Enter]は必要無い。0～F以外のキーを押すと？が出てもう一
8101 87－ 度同じアドレスが表示される。データ（2桁の16進数）を入力すると、次のアドレスが同じように続けて表示される

③前のアドレスに戻る
8100 2A－31 入力ミスなどでもう一度前のアドレスに戻りたいときは[←] を押す。前のアドレスとその
8101 87－[←] 内容が下に表示される
8100 31－

④処理の終了
この作業を打ち切りたい時は[Enter]を押すか又は[Ctrl]B を入力します。するとプロンプトマーク（＞）が出てコマンド入力待ちに戻り、カーソルマークが表示されます。
この時までに行った書き換え作業はすでに実行済みなので、処理を打ち切っても取り消しはできません。

[注意]
スペースキー、[←] 、[Enter]、[Ctrl]B は1桁目のデータ入力では有効ですが、2桁目の入力では受け付けられません。？マークが表示されます。

8105　3A－_　　ここでのスペースキー、[←] 、[Enter]、[Ctrl]B[の入力は有効
8105　3A－5_　ここでスペースキー、[←] 、[Enter]、[Ctrl]Bを入力すると？が出る

CP ［省略形］なし

［書式］CP　aaaa,bbbb,cccc

　指定アドレス範囲（aaaa～bbbb）のメモリの内容を、指定アドレス（cccc）からはじまるメモリの内容と1バイトずつ比較し、一致しない場合はそのアドレスとデータを表示します。一致した箇所は表示されません。（CPはcompareの略です）
　比較作業が完了すると、END＝bbbbと表示して、コマンドモードに戻ります。
　aaaa、bbbb，ccccは4桁の16進数で、ccccに制限はありませんが、aaaa≦bbbbでなければいけません。aaaa＞bbbbのときはHOW？メッセージが出ます。

CS ［省略形］なし

　指定範囲のメモリの内容を1バイトずつ加算して、そのトータル値を16進4桁で表示します（CS＝Check　Sum）。

［書式］CS　aaaa，bbbb

　aaaaは開始アドレス、bbbbは終了アドレスです。

DM ［省略形］D.

［書式］DM　aaaa,bbbb

　CMが1バイトずつの表示であったのに対し、DMコマンドは指定した範囲（aaaa～bbbb）のメモリ内容を1行16バイトずつ表示します。（DMはdump　memory　&　restoreの略です）
　aaaa、bbbbは4桁の16進数でaaaa≦bbbbでなければいけません。aaaa＞bbbbのときはHOW？メッセージが出ます。
　1行16バイトで表示する結果、最後の行が16バイトに満たなくなった場合でも、指定範囲を越えて16バイト分の表示が行われます。
　メモリ内容の表示はCMと同じように16進数2桁で行い、さらに各行の右側にその16進数を文字コード（JIS、ASCII）とみなして、対応する文字を一緒に表示します。（＄00～＄1F、＄80～＄9F、＄E0～＄FFのコードに対しては、ピリオド（　.　）が表示されます）

／EXIT ［省略形］なし

　ZBKシステムを終了します。ログファイルもクローズされます。
　ZBKシステムの終了は／EXITのほか［Ctrl］＋［C］入力でも行えますが、プリンタポート接続の場合、［Ctrl］＋［C］で終了するとログファイルが正しく保存されません。USB接続の場合には［Ctrl］＋［C］で終了してもログファイルは正しく保存されます。

JP ［省略形］J.

［書式］JP　aaaa

　指定する16進アドレス（aaaa）にジャンプします。（JPはjumpの略です）
　ユーザーが書いたマシン語のプログラムを実行する場合に使います。そのプログラムの先頭アドレスを指定

することによって、これ以後はCPUの制御はユーザープログラムに移ります。

●ユーザープログラムの終りの処理
ユーザープログラムの最後の命令は重要です。BASICプログラムなら終りっぱなしでも構いませんが、マシン語プログラムを終りっぱなしにすると大抵は暴走して止まらなくなったりハングUPしてしまいます。
ユーザープログラムの最後にはシステムのリエントリアドレスへのジャンプ命令を書いてください。リエントリアドレスは＄1033です。
JP ＄1033（コード C33310）と書いてください。
ユーザープログラムの最後にJP ＄1033があるとユーザープログラム終了後そのままシステムのリエントリプログラムが実行されます。そこではスタックの値やその他システムの処理を継続するのに必要なワークアドレスの値が再設定されるため、ユーザーは終りっぱなしに近い感覚でプログラムを終ることができます。

[参考]USR命令
マシン語のユーザープログラムを実行するには、このJPコマンドの他にBASICのUSR（＄aaaa）命令も使えます。
USR（＄aaaa）もコマンドモードで実行すれば、動作はこのJPコマンドと同じになります。
終わりの処理は、JPの場合には既に説明したように、＄1033へのジャンプ命令でなければいけませんが、このUSR命令の場合には、RET命令で終わることもできます（USR命令をBASICプログラム中で使う場合には、終わりは必ずRET命令でなければいけません）。
ただし終わりがRET命令の場合には、プログラム中でSPの値を不用意に変更してはいけません。PUSH、POPの使い方にも注意して、ユーザープログラムのスタート時のSPの値と、最後のRET命令の実行直前のSPの値が同じになるように注意する必要があります。

／LD ［省略形]なし

[書式]／LD ファイルネーム, aaaa

バイナリファイルをRAMの指定アドレスにロードします。ファイル名にはドライブ名やディレクトリ名も付加することができます。aaaaは4桁の16進数でLOAD先のRAMアドレスを示します。aaaaを省略することはできません。ファイルがみつからないときは FILE NOT FOUND と表示されます。
／SVでBASICプログラムとマシン語サブルーチンを一括して保存作成したファイルの場合にはロード後にBSSETコマンドでBASICプログラムをLIST表示可能な形にすることができます（使用例2）。

[使用例1]
＞／LD TEST__SUB. BIN, 4004[Enter]

[使用例2]
＞／LD A:￥BASIC￥TESTPRO. ROM, 4000[Enter]
＞BSSET[Enter]
TEXT 4500－6543
ﾍﾝｽｳ DBC9－DFFF

MV ［省略形]M.

[書式]MV aaaa,bbbb,cccc

指定アドレス範囲（aaaa～bbbb）のメモリの内容を、指定アドレス（cccc）からはじまるメモリエリアに転送します。（MVはmoveの略です）
この転送作業によってもとのメモリエリア（aaaa～bbbb）の内容は変化しません。
ただしもとのメモリエリアと転送先のメモリエリアに重なりがある場合には、重なった部分のメモリ内容は転送

後の内容に置き代わります（メモリエリアの重なり方に制限はありません。どういう重なり方でも、正しく転送されます）。
aaaa、bbbb，ccccは4桁の16進数で、ccccに制限はありませんが、aaaa≦bbbbでなければいけません。aaaa＞bbbbのときはHOW？メッセージが出ます。
転送先のメモリエリアがROMの場合でもエラーにはなりませんが、結果的には書き込みができないため、実行されなかったのと同じことになります。
元のメモリエリアはRAMでもROMでも構いません。

［注意］
このコマンドは、指定範囲のメモリ内容がプログラムであっても、ただのデータとしてそのまま転送します。
したがって例えばC000から実行される形で書かれたマシン語のプログラムをこのコマンドで他のアドレス（例えばB000）に移動させた場合、そのアドレスではそのプログラムは実行できないことに注意して下さい。
そのプログラムを他のアドレスで実行できるように移動したい場合には、一度逆アセンブラ（ZDAS．COM）にかけて、ソースプログラムを再生した後、アセンブラ（ZASM．COM）で、そのアドレスにオブジェクトプログラムを作るようにします（詳しくは9章を参照して下さい）。

R64／R128／R256 ［省略形］なし

開発セットについているROM　WRITERを実装して、ブルーの書き込み用ソケットに書込み済みのROMをセットして、このコマンドを実行するとROMからRAMのテキストエリア（＄4000～）にROMの内容がそのままCOPYされます。R64は27C64、R128は27C128、R256は27C256に対するコマンドです。
＞R64［Enter］ 27C64の内容が4000～5FFFに読み込まれる
＞R128［Enter］ 27C128の内容が4000～7FFFに読み込まれる
＞R256［Enter］ 27C256の内容が4000～BFFFに読み込まれる。ROMの前半16KBが8000～BFFFに、後半16KBが4000～7FFFに入る。

［注意］
書き込み用ソケットは通電状態でROMをセットするため、逆差しをするとその瞬間にROMが破損することになります。十分注意して作業してください。

R1M　B／R64　B／R128　B／R256　B ［省略形］なし

開発セットについているUSB／WRITER基板とZBKボードとを第5章で説明する方法で接続して、ブルーの書き込み用ソケットに、ユーザーのBASICプログラム書込み済みのROMをセットして、このコマンドを実行するとROMからRAMのテキストエリア（＄4000～）にユーザーのBASICプログラム（およびマシン語サブルーチン）がCOPYされ、実行、デバッグが可能になります。R1M　Bは27C010（27C1001）、R64　Bは27C64、R128　Bは27C128、R256　Bは27C256に対するコマンドです。

＞R1M　B［Enter］
TEXT　5000－9235　……BASICプログラムの格納されたエリア
ヘンスウ　C651－DFFF　……使用している変数エリア
（R64　B、R128　B、R256　Bの場合も実行結果の表示は同じです。）
［注記］
ROMにユーザーのBASICプログラムが正しく書かれていないと、HOW？と表示されます。
［注意］
書き込み用ソケットは通電状態でROMをセットするため、逆差しをするとその瞬間にROMが破損することになります。十分注意して作業してください。

RRB ［省略形］なし

ZB10K～ZB28K、ND80KのシステムROMにユーザーのBASICプログラムが書きこまれている場合にRRB[Enter]を実行すると、RAMのテキストエリア（＄4000～）にユーザーのBASICプログラム（およびマシン語サブルーチン）がCOPYされ、実行、デバッグが可能になります。
R1M B が書込済ROMをROM WRITERから読みこむのに対し、RRBは使用中のシステムROMからユーザーのBASICプログラム部分を読みこみます（この機能はUSB／WRITER基板のWRITER部分を必要としません。プリンタポート接続の場合にはUSB／WRITER基板を接続する必要はありません。USB接続の場合も5章のWRITERのための接続をする必要はありません）。

＞RRB[Enter]
TEXT 4004－478A ……BASICプログラムの格納されたエリア
ヘンスウ DEF7－DFFF ……使用している変数エリア

[注記]
システムROMにユーザーのBASICプログラムが正しく書かれていないと、HOW？と表示されます。

SD [省略形]なし

[書式1]SD aaaa, bbbb, hhhhhh……
[書式2]SD aaaa, bbbb, ”文字列”

指定アドレス範囲（aaaa～bbbb）のメモリをサーチし、指定データと同じ内容をみつけると、その前後の32バイトを表示します（SD＝Search Data）。
[書式1]は指定データを16進数で示したもので、データの組合せに制限はありません。
[書式2]は指定データを文字列で示します。したがってサーチできるデータは文字コードに限られます。

[使用例]
>SD 5000,57FF,C33310[Enter]

```
51C3  11 55 1F C3 CC 11 11 5B 1F CD 61 11 AF CD 18 10  .U.ﾃﾌ..[.ﾍ a.ｯﾍ..
51D3  C3 33 10 CD E0 11 7C FE FF CA BD 11 C9 21 00 00  ﾃ3.ﾍ..:..ﾊｽ.ﾉ!..

5289  00 DB 01 E6 50 C9 CD 86 12 C0 CD A2 15 CD 03 1A  .ﾛ..Pﾉﾍ .ﾀﾍ｢.ﾍ..
5299  C3 33 10 CD 6F 11 0E 06 1A 6F 13 1A 67 13 CD 5D  ﾃ3.ﾍo....o..g.ﾍ]
```

>SD 4000,4FFF,"ERR" [Enter]

```
4641  55 54 D2 45 41 44 D2 45 53 54 4F 52 45 CF 4E 20  UTﾒEADﾒESTOREﾏN
4651  45 52 52 4F 52 20 47 4F 54 4F D2 45 53 55 4D 45  ERROR GOTOﾒESUME
```

／SV [省略形]C.

[書式]／SV ファイルネーム, aaaa, bbbb

指定アドレス範囲（aaaa～bbbb）のマシン語プログラム（データ）をファイル名をつけてディスクに保存します。aaaa, bbbbは4桁の16進数でaaaa＜＝bbbbです。ファイル名はMSDOSの規則に従います。マシン語プログラムはZ80アセンブラによってバイナリファイルを作成することができますから、このコマンドを使うことは希です。BASICプログラムとマシン語サブルーチンを一括して保存する目的で使うと効果的です（使用例2）。その場合には／SVコマンドにさきだってBROMコマンドの実行が必要です。[使用例2]ではユーザープログラムのROMイメージを保存することと同じです。この場合も拡張子は任意につけられます。

[使用例1]
＞／SV TEST__SUB. BIN, 4004, 4365[Enter]

[使用例2]
＞BROM[Enter]
4500－5732
＞／SV TESTPRO. ROM, 4000, 5732[Enter]

[注記]
ファイル名は名前部が8桁以内の英数字および－＿で拡張子は3桁以内です。拡張子は任意です。付けなくても構いません。ファイル名にはドライブ名やディレクトリ名も付加することができます。
例: ／SV A:￥BASIC￥TESTPRO. ROM, 4000, 7FFF
[注意1]
使用例2ではaaaaは必ず4000にします。先にBROMを実行しておなかいと、／LD作業でBASICプログラムを再生させることができません。bbbbはBROMの実行により表示される2番目のアドレス値にします。使用例でaaaa, bbbbは4500, 5732ではなくて4000, 5732になる点に注意してください
[注意2]
使用例2で保存したファイルにはBASICプログラムも含まれていますがバイナリイメージで保存しているため、Notepadなどのテキストエディタで参照することはできません。

W1M B[省略形]なし

ZB10K～ZB28K、ND80KのシステムROM（27C010）にユーザー用BASICプログラムを書き込みます。第5章および第6章参照。

＞W1M B[Enter]
と入力することでRAMのBASICプログラム（およびマシン語サブルーチン）がROMに書込まれます。書込み中は＊が左から右に表示されていきます。＊1個が256バイト書込み済みを示します。
書き込み中にエラーが発生すると書き込みを打ち切り、そのときのアドレスとその後ろにもとのデータとそれに対するエラーデータを表示したあと、その下にERR, Wと表示してコマンドモードに戻ります。

W1M S[省略形]なし

ZB10K～ZB28K、ND80KのシステムROM（27C010）の複製を作ります。ユーザーのプログラムはコピーされません。ユーザーのプログラム部分は前述の W1M Bコマンドを使います。第5章および第6章参照。

＞W1M S[Enter]
と入力することでシステムプログラムがROMに書込まれます。書込み中は＊が左から右に表示されていきます。＊1個が256バイト書込み済みを示します。
書き込み中にエラーが発生すると書き込みを打ち切り、そのときのアドレスとその後ろにもとのデータとそれに対するエラーデータを表示したあと、その下にERR, Wと表示してコマンドモードに戻ります。

W256 [省略形]なし

[書式]W256 aaaa,bbbb

27C256用の書き込みコマンドです。
aaaa、bbbbは4桁の16進数でそれぞれもとになるメモリの、書き込みを開始したいアドレスと、書き込みを終了したいアドレスを示します。
＄4000～＄7FFFが書き込み用ROMの後半16kBに対応し、＄8000～＄BFFFが前半16kBに対応し

ます（5章ROM　WRITER　3．1　もとになるデータがROMの場合［注意3］参照）。しかしコマンドパラメータとしては必ずaaaa＜＝bbbbです。
　W256　4000，BFFF　（正）
　W256　8000，7FFF　（誤）

　W256　4000，BFFF[Enter]と入力すると、＊が左から右に表示されていきます。＊1個が256バイト書込み済みを示します。27C256は32KBですから4000～BFFFまで書く場合には＊が128個表示されることになります。
　書き込み中にエラーが発生すると書き込みを打ち切り、そのときのアドレスとその後ろにもとのデータとそれに対するエラーデータを表示したあと、その下にERR，Wと表示してコマンドモードに戻ります。

［注記］
　部分的に書き込むことも可能です。
　ROMの先頭から書き込まなくても、途中からでも書き込みを開始できます。 たとえばW256　4000,435Fとか、W256　5300,56EFというような指定をしても構いません。
　途中のアドレスを指定した場合は、ROMの先頭から書き込まれるのではなく、対応するアドレス位置から書き込みが開始されます。

W256　B［省略形］なし

　ZB11V3、ZB11V2用のBASICプログラムROM作成コマンドです。
＞W256　B[Enter]と入力することで、RAMのBASICプログラム（およびマシン語サブルーチン）が27C256に書込まれます。

XD　［省略形］なし

［書式1］XD　aaaa，bbbb，nnnn……，mmmm……

　指定アドレス範囲（aaaa～bbbb）のメモリをサーチし、指定データ（nnnn……）と同じ内容をみつけるとその前後の32バイトを表示したのち、mmmm……で置きかえます（XD＝Exchange　Data）。
　nnnn……，mmmm……は16進2桁を単位として複数バイト記述できます。nnnn……，mmmm……は同じ長さでなければいけません。

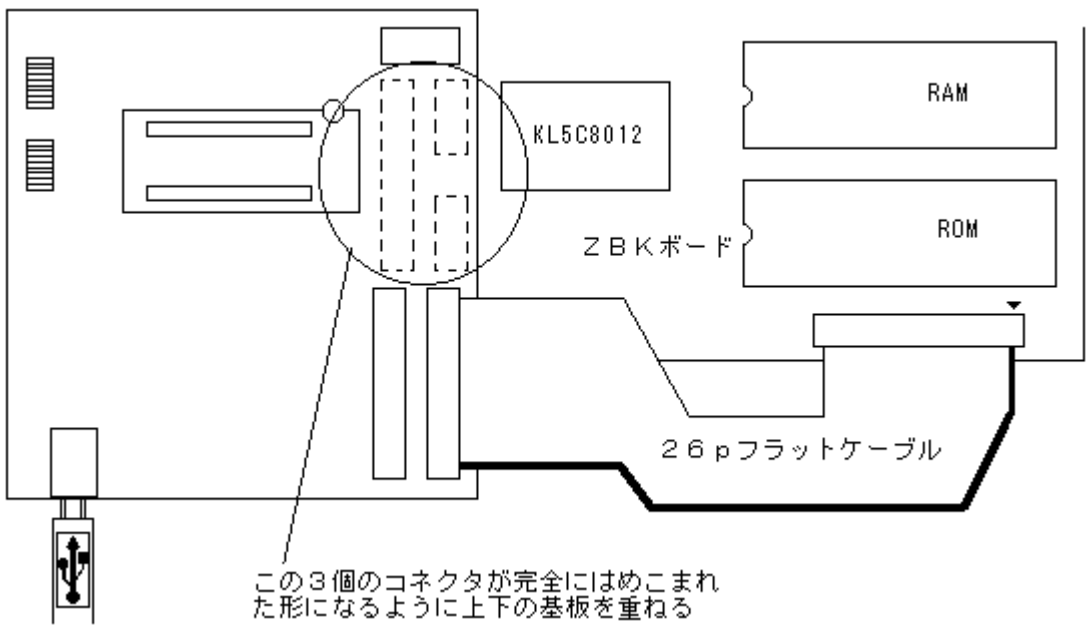
［使用例］
```
>XD 5000,57FF,C33310,C33010[Enter]

51C3  11 55 1F C3 CC 11 11 5B 1F CD 61 11 AF CD 18 10  .U.ﾃﾌ..[.ﾍ a.ｯﾍ..
51D3  C3 33 10 CD E0 11 7C FE FF CA BD 11 C9 21 00 00  ﾃ3.ﾍ..:..ﾊｽ.ﾉ!..

5289  00 DB 01 E6 50 C9 CD 86 12 C0 CD A2 15 CD 03 1A  .ﾛ..Pﾉﾍ .ﾀﾍ｢.ﾍ..
5299  C3 33 10 CD 6F 11 0E 06 1A 6F 13 1A 67 13 CD 5D  ﾃ3.ﾍo....o..g.ﾍ]

>SD 5000,57FF, C33010[Enter]

51C3  11 55 1F C3 CC 11 11 5B 1F CD 61 11 AF CD 18 10  .U.ﾃﾌ..[.ﾍ a.ｯﾍ..
51D3  C3 30 10 CD E0 11 7C FE FF CA BD 11 C9 21 00 00  ﾃ3.ﾍ..:..ﾊｽ.ﾉ!..

5289  00 DB 01 E6 50 C9 CD 86 12 C0 CD A2 15 CD 03 1A  .ﾛ..Pﾉﾍ .ﾀﾍ｢.ﾍ..
5299  C3 30 10 CD 6F 11 0E 06 1A 6F 13 1A 67 13 CD 5D  ﾃ3.ﾍo....o..g.ﾍ]
```

ZBK開発セットには紫外線消去型ROMの27C256／27C010用のWRITERボードが含まれていて簡単なコマンドで高速書き込みができるようになっています。
ROMにデータを書き込むためには、＋5Vのほかに＋12．5Vの電圧が必要です。また高速書き込みをするには、ROMのVCC を＋6Vにしなければなりません。
ZBK開発セットのROM　WRITERはそれらの電圧をすべてボード内のDC／DC回路によって、＋5Vから作り出しているため他の電圧は必要ありません。

この章では、ROMへの書き込み方法について説明します。
なおBASICプログラムをROM化する場合や、すでにROM化したBASICプログラムを修正する場合には次の6章も参照して下さい。

1．　ROM　WRITERの接続

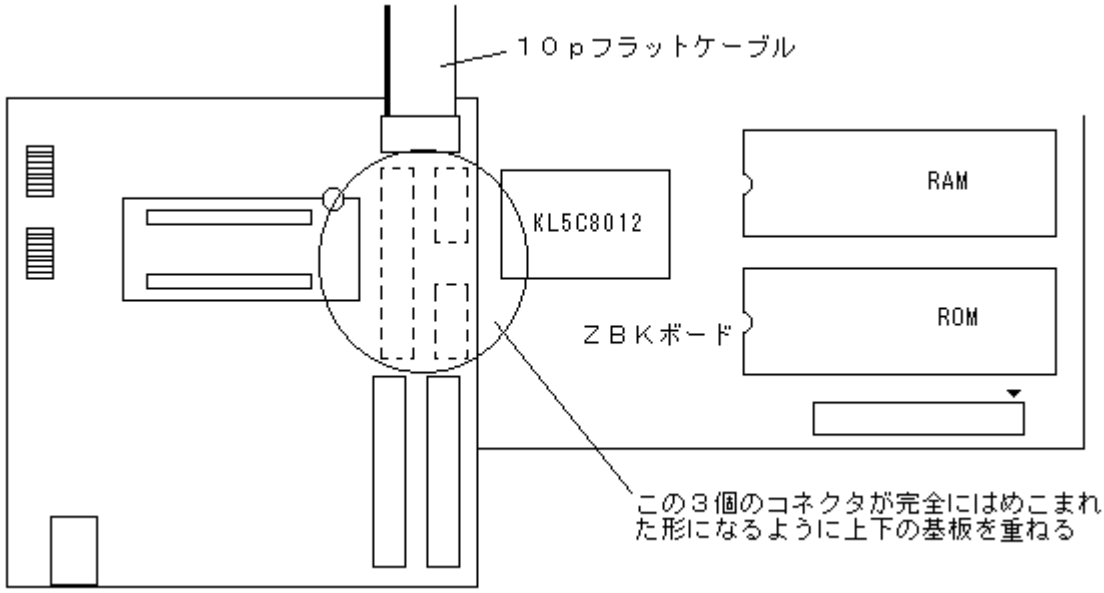
［USB接続の場合］
ZBKシステムを終了し、ZBKボードの電源をOFFにした状態で接続作業をします。
ROMWRITERの機能を使う時にはLCD表示器は接続できません。LCD表示器は外してください。
ZB10K～ZB28K、ND80Kの3個のコネクタとUSB／WRITER基板の裏側のコネクタが合うようにUSB／WRITER基板を上にして重ねます。USBケーブル、26pフラットケーブルもつないだままにします。



この状態にしてから、ZBKボードの電源をONにしてZBKシステムを起動します。起動の方法は今までの説明と同じです。この状態でもプログラム開発操作などは通常と同じように行えますが、LCD表示、232C機能は使うことができません。

［プリンタポート接続の場合］
プリンタポート接続ではZBKボードに直接10pフラットケーブルコネクタを接続して、DOS／Vパソコンにプリンタポート接続をして使いますが、ROMWRITERを使うときは、USB接続と同じようにUSB／WRITER基板とZBKボードとを重ねる形で接続する必要があります。
ZBKシステムを終了し、ZBKボードの電源をOFFにした状態で接続作業をします。
ROMWRITERの機能を使う時にはLCD表示器は接続できません。LCD表示器は外してください。

[注意]USBコネクタ、26pフラットケーブルは接続しません。
プリンタケーブル・10pケーブル変換基板につないで使っていた10pフラットケーブルをZBKボードから外してそこに図のようにUSB／WRITER基板を重ねます。そうしておいて、今外した10pケーブルを図のようにUSB／WRITER基板の10pコネクタに接続します。



この状態にしてから、ZBKシステムを起動します。起動の方法は今までの説明と同じです。この状態でもプログラム開発操作などは通常と同じように行えますが、LCD表示、232C機能は使うことができません。

2. ROM書き込みの種類

ROMの書き込みは基本的にはデータの複製（コピー）を作る作業です。
もとになるデータが何か、によって前準備の作業が異なってきます。ROMの書き込み作業には、次の種類が考えられます。
①すでにあるROMと全く同じ複製が欲しい。
②すでにあるROMのデータを違う種類のROMにコピーしたい。
③すでにあるROMの内容を部分的に修正して、新しいROMにしたい。
④RAM上で新たに作成したデータ、プログラムをROM化したい。
以上の内③では②の作業を伴うことも考えられます。
またいずれの場合にも、ROMのメモリ容量一杯に書き込む場合と、その一部にしか書き込まない場合とが考えられます。
さらに、一般的には書き込む対象となるROMは新品か或いは紫外線消去器によって消去済みのROMを使用しますが、場合によっては部分的に書き込み済みになっているROMの、未使用部分に追加書き込みしたいこともあります。
このシステムでは①は27C256に対してのみ可能です。27C64、27C128は27C256にコピーすることはできますが、27C64、27C128に書込むことはできません。②③についても同じことになります。
④についても27C256に書込んで新規作成することのみ可能です（BASICプログラムROMに限って読出し、書込みともに27C256と27C010を扱うことができます）。
上記の作業は整理すると結局次の2種類にまとめることができます。
（1）ROMからROMへのコピー
（2）RAMからROMへのコピー
ZBK開発システムでは（1）（2）を同じ手続きで行います。（1）の場合にはもとになるROMからRAMにCOPYして次にそのCOPYしたRAMの内容をROMに書き込みます。必要なら書込む前にデータの修正や追加を行います。（2）はROM→RAMへのCOPYを行わないだけで、それ以後は同じ作業（RAM→ROM書込み）にな

ります。

3.　もとになるデータの準備

3．1　もとになるデータがROMの場合

　すでにあるROM（27C64、27C128、27C256）をもとにして、そのコピーをとる場合には、まずそのもとになるROMをROM　WRITERのブルーの書込み用ソケットにセットします。電源は入ったまま作業しますから、逆にセットしたり、27C64～27C256の場合には差し込む位置を間違えないように注意してください（「1.　ROM　WRITERの接続」図参照）。

［注意1］
　逆差しをしたり位置を間違えるとROMが破損してしまうことがあります。最悪の場合にはZB10K～ZB28K、ND80Kが破損することもありますから十分注意して作業してください。

　ROMが27C64の時は、**R64**[Enter]、27C128なら**R128**[Enter]、27C256は**R256**[Enter]と入力します。ROMのデータがRAMの＄4000～＄5FFF（27C64の場合。271C28の場合には＄4000～＄7FFF、27C256は＄4000～＄BFFF）に転送されます（この時、トータルチェックが行われその値が表示されます）。
　＞R64［Enter］　4000～5FFFに読み込まれる
　＞R128［Enter］　4000～7FFFに読み込まれる
　＞R256［Enter］　4000～BFFFに読み込まれる。ROMの前半16KBが8000～BFFFに、後半16KBが4000～7FFFに入る。

　もとになるROMが27C64または27C128でそれを27C256にコピーする場合には［注意2］に示す理由から、アドレスの移動が必要です。27C64はR64コマンドのあと
　MV　4000，5FFF，8000［Enter］
を実行して、4000～5FFFのデータを8000～9FFFにコピーしてください。27C128はR128コマンドのあと
　MV　4000，7FFF，8000［Enter］
を実行して、4000～7FFFのデータを8000～BFFFにコピーしてください。

［注意2］
　もとになるROMの内容がBASICプログラムの場合には、ここで説明した方法では内容の変更はできません。次の6章を参照して下さい。またもとになるROMがZBKシステムで開発した27C010（27C1001）の場合も6章を参照してください。
　もとになるROMが27C64～27C256の場合には他社システムで書込まれたデータROMであってもコピーしたり16進コードでの編集作業をすることもできますが27C010（27C1001）の場合には他社システムで書込まれたデータROMを正しく読み込むことはできません。
［注意3］
　もとになるROMが27C256の場合にはROMの前半16kBが、＄8000～＄BFFFに入り、後半の16kBが＄4000～＄7FFFに入ります。前半と後半が逆になることに注意してください。これはユーザー用RAMエリアが＄8000～＄FFFFではなくて＄4000～＄BFFFと16KBずれていることに起因します。27C256はROM単独では0000～7FFFの範囲のアドレスでアクセスされます。上位8ビットをビット表現で示せばX0000000～X1111111です（ビット7は0でも1でもよい）。RAMエリアの4000～7FFF、8000～BFFFの上位8ビットを同じ表現で示すと、01000000～0111111、10000000～10111111になります。ここでRAMのビット7を無視してROMのアドレスと比較すると、前半と後半が入れ替わっていることになります。

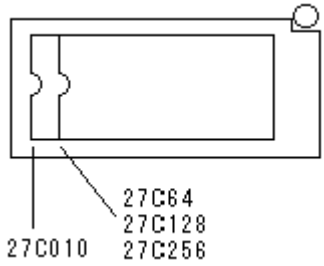
3．2　もとになるデータがRAMの場合

　3．1と基本的には同じです。
　書込みの対象は27C256ですから、ROMの前半16kBが、＄8000～＄BFFFに対応し、後半の16kBが

＄4000～＄7FFFに対応します（3．1[注意3]）。
27C256の先頭に入れるべきデータは＄4000ではなくて＄8000におかなければなりません。その次のデータは＄8001、……で＄BFFFまで進んだあと、次は＄4000に行って、＄4001、……と続き最後のデータは＄7FFFにおくことになります。

4． ROMのセット

書き込み用のソケット（ブルーのレバー付ソケット）に消去済みのROMをセットします。
レバーを上げてROMをしっかりセットしてから、レバーを下ろしてROMを固定します。
ROMの向きを間違えないように注意して下さい。逆差しをしたり位置を間違えるとROMが破損してしまうことがあります。最悪の場合にはZB10K～ZB28K、ND80Kが破損することもありますから十分注意して作業してください（ROMの切り欠きを左にします。）。



5． 書き込みコマンド

[注記]ZBKシステムでは27C256にプログラムやデータを書込むことはありません。ここで説明するコマンドは旧ZBシステムのためのものです。新ZBKシステムのためのROM作成作業については次の第6章を参照してください。

W256 ［省略形]なし

[書式]W256 aaaa,bbbb

27C256用の書き込みコマンドです。
aaaa、bbbbは4桁の16進数でそれぞれもとになるメモリの、書き込みを開始したいアドレスと、書き込みを終了したいアドレスを示します。
アドレスについては、すでに説明したように＄4000～＄7FFFが書き込み用ROMの後半16kBに対応し、＄8000～＄BFFFが前半16kBに対応します。しかしコマンドパラメータとしては必ずaaaa＜＝bbbbです。
W256 4000，BFFF （正）
W256 8000，7FFF （誤）

W256 4000，BFFF[Enter]と入力すると、＊が左から右に表示されていきます。＊1個が256バイト書込み済みを示します。27C256は32KBですから4000～BFFFまで書く場合には＊が128個表示されることになります。
書き込み中にエラーが発生すると書き込みを打ち切り、そのときのアドレスとその後ろにもとのデータとそれに対するエラーデータを表示したあと、その下にERR，Wと表示してコマンドモードに戻ります。

[注記]
部分的に書き込むことも可能です。
ROMの先頭から書き込まなくても、途中からでも書き込みを開始できます。 たとえばW256 4000,435Fとか、W256 5300,56EFというような指定をしても構いません。

途中のアドレスを指定した場合は、ROMの先頭から書き込まれるのではなく、対応するアドレス位置から書き込みが開始されます。

ZB10K～ZB28K、ND80Kはユーザーが書いたBASICのプログラムをROM化して、パワーONで実行させることができます。
一般にRAM上でBASICプログラムを実行させる場合には、パワーONしてすぐに実行させるわけにはいきません。まずRAMにBASICプログラムをハードディスクからLOADしなければなりません。LOADが完了してから、RUN[Enter]と入力してはじめて実行を開始します。
BASICプログラムをROM化した場合はDOS／Vパソコンと接続するコネクタにショートピンをつけることで、パワーON後ただちにユーザープログラムの実行を開始します。
ショートピンを外してDOS／Vパソコンと接続して立ち上げれば、通常の開発用画面になります。RRBコマンドの入力により、ユーザープログラムをROMからRAMにCOPYしてデバッグすることができます。
RAM上に作成したBASICプログラムをROMに書き込むためには、通常のデータをROMに書き込むのと違い特別の制御データの作成が必要ですが、そのためのコマンドが用意されているので面倒な手続きは必要ありません。説明通りに作業すれば簡単にBASICのROMができあがります。
BASICプログラムをROMに書き込むには、W256　BまたはW1M　Bを使います。

1．　ZB10K～ZB28K、ND80Kのためのプログラム ROM作成

ZB10K～ZB28K、ND80Kは1個のROM（27C010）にシステムプログラムとユーザープログラムを書き込みます。そのためシステムとユーザープログラムの2回に分けて書込みを行います

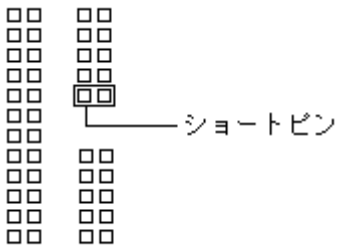
①ブルーの書き込みソケットに消去済の27C010（27C1001）をセットします。
②システムプログラムのコピー。。
W1M　S[Enter]と入力します。
③次にユーザープログラムをLOADします。念のためLISTコマンドで確認します。
④ユーザープログラムを書込みます。
W1M　B[Enter]と入力します。
②④ともに書き込み中は256バイト毎に＊が表示されます。
②と③④の順序を逆にしても構いません。

[注意1]
ROMで実行するプログラムにINPUT、INPUT＄、INKEY＄があるとそこでハングUPしてしまいます。ROM化の前にそれらの文は　’　をつけて注釈行にしておいてください。INPUT＄はキー入力に対する使用は不可ですが232C入力はできます。
ZB25K、ZB27Kはキー入力の可能なインターフェースを内臓しているため、これらの命令をROMでも実行することができます。
なおPRINT文はそのままでも構いません。ROM化した場合LCD表示器を接続するとPRINT文はLCDに対して行われます。LCD表示器が接続されていない場合PRINT文は無視されます。ROM化したプログラムでLCD表示を行うためにPRINT文を使う場合にはPRINT文の終わりを；（セミコロン）にする必要があります。

ROMでの実行時にはシステムによって無視されます。
[注意2]
プログラムサイズによってはROM／RAMエリアの設定が必要になってきます。[6．　ROM／RAMエリアの設定]を参照してください。
[注意3]
書き込みを終わったROMは電源が入ったままの状態でソケットから外して、透明な窓部分の上に不透明なシールを貼って、その上にアドレスや日付などを書いておきます（シールを貼らなくても直射日光に長時間さらしたりしなければ、蛍光灯の紫外線くらいでは、データが消えたりするものではありませんが念のためにこうしておいたほうが安全です）。

2．　プログラムROMの実装と実行

　／EXIT［Enter］を入力してZBK－V3BASICを終了します。電源を切って、DOS／Vとの接続ケーブルを外します。ROM　WRITERも取り外します。
　次にシステムROMを外して、代わりにいま書込み終了したプログラムROMを取り付けます。ユーザープログラムが追加されていますが、システムプログラム部分は今まで実装されていたROMと同じです。
　図のように10Pコネクタの2－4PINをショートするとROMプログラム起動モードになります。



（2005年4月出荷分から図のように変更しました）

3．　マシン語サブルーチンを使用している場合

　いままでの説明はBASICプログラムだけの場合についてでしたが、マシン語のサブルーチンを使っている場合でも同じようにROM化することができます。
　RAM上でプログラムを実行させる場合にはマシン語サブルーチンも適当な空きエリアに書き込んでおいてそれをコールすればよいのですが、ROM化する場合には、BASICプログラムと共にマシン語のプログラムもROMに書かなくてはいけないので、メモリ上の何処にマシン語のプログラムを置くかということがかなり重要になってきます。
　マシン語のプログラムは必ず＄4004番地から格納するようにします（アセンブラソースプログラムの先頭をORG　＄4004にする）。Z80アセンブラによってバイナリファイルを作成し、／LDコマンドで4004番地にロードします。
　BASICプログラムはそのマシン語プログラムの後でアドレスが重ならないところにロードします。マシン語プログラムが4004～4735番地だったとすれば4736番地以後ならよいわけですが、プログラムの修正や変更も考えられますから少しは余裕があった方がよいでしょう。4800番地とか5000番地からBASICプログラムをロードします。

［注意］
　＄4000～＄4003は使用してはいけません。この4バイトにはシステムの情報が入れられます。

　このあとは　1．　ZB10K～ZB28K、ND80Kのためのプログラム ROM作成　および　2．　プログラムROMの実装と実行　の説明と同じです。

［参考］
　W1M　Bコマンドはユーザーの BASICプログラムの開始アドレスと終了アドレスを4000～4003番地に書込んだあと、4000番地からユーザーの BASICプログラムの終了アドレスまでのRAMの内容をROMにCOPYします。マシン語プログラムを4004番地から書いて、BASICプログラムをその後ろに置くことで、マシン語プログラムとBASICプログラムをともにROM化することができます。

4．　マシン語プログラムだけの場合

　この場合も上の3．と同じようにマシン語プログラムをサブルーチンとみなします。そしてBASICプログラムは、USR文を1行だけ書きます。

BASICプログラムは次の1行だけ書きます。

10 USR(＄4004)

マシン語プログラムは＄4004から実行する形で書き、BASICプログラム(上の1行のプログラム)はマシン語プログラムの後ろのエリアに書くことも3. で説明した通りです。その後の書き込み作業も全て3. で説明した通りに行います。

5. BASICプログラムROMの読み込み

BASICプログラムを書き込んだシステムROMを実装して、DOS／Vパソコンを接続して、ROMのBASICプログラムを修正したい場合があります。
ZBK－V3BASICにエントリした状態で

＞RRB[Enter]

と入力します。ROMからRAMにプログラムがCOPYされ、次のように表示されます。LISTコマンドを入力してリストが表示されることを確認してください。

TEXT 5000－7365
ﾍﾝｽｳ D35B－DFFF

6. ROM／RAMエリアの設定

ROMのBASICプログラムが実行されるとき、初期設定ではユーザーのBASICプログラム(ROM)はアドレス4000～7FFFに限定され、RAMエリアは8000～FFFFになります。
BASICプログラムは4004から後ろに書き込まれて行き、変数データはDFFFから前に割り付けられて行きます。開発セットを接続しているときは4000～FFFFは全てRAMになっていますからユーザープログラムと変数エリアがぶつからない限りは正しく実行されます。しかしプログラムをROM化するとROM／RAMアドレスが重要になってきます。ユーザープログラムが7FFFまでで終り、また変数エリアが8000より前にあればROM化しても初期設定のままでできますから何も意識する必要はありません。
プログラムと変数エリアの情報はHELPコマンドで得られます。
＞HELP[Enter]
TEXT 4004－5736 ……プログラムの入っているエリア
ﾍﾝｽｳ B753－DFFF ……変数の割り付けられているエリア
のように表示されます。TEXTの終りの値が8000を超えているか、またはヘンスウの1番目の値が8000より小さいときはROM化の前にROM／RAMエリアの設定が必要です。
BASICプログラムの一番前に次のOUT文を書き加えてください。

10 OUT 4, hh (hhは＄10～＄37の範囲の2桁の16進数)

この命令によりROM／RAMの境が移動します。初期値は＄1Fで境は8000番地になります。値と境のアドレスは下記の関係になっています。

＄10	4400
＄11	4800
＄12	4C00
＄13	5000
＄14	5400
＄15	5800
＄16	5C00

＄17　6000
＄18　6400
＄19　6800
＄1A　6C00
＄1B　7000
＄1C　7400
＄1D　7800
＄1E　7C00
＄1F　8000
＄20　8400
＄21　8800
＄22　8C00
＄23　9000
＄24　9400
＄25　9800
＄26　9C00
＄27　A000
＄28　A400
＄29　A800
＄2A　AC00
＄2B　B000
＄2C　B400
＄2D　B800
＄2E　BC00
＄2F　C000
＄30　C400
＄31　C800
＄32　CC00
＄33　D000
＄34　D400
＄35　D800
＄36　DC00
＄37　E000

hh＝＄37のときはユーザーが定義する変数が無い場合です。A%～Z%、A＄～H＄以外の変数を使用していないときはヘンスウの値がDFFF－DFFFになって変数エリアが0になり、このときユーザープログラムは最大の4000～DFFFまでを使用することができます。

上記のようにアドレスは1KB単位で設定します。たとえばユーザープログラムが4004～A765のとき、ROMにすべきアドレスはA7FFまでですから、境界アドレスはA800を選択し、hh＝＄29にします。

［注記］

RAM上でのデバッグ時にこのOUT命令を実行しても無意味です。実行に影響はありません。

［注意］

ZB10K～ZB28K、ND80Kは標準では32KBのRAM62256を実装して出荷しています。この場合境界アドレスは8000～E000が有効です（hh＝＄1F～＄37）。RAMエリアをもっと大きくしたいときはRAMを628128にしてください。

7．ZB11V2、ZB11V3シリーズ用プログラムのROM化

ZBK－V3BASICはZB11V3のBASICと互換性があり、ZBK－V3BASIC上でZB11V3BASICのROM作成や修正を行うことができます。

ZB11V3上でおこなってきたROMの作成手順と全く同じです。

ZB11V3（またはZB11V2）によって作成されたBASICプログラムROMはROM　WRITERのブルーの書込み用ソケットにセットした上で、27C64の場合はR64　Bコマンドで RAMに読み込むことができます。27C128はR128　Bコマンドで、また27C256はR256　BコマンドでRAMに読み込むことができます。

＞R64　B［Enter］　27C64に書かれたBASICプログラムがRAMに読み込まれる
＞R128　B［Enter］ 27C128に書かれたBASICプログラムがRAMに読み込まれる
＞R256　B［Enter］ 27C256に書かれたBASICプログラムがRAMに読み込まれる

各コマンドでRAMに読み込まれたプログラムはLISTコマンドで表示することや修正、変更することが自由にできます。また／SAVEコマンドでディスクにテキストファイルとして保存することもできます。
ZB11V3、ZB11V2用（ZBシリーズ用）にBASICプログラムROMを書込み、作成できますが、27C256に限られます。27C64、27C128に書込むことはできません。
消去済みの27C256をROM　WRITERのブルーの書込み用ソケットに実装して
＞W256　B［Enter］
と入力することで、ROM作成が行われます。

マシン語の短いプログラムを書くのにはCMコマンドが便利です。
しかしそれにしてもマシン語の命令を16進コードで書くのは根気の要る作業です。
ジャンプ命令がC3でOUTがD3、とこのくらいはすぐに覚えられますが、LD命令になると種類が多くてとても覚えられません。一つ一つの命令毎に命令表を引きながら16進コードに置き換えるのは、面倒で大変根気の要る作業です。
この章で説明するラインアセンブラの機能は、その退屈な置き換え作業を一気にゼロにしてしまいます。一度使い出したら、手放せない有り難い機能です。
ニーモニックコードをキーから入力すると、ただちに16進コードに置き換えて、メモリに書き込んでくれます。

1．ラインアセンブラの起動

下のようにキー入力します。

＞AS　aaaa[Enter]

aaaaはマシン語プログラムの先頭のメモリアドレスで4桁の16進数です。

[使用例]
>AS 8000[Enter]　　……このように入力すると
8000 __　　……アドレスに続いてカーソルが表示される。そこで
8000 LD A,($8100)[Enter]　　……と入力すると
8000 3A0081　　LD A,($8100)　　……すぐに16進コードが表示されて次のアドレスとカーソルが表示される。
8003 __
8003 LD B,A[Enter]　　……続けて次の命令を入力すると
8003 47　　LD B,A　　……16進コードが表示されて次のアドレスとカーソルが表示される。
8004 __

処理を終了（中止）したいときは[Ctrl]Bを入力します。
この機能はニーモニックコードを16進コードに翻訳し画面に表示すると同時にそのメモリアドレスに16進コードを書き込む作業も行っています。
したがってどの時点で処理を中止しても、それ以前に翻訳された部分は16進コードでメモリに書き込まれています。
本当に書き込まれているかどうか、CMコマンドかDMコマンドで確認してみて下さい。
なおニーモニックコードを入力している途中で、キーの押し間違いに気がついたら、[Enter]を押す前ならば、[←]　[→] [INST][DEL] などの機能を使って、書き直すことができます。
[Enter]を入れてしまったあとでは、取り消しはできません。
済んでしまったアドレスに戻ってやりなおしたい場合には、[Ctrl]Bを入力して作業を打ち切ってから、もう一度希望するアドレスを指定してASコマンドを入力します。
またORG命令を書くことでも、アドレスを指定しなおすことができます。

8050 ORG $8035[Enter]　　……このように入力すると
8050　　ORG $8035
8035 __　　……次のアドレスが変更されて表示される。

2．ラベル・変数の使用

このライン・アセンブラは、疑似命令の機能なども一定の条件付で使用することができます。

ラベル・変数も使用することは可能ですが、ライン・アセンブラの性質上、その値がそれ以前に確定している場合に限ります。

たとえば次のようなプログラムを考えます。

```
        LD B,A
LOOP1:INC C
        JP LOOP1
```

ここで2行目のINC命令の前に書いてあるLOOP1というのがラベルです。その下のJP命令のジャンプ先を示しています。この例ではJP命令でそのジャンプ先としてLOOP1というラベルを使用した時点では、そのアドレスはすでに確定しています。

というのは、LOOP1:INC C という行を入力した時点で、システムによってその時のアドレスがLOOP1として記憶されるからです。

したがってそれからあとで、JP LOOP1 という行を入力したとき、システムによって、このLOOP1として記憶されていたアドレスが読み出されて使用されます。

しかし、もしジャンプ先がそのJP命令よりもさらに先にあった場合に、そのジャンプ先がラベルで表現されていたとすると、このJP命令を翻訳する時点では、まだそのジャンプ先のラベルはどのアドレスかわかりませんから、エラーになります。

したがってこの場合には、直接16進数で入力するしか方法はありません。

なおこれはあくまでライン・アセンブラに限っての制約で、ZBK開発セットに添付されているZ80アセンブラ(ZASM. COM)ではラベルがJP命令の後にあっても全く支障ありません。

同様に、このアセンブラでは、LD HL,DATA1 というように16進4桁の数値の代わりに変数を用いることができますが、この場合にもライン・アセンブラに限り、それ以前に変数の値が確定している必要があります。

変数の値を確定するには、＝(イコール)を使います(9章参照)。

[注意1]

ラベルも変数も、使用する以前に確定していなければなりませんが、それは一続きのASコマンドの作業内に限ります。

例えば一度[Ctrl]Bで処理を打ち切ると、それ以前に定義されていたラベルや変数の値はクリアされてしまうため、再びASコマンドで作業を再開しても、前のASコマンドの作業中に定義したラベルや変数を利用することはできません。

[注意2]

ラインアセンブラの作業エリアとしてB000～のメモリを使用します。B000より大きいアドレスを指定してASコマンドを実行すると、ニーモニック入力時に、SORRY と表示して処理が中止されます。

[注意3]

BASICプログラムの書かれているメモリアドレスにマシン語のプログラムを書くと、BASICプログラムは正しく実行されなくなりますから、アドレスには注意が必要です。

7章では、ラインアセンブラについて説明しました。
マシン語のプログラムを作成するのに、アセンブラの機能がとても便利だということが充分理解できたと思います。
ところで時にはプログラムを作るのではなくて、「読む」ことも必要になる場合があります。
例えば、以前に作成したプログラムだが、ROMは残っているが、ノートがどこかへ行ってしまって、どういうプログラムだったかもう一度調べなければならない、というような場合です。
また時には他人の作ったプログラム（16進コードのリストかROMしか無い）を解析しなければならない、という場合もあると思います。
すでに説明したCMコマンドやDMコマンドは、ROMまたはRAMの中味を読むのに便利な機能です。
しかし16進コードで表示されるため、そのコードを命令説明書をもとに逆にニーモニックに翻訳していかなければ、何が書いてあるか理解することができません。
そんな時このライン逆アセンブラを使えば、一命令毎にニーモニックコードに逆翻訳してくれます（逆アセンブラという名前は、アセンブラの逆の働きをするところから来ています）。
この機能も使ってみると、便利なことがよくわかります。

1．　ライン逆アセンブラの使い方

下のようにキー入力します。

＞DA　aaaa[Enter]

aaaaは読み出したいメモリの先頭アドレスで、4桁の16進数です。

[使用例]

```
>DA 8000[Enter]                    ……このように入力すると
8000 3A0081      LD A,($8100)  ……すぐに命令が翻訳されニーモニックが表示されます。ここで適当なキーを
                                        押すと
8003 47          LD B,A          ……次の命令が翻訳されて表示されます
```

何かキーを押す度に次の命令が16進コードと共に表示されて行きます。
実行を中止したいときは[Ctrl]B を押します。

[注意]
この機能は、指定するメモリアドレスの内容がZ－80のマシン語プログラムである、という前提で翻訳作業を行います。したがって命令以外のデータなどを読んだ場合にも、もしその16進数に該当する命令コードがあれば、そのニーモニックに変換して表示を行います。
また2バイト以上の長さの命令の場合には、その第一バイトを指定して、逆アセンブラを開始しないと、別の命令と解釈して誤訳してしまいます。 例えば上の例で、DA　8001[Enter]と入力して、スタートしたらどうなるでしょうか？下にその場合の結果を示します。

```
>DA 8001[Enter]
8001 00          NOP
8002 81          AD A,C
8003 47          LD B,A                ……ここでやっと正しくなりました
```

逆アセンブラはオールマイティ（万能）ではない、ということをよく認識して使用して下さい。
なお、命令コードとしては存在しない16進数にぶつかったときは、ニーモニックを表示する代わりに、？が表示されます。

マシン語プログラムを書く場合に、短いプログラムならCMコマンドでも作業できますが少し長いプログラムになると書くだけでも大変な作業になります。
一度で完全なプログラムができる事は希なので、途中何回も変更がでてきます。こうなるとマシン語プログラミングは、更に大変になります。
簡単に操作できるアセンブラとして7章でラインアセンブラの説明をしました。
短いマシン語のプログラムを即席で作るのなら、ラインアセンブラほど便利な機能は他にはありません。
しかしマシン語のプログラムも長いものになってくると、一度でOKになることはまずありません。何回もデバッグ、修正を繰り返さなければなりません。そうなるとラインアセンブラに欠けている機能が欲しくなります。
それは次のようなことのできる機能です。

①プログラムの追加、削除を簡単に行いたい。
②後で理解し易いように、プログラムにコメントなどをつけておきたい。
③ニーモニックコードの状態でテープやディスクにSAVEしておきたい。

ラインアセンブラは1ステップずつ命令をキー入力して、その都度マシン語に翻訳してメモリに書き込んでしまうため、あとからの修正が困難です。
特に命令の追加は、そのままでは不可能です（マシン語を直接メモリに書き込む場合と同じで、その部分にJP命令を書いて空いているメモリアドレスにジャンプさせて、そこに追加の命令を書いたあと、また先程のJP命令の次にジャンプして戻る、という面倒な作業をするしか方法はありません）。
プログラムの中でポイントになる部分にコメントをつけておくと、後で読んだときに理解を助けます。
ラインアセンブラには残念ながらその機能がありません。そして追加や修正などは、何日も、場合によっては何か月も後で行うときもあるため、追加修正ができる形でディスクに保存しておくことができると大変助かります。
この章で説明するアセンブラ（ZASM．COM）は、それらの点を全てクリアしてくれます。
しかしラインアセンブラよりも少し手間がかかります（かけただけの価値はありますが）。
どちらを使うかは、目的によって選択して下さい（小さなプログラムで保存の必要がなければ、ラインアセンブラで手軽に作る。大きいものは、このアセンブラで作って保存する、というように）。

1．　アセンブラプログラムの作成

マシン語やラインアセンブラの場合には、命令をいきなりメモリに書き込んでいきました。
しかしこのアセンブラでは、まずプログラムをファイルの形で書きます。
といっても難しく考える必要はありません。まずもとになるプログラム（これをソースプログラムといいます）を書きます（この時点では、まだマシン語にはなりません）。それから全部を一度にマシン語に翻訳します。
まずプログラムの書き方から説明します。
Z80アセンブラの作業はZBK－V3BASICを起動させてその上で行うのではなくて、通常のWindows95やWindows98の上で行います。元になるプログラム（ソースプログラム）はNotepad（メモ帳）やWriteなどのテキストエディタを使って普通の文章を書くときと同じようにして作ります。
Wordなども使えますが、ZASM．COMはテキストイメージでSAVEされたファイルしか読み込めないのでSAVEするときにファイルの種類に注意する必要があります。SAVEするときは通常のテキスト形式（プレーンテキスト）を選んでSAVEします。それでも時としてファイルの前後に変なゴミが出来てしまうことがあります。勿論アセンブラではエラーになります。そのようなときはファイルを一度Notepadで開いて、ゴミを削除してからSAVEします。

1．1　プログラムの作成

テキストエディタを使って下のリストを作成して下さい。アセンブラプログラムは必ず半角英大文字で書いてく

ださい。

```
;;; TEST PROGRAM
;
ORG $4004
ADRS=$8000
REENT=$1033
;
LD B,00
LD HL,ADRS
XOR A
LOOP:LD (HL),A
INC HL
DJNZ LOOP
JP REENT
;END
```

アセンブラソースプログラムはBASICと異なり行番号はつけません。
ORGは通常はプログラムの先頭に書いておきます。
このプログラムがマシン語に翻訳された時に格納されるメモリの先頭アドレスを指定する命令です。
この例では4004番地からのアドレスに作成したいわけですから、ORG ＄4004と指定します。
この行によって、このプログラムはマシン語に翻訳されたとき、4004番地からメモリに書き込まれたときに正しく実行されるプログラムとして作成されます。
プログラムの最後は ;END で終るようにしてください。;ENDはコメント行なのでこの通りでなくてもよいのですが、アセンブラの都合で最後にコメント行がないとうまく翻訳できないことがあるので、この1行を最後に書くようにしてください。

1．2 ソースプログラムの保存

テキストエディタで作成したプログラムは適当な名前でディスクに保存します。
できればZASM. COMと同じフォルダ(ZBKフォルダ)に保存してください。テキストエディタで保存するとたいていは .TXTという拡張子がつけられますが、ZASM. COMは拡張子があってもなくても支障なく読み込みます。ただし内容はテキストイメージである必要があります。ファイル名も半角英数字を使ってください。ファイル名は小文字でも構いません。ファイル名は8字以内、拡張子は3字以内です。

1．3 アセンブラの実行

Z80アセンブラ(ZASM. COM)はZBKフォルダにあります。
ZBKシステムを起動するときと同じように、デスクトップのZBKアイコンをクリックしてDOSプロンプト(コマンドプロンプト)を開きます。
[注記]Z80アセンブラ、Z80逆アセンブラの作業を行うときにはZBKボードを接続しておく必要はありません。

ZASM TESTASM. TXT[Enter]と入力します(1．2で作ったソースファイルはTESTASM. TXTの名前でSAVEされていることとします)。

C:￥ZBK＞ZASM TESTASM. TXT[Enter]

短いプログラムならば瞬時に翻訳が完了して、下のように表示されます。

2000/12/12 22:59 TESTASM.TXT

```
END=4010
                ;;; TEST PROGRAM
                ;
                ORG $4004
                ADRS=$8000
                REENT=$1033
                ;
4004 0600       LD B,00
4006 210080     LD HL,ADRS
4009 AF         XOR A
400A 77         LOOP:LD (HL),A
400B 23         INC HL
400C 10FC       DJNZ LOOP
400E C33310     JP REENT
                ;END
ADRS           =8000  LOOP           =400A  REENT          =1033
```

プログラムの表記に誤りがあれば、WHAT？とかHOW？という表示に続いて、そのエラーのある行が表示されて、翻訳作業は打ち切られます。

エラーの原因を確認した上で、修正してSAVEした後、もう一度、ZASMコマンドを実行して下さい。

エラーのチェックは2段階に別けて行われます。まず最初のチェックで発見されたエラーを全て表示して処理は打ち切られます。最初のチェックでエラーが無かった場合には第2のチェックを行います。ここでエラーが発見されるとやはりそのエラーを全て表示して処理は打ち切られます。2段階のチェックでエラーが無かったときは、リスト表示とともにソースプログラムと同じファイル名で拡張子が. BINのファイルが作成されます。. BINファイルはZBK－V3BASIC上で／LDコマンドによってRAMにLOADすればKL5C8012で実行できる形で作成されます。

ZアセンブラはZBKボードの接続を必要としませんが、ZASM. COMの実行によって作られる実行形式のバイナリファイルを読み込んで実行するにはZBKボードの接続が必要になります。

ZBKボードを接続してZBKシステムを起動してください。

)Z[Enter]でZBK－V3BASICを起動して、

＞／LD　TESTASM. BIN, 4004[Enter]　と入力してください。

入力後にCMコマンドで4000～4010にマシン語プログラムが読み込まれていることを確認してください。

このプログラムは8000番地～80FF番地に00を格納するものです。

＞DM　8000, 80FF[Enter]と入力して8000～80FF番地の内容を確認しておいてください。

＞JP　4004[Enter]と入力します。または

＞USR(＄4004)[Enter]　と入力して実行したあと(瞬間的に完了します)、もう一度DMコマンドで8000～80FF番地の内容を確認してみてください。

1．4　アセンブラリストファイルの作成

ZASMの実行により、画面には翻訳されたリストが表示されますが、長いプログラムではスクロールされてしまい、はじめの部分を見ることはできません。また表示が完了するのに時間がかかります。

下のように＞を使ったリダイレクト処理を行うことで、出力リストを画面に出す代わりにファイルに出力して保存することができます。

C:￥ZBK＞ZASM　TESTASM. TXT＞TESTASM. WK[Enter]

出力先のファイル名は任意で拡張子も任意につけてかまいません。

[注意]この場合エラーメッセージも画面に表示されず、TESTASM．WKに書込まれることに注意してください。

2．　アセンブラで使用できる命令

2．1　Z80ニモニックの全命令

　表現形式、文法はザイログ社のアセンブラに準拠します。詳しくは別刷りの「Z80命令説明書」を参照して下さい。

2．2　疑似命令

　疑似命令とはZ80の命令ではありませんが、アセンブラプログラムを作るときにあると便利なために考えられたもので、このアセンブラでは次のものがあります。

ORG
DB
;
＝
:
DW
” ”

2．2．1　**ORG**

[書式]ORG　＄nnnn

　マシン語プログラムを割りつけるメモリの先頭アドレスを指定します。同一ソースプログラム内で何回でも使用できます。nnnnは4桁の16進数。
　この命令を使うと、これ以後はこの命令で指定したメモリアドレスからマシン語のプログラムが割りつけられます。

[使用例]
ORG　＄4004

2．2．2　**DB**

[書式]DB　nn

　1バイトの16進数をそのまま割りつけます。メッセージデータなどの文字列を表示するには、文字コードを直接メモリに書き込みますが、そのようなときに使用します。
　nnは2けたの16進数。

[使用例]
DB　45　;E
DB　4E　;N
DB　44　;D

2．2．3　;（セミコロン）

この記号(;)を書くと、それ以後はその行の終わりまでの間に何を書いても、アセンブラの翻訳作業に影響は与えません(前項の使用例や下の使用例を参照して下さい)。

[使用例]
;;; TEST PROGRAM
ORG ＄4004
LD A ,(＄8000) ;first data

---

2. 2. 4 ＝(イコール)

2バイトのデータを変数に代入します。(変数については後述)
＝の左辺に変数名を置き、右辺には4桁の16進定数、＄nnnnを置くことにより、変数にデータを代入します。(1バイトのデータは代入できません)

[使用例]
XYZ1＝＄8000

---

2. 2. 5 :(コロン)

ジャンプ先などを示す時、変数名を使ってそのアドレスにラベルをつけることができます。変数名の後ろに、この記号(:)をつけて示したあとに、普通に命令を書くと、その位置のアドレスがその変数に代入され、ラベルとして使用できるようになります。

[使用例]
LOOP:INC HL
:
JP NZ ,LOOP

---

2. 2. 6 **DW**

[書式]DW ＄nnnn

2バイトの16進数をそのまま割りつけます。DBは1バイトでしたが、ジャンプ先テーブルなどを作成する場合など2バイト単位の方が都合のよいことがあります。そのような時に使用します。nnnnは4けたの16進数で、この代わりにラベルや変数を置くこともできます(特殊な命令なので普通は使用しません)。

[使用例]
(ZASM実行後のリストで示す。JP命令などと同じく、下位、上位の順に割り付けられることに注意)

```
4007 3412       DW $1234
4009 0B40       DW END
400B C33310     END:JP $1033
```

---

2. 2. 7 ” ”(ダブルクォーテーション)

[書式] ”ABCD”

1～4桁の文字を” ”で囲って使うことで、その文字に対応するASCIIコードが作成されます。

[使用例]

（ZASMコマンド実行後のリストです）

```
4050  41424344      "ABCD"
4054  3233334       "234"
```

2．3　定数

Z80アセンブラでは1バイトまたは2バイトの16進定数が使用できますが10進数は使えません。
1バイトのときはそのまま、0A、E7のように表しますが、2バイトのときは、＄マークを付けて、＄1234、＄FEDCのように表します。
－7、＋2Cのように演算子を付けて使う事はできません。
相対ジャンプ命令のオペランド部は、したがってマシン語オブジェクトと同じ、00～FFの1バイト16進表記になります。（JR　NZ，3Eのように表記します）
またインデックスレジスタIX，IYを含む命令の増分パラメータdは＋記号に続けて16進2桁で表記します。（LD　A,(IX＋F3）のように表記します）

2．4　変数

2バイトの定数の代わりとして使います（1バイトの定数の代わりにはできません）。LD命令やCALL、JP命令などのオペランド部に自由に使う事ができます。
変数名は、1桁目がアルファベットA～Zで始まる、13桁以内の英数字および－＿の文字列で示します。
なおBASICとは異なり、アセンブラの変数名には予約語の制約はありません。1桁目がアルファベットで始まる13桁以内の英数字の並びなら、どのような文字列でも使用できます（RET、JP、LDなどニーモニックと同じ綴りでも構いません）。
変数の後ろに：（コロン）をつけて、命令の前に置くことにより、JP命令やCALL命令などで参照できるラベルになります。ラベルとして使用された変数には、そのメモリアドレスが値として入ります。

［注意］二重定義の禁止
同じ変数名を2度以上＝（イコール）で定義したり、＝（イコール）と：（コロン）で同じ変数を使用したり、同じ変数名を2か所以上で：（コロン）をつけてラベルとして使用してはいけません（HOW？メッセージが出てエラーになります）。
二重定義は禁止ですが、参照することに制限はありませんから、LDやJP、CALLなどでは何回使用しても構いません。
使用例は2．2．4～2．2．6を参照して下さい。

2．5　相対ジャンプ命令のラベル、変数

相対ジャンプ命令、JRやDJNZなどでオペランド部に1バイトの移動値を記述する代わりにジャンプ先を示すラベル名を書くことができます。
相対ジャンプ命令は今までのバージョンでは1バイトの16進数をそのまま表記することになっていました。V3システムでも互換性を維持するために1バイトの16進数をそのまま表記することと、ラベル名で表記することのいずれも可能にしたため、特定のラベル名では都合が悪い場合が出てきました。
［都合が悪いラベル名の例］
①JR　NZ，ABC
②JR　Z，DTIN
①の場合は下線部が16進数2桁のABと同じため、JR　NZ，ABと解釈されてしまいます。また②は16進数の始めの1桁がDと読み取られ、次の2桁目が0～Fではないため、エラーになってしまいます。
つまり相対ジャンプ命令のオペランドにラベルを使う場合には、そのラベル名はA～Fで始まっては都合が悪いことになります。
せっかく相対ジャンプ命令でもラベル使用ができるようにしたのに、これではありがたさが半減してしまいま

す。
　そこでラベル名の先頭にラベルを示すマークとして、＊をつけて表示してもよいことにしました。BASICの場合とは異なり、この＊マークはラベルか2桁の16進数かを区別するためにだけ必要なものですから、不要な場合にはつけなくても構いません。＊ABCとABCは全く同じものとして扱われます（下例参照）。また相対ジャンプ命令以外の命令のオペランドに使用しても構いません。

［使用例］
＊ABC：LD　HL，＊DT01　……ABC：LD　HL，DT01でも同じ
ABC5：INC　A　……＊ABC5：INC　Aでも同じ
JR　NZ，＊ABC5　……JR　NZ，ABC5と書いてはいけない
INC　HL
JP　C，＊ABC5　……JP　C，ABC5でも同じ
JP　ABC　……JP　＊ABCでも同じ
DT01：DB　43　……＊DT01：DB　43でも同じ。

2．6　ZASMコマンド実行時のエラーメッセージ

　ZASMコマンド実行時にエラーがあると、エラーメッセージが表示され、処理は中止されます。エラーメッセージは次の3種です。

●WHAT？
　このメッセージに続いて、エラーの発生した行が表示されます。
　ニーモニックのつづりを間違えたり、規則に合わない命令の使い方をした場合などに出されます。
　BASICと異なり、スペースにも意味があります。例えば、LD　A,Bと書くべきところを、LDA,B（DとAの間にスペースが無い）やLD　A　,B（Aと　，の間に余分なスペースが有る）と書いたりすると、エラーになります。

●HOW？
　このメッセージに続いて、エラーの発生した行が表示されます。
　同じ変数名を2度定義した時に出されます。（異なった箇所に同じ変数名のラベルをつけたときなど）
　未定義の変数名が使用された時にも出されます。
　例えば、CALL　A5　という文があって、プログラムのどこにも、A5：のラベルのついた行が無く、またA5＝＄nnnnという定義文も無い場合などがエラーになります。

●SORRY
　このメッセージに続いて、エラーの発生した行が表示されます。
　プログラムが大きすぎて、ワークエリアが足りなくなった時に出されます。
　この場合には、その行以後を削除しなければなりません。
　このように大きくてアセンブルできないプログラムは適当なところで2つに分けて、2本のプログラムにしてそれぞれアセンブルするようにします。

この章で説明する逆アセンブラは8章で説明したライン逆アセンブラと基本的には同じ動作をします。
ライン逆アセンブラはマシン語を逆アセンブルして、ニーモニックコードに置き換えますが、ここで説明するアセンブラはその後さらに加工して再びアセンブルすることが可能な、アセンブラソースプログラムファイルを逆作成します。 勿論そのようにして作られたソースプログラムは、前章で説明したZASMコマンドで再びアセンブルしてマシン語プログラムを作り出すことができます。
例えばここにあるマシン語プログラムがあって、それはROMになっていてアセンブラのソースプログラムは残っていないが、そのプログラムを大幅に書き直さなくてはならなくなった、というような場合を考えて下さい。
逆アセンブラを利用すれば、そのプログラムの解析はできます。しかし部分的に書き直すにしても、これはかなり厄介な作業を覚悟しなければなりません。
また同じように、ソースプログラムが無くて、マシン語プログラムROMだけあって、そのプログラムを別のアドレスに移し変えたいというような場合もあります。これはさらに大変でジャンプアドレスやコール命令のアドレスなどを全部ていねいに直していかなければなりません。一箇所でも見落とすと、プログラムが暴走してしまいます。
アセンブラソースプログラムファイルを逆作成する機能はこれらの問題を一度にクリアしてしまいます。
逆アセンブルコマンドはZDASです。
Z80逆アセンブラ(ZDAS. COM)はZBKフォルダに入っています。

[書式]ZDAS　ファイルネーム　aaaa

ここで対象になるのは、Z80マシン語プログラムをバイナリ形式で保存したファイルです。ZBK－V3BASICを起動して、／SVコマンドでSAVEしたファイルはバイナリファイルです。
aaaaはそのマシン語プログラムが本来おかれていたアドレスです。バイナリファイルにはアドレス情報はありません。aaaaを指定しないでZDASコマンドを実行すると、逆アセンブラはそのプログラムが0000番地からかかれていたものと仮定して翻訳を行います。
マシン語プログラムの場合、途中のアドレス、たとえば4004番地から書かれていたプログラムは0000番地に書かれるプログラムとは異なります。この理由でパラメータaaaaをつけることが必要になります。
もとのプログラムが4004番地からスタートしている場合はaaaaに4004を指定することによって、正しいソースプログラムが作り出されます。

[注記]ニモニックコード変換のルール
この逆アセンブラは指定範囲のメモリの内容を、すべてプログラムとみなしてコード変換します。したがって、もし指定範囲内にJISコードの文字列テーブルや、何かのコード変換テーブルが有っても、すべてプログラムとみなしてZ80ニモニックに変換してしまいます。
またJP命令、CALL命令のように3バイトある命令の途中(2バイト目や3バイト目)から逆アセンブルした場合には、当然異なった命令に翻訳されてしまいます。(この事は2バイト、4バイト長の命令についても同じことが言えます)
もしもZ80の命令コード以外のコードをみつけた時は、その行のニモニック部分に、？マークを表示します。

[使用例]
9章で作成したプログラムを少し変更します。TESTASM. TXTを下のように変更してください(下線部のように変更します)。

```
;;; TEST PROGRAM
;
ORG $4004
ADRS=$8000
```

```
REENT=$1033
;
LD B,00
LD HL,ADRS
XOR A
LOOP:LD (HL),A
INC HL
DEC B
JP NZ,LOOP
JP REENT
;END
```

TESTASM2．TXTとしてSAVEします。

C:￥ZBK＞ZASM　TESTASM2．TXT＞TESTASM2．WK[Enter]を実行します。

逆アセンブラもZBKボードを接続しておく必要はないのですが、ここではテストのためZBKボードを接続してZBKシステムを起動してください。
)Z[Enter]でZBK－V3BASICにエントリして
＞／LD　TESTASM2．BIN，4004[Enter]でロードして9章での説明と同様にして実行してみます（その前に8000～80FFを00以外にしておくために、MV　0000，00FF，8000[Enter]と入力してください）。
実行の結果は最初のプログラムと同じ結果が得られます。

ここまで確認したところで逆アセンブラを使ってみます。
TESTASM2．BINはZ80マシン語プログラムのバイナリファイルです。内容は確認済みです。
ZBK－V3BASICを起動しているMSDOSプロンプト（コマンドプロンプト）はそのままにしておいて、別のMSDOSプロンプト（コマンドプロンプト）もうひとつ開きます。

C:￥ZBK＞ZDAS　TESTASM2．BIN[Enter]
と入力してください。画面には次のリストが表示されます（ZDAS　TESTASM2．BIN＞TESTASM2．WK2としてファイルに出力することもできます。TESTASM2．WK2は任意のファイル名で構いません）。

```
0000 0600          LD B,00
0002 210080        LD HL,$8000
0005 AF            XOR A
0006 77            LD (HL),A
0007 23            INC HL
0008 05            DEC B
0009 C20A40        JP NZ,$400A
000C C33310        JP $1033
END
```

マシン語プログラムをある程度経験している人ならば、これではだめだ、ということがわかるはずです。
その理由については先で説明することにして、ZDASコマンドを実行すると、これとは別のファイルが作られます。TESTASM2．SOU　のように　．SOUという拡張子がつけられます。これが逆アセンブルの結果作成されたソースプログラムファイルです。テキストエディタで開いてみてもよいのですが、今回は短いファイルですからMSDOSのTYPEコマンドを使ってみます。

C:￥ZBK＞TYPE　TESTASM2．SOU[Enter]

```
      ORG $0000
```

```
        Z8000=$8000
        Z400A=$400A
        Z1033=$1033
        LD B,00
        LD HL,Z8000
        XOR A
        LD (HL),A
        INC HL
        DEC B
        JP NZ,Z400A
        JP Z1033
```

このようにZDAS．COMはバイナリファイルを読んで再アセンブル可能なZ80ソースプログラムを作成します。

じつはこれはZDASの誤った使い方の例なのです。もとのプログラムはORG ＄4004のはずだったのにここではORG ＄0000になっています。ORGについてはあとから直すことも可能です。問題は下から2行目、JP NZ, Z400Aです。逆アセンブラはこのようにJP命令の飛び先やLD命令の定数などをラベルにして示します（先頭にZをつけてラベル名にしている）。Z400Aはリストのはじめの方でZ400A＝＄400Aとして定義しています。もとのプログラムではLD （HL), Aの命令の書かれたアドレスを指定していたのですが、このプログラムでは全く違ったアドレスを指定していることになります。

もう一度ZDASをやり直してみます。今度は下のように4004番地をパラメタ指定して実行します。

```
C:￥ZBK＞ZDAS TESTASM2.BIN 4004[Enter]
4004 0600           LD B,00
4006 210080         LD HL,$8000
4009 AF             XOR A
400A 77             LD (HL),A
400B 23             INC HL
400C 05             DEC B
400D C20A40         JP NZ,$400A
4010 C33310         JP $1033
END
```

今度はよさそうです。4004番地からスタートしています。TESTASM2．SOUも確認してみます。

```
C:￥ZBK＞TYPE TESTASM2.SOU[Enter]
        ORG $4004
        Z8000=$8000
        Z1033=$1033
        LD B,00
        LD HL,Z8000
        XOR A
Z400A:LD (HL),A
        INC HL
        DEC B
        JP NZ,Z400A
        JP Z1033
```

ORG ＄4004になりました。プログラムの中ほど、Z400A:LD （HL), A に注目してください。

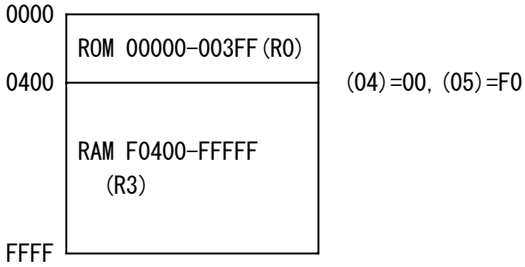
このようにZDAS．COMはJP命令の飛び先アドレスをチェックして正しい位置にラベルを設定してくれます。かしこいプログラムだと思いませんか？

以上の説明でZDASのパラメタaaaaの使い方が理解できたと思います。

なお9章で作ったプログラムを変更したわけは、aaaaをつけないで逆アセンブルしたときの問題点をはっきり示すためです。もとのプログラムは相対ジャンプ命令なので、この部分に限って言えば問題は発生しないのです。このもとのようなプログラムをリロケータブル（再配置可能）なプログラムであるといいます。興味の有る方は9章のTESTASM．BINを逆アセンブルして、なぜリロケータブルなのかを考えてみてください。

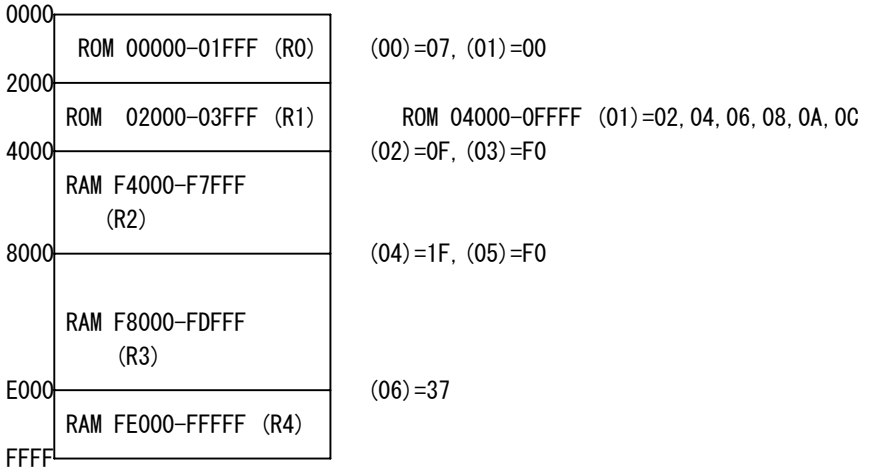
ところでそれならば全て相対ジャンプ命令だけで書けばよいではないか、という疑問がでてくるかもしれません。じつは相対ジャンプ命令にも短所があるのです。またCALL命令は絶対アドレスしか許されません。相対ジャンプ命令の短所とはなにか？これも各自がプログラムを書くのになれてくる過程で明らかになってくると思いますからここでは説明を省きます。

ZBK開発セットで起動するとブートプログラムが実行されます。そのときのメモリマップは次の通りです。

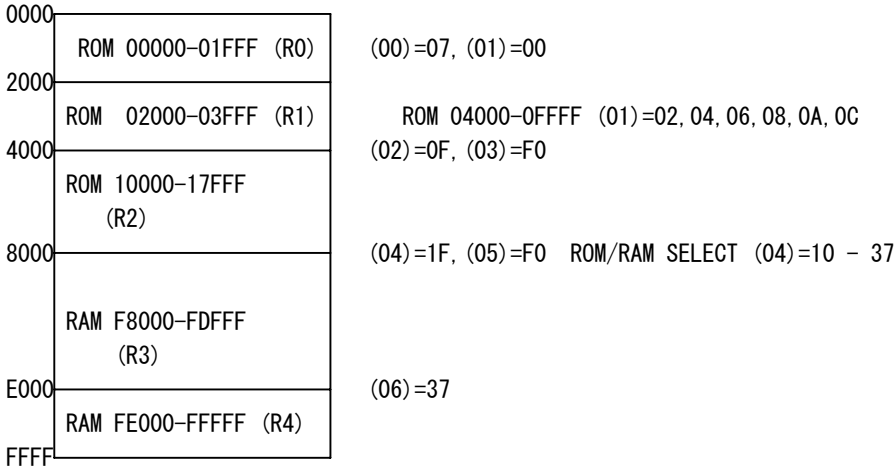


0000～03FFがROMで残りはRAMになります。メモリバンクはR0とR3のみが有効です。メモリバンク用レジスタには(04)＝00、(05)＝F0がセットされます。メモリバンクについては、ZBKボードハードウェア説明書3章7. を参照してください。

Zコマンドを入力するか、メモリバンク用レジスタ(00)～(06)を書きかえるまでは、このメモリ構成になっています。 ZコマンドによってZBK－V3BASICが起動すると次のメモリ構成になります。



ROMスタートした場合には次の構成になります。



2000～3FFFはシステムがROMの02000～0FFFFをバンク切り替えによって割り当てて使用します。この部分にユーザーが関与することはできません。

4000～7FFFはユーザーのROMプログラムエリア10000～17FFFが割り当てられ、8000～DFFFにはRAMのF8000～FDFFFが割り当てられますが、ユーザープログラムの先頭で、I／Oアドレス （04） に10H～37Hを出力することでROMとRAMの境界を4400～E000の範囲で変更することができます。方法については6章6. を参照してください。

1. ND80KモニタとZBKブートプログラム

ND80KはZB10K～ZB28Kと異なり、ボード上のキーボードとLEDにより単独でマシン語プログラムの作成、実行などが行えます。
ZBK開発セットと接続することで、より高度な処理が行えますが、ZBK－V3BASICの機能によってプログラムを作成し、ふたたびND80Kモニタに戻って実行する、というような処理がしたい場合もあります。
開発セットのブートプログラムとND80Kモニタとの間は自由に往復できるように考慮してあります。
ND80KからZBKブートプログラム（13章参照）に行くには、ND80Kのキーボードから0200[ADRSSET][RUN]と操作します。
ND80Kモニタから離れるとLED表示は消灯し、ND80Kのキー入力はできなくなります（リセットのみ可能ですが、ブートプログラムにエントリ中にリセットすると、DOS／V側プログラムがハングアップします）。
ブートプログラムにエントリするとレジスタダンプが表示されますが、その後ブートプログラムのコマンド入力が可能になります。
ここからZBK－V3BASICにエントリするには、Z[Enter]を入力します。
ブートプログラムからND80Kモニタに戻るには、N[Enter]を入力します。
ZBK－V3BASICからND80Kモニタに直接戻ることはできません。一旦、／BOOTコマンドでブートプログラムに戻ってから、NコマンドでND80Kモニタに行きます。

2. ND80KモニタとZBK－V3BASIC

ZBK－V3BASICから直接ND80Kモニタに戻ることはできませんが、ユーザーのBASICプログラムの中でND80Kモニタの機能を実行することはできます。
ZBK－V3BASICは特別のモードで実行されており、ND80Kモニタとは切り離されているため、ND80Kモニタのサブルーチンアドレスを直接コールすることはできません。特定の機能のみ、別に用意したエントリアドレスをコールすることで実行できます。
ユーザーのBASICプログラムの中でマシン語サブルーチンコール命令（USR関数）により実行します。
BASICで使用できるND80Kモニタの機能は次の通りです。

①NDLEDON （LED表示ON）　　エントリアドレス　＄020C
②NDLEDOFF （LED表示OFF）　　エントリアドレス　＄020F
③NDSEGDP （LEDセグメント表示）　　エントリアドレス　＄0212
④NDADDSP （LEDアドレスデータ表示）　エントリアドレス　＄0215
⑤NDKEYIN （KEY入力①）　　エントリアドレス　＄0218
⑥NDINPUT （KEY入力②）　　エントリアドレス　＄021B
⑦NDSOUND （サウンド出力）　　エントリアドレス　＄021E
⑧NDCHRPRT （1文字プリント）　　エントリアドレス　＄0221

各サブルーチンの説明

①NDLEDON （LED表示ON）　　エントリアドレス　＄020C

ZBKブートプログラムやZBK－V3BASICにエントリ中はND80KのLEDは消灯しています。しかしユーザープログラムの中ではLEDを点灯させることができます。
ユーザープログラムの中で
　USR（＄020C）
を実行するとそれ以後はUSR（＄020F）を実行するかブレイクするまでLEDが点灯します。
[注意]

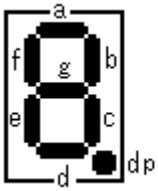
LED表示ルーチンはカウンタ／タイマーBチャネル2による割り込みを利用して2ms毎にダイナミック点灯を行っています。プログラム中でINTOFFを実行するとLEDを点灯させることができなくなります。またLED点灯を行う場合にはユーザーはカウンタ／タイマーBチャネル2を使用できません。

②NDLEDOFF （LED表示OFF）　　エントリアドレス　＄020F

USR（＄020F） の実行でLEDが消灯します。

③NDSEGDP （LEDセグメント表示）　　エントリアドレス　＄0212

8個のLEDの1個を指定して、セグメント表示を行います。B%に表示桁位置（0～7）、A%にセグメントデータを入れてUSR（＄0212）を実行します。
LED表示ONルーチン　USR（＄020C）が実行されていないと、表示結果を見ることができません。
セグメントデータバッファ内のデータの各ビットはLED表示器1桁のセグメントと下図のように関係しています。対応するビットが1のとき、そのセグメントが点灯します。



7	6	5	4	3	2	1	0
dp	g	f	e	d	c	b	a

たとえば　2　という数字を表示するには、a、b、d、e、g＝1なので、01011011（5BH）を表示バッファに書き込みます。
[プログラム例]
LEDの左3桁に　End　と表示します（残り5桁はブランク）。

```
 10 DIM SEGDT%(7)
 20 DATA $79,$54,$5E
 30 FOR N%=0 TO 2
 40 READ SEGDT%(N%)
 50 NEXT N%
 60 FOR N%=3 TO 7
 70 SEGDT%(N%)=0
 80 NEXT N%
 90 FOR B%=0 TO 7
100 A%=SEGDT%(B%):USR($0212)
110 NEXT B%
120 USR($020C)
130 GOTO 130
```

ブレイクすると表示OFFになってしまうので130行でループさせてプログラムが終らないようにしてあります。

④NDADDSP （LEDアドレスデータ表示）　エントリアドレス　＄0215

アドレスレジスタ（LED左4桁）にB%の値を、データレジスタ（LED右4桁）にA%の値をそれぞれ16進数で表示します。
B%＝＄1234、A%＝＄ABCDのとき、USR（＄0215）を実行するとLEDに　1234AbCd　と表示されます。LED表示ONルーチン　USR（＄020C）が実行されていないと、表示結果を見ることができません。

⑤NDKEYIN （KEY入力①）　　エントリアドレス　＄0218

USR(＄0218)を実行するとND80Kのキー入力待ちになります。キーが押されるとそのキーに対応する値(0～23)がA%に入ります。

キーとA%に入る値の関係は0～Fキーが0～15で以下RUN＝16、CONT＝17、ADRSSET＝18、READDEC＝19、READINC＝20、WWITEINC＝21、＊(I／O)＝22、REG＝23です。

[プログラム例]

```
10 USR($0218)
20 IF A%=22 THEN STOP
30 PRINT A%
40 GOTO 10
```

[注意]

USR(＄0218)はキー入力されるまでメインルーチンに戻って来ません。[CTRL]Bによってブレイクすることはできません。[CTRL]Bを入力するとハングUPしてしまいます。プログラム例のように特定のキーが押されたらブレイクするようにしてください(この例では＊(I／O)キーでブレイク)。

⑥NDINPUT (KEY入力②)　　エントリアドレス　＄021B

USR(＄021B)が実行されたときにND80Kのキーが押されていると、そのキーに対応する値がA%に入り、何も押されていないときは255がA%に入ります。前述のUSR(＄0218)と異なり、キーチェック後すぐにリターンするので、ブレイクに対する考慮は必要ありません。

[プログラム例]

```
10 USR($021B)
20 PRINT A%
30 GOTO 10
```

⑦NDSOUND (サウンド出力)　　エントリアドレス　＄021E

A%の値に対応する高さの音が一定時間スピーカーから出力されます。A%の値と音の高さの関係についてはND80K操作説明書8章を参照してください。

[プログラム例]

```
10 USR($021B)
20 IF A%=255 GOTO 10
30 USR($021E)
40 GOTO 10
```

ND80K操作説明書8章の電子オルガンプログラムと同じ働きをします。

⑧NDCHRPRT (1文字プリント)　　エントリアドレス　＄0221

ND80Kにセントロニクスプリンタを接続して、プリンタに文字を出力します。A%の値に対応する文字コードがプリンタに送られます。

［1章5. スタート］で説明した、スタート時の ） プロンプト表示がブートプログラムが実行されている状態です。ZBK－V3BASICにエントリするために入力する Z はブートプログラムのコマンドです。

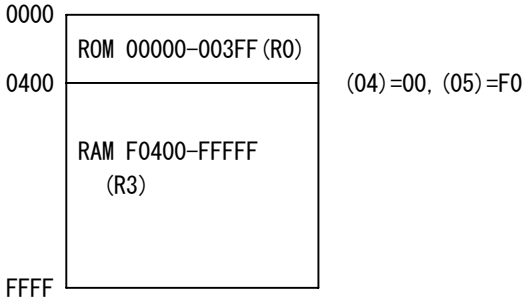
ブートプログラムは開発システムが起動するとスタートする、小さなプログラムです（0000～03FFの1KB）。BASICを使う上では何の役にも立ちません。普通のユーザーはZコマンドでZBK－V3BASICを起動させてしまうとあとは終了するまでここに戻ってくることはありません。

実はブートプログラムは、最初にKL5C8012の動作を確認し、そしてZBK－V3BASICを移植しデバッグするために必要になって作成したものです。マシン語プログラムのためのデバッグ機能はZBK－V3BASICの中にもあります（4章）が、ブートプログラムはそれよりももっと下層で働くマシン語プログラムのためのツールです。ZBKボードをBASICで制御するのではなくて、マシン語のみで制御するような使い方をする場合に役に立ちます。

ブートプログラムの実行中もZBK－V3BASICと同様、スクリーンエディタが機能しています。またログファイルも作成されます。

1. 起動時（ブートプログラムエントリ時）のメモリマップ

ZBK開発セットの構成で起動すると、プログラムROM128KB、RAM128KBのうちCPUが直接アクセスする64KBのアドレスには、次のように割り当てられます。



0000～03FFがROMで残りはRAMになります。メモリバンクはR0とR3のみが有効です。メモリバンク用レジスタには（04）＝00、（05）＝F0がセットされます。メモリバンクについては、ZBKボードハードウェア説明書3章7. を参照してください。

Zコマンドを入力するか、メモリバンク用レジスタ（00）～（06）を書きかえるまでは、このメモリ構成になっています。ブートプログラムのコマンドは0400～FFFFのRAMに対して使用します。

ROMにユーザープログラムを書いて、ショートストラップの設定でROMスタートしたときは、11章で説明したメモリマップにしたのち、1800番地にジャンプします（BASICシステムを使用しない場合）。

［注意］

ブートプログラム自身のワークやスタックのためにF×××番地のメモリを使用します。この範囲のメモリを不用意に書きかえると、ブートプログラムが機能しなくなりますから、F000～FFFFは書き換えないようにしてください。

2. コマンド

次のコマンドがあります。

C （Change memory）
D （Dump memory）
G （Go）
I （In）

L （Load）
N （ND80K JUMP）
O （Out）
Q （Quit）
R （Return）
S （Save）
T （Trace）
Z （ZBK－V3BASIC entry）

各コマンドについて簡単に説明します。コマンド（およびパラメータ）入力後［Enter］入力により実行されます。

①**C** （Change memory）

［書式］C aaaa

指定メモリアドレスの内容を書き換えます。ZBK－V3BASICマシン語モニタのCMコマンドとほぼ同じ動作をします。aaaaは4桁の16進数でアドレスを示します。

②**D** （Dump memory）

［書式］D aaaa

指定メモリアドレスからはじまる128バイトの内容を表示します。ZBK－V3BASICマシン語モニタのDMコマンドとほぼ同じ動作をします。aaaaは4桁の16進数でアドレスを示します。

③**G** （Go）

［書式1］G＝aaaa bbbb
［書式2］G bbbb

指定メモリアドレスにジャンプしてユーザープログラムを実行します。［書式1］でaaaaはジャンプ先ユーザーアドレスで、bbbbはブレイクアドレスです。bbbbは省略できます。
［書式2］はブレイク後に次のブレイクアドレス（bbbb）を指定してユーザープログラムにリターンします。
aaaa、bbbbは4桁の16進数で命令の第一バイトが存在するアドレスでなければいけません。
ZBK－V3BASICマシン語モニタのJP、BP、RTコマンドとほぼ同じ動作をします。

④ **I** （In）

［書式］ I aa

I／Oアドレス aa に対してIN命令を実行し、入力した値を16進数で表示します。ZBK－V3BASICマシン語モニタのINコマンドとほぼ同じ動作をします。aaは2桁の16進数でI／Oアドレスを示します。

⑤**L** （Load）

［書式］ L ファイルネーム, aaaa

バイナリファイルをアドレスaaaaからLOADします。ファイルネームはMSDOSのルールに従います。aaaaは4桁の16進数で省略はできません。ZBK－V3BASICマシン語モニタの／LDコマンドとほぼ同じ動作をします。LOADできるファイルサイズはB000バイト以下でなければいけません。

⑥**N** （ND80K JUMP）

ND80Kの場合、N[Enter]でND80Kモニタに戻ることができます。
ND80Kからブートプログラムに戻るには、ND80Kのキーボードから0200[ADRSSET][RUN]と操作します。ZBK－V3BASICからND80Kモニタに戻ることはできません。ZBK－V3BASICからは、まず／BOOTコマンドでブートプログラムに戻ってから、NコマンドでND80Kモニタに戻ります。
なおNコマンドでND80Kモニタに戻ったあと、ふたたび0200[ADRSSET][RUN]でブートプログラムにリターンすると、レジスタダンプが表示されますが、それ以後は普通にブートコマンドの入力ができます。

⑦**O** （Out）

[書式] O aa, nn

I／Oアドレス aa に2桁の16進数nnを出力します。ZBK－V3BASICマシン語モニタのOTコマンドとほぼ同じ動作をします。aaは2桁の16進数でI／Oアドレスを示します。

⑧**Q** （Quit）

処理を終了してMSDOSに戻ります。ブートプログラムの実行中もログファイルの機能が働いていて全ての表示内容が保存されます。Qコマンドで終了すると完全なログファイルが作成されます。[CTRL]CでMSDOSに戻った場合、最後の1ページは保存されません。

⑨**S** （Save）

[書式] S ファイルネーム, aaaa, bbbb

バイナリファイルをアドレスaaaa～bbbbの範囲のマシン語プログラム（データ）LOADします。ファイルネームはMSDOSのルールに従います。aaaa、bbbbは4桁の16進数で省略はできません。ZBK－V3BASICマシン語モニタの／SVコマンドとほぼ同じ動作をします。

⑩**T** （Trace）

[書式1] T＝aaaa n
[書式2] T n

ブレイクポイントから指定ステップだけトレース実行し、各ステップ毎にレジスタの値をダンプ表示します。書式1は G＝aaaa ＋ T n の動作になります。アドレスaaaaからスタートしてnステップ、トレース実行します。
書式2はブレイクしているポイントからnステップ、トレース実行します。
トレースはKL5C8012のタイマ／カウンタBチャネル0を割り込み使用しています。
ステップ回数nは1桁以上の16進数で、nを省略するとn＝1を指定したのと同じになります。

[注意1]
トレース機能はタイマーの割り込みを利用しています。ユーザープログラムに割り込み禁止命令DIがあると実行できなくなります。そのためトレース中にDI命令をみつけると！を表示してトレースを中止します。
[注意2]
トレースはブレイク機能と異なり、ROMに書かれたプログラムに対しても実行できますが、ブートプログラム自身をトレースすると、トレースそのもので使用している機能をトレースすることになり正しく実行されません。

⑪**Z**（ZBK－V3BASIC entry）

ZBK－V3BASICにエントリします。なおZBK－V3BASICからブートプログラムに戻るには／BOOTコマンドを使います。

3. プログラムリスト

ZB10K～ZB28K、ND80KモニタROMに入っているブートプログラムリストです（このリストの作成後にプログラムの改良を行ったため、実際のROMの内容とは異なる部分もあります）。
このプログラムはZ80クロスアセンブラ（ZASM. COM）で作成したものです。
［注意1］
このリストはユーザーの学習の手助けとなるように参考として公開したものです。プログラムの著作権は㈲中日電工が所有します。個人目的以外での複製、改変を禁じます。
プログラムの内容に関するお問い合わせ、ご質問にはお答えできません。
［注意2］
ユーザーが作成したプログラムをROMの空き部分に追加書込みすることで、電源ON後ただちにユーザープログラムを実行することができます。この機能を使って機器などに組込んで制御することが可能ですが、人体、生命、財産に影響のある機器への組込みや、応用には絶対に使用しないでください。
マイクロコンピュータを含む電子回路は極めて微弱な電圧、電流で高速動作をしているためノイズなどの影響を受けやすく、思わぬ誤動作をする可能性があります。またICその他の部品自体の劣化、欠陥により誤動作する可能性やシステムプログラムのバグによる誤動作の可能性もあります。
ZB10K～ZB28K、ND80Kの使用に際し、当社は予想可能、不可能なトラブルの全てに対して一切の保証を致しません。製品、プログラムに欠陥、不良が発見された場合には現品の修理、交換または返品の受け付けを行う以外の責は負いません。

```
2000/12/23  22:20  ZBOOTP.TXT
END=0361
              ;;; BOOT LOADER for ZBK 00/10/13 10/14 10/17 10/22 12/19 12/20
              ;;;
                    ORG $0000
              ;
                    BKCG6=$051B
              ;
                    ZBENTRY=$1800
              ;
                    NDMON=$2000
                    SEGDP=$200C
                    ADDSP=$2012
                    KEYIN=$2015
                    INPUT=$2018
                    SOUND=$201B
                    CHRPRT=$202A
              ;
                    ;
                    LSEG8=$FFFF
                    LSEG7=$FFFE
                    LSEG5=$FFFC
                    LSEG4=$FFFB
                    LSEG3=$FFFA
```

```
        LSEG2=$FFF9
        LSEG1=$FFF8
        DISP3=$FFF6
        DISP1=$FFF4
        BRKC=$FFF2
        BRKA=$FFF0
        ADRSH=$FFEF
        ADRSL=$FFEE
        DATAH=$FFED
        DATAL=$FFEC
        AREG=$FFEB
        FREG=$FFEA
        CREG=$FFE8
        EREG=$FFE6
        LREG=$FFE4
        SPL=$FFE2
        PCL=$FFE0
        LDSH=$FFD4
        IREG=$FFD3
        TRSW=$FFD1
        IOREG=$FFD0
        RMODE=$FFCF
        BRKCODE=$FFCE
        TRADRS=$FFCC
        LEDAD=$FFCA
        S_MODE=$FFC9
        S_RATE=$FFC8
        S_STATUS=$FFC7
        EIMK=$FFC6
        ATRNMK=$FFC4
;
        RS7JA=$FFBB
        RS7JC=$FFBA
        SPTOP=$FFBA
;
        IR15=$FEFE
        IR13=$FEFA
;
        USTOP=$F800
;
        BP=$F442
        AP=$F440
        AD=$F300
;
;
        BF=$F000;for NDLPRT
;
        Z6000=$6000
;
;
```

```
                    DAS=$1400
                ;
                ;
0000 F3             DI
0001 AF             XOR A
0002 D32E           OUT (2E),A;RTC
0004 D304           OUT (04),A
0006 3EF0           LD A,F0
0008 D305           OUT (05),A;ROM1K 0-03FF,RAM63K 0400-FFFF
000A 3EFF           LD A,FF
000C D32F           OUT (2F),A
000E D398           OUT (98),A;PCCARD
0010 D330           OUT (30),A
0012 D331           OUT (31),A
0014 C3C901         JP START2
                ;
                ; BOOT LOADER
                ;
0017 3E23       BOOTENTRY:LD A,23
0019 D333           OUT (33),A;P20-P27 OUT P40-P43 IN P44-P47 OUT
001B 214301         LD HL,BOOT_BREAK
001E 22BBFF         LD (RS7JA),HL
0021 3EC3           LD A,C3
0023 32BAFF         LD (RS7JC),A
                ;
0026 31BAFF         LD SP,SPTOP
0029 CDD600     LOOP:CALL RBH(HIGH 4BIT)
002C C23B00         JP NZ,CMDRD
002F CDF000         CALL RBL(LOW 4BIT)
0032 77             LD (HL),A
0033 23             INC HL
0034 C32900         JP LOOP
                ;
                    ORG $0038
                ;
0038 C3BAFF         JP RS7JC;RST 7
                ;
003B 79         CMDRD:LD A,C
003C E6F0           AND F0
003E 2830           JR Z,TEST
0040 FE10           CP 10
0042 CA9100         JP Z,ADSET
0045 FE20           CP 20
0047 CA2900         JP Z,LOOP;WRITE INC
004A FE30           CP 30
004C CAA800         JP Z,READ
004F FE40           CP 40
0051 CA9401         JP Z,BOOT_RUN
0054 FE50           CP 50
0056 CABE00         JP Z,BOOT_OUT
```

```
0059 FE60           CP 60
005B CACD00         JP Z,BOOT_IN
005E FE70           CP 70
0060 CAAF02         JP Z,BRSET
0063 FE80           CP 80
0065 CA9701         JP Z,RET
0068 FEA0           CP A0
006A CAB802         JP Z,TRACE
006D C32900         JP LOOP
                    ;
0070 DB2C     TEST:IN A,(2C)
0072 CB57           BIT 2,A
0074 28FA           JR Z,TEST
0076 3E00           LD A,00;READY OUT
0078 D332           OUT (32),A
007A DB2C     TEST1:IN A,(2C)
007C CB57           BIT 2,A
007E 20FA           JR NZ,TEST1;IF STB OFF
0080 3E80           LD A,80
0082 D332           OUT (32),A;BUSY OUT
0084 DB2C     TEST2:IN A,(2C)
0086 CB57           BIT 2,A
0088 28FA           JR Z,TEST2;IF STB ON
008A 3E00           LD A,00
008C D332           OUT (32),A;READY OUT
008E C32900         JP LOOP
                    ;
0091 CDD600   ADSET:CALL RBH
0094 C23B00         JP NZ,CMDRD
0097 CDF000         CALL RBL
009A 6F             LD L,A
009B CDD600         CALL RBH
009E C23B00         JP NZ,CMDRD
00A1 CDF000         CALL RBL
00A4 67             LD H,A
00A5 C32900         JP LOOP
                    ;
00A8 CDD600   READ:CALL RBH
00AB C23B00         JP NZ,CMDRD
00AE CDF000         CALL RBL
00B1 47             LD B,A
00B2 7E       READ2:LD A,(HL)
00B3 CD1401         CALL BOUT
00B6 23             INC HL
00B7 05             DEC B
00B8 C2B200         JP NZ,READ2
00BB C32900         JP LOOP
                    ;
00BE CDD600   BOOT_OUT:CALL RBH
00C1 C23B00         JP NZ,CMDRD
```

```
00C4 CDF000          CALL RBL
00C7 4D              LD C,L
00C8 ED79            OUT (C),A
00CA C32900          JP LOOP
                ;
00CD 4D         BOOT_IN:LD C,L
00CE ED78            IN A,(C)
00D0 CD1401          CALL BOUT
00D3 C32900          JP LOOP
                ;
00D6 DB2C       RBH:IN A,(2C)
00D8 CB57            BIT 2,A;STB
00DA CAD600          JP Z,RBH;IF STB ON
00DD 3EEE            LD A,EE
00DF D333            OUT (33),A;READY
00E1 DB2C       RBH1:IN A,(2C)
00E3 CB57            BIT 2,A;STB
00E5 C2E100          JP NZ,RBH1
00E8 4F              LD C,A;DATA(HIGH 4BIT)
00E9 3EEF            LD A,EF
00EB D333            OUT (33),A;BUSY
00ED CB59            BIT 3,C
00EF C9              RET
                ;
00F0 DB2C       RBL:IN A,(2C)
00F2 CB57            BIT 2,A;STB
00F4 CAF000          JP Z,RBL
00F7 3EEE            LD A,EE
00F9 D333            OUT (33),A;READY
00FB DB2C       RBL2:IN A,(2C)
00FD CB57            BIT 2,A;STB
00FF C2FB00          JP NZ,RBL2
0102 47              LD B,A;DATA(LOW)
0103 3EEF            LD A,EF
0105 D333            OUT (33),A;BUSY
0107 CB38            SRL B
0109 CB38            SRL B
010B CB38            SRL B
010D CB38            SRL B
010F 79              LD A,C
0110 E6F0            AND F0
0112 B0              OR B
0113 C9              RET
                ;
0114 F3         BOUT:DI
0115 5F              LD E,A
0116 DB2C       BOUT2:IN A,(2C)
0118 CB57            BIT 2,A;STB
011A 28FA            JR Z,*BOUT2
011C 1609            LD D,09
```

```
011E 3EEE           LD A,EE
0120 D333           OUT (33),A;START BIT
0122 DB2C     BOUT3:IN A,(2C)
0124 CB57           BIT 2,A;STB
0126 20FA           JR NZ,*BOUT3
0128 3E77           LD A,77
012A CB13           RL E
012C 17             RLA
012D D333           OUT (33),A
012F DB2C     BOUT4:IN A,(2C)
0131 CB57           BIT 2,A;STB
0133 28FA           JR Z,*BOUT4
0135 15             DEC D
0136 20EA           JR NZ,*BOUT3
0138 3EEF           LD A,EF
013A D333           OUT (33),A
013C 3AC6FF         LD A,(EIMK)
013F B7             OR A
0140 C8             RET Z
0141 FB             EI
0142 C9             RET
                ;;;
0143 F3       BOOT_BREAK:DI
0144 22E4FF         LD (LREG),HL
0147 E1             POP HL
0148 F5             PUSH AF
0149 3AD1FF         LD A,(TRSW)
014C B7             OR A
014D 3E00           LD A,00
014F 32D1FF         LD (TRSW),A
0152 FA5A01         JP M,B_BRK2
0155 2B             DEC HL
0156 3ACEFF         LD A,(BRKCODE)
0159 77             LD (HL),A
015A ED53E6FF B_BRK2:LD (EREG),DE
015E ED43E8FF       LD (CREG),BC
0162 22E0FF         LD (PCL),HL;PC
0165 E1             POP HL;=AF
0166 22EAFF         LD (FREG),HL;AF
0169 ED73E2FF       LD (SPL),SP
016D 31E0FF         LD SP,PCL
0170 DDE5           PUSH IX
0172 FDE5           PUSH IY
0174 08             EX AF,AF'
0175 F5             PUSH AF
0176 D9             EXX
0177 C5             PUSH BC
0178 D5             PUSH DE
0179 E5             PUSH HL
017A ED57           LD A,I
```

```
017C 67             LD H,A
017D ED5F           LD A,R
017F 6F             LD L,A
0180 E5             PUSH HL
0181 31BAFF         LD SP,SPTOP
0184 3E02           LD A,02
0186 CD1401         CALL BOUT
0189 3AC6FF         LD A,(EIMK)
018C B7             OR A
018D CA2900         JP Z,LOOP
0190 FB             EI
0191 C32900         JP LOOP
                ;
0194 22E0FF     BOOT_RUN:LD (PCL),HL
0197 F3         RET:DI
0198 31D4FF         LD SP,LDSH
019B E1             POP HL
019C D1             POP DE
019D C1             POP BC
019E D9             EXX
019F F1             POP AF
01A0 08             EX AF,AF'
01A1 FDE1           POP IY
01A3 DDE1           POP IX
01A5 31E6FF         LD SP,EREG
01A8 D1             POP DE
01A9 C1             POP BC
01AA F1             POP AF
01AB ED7BE2FF       LD SP,(SPL)
01AF 2AE0FF         LD HL,(PCL);PC
01B2 E5             PUSH HL
01B3 2AE4FF         LD HL,(LREG)
01B6 F5             PUSH AF
01B7 3AD1FF         LD A,(TRSW)
01BA B7             OR A
01BB FAC602         JP M,TRSET
01BE 3AC6FF         LD A,(EIMK)
01C1 B7             OR A
01C2 2803           JR Z,RET2
01C4 F1             POP AF
01C5 FB             EI
01C6 C9             RET
01C7 F1         RET2:POP AF
01C8 C9             RET
                ;
01C9 D332       START2:OUT (32),A
01CB 3E9B           LD A,9B
01CD D383           OUT (83),A
01CF D387           OUT (87),A
01D1 D38B           OUT (8B),A
```

```
01D3 D38F          OUT (8F),A
01D5 3E37          LD A,37
01D7 D306          OUT (06),A
01D9 3EFF          LD A,FF
01DB D334          OUT (34),A;LERL
01DD D335          OUT (35),A;LERH
01DF 3EE0          LD A,E0
01E1 D337          OUT (37),A;IVR
01E3 AF            XOR A
01E4 32C6FF        LD (EIMK),A
01E7 32D1FF        LD (TRSW),A
01EA 3EFE          LD A,FE
01EC ED47          LD I,A
01EE ED5E          IM 2
01F0 3EBF          LD A,BF
01F2 D3C4          OUT (C4),A;ND CHECK
01F4 DBC0          IN A,(C0)
01F6 E602          AND 02
01F8 C23602        JP NZ,ZBMODE
01FB C35102        JP NDJMP
                ;
                ;
                     ORG $0200
                ;
                ;   from ND to BOOT ENTRY
                ;
0200 C32702        JP ZBOOT
0203 C36C02        JP RAMST
0206 C35102        JP NDJMP
0209 C3E102        JP INTDP
020C C3F802        JP NDLEDON
020F C30C03        JP NDLEDOFF
0212 C31603        JP NDSEGDP
0215 C32703        JP NDADDSP
0218 C33A03        JP NDKEYIN
021B C34103        JP NDINPUT
021E C34F03        JP NDSOUND
0221 C35A03        JP NDCHRPRT
0224 C3C602        JP TRSET
                ;
0227 CD0C03   ZBOOT:CALL NDLEDOFF
022A DB2C          IN A,(2C)
022C E6F8          AND F8
022E FE08          CP 08;TEST
0230 CA1700        JP Z,BOOTENTRY;FIRST BOOT
0233 C3CD00        JP BOOT_IN;REBOOT
                ;
0236 DB2C     ZBMODE:IN A,(2C)
0238 E608          AND 08
023A CA8F02        JP Z,ROMST
```

```
023D 3EFF     ZBMODE2:LD A,FF
023F 32C4FF           LD (ATRNMK),A
0242 214301           LD HL,BOOT_BREAK
0245 22FAFE           LD (IR13),HL
0248 2100F8           LD HL,USTOP
024B 22E2FF           LD (SPL),HL
024E C31700           JP BOOTENTRY
                     ;
                ; JUMP TO ND MONITER
0251 3E1F     NDJMP:LD A,1F
0253 D304             OUT (04),A
0255 3EF0             LD A,F0
0257 D305             OUT (05),A
0259 3E0F             LD A,0F
025B D302             OUT (02),A
025D 3EF0             LD A,F0
025F D303             OUT (03),A
0261 3E07             LD A,07
0263 D300             OUT (00),A
0265 3E0C             LD A,0C;  ND BANK (BANK6)/ 00 for 256ROM
0267 D301             OUT (01),A
0269 C30020           JP NDMON
                     ;
026C CD0C03   RAMST:CALL NDLEDOFF
026F 3EFF             LD A,FF
0271 32C4FF           LD (ATRNMK),A
0274 3E1F             LD A,1F
0276 D304             OUT (04),A
0278 3EF0             LD A,F0
027A D305             OUT (05),A
027C 3E0F             LD A,0F
027E D302             OUT (02),A
0280 3EF0             LD A,F0
0282 D303             OUT (03),A
0284 3E07             LD A,07
0286 D300             OUT (00),A
0288 3E00             LD A,00
028A D301             OUT (01),A
028C C30018           JP ZBENTRY
                ;
028F 3E00     ROMST:LD A,00
0291 32C4FF           LD (ATRNMK),A
0294 3E1F             LD A,1F
0296 D304             OUT (04),A
0298 3EF0             LD A,F0
029A D305             OUT (05),A
029C 3E0F             LD A,0F
029E D302             OUT (02),A
02A0 3E0C             LD A,0C
02A2 D303             OUT (03),A
```

```
02A4 3E07           LD A,07
02A6 D300           OUT (00),A
02A8 3E00           LD A,00
02AA D301           OUT (01),A
02AC C30018         JP ZBENTRY
                    ;
02AF 7E       BRSET:LD A,(HL)
02B0 32CEFF         LD (BRKCODE),A
02B3 36FF           LD (HL),FF
02B5 C32900         JP LOOP
                    ;
02B8 3E80     TRACE:LD A,80
02BA 32D1FF   LD (TRSW),A
02BD DB37           IN A,(37)
02BF E6DF           AND DF
02C1 D337           OUT (37),A
02C3 C39701   JP RET
                    ;
02C6 21DB02   TRSET:LD HL,TRSET2
02C9 3E02           LD A,02;1/4 PRE-SCALL SINGLE PULSE
02CB D321           OUT (21),A;COUNTER-B CH1 CONTROLL
02CD DB21           IN A,(21);DUMMY
02CF 3E07           LD A,07
02D1 D320           OUT (20),A
02D3 3E00           LD A,00
02D5 D320           OUT (20),A
02D7 F1             POP AF;3+3=6
02D8 E5             PUSH HL;4+3=7
02D9 ED4D           RETI;7+4=11
02DB 2AE4FF   TRSET2:LD HL,(LREG);5+3=8 TOTAL 32
02DE FB             EI;2+1=3
02DF ED45           RETN;4+4=8
                    ;
02E1 E5       INTDP:PUSH HL;4+1 CLOCK
02E2 F5             PUSH AF;4+1
02E3 2ACAFF         LD HL,(LEDAD);5+3
02E6 7E             LD A,(HL);2+1
02E7 D3CC           OUT (CC),A;4+2+2
02E9 7D             LD A,L;1+1
02EA D3C8           OUT (C8),A;4+2+2
02EC 3D             DEC A;1+1
02ED F6F8           OR F8;2+2
02EF 6F             LD L,A;1+1
02F0 22CAFF         LD (LEDAD),HL;5+3
02F3 F1             POP AF;3+1
02F4 E1             POP HL;3+1
02F5 FB             EI;2+1
02F6 ED4D           RETI;7+2
                    ;TOTAL=75(7.5mycrosec)
                    ;
```

```
02F8 F3       NDLEDON:DI
02F9 21E102           LD HL,INTDP
02FC 22FEFE           LD (IR15),HL
02FF DB37             IN A,(37),A
0301 E67F             AND 7F
0303 D337             OUT (37),A
0305 3E01             LD A,01
0307 32C6FF           LD (EIMK),A
030A FB               EI
030B C9               RET
                ;
030C DB37     NDLEDOFF:IN A,(37)
030E F680             OR 80
0310 D337             OUT (37),A
0312 AF               XOR A
0313 D3CC             OUT (CC),A
0315 C9               RET
                ;
0316 3A42F4   NDSEGDP:LD A,(BP)
0319 E60F             AND 0F
031B 6F               LD L,A
031C 2600             LD H,00
031E 11F8FF           LD DE,LSEG1
0321 19               ADD HL,DE
0322 3A40F4           LD A,(AP)
0325 77               LD (HL),A
0326 C9               RET
                ;
0327 2140F4   NDADDSP:LD HL,AP
032A 11ECFF           LD DE,DATAL
032D 010400           LD BC,$0004
0330 EDB0             LDIR
0332 DD211220         LD IX,ADDSP
0336 CD1B05           CALL BKCG6
0339 C9               RET
                ;
033A DD211520 NDKEYIN:LD IX,KEYIN
033E C34503           JP NDINPUT2
                ;
0341 DD211820 NDINPUT:LD IX,INPUT
0345 CD1B05   NDINPUT2:CALL BKCG6
0348 6F               LD L,A
0349 2600             LD H,00
034B 2240F4           LD (AP),HL
034E C9               RET
                ;
034F 3A40F4   NDSOUND:LD A,(AP)
0352 DD211B20         LD IX,SOUND
0356 CD1B05           CALL BKCG6
0359 C9               RET
```

```
                ;
035A DD212A20 NDCHRPRT:LD IX,CHRPRT
035E CD1B05           CALL BKCG6
0361 C9               RET
                ;
AD              =F300  ADDSP          =2012  ADRSH          =FFEF
ADRSL           =FFEE  ADSET          =0091  AP             =F440
AREG            =FFEB  ATRNMK         =FFC4  BF             =F000
BKCG6           =051B  BOOTENTRY      =0017  BOOT_BREAK     =0143
BOOT_IN         =00CD  BOOT_OUT       =00BE  BOOT_RUN       =0194
BOUT            =0114  BOUT2          =0116  BOUT3          =0122
BOUT4           =012F  BP             =F442  BRKA           =FFF0
BRKC            =FFF2  BRKCODE        =FFCE  BRSET          =02AF
B_BRK2          =015A  CHRPRT         =202A  CMDRD          =003B
CREG            =FFE8  DAS            =1400  DATAH          =FFED
DATAL           =FFEC  DISP1          =FFF4  DISP3          =FFF6
EIMK            =FFC6  EREG           =FFE6  FREG           =FFEA
INPUT           =2018  INTDP          =02E1  IOREG          =FFD0
IR13            =FEFA  IR15           =FEFE  IREG           =FFD3
KEYIN           =2015  LDSH           =FFD4  LEDAD          =FFCA
LOOP            =0029  LREG           =FFE4  LSEG1          =FFF8
LSEG2           =FFF9  LSEG3          =FFFA  LSEG4          =FFFB
LSEG5           =FFFC  LSEG7          =FFFE  LSEG8          =FFFF
NDADDSP         =0327  NDCHRPRT       =035A  NDINPUT        =0341
NDINPUT2        =0345  NDJMP          =0251  NDKEYIN        =033A
NDLEDOFF        =030C  NDLEDON        =02F8  NDMON          =2000
NDSEGDP         =0316  NDSOUND        =034F  PCL            =FFE0
RAMST           =026C  RBH            =00D6  RBH1           =00E1
RBL             =00F0  RBL2           =00FB  READ           =00A8
READ2           =00B2  RET            =0197  RET2           =01C7
RMODE           =FFCF  ROMST          =028F  RS7JA          =FFBB
RS7JC           =FFBA  SEGDP          =200C  SOUND          =201B
SPL             =FFE2  SPTOP          =FFBA  START2         =01C9
S_MODE          =FFC9  S_RATE         =FFC8  S_STATUS       =FFC7
TEST            =0070  TEST1          =007A  TEST2          =0084
TRACE           =02B8  TRADRS         =FFCC  TRSET          =02C6
TRSET2          =02DB  TRSW           =FFD1  USTOP          =F800
Z6000           =6000  ZBENTRY        =1800  ZBMODE         =0236
ZBMODE2         =023D  ZBOOT          =0227
```