# ND80ZⅢ　アセンブラ・逆アセンブラ操作説明書

(有)中日電工

# 目次




〒463-0067 名古屋市守山区守山2－8－14
パレス守山305
有限会社中日電工
TEL052-791-6254 Fax052-791-1391
E-mail thisida@alles.or.jp
Homepage http://www.alles.or.jp/~thisida/



2010．9．29 Rev． 1．0

# Ⅰ．　ND80ZⅢ附属CDROMに含まれているアセンブラ、逆アセンブラ

ND80ZⅢ組立キットに附属しているCDROMには、ND80ZⅢのためのプログラム開発ツールとして、下記のアセンブラ、逆アセンブラが含まれています。

1）8080アセンブラ　ASM80．COM
2）Z80アセンブラ　ZASM．COM
3）Z80逆アセンブラ　ZDAS．COM

1）8080アセンブラは、8080のための命令語を記述するのに、インテル表現（インテルニーモニック）を使って書いたソースプログラムをマシン語に翻訳するのに使います。
　TK80モニタプログラムのリストはインテルニーモニックで書かれています。
　TK80のために書かれたソースプログラムもその多くはインテルニーモニックで書かれています。
　Z80に比べて8080の命令はうんと少ないので、マシン語ははじめてという方は、8080アセンブラから入ったほうが、わかりやすいでしょう。
2）Z80アセンブラは、Z80のための命令語を記述するのに、ザイログ表現（ザイログニーモニック）を使って書いたソースプログラムをマシン語に翻訳するのに使います。
Z80はマシン語レベルでは、8080の全ての命令をそのまま実行することができますから、8080アセンブラでZ80のためのプログラムを作成しても全く問題はありません。
しかし8080アセンブラは、8080にはなくてZ80のみにあるZ80固有の命令をマシン語に翻訳することはできません。8080の命令に慣れてきたら、つぎはZ80アセンブラに挑戦してみてください。
　ZB3BASICは、Z80アセンブラで作成されたものです。
3）逆アセンブラは特殊な機能なので、一般に使われることは余りないと思われます。
対象をND80ZⅢで実行できるプログラムに限定した場合、8080のために書かれたプログラムは、Z80ニーモニックに翻訳することが可能ですから、「8080逆アセンブラ」がなくても、Z80逆アセンブラがあれば十分なはずです。

以下、簡単にアセンブラ、逆アセンブラについて説明したあと、ASM80．COM、ZASM．COM、ZDAS．COMのそれぞれについて、使い方を説明します。

[注記1]ASM80．COM、ZASM．COM、ZDAS．COMは、x86アプリケーション（16ビットアプリケーション）なので16ビット互換モードが使えない、Windows7（64ビット版）、WindowsXP（64ビット版）では実行することができません。
Windows7（32ビット版）Home　Premiumエディションでは動作することを確認しています。

[注記2]WindowsXPは、コマンドプロンプトを開いて、ASM80．COM、ZASM．COM、ZDAS．COMを実行する前に少なくとも1回は、下記の操作をしないとメッセージがコマンドプロンプト画面に表示されません（debug[Enter]と入力すると、－が表示されますから、q[Enter]と入力してください）。
　＞debug[Enter]
　－q[Enter]　　q　はアルファベットの　Q　キーです。

[注記3]Windows7で、ASM80．COM、ZASM．COM、ZDAS．COMを実行すると、それ以前にコマンドプロンプト画面に表示されていた日本語の表示が？に文字化けしてしまいます。これはWindowsVista以降16ビットアプリケーションが日本語環境で実行できなくなったため、実行時に一時的に英語表記のコマンドプロンプトが起動されるためで、エラーではありません。プログラムの実行は正しく行われます。

## 1．1　アセンブラ

メモリにマシン語プログラムを書き込む場合に、短いプログラムならマシン語コードを直接、ND80ZⅢの16進キーボードから入力して書いていくこともできます。
　もともとTK80の時代にはアセンブラなどという機能は使いたくてもできなかったので、そうするしか仕方がなかったのです。
　人間がマシン語（2進数あるいは16進数コード）でプログラムを書くという場合、長いプログラムになればなるほど、その作業の困難さは飛躍的に増大してきます。
　マシン語コードそのままの形でプログラムを書くことは実用的ではありませんから、普通はまず紙の上にエンピツでニーモニック（注＊）でプログラムを書いていくことになります。
　たとえば、
　MVI　A，12
　INR　A

というような調子です。
これなら命令全部を覚えていなくてもわからなくなったら8080命令説明書を参照するようにしながらプログラムを書いていくことができます。
あとから見なおしたときなどでも、どういうプログラムを書いたのか、ということがわかります。
これをいきなりマシン語コードで、
3E123C…
などと書いたりすると、あとから見たとき、何が書いてあるのかさっぱりわからなくなってしまいます。

さて紙の上にニーモニックでまずプログラムを書くところまではできたとして、それからあとが大変です。
ニーモニックで書いた命令を今度はひとつずつマシン語コードに置き換えていかなくてはなりません。
ひとつずつ8080命令説明書を見ながらマシン語コードに置き換えていく作業は退屈きわまりない作業です。
30バイトや50バイトていどのプログラムならば、それでもなんとかやりとげることもできるでしょうが、数百バイト以上の規模ともなると、もういやになって投げ出したくなってしまいます。

そこを辛抱強く我慢して最後までやりとげたとしても、人間がする作業ですから、なかにはマシン語コードを間違えて書いてしまったりすることもでてきます。
それに気がつかないでND80Z3のメモリに書き込んで実行させると、いきなり暴走してしまったり、そこまではいかなくても、期待した通りに動作してくれなくて悩むことになります。
マシン語コードはミスすることなく正しく書いたとしても、長いプログラムともなれば、一度で完全なプログラムができる事は希なことになります。途中何回も変更したいところがでてきます。
こうなるとマシン語プログラミングは、大変な作業になります。
途中に命令を追加したいと思っても、直接そこに追加することはできません。
たとえ1命令でも追加するとなると、そこから後ろの命令のアドレスが全部変わってしまうために、JMP命令やCALL命令のアドレス指定部分を全部直さなくてはいけません。
それは現実的ではありませんから、普通は追加したい部分にプログラムの後ろの方のアドレスへのJMP命令を書いて、そのJMP先のメモリアドレスに追加したい命令を書いたあとで、またもとの続きのアドレスに戻るようなプログラムを書くことになります。
修正が重なってくると、JMP命令ばかりが目立つわかりにくいプログラムになってしまいます。

（注＊）ニーモニックとはマシン語の命令に、人間が理解できるような名前をつけたものです。
たとえばコード3CはAレジスタをインクリメント（＋1）する命令なので、INR　Aと書きます。INRは英語のIncrementという単語がもとになっています。

そのような困難さは、アセンブラを使うことでほぼ解消することができます。
アセンブラとはニーモニックで書いたプログラムを自動的にマシン語コードに翻訳するプログラムのことです。
ニーモニックのプログラムはファイルの形でハードディスクに格納しておきます。もとになるプログラムファイルですからソースプログラムファイルといいます。
人間が普通に読める文字（アルファベットと数字、記号）で書きますから通常はNotepad（メモ帳）などのテキストエディタを使います。もっともシンプルな形のテキストファイル（．txt）です。
アセンブラはマシン語プログラムでは困難な以下のような機能をもっています。

①プログラムの追加、削除を簡単に行うことができる。
②JMP先アドレスやCALLアドレスを直接指定しなくても、ラベルの形で指定することができる。
③ワークエリアなどのメモリアドレスも直接指定しなくても、変数名として指定することができる。
④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くことができる。
⑤ニーモニックプログラムのまま（テキストファイルとして）、ハードディスクにSAVEしておくことができる。

## 1．2　逆アセンブラ

アセンブラはマシン語の命令コード（16進数コード）で直接書く代りに、人間が理解しやすいように簡単な英語をベースにした命令（ニーモニック）で書いたソースプログラムを、翻訳プログラムにかけてマシン語に翻訳するものですが、これと全く逆の働きをするのが逆アセンブラです。
逆アセンブラはマシン語の16進コードで書かれたプログラムファイル（バイナリファイル）を読み込んで、それから逆にニーモニック命令に翻訳します。
アセンブラに比べて逆アセンブラは特殊な機能なので、一般には使うことは余りありません。
アセンブラのソースプログラムがなくなってしまって、バイナリファイルだけが残っているときに、そのプログラムを変更したい、というような場合に、逆アセンブラが必要になります。
逆アセンブラは、他人が作ったマシン語のプログラムを解析する目的で使われることが多いようです。

## Ⅱ． 8080アセンブラ ASM80．COM

### 1．1 アセンブラプログラムの作成

マシン語コードのプログラムは、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（ソースプログラム）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。
8080アセンブラの作業はND80ZⅡをUSBで接続しておく必要はありません。
Windows98やWindowsXPやWindows7の上だけで行います。
元になるプログラム（ソースプログラム）はNotepad（メモ帳）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などで作成するとファイルの前後に余計な情報をいっぱい書き込んでしまいます。
ASM80．COMはテキストイメージ（プレーンテキスト）で保存されたファイルしか読み込めないので、それ以外のファイルをASM80．COMに読み込ませると理解できないエラーで止まってしまいます。
ソースプログラムの作成にはNotepadか、あるいはプレーンテキストで保存できるようなテキストエディタを使うようにしてください。

### 1．2 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 $8000
ADRS=$8100
MON_ENTRY=$003B
;
MVI B,00
LXI H,ADRS
XRA A
LOOP:MOV M,A
INX H
DCR B
JNZ LOOP
JMP MON_ENTRY
;END
```

ORGはプログラムの先頭に書きます。
ORGは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8000番地からのアドレスに作成したいわけですから、ORG ＄8000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、8000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。

[注記1]ND80ZⅢのRAMアドレスは8000から始まります。
RAMは8000～FFFFのアドレスに割り当てられていますが、83××番地台はTK80モニタプログラムのワークエリアになっていますから、そこにユーザーがプログラムを書くことはできません。
それ以外の、8000～FFFFの範囲ならどこにでもプログラムを書くことができますが、特別の目的がなければ、8000番地からプログラムを書くようにするとよいでしょう。
ND80Zモニタプログラムではffxx番地台をワークエリアとして使っています。
またZB3BASICではE000～FFFFをワークエリアとしてリザーブしています。
TK80モニタのもとでの動作に限定すれば、83XX番地台にプログラムやデータを置かないようにさえすれば支障はありませんが、8080アセンブラで作ったプログラムを、あるいはND80Zモニタ やZB3BASICでも実行させるかもしれない、と考えた場合には、E000～FFFFの範囲にもプログラムやデータを置かないようにしたほうが無難です。

[注記2]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。

[注意]行のはじめから命令までの間の空白は許されますが、命令の中での空白は、必要な1文字分のスペース以外には許されません。
例)空白を△で示す
△△△MVI△A，１２　……正しい
JMP△△△LOOP　……誤用(JMP△LOOPにしなければならない)

１．３　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ASM80．COMと同じ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、ASM80．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。

１．４　アセンブラの実行

8080アセンブラ(ASM80．COM)はnd80z3フォルダにあります。
ND80ZⅢリモート接続プログラムを起動するときと同じように、DOSプロンプト(コマンドプロンプト)を開きます。
[注記]8080アセンブラの作業を行うときにはND80ZⅢを接続しておく必要はありません。

ASM80　ASM8TST．TXT[Enter]と入力します(１．２で作ったソースファイルはASM8TST．TXTの名前でSAVEされていることとします)。

C:￥nd80z3＞ASM80　ASM8TST．TXT[Enter]

[注記]ソースプログラムは半角英大文字を使いますが、コマンドプロンプトで入力するコマンドやファイル名は大文字でも小文字でも構いませんが、半角でなければなりません。

短いプログラムならば瞬時に翻訳が完了して、エラーがなければ下のように表示されます。

```
2009/10/6  8:53  asm8tst.txt
END=800E
```

このとき作業中のフォルダ(nd80z3)には、ソースファイルと同じファイル名で拡張子だけが違う、次のファイルが作られます。
ASM8TST．BIN　マシン語のバイナリファイルです。ZB3BASICの／LDコマンドでロードするファイルです。
ASM8TST．BTK　TK80モニタ、ND80ZモニタのLOADキー操作でロードするファイルです。
ASM8TST．LST　アドレスとマシン語コードとアセンブラニーモニックが対比できるリストです。

ASM8TST．LSTはNotepadなどのテキストエディタで開くことができます。
下はASM8TST．LSTの内容です。

```
2009/10/6  8:53  ASM8TST.TXT
END=800E
                ;;; TEST PROGRAM
                ;
                ORG $8000
                ADRS=$8100
                MON_ENTRY=$003B
                ;
8000 0600       MVI B,00
8002 210081     LXI H,ADRS
8005 AF         XRA A
```

```
8006 77        LOOP:MOV M,A
8007 23        INX H
8008 05        DCR B
8009 C20680    JNZ LOOP
800C C33B00    JMP MON_ENTRY
               ;END
ADRS          =8100  LOOP          =8006  MON_ENTRY     =003B
```

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、上で説明した3種類のファイルが作成されます。

## 1．4　ND80ZⅢ（TK80モニタ）での実行

ASM80アセンブラはND80ZⅢの接続を必要としませんが、ASM80．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ND80ZⅢの接続が必要になります。
以下、上で作成したASM8TST．BTKファイルをND80ZⅢ（TK80モニタモード）に送って、実行するまでの順序を説明します。

1）先にND80ZⅢの［＊（I／O）］キーを押します（TK80モニタモードでは、［＊（I／O）］キーは［LOAD］キーとして働きます）。

2）次にパソコンのコマンドプロンプト上で次のように入力します。

＞HIDWR　ASM8TST．BTK［Enter］

ただちに送信が開始され、送信が完了すると、ND80ZⅢの7セグメントLEDには、プログラムの開始アドレスと終了アドレスが表示されます。

3）受信が完了してプログラムの開始アドレスがND80ZⅢの7セグメントLEDのアドレス表示部に表示された状態でそのまま［RUN］キーを押せば、プログラムが実行されます。
［MON］キーを押してリセットしたあとも、プログラムはそのまま残っていますから、［8］［0］［0］［0］［ADRSSET］［RUN］とキー入力して、プログラムを再度実行させることができます（プログラムの開始アドレスが8000番地の場合の例です）。

# 2．　アセンブラで使用できる命令

## 2．1　8080ニーモニックの全命令

表現形式、文法はインテル社のアセンブラに準拠します。詳しくは「8080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8080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。プログラムの先頭に一回のみ使用することができます。省略することや、複数回使用すると、ND80Z Ⅲの上で正しく実行されません。
nnnnは4桁の１６進数です。
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 ＄8000

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)。
; の後ろに限って(その行の終わりまでの間に)、半角の英小文字、記号も書くことができます(全角は使えません)。

[使用例]
;;; TEST PROGRAM
ORG $8000
MVI A,80;*** test 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＝＄8100

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:INX H
:
JNZ 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］
（ASM80実行後のリストで示します。JMP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してください）

```
8007 3412          DW $1234
8009 0B80          DW END
800B C33310      END:JMP $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］
（ASM80コマンド実行後のリストです）

```
8050　41424344　　”ABCD”
8054　3233334　　”234”
```

2．3　定数

　8080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。

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）。STA、LDA、LXI命令やCALL、JMP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、JMP、MOV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、JMP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、LXIやJMP、CALLなどでは何回使用しても構いません。

2．5　ASM80実行時のエラーメッセージ

　ASM80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。
　スペースにも意味があります。例えば、MOV　A,Bと書くべきところを、MOVA,B（VとAの間にスペースが無い）やMOV　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。

3. ソースプログラムのレイアウト

プログラムの見やすさを考えて、行の前には複数の空白を置くことができるようになっています。
なお、;(セミコロン)の後ろ以外では、命令やラベルの間やその後ろには、1文字の空白が必要なところを除いては余分な空白を置くことはできません。
TABも使用できますが、アセンブル後に出力されるリストにはTAB位置通りの空白は置かれません。
行の前にスペースを置いたプログラム例です。

[ソースプログラム]
```
;;; TEST PROGRAM
;
     ORG $8000
      ADRS=$8100
       MON_ENTRY=$003B
;
     MVI B,00
      LXI H,ADRS
       XRA A
LOOP:MOV M,A
      INX H
     DCR B
    JNZ LOOP
   JMP MON_ENTRY
;END
```

[ASM80の実行によって出力されたリスト]
```
2009/10/6  9:35  ASM8TST3.TXT
END=800E
               ;;; TEST PROGRAM
               ;
                     ORG $8000
                      ADRS=$8100
                       MON_ENTRY=$003B
               ;
8000 0600             MVI B,00
8002 210081            LXI H,ADRS
8005 AF                 XRA A
8006 77          LOOP:MOV M,A
8007 23                INX H
8008 05               DCR B
8009 C20680          JNZ LOOP
800C C33B00         JMP MON_ENTRY
               ;END
ADRS          =8100  LOOP          =8006  MON_ENTRY     =003B
```

## Ⅲ. Z80アセンブラ ZASM. COM

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。
そうなると次のようなことのできる機能が欲しくなります。

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(ソースプログラム)の状態でハードディスクにSAVEしておきたい。

マシン語プログラムは、たとえばリモート接続をしないND80ZⅢでしたら、電源ONのあと、プログラムを書き始める先頭アドレスを指定するだけで、すぐに書き始めることができます。
またZB3BASICのマシン語モニタコマンドのCMコマンドも、ZB3BASICが起動している状態なら、いつでもマシン語の命令コードを任意のアドレスに書くことができます。
実行する場合も、スタートアドレスをセットして[RUN]キーを押すか、JPコマンドを使うだけで、実行を開始することができます。
それに比べると、アセンブラは少し手間がかかります(かけただけの価値はありますが)。

### 1. アセンブラプログラムの作成

マシン語プログラムの場合には、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。
最初に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。

それでは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ZB3BASICを起動させてその上で行うのではなくて、通常のWindows98やWindowsXPやWindows7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)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。
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 $8004
ADRS=$8100
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。
ORGは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。
ZB3BASICシステムで実行するマシン語プログラムは、ふつうは8004番地からのアドレスに作成します。
ですからORG ＄8004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、8004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。
［注記]ZB3BASICで使うのではなくて、ND80Zモニタで動作するプログラムとして作成する場合には、プログラムの開始アドレスは8000にしても構いません。

## 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ハードディスクに保存します。
できればZASM. COMと同じフォルダ（nd80z3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nd80z3フォルダにあります。
ND80ZⅢリモート接続プログラムを起動するときと同じように、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ND80ZⅢ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:￥nd80z3＞ZASM TESTASM. TXT[Enter]

[注記]ソースプログラムは半角英大文字を使いますが、コマンドプロンプトで入力するコマンドやファイル名は大文字でも小文字でも構いませんが、半角でなければなりません。

短いプログラムならば瞬時に翻訳が完了して、エラーがなければ下のように表示されます。

```
2010/9/16  22:41  testasm.txt
END=8010
```

このとき作業中のフォルダ（nd80z3）には、ソースファイルと同じファイル名で拡張子だけが違う、次のファイルが作られます。
TESTASM. BIN　マシン語のバイナリファイルです。ZB3BASICの／LDコマンドでロードするファイルです。
TESTASM. BTK　TK80モニタ、ND80ZモニタのLOADキー操作でロードするファイルです。
TESTASM. LST　アドレスとマシン語コードとアセンブラニーモニックが対比できるリストです。

TESTASM. LSTはNotepadなどのテキストエディタで開くことができます。
下はTESTASM. LSTの内容です。

```
2010/9/16  22:41  testasm.txt
END=8010
                ;;; TESTPROGRAM
                ;
                ORG $8004
                ADRS=$8100
                REENT=$1033
                ;
8004 0600       LD B,00
8006 210081     LD HL,ADRS
```

```
8009 AF        XOR A
800A 77        LOOP:LD (HL),A
800B 23        INC HL
800C 10FC      DJNZ LOOP
800E C33310    JP REENT
               ;ENDADRS          =8100  LOOP          =800A  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、上で説明した3種類のファイルが作成されます。

## 1．4 ND80Z Ⅲ（ZB3BASIC）での実行

Z80アセンブラはND80Z Ⅲ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ND80Z Ⅲの接続が必要になります。

ND80Z ⅢをUSB接続してZB3BASICを起動してください。

ZB3BASICを起動して

＞／LD TESTASM. BIN, 8004[Enter] と入力してください。

入力後にCMコマンドで8000～8010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8100番地～81FF番地に00を格納するものです。

＞DM 8100, 81FF[Enter]と入力して8100～81FF番地のプログラム実行前の内容を確認してください。

＞JP 8004[Enter] と入力します。または

＞USR（＄8004）[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81000～81FF番地の内容を確認してみてください。

## 1．5 ND80Z Ⅲ（ND80Zモニタ）での実行

前項では、Z80BASICの／LDコマンドで、バイナリファイル（拡張子 ．bin）をロードして実行しましたが、ND80Zモニタでロードするときは、先頭に開始アドレス、終了アドレスが付加されたバイナリファイル（拡張子 ．btk）をロードします。

．btkファイルはND80Z ⅢをWindowsパソコンとUSBケーブルで接続して、HIDWR. EXEで、ND80Z Ⅲに送信します。

1）先にND80Zモニタプログラムの[＊（I／O）][D LD]キーを押して、受信スタンバイにします。

2）コマンドプロンプトで、下のように入力します。

＞HIDWR TESTASM. BTK[Enter]

すると下のように表示され、

```
s=8004,e=8010
send data 16(=10)bytes
```

ND80Z Ⅲの7セグメントLEDには、プログラムの開始アドレスと終了アドレスが 80048010 のように表示されます。

3）この状態で[RUN]キーを押せばただちにプログラムが実行されます。

# 2． アセンブラで使用できる命令

## 2．1 Z80ニーモニックの全命令

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「Z80命令説明書」を参照して下さい。

## 2．2 疑似命令

疑似命令は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 ＄8004

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。
; の後には小文字を使っても構いません。

[使用例]
```
;;; TEST PROGRAM
ORG $8004
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)
```
8007 3412       DW $1234
8009 0B80       DW END
800B 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)
```
8050  41424344       ”ABCD”
8054  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。

## 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。
しかし16進数での表記もできますから、特定のラベル名では都合が悪い場合が出てきます。
[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。
そこでこのような場合には、ラベル名の先頭にラベルを示すマークとして、＊をつけて表示します。
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[注記]
INC　HL
JP　C, ＊ABC5　……JP　C, ABC5でも同じ[注記]
JP　ABC　……JP　＊ABCでも同じ[注記]
DT01：DB　43　……＊DT01：DB　43でも同じ。

[注記]JP命令で16進数を直接表記するときは＄XXXXの形にするため、ラベルでABCDと表記しても16進数表記とは区別されます。JR命令の場合は1バイトの数値を16進数2桁でそのまま表記しますから、最初の文字がA～Fで始まるラベル名を使うと、区別できなくなるために、＊が必要になります。

## 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。

## Ⅳ. Z80逆アセンブラ ZDAS.COM

Z80逆アセンブラZDAS.COMはZ80マシン語プログラムが書かれたバイナリファイルを読み込み、マシン語の命令コード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。
そのようにして作られたソースプログラムは、Z80アセンブラZASM.COM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。
Z80逆アセンブラZDAS.COMはnd80z3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。ZB3BASIC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.COMを実行すると、逆アセンブラはそのプログラムが0000番地から書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、8004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8004番地からスタートしている場合はaaaaに8004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ASCII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]
Z80アセンブラZASM.COMのサンプルとして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 $8004
ADRS=$8100
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したあとで、コマンドプロンプトで
＞ZASM TESTASM2.TXT[Enter]と入力すると、下のように表示されて、バイナリファイル他のファイルが作られます。

```
2010/9/19  8:37  testasm2.txt
END=8012
```

逆アセンブラもND80ZⅢ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ND80ZⅢをUSB接続してZB3BASICを起動してください。

① MV 0F00, 0FFF, 8100[Enter]と入力してください。
これはプログラムを実行する前に、8100～81FFを00以外にして、プログラムを実行した結果を確認しやすくするためです。
② DM 8100, 81FF[Enter]を実行して、8100～81FFが00ではないことを確認しておいてください。
③ ／LD TESTASM2. BIN, 8004[Enter]でロードします（拡張子を間違えないように！. BINです）。
④ JP 8004[Enter]で、プログラムを実行します。
⑤ DM 8100, 81FF[Enter]で、8100～81FFがすべて00になったことを確認してください。

今回のTESTASM2. BINの実行の結果は、TESTASM. BINのときと同じ結果になります。
これでTESTASM2. TXTのようにプログラムを変更してもTESTASM. BINと同じ動作をすることが確認できました。

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。
ZB3BASICを終了して、コマンドプロンプトに戻ります。

＞ZDAS TESTASM2. BIN[Enter]
と入力してください（拡張子を間違えないように！. BINです）。
画面にはすぐに、
END
と表示されます。
＞DIR TESTASM2. ＊[Enter]と入力してみてください。
TSTASM2. TXT
TSTASM2. BIN
TSTASM2. BTK
TSTASM2. LST
TSTASM2. DTX
TSTASM2. DLS
の6つのファイル名が表示されます。
BIN、BTK、LSTはさきほどのZASMの実行によって作られたファイルです。
DTX、DLSが、今回のZDASの実行によって作られた逆アセンブラファイルです。
TESTASM2. DTXもTESTASM2. DLSも、中身はTEXTファイルです。Notepad（メモ帳）で開くことができます。
まず、TESTASM2. DLSをNotepad（メモ帳）で開いて見てみます（ファイルの種類を、「すべてのファイルにしてください」）。
TESTASM2. DLSを見ると、もとのバイナリコードがどように逆アセンブルされたかがわかります。

```
0000 0600          LD B,00
0002 210081        LD HL,$8100
0005 AF            XOR A
0006 77            LD (HL),A
0007 23            INC HL
0008 05            DEC B
0009 C20A80        JP NZ,$800A
000C C33310        JP $1033
```

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のちほど説明することにして、もうひとつのファイルTESTASM2. DTXを見てみることにします。
この拡張子が. DTXのファイル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、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:￥nd80z3＞TYPE TESTASM2. DTX[Enter]
        ORG $0000
        Z8100=$8100
        Z800A=$800A
        Z1033=$1033
        LD B,00
        LD HL,Z8100
        XOR A
        LD (HL),A
```

```
        INC HL
        DEC B
        JP NZ,Z800A
        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 ＄8004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，Z800A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ます）。Z800Aはリストのはじめの方でZ800A＝＄800A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8004番地をパラメタ指定して実行します。

```
C:￥nd80z3＞ZDAS TESTASM2．BIN 8004[Enter]
END
```

TYPEコマンドで確認してみます。

```
C:￥nd80z3＞TYPE TESTASM2．DLS[Enter]
8004 0600           LD B,00
8006 210081         LD HL,$8100
8009 AF             XOR A
800A 77             LD (HL),A
800B 23             INC HL
800C 05             DEC B
800D C20A80         JP NZ,$8400A
8010 C33310         JP $1033
```

今度はよさそうです。8004番地からスタートしています。TESTASM2．DTXも確認してみます。

```
C:￥nd80z3＞TYPE TESTASM2．DTX[Enter]
        ORG $8004
        Z8100=$8100
        Z1033=$1033
        LD B,00
        LD HL,Z8100
        XOR A
Z800A:LD (HL),A
        INC HL
        DEC B
        JP NZ,Z800A
        JP Z1033
```

ORG ＄8004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。
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TESTASM．TXT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。