Ｃｒｏｓｓ　Ａｓｓｅｍｂｌｅｒ－ＭＳ

ユーザーズ　マニュアル

も　く　じ

第４章　擬似命令

第５章　条件付きアセンブリ

第６章　マクロ機能

第７章　オブジェクトコード

第８章　エラーメッセージ

第９章　各アセンブラの詳細

付　録

第１章　はじめに

１－１　紹　介

「Cross Assembler-MS」は、アブソリュートタイプの完全な２パス形式のクロスマクロアセンブラです。

豊富な擬似命令、条件付きアセンブル、マクロ機能、クロスリファレンスの出力と高機能です。
ラベル文字は10文字まで認識し、ラベル(シンボルも含む)数は最大6500個まで使用できます。この数はソースラインでおよそ30000行以上のアセンブルが可能ということになります(ラベルの登録を5行で1回行ったという計算より)。

いずれにせよ、8ビットCPUのアドレス限界値は、FFFF(16進)ですのでそれを越えてのプログラムの作成はできません。

各CPUにおける標準ニーモニックの使い方は、本マニュアルでは説明しません。第9章の「各アセンブラの詳細」で参考書籍を記しますので、もしニーモニックの使用法が分からない方は是非とも購入してください。アセンブラを使われる方の必需品です。ただし、他にも多くの書籍があると思いますので自分にあった最高のものを選んでください。

本アセンブラには、モステック社の6502、モトローラ社の6801、6805、6809、68HC11(68HC11のアセンブラは、本文中では6811と省略する場合があります)、インテル社の8048、8051、8085、ザイログ社のZ8、Z80のバージョンがあります。
また、6801でも6800、6301などもサポートしています。詳しくは付録Aの「対応CPU一覧表」をご覧ください。

本書は、「Cross Assembler-MS」の共通マニュアルです。本文中に記載されている「X6809」などのファイルは、購入されたアセンブラの種類によっては入っていないものもあります。

本文中に68系と記述しているものは、「6502、6801、6805、6809、6811」のアセンブラを示し、80系と記述しているものは、「8048、8051、8085、Z8、Z80」のアセンブラを示します。

１－２　アブソリュートアセンブラとは?

ここ数年、コンピュータは目まぐるしく進歩しました。現在使用しているパソコンも大きく変わりました。
数年前までは、8ビットパソコンで開発していましたが、現在では8ビットパソコンは眠っており、変わりに16ビットパソコンが活躍しているといった所が現状です。
国産の8ビット機では、PC-8800シリーズ、FM-7シリーズなどがあります。しかし、一般にそれらは開発マシンとしては使用されなくなっています。

PC-8800シリーズには「CP/M-80」、FM-7シリーズには「FLEX」や「OS/9」といったOSが走ります。そして、それらのOSにはアセンブラも標準で付属していました。それらのアセンブラはいずれもアブソリュートアセンブラです。しかし、CP/M-80では、もともと8080CPU用として開発されたOSだった為、付属のアセンブラは8080アセンブラでした。その為、ほとんどのユーザーはZ80のプログラムを開発する為に、別途「MACRO-80」を購入したのではないでしょうか。そして、この「MACRO-80」は、リロケータブルアセンブラです。

アブソリュートアセンブラとは、ソース中にあらかじめそのコードを配置するアドレスを設定しておくタイプのものです。本アセンブラでは、「ORG」「CSEG」命令や「DSEG」命令を使ってアドレスを設定します。
それに対し、リロケータブルアセンブラは、ソース中にはアドレス配置情報をいっさい記述せず、リンカー実行時にアドレスを決定するといった方法を用います。ソースをモジュールごとに分割してそれぞれアセンブルし、最後にリンカーで1本にまとめるといった使い方がもっぱらです。

しかし、8ビットCPUのソフトウェアの開発は高級言語ならともかく、アセンブリ言語で記述するのならばアブソリュートで十分です。むしろ、アブソリュートの方がコードを配置するアドレスがはっきりしている為、適しているのではないかと思われます。

既に皆様は、「Cross Assembler-MS」を購入されたのですから、少なくともリロケータブルでなくては使えないなどと言われる方はいないと思います。

１－３　ソフトウェア開発手順

「Cross Assembler-MS」を使って開発するプログラムの種類は多種多様です。一番多い使用目的は制御プログラムの作成です。その為、一般にROMに焼くといった方が多いようです。
しかし、そればかりではなく、先ほど登場した8ビットパソコンのソフトの開発などを行っている方もいるのではないでしょうか。

いずれにしても、まず必要なことはメモリマップの作成です。もちろん、作成するソフトの設計書ができているということが前提です。

(1)メモリマップの作成
基本的に8ビットCPUは、64Kという小さな空間に「プログラム」「データ」「ワークエリア」などを詰め込まなくてはいけません。そして、肝心な所がRAM部とROM部の振り分けです。しかし、ROM化をしないプログラムは、そういうことを意識せず作成できます。しかし、8ビットパソコンでは、使用できるエリアとできないエリアがありますので注意してください。
ROM化する場合は、「初期化されたデータ」が問題になります。この問題に関しては「付録C」で説明することにします。

(2)プログラムの作成
プログラムの作成は各種のエディタで行います。エディタは数多くあり、こればっかりは好みというものがありますので、特にどのエディタが良いなどとは言えません。
作成したソースがMS-DOSのTYPEコマンドで見れれば、エディタに関わらずワープロでも良いわけです。
プログラム中で先ほど決めたメモリマップに従い、「ORG」「CSEG」「DSEG」などを用いアドレスを設定することを忘れてはいけません（４－１参照)。

(3)アセンブル
ここで本アセンブラを使用します。一番簡単なアセンブルの仕方は以下の通りです。

```
A>XZ80 SAMPLE;□                    (アンダーラインの所が実際入力する所)
  ~~~~~~~~~~~~~~                  (□はリターンキー、以下同じ)
```

この例ではドライブAにZ80のアセンブラ、「SAMPLE.TXT」というソースプログラムが入っている場合で、オブジェクトもドライブAに「SAMPLE.HEX」という名で作成されます。もしエラーが表示された場合は、(2)に戻ってソースを編集してください。エラーメッセージについては第８章を読んでください。
また、オブジェクトの作成とエラー行のみの表示をさせたい場合は以下のコマンドラインで実行します。

```
A>XZ80 SAMPLE; /LS□                (「/LS」はオプションです)
  ~~~~~~~~~~~~~~~~~
```

(4)ターゲットへの転送

ROM化する場合は、ROMライターの方にオブジェクトを転送しなければなりません。汎用のROMライターでは、RS-232C経由で直接オブジェクトを転送できる為、比較的簡単に行えます（７－４参照)。

また、パソコンのスロットに差し込むタイプのもので、本体のメモリの内容を直接ROMに書き込むようなものでも、付属の「LOAD」ユーティリティーを使用することでROM化が行えます（７－２、付録D参照)。

他のパソコンに転送する方法としては2通り考えられます。1つは、ディスクに直接書き込むことで、もう1つはRS-232Cで送ることです。

スタンドアローンのシステムを作ろうという時は、ディスクに直接書いた方が後々楽ですが、ディスクに対する知識が必要になってきます。この場合は、オブジェクトを読み、そのコードを自分で決めたディスクマップの通りに直接ディスクに書くプログラムを作成しなければならないということは言うまでもありません。

DISK-BASIC上で動作させるような機械語プログラムは、RS-232Cで送る方が良いでしょう。もちろん、この場合は、受け取る側でBASICなどで、オブジェクトを受け取ってディスクにセーブするプログラムを作成しなくてはなりません。

(5)デバッグ

一発で完ぺきに動けばデバッグ作業はいらないのですが、やはり人間の作るもの、必ずと言ってよいほどどこかに欠陥があります。この欠陥を捜すのがまた大変な作業です。デバッグの方法だけについて述べられている書籍もあるぐらいで、とてもここではそんなことを述べられません。

一般にクロス開発では、デバッグが一番たいへんな作業です。オブジェクトを作成した機械では、そのまま実行できないからです。やはり、実機で行うのがよいでしょう。

また、クロスのデバッガーもあります。個々の内部ルーチンのチェックなどは、これで十分できます。

当社の「Cross Debbuger-MS」シリーズには、現在の所、Z80用と6301用の2種類があります。

もし、プログラムの欠陥を発見した場合は、(2)へ戻ってソースの変更をします。

第２章　アセンブラの実行

２－１　アセンブラの実行方法

アセンブラを実行するには次のコマンドラインで行います。

```
Xxxxx [ソース], [オブジェクト], [リスティング], [クロスリファレンス],
      [シンボルテーブル]  [+/-（オプション）．．．．]
```

「Xxxxx」はアセンブラ名です。アセンブラ名に付いては付録Aを参照して下さい。

オプションを指定する場合は、[ソース]の前か、[シンボルテーブル]の後に、「/」や「+」や「-」の後に続けて指定します。
オプションの種類は、後述の「オプション」の所で説明します。

```
(1)XZ80 /BLS TEST;
(2)XZ80 TEST; /BLS
(3)XZ80 /B /L /S TEST;
(4)XZ80 /B TEST; /LS
(5)XZ80 /B TEST; /L /S
```

上記の5つの例は、いずれも同じ動作をします。

コマンドラインで指定するファイル名はすべて省略でき、省略した場合はそれぞれのファイル名を問い合わせてきます。ただし、上記の例の様にファイル名の後ろに「;」を付けた場合は、「ここでファイルの指定は終了」という意味で、これより先のファイルは問い合わせず、すぐさまアセンブル実行に入ります。

【実行その1】

ファイル名をまったく指定しなかった場合は、ファイル名の入力を促すプロンプトが表示されます。

```
A>XZ80□
  ~~~~~~
```

```
┌──────────────────────────────┐
│Source Filename [.TXT] : TEST□ │
│                         ~~~~~~ │
└──────────────────────────────┘
```

アセンブルしたいソースファイル名を指定します。
拡張子を省略した場合は、自動的に「.TXT」を付けます。つまり、上記の場合は「TEST.TXT」というファイルを指定した時と同じです。

ソースファイルは複数個指定できます。つまり、コマンドラインで「LIB」の動作を行うことができるのです。
複数個指定する場合は、「+」でファイルを結合します。

```
Source Filename [.TXT] : TEST+TEST2+TEST3+TEST4□
                         ~~~~~~~~~~~~~~~~~~~~~~~~
```

この場合はプログラム中で、

```
┌──────────────────┐
│  LIB    TEST.TXT  │
│  LIB    TEST2.TXT │
│  LIB    TEST3.TXT │
│  LIB    TEST4.TXT │
└──────────────────┘
```

と行った場合と同じ結果が得られます。
(「LIB」についての詳しいことは「４－５」を参照してください。)

「LIB」を使用すると、環境変数「ASMLIB」で指定したパスが有効になりますが、ここで指定したファイルは「ASMLIB」を無視します。

1つのプロンプトで入力できる文字数は最高60文字までです。その為、結合するソースファイルが多すぎ、1ラインで指定できない場合は、いくつかに分けて指定することができます。その場合は、一番最後に「+」を付けます。

```
Source Filename [.TXT] : TEST+TEST2+TEST3+TEST4+□  (最後に「+」を付けると
                         ~~~~~~~~~~~~~~~~~~~~~~~~   もう一度ソースファイル
Source Filename [.TXT] : TEST5+TEST6+TEST7+TEST8□   を問い合わせてくる)
                         ~~~~~~~~~~~~~~~~~~~~~~~~
```

しかし、指定できるファイル名は最高20個までです。もし、それ以上を指定した場合は、21個目以降のファイルは無視されます。

```
┌──────────────────────────────┐
│Objest Filename [TEST.HEX] : □ │
│                              │
└──────────────────────────────┘
```

出力されるオブジェクトファイル名を指定します。
単に「リターンキー」のみをタイプすると、[ ]内に表示されているファイル名を指定したことになります。このファイル名はソースファイル名の拡張子を「.HEX」(68系のアセンブラでは「.S」)に変更したものです。
また、パス名の指定もでき、その場合は、

```
Object Filename [TEST.HEX] : OBJ¥□
```

とパス名の最後に「¥」を付けます。
この場合は、「OBJ¥TEST.HEX」と指定したことになります。

もし、コマンドラインで、

```
A>XZ80 /B□          (オプションBはオブジェクトの出力を禁止するもの)
```

とオブジェクトの出力を禁止するオプションを指定した場合は、ここのプロンプトは表示されません。

```
┌──────────────────────────────┐
│Source Listting [NUL.LST] : □  │
│                              │
└──────────────────────────────┘
```

アセンブルリストファイル名を指定します。
ここではデフォルトで「NUL」、つまり出力しないという形になっています。その為、「リターンキー」のみをタイプした場合は出力しません。
ファイル名を指定した場合、そのファイル名でアセンブルリストファイルが出力されます。拡張子を省略した場合は、自動的に「.LST」が付きます。

画面上に出力されるアセンブルリストはページングしますが、ファイルに作成されるアセンブルリストファイルはページングを行いません。また、次のクロスリファレンスファイル及びシンボルテーブルファイルもページングしません。

```
┌──────────────────────────────┐
│Cross Reference [NUL.CRF] : □  │
│                              │
└──────────────────────────────┘
```

クロスリファレンスファイル名を指定します。
ここではデフォルトで「NUL」、つまり出力しないという形になっています。その為、「リターンキー」のみをタイプした場合は出力しません。
ファイル名を指定した場合、そのファイル名でクロスリファレンスファイルが出力されます。拡張子を省略した場合は、自動的に「.CRF」が付きます。

```
┌──────────────────────────────┐
│Symbol Table [NUL.CRF] : □    │
│                        ~~    │
└──────────────────────────────┘
```

シンボルテーブルファイル名を指定します。

ここではデフォルトで「NUL」、つまり出力しないという形になっています。その為、「リターンキー」のみをタイプした場合は出力しません。

ファイル名を指定した場合、そのファイル名でクロスリファレンスファイルが出力されます。拡張子を省略した場合は、自動的に「.MAP」が付きます。

上記のプロンプトでの指定は、コマンドラインで以下のように行えます。

```
A>XZ80 TEST;□
  ~~~~~~~~~~~~
```

ファイル名の後の「;」は、これより先はデフォルト名で行うということです。
また、この「;」はコマンドライン以外にファイル名入力プロンプトでも使用できます。

```
Source Filename [.TXT] : TEST;□
                         ~~~~~~~
```

【実行その2】

コマンドラインですべて指定した場合は、ただちにアセンブルを開始します。

```
A>XZ80 TEST,TEST,TEST,TEST,TEST□
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

この場合は「TEST.TXT」をアセンブルし、「TEST.HEX」という名でオブジェクトファイルを作成します。
そして、「TEST.LST」(リスティング)、「TEST.CRF」(クロスリファレンス)
「TEST.MAP」(シンボルテーブル)ファイルをそれぞれ作成します。

また、この場合のオブジェクトファイルは、デフォルトで「TEST.HEX」という名前になる為、

```
A>XZ80 TEST,,TEST,TEST,TEST□
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

と省略できます。
ただし、リスティング、クロスリファレンス、シンボルテーブルファイルは、デフォルトでは作成しませんので、作成したい場合は必ずファイル名の指定が必要です。

※ファイル名とファイル名の間は、「,」で区切り、スペースを開けてはいけません。
^^

【実行その3】

オプションやファイル名などをファイルで指定することができます。

```
A>XZ80 @TEST□
```

「@」に続き、ファイル名を指定します。

「TEST」ファイルには、

```
┌──────────────────┐
│/LS□              │※オプションを指定するときは、必ず第一行目で行って
│TEST□             │
│□                 │  下さい。
│TEST□             │
│TEST□             │
│TEST□             │
└──────────────────┘
```

が入っていると、上記のコマンドラインでは、

```
A>XZ80 /LS TEST,,TEST,TEST,TEST□
```

と入力した時と同じ動作をします。

２－２　オプション

オプションとは、アセンブル時のデフォルトの動作を変更したりするものです。

次の項目がデフォルト値で、これらの動作はオプション指定により切り替えられます。

・「TTL」「STTL」「PAG」などの命令行は表示しない
・オブジェクトコードを生成する
・条件付きコードの表示をする
・マクロ拡張ラインの表示をしない
・1ラインの表示を80文字に切り詰める
・複数行に渡るオブジェクトの表示を省略する
・68系のアセンブラはSフォーマット、80系はHEXフォーマットで出力する
・アセンブルリストを画面に表示する
・マクロ定義、マクロコール行の表示を行う
・「ORG」などで設定したアドレス通りにアセンブルする
・ページングを行う
・シンボルテーブルを画面に表示する
・ファイル単位で、行番号を表示する
・ワーニングエラーも表示する
・クロスリファレンスは画面に表示しない
・行番号が5桁より小さい場合は頭の所をスペースで表示する

オプション	A
説　明	ページングを行うと「STTL」「TTL」や「PAG」などがリスト上に反映される為、これらの命令は行から消えますがこのオプションを指定すると行の方にも表示します。 具体的に反映される命令は「PAGE,PAG,PG,*EJECT,LIST,*LIST ON,XLIST,*LIST OFF」のリストページング命令と「TTL,NAM,NAME,TITLE,STTL,SUBTTL,*HEADING」のリストタイトル、サブタイトル命令です。
用　途	「LIST」ユーティリティーを使ってリストファイルを出力する場合に有用です。
例	X6502 /A TEST6502;

オプション	B
説　明	オブジェクトファイルの生成を禁止します。
用　途	エラーのチェックやアセンブルリストをプリンタへ出力する時などに使用します。
例	X6801 /B TEST6801;

オプション	C
説　明	条件付きコードの表示を禁止します。 具体的には「IFxx,COND,ELSE」などの条件付きアセンブリに関する命令です。
用　途	真のリスト表示が行えます。
例	X6805 /C TEST6805;

オプション	D<シンボル名>[=数値]
説　明	シンボルを外部から定義します。「D」に続いてシンボル名を指定しなくてはいけません。その後に「=」に続けて数値を指定するとその値がシンボルに設定されます。 値を指定しなかった場合は、シンボルの値は「1」に設定されます。 また、数値は10進数値で指定してください。16進数値などは受け付けません。
用　途	「IF」や「IFDEF」などの条件付きアセンブリと共に使用すると良いでしょう。
例	X6809 /DDEBUG TEST;　　（「DEBUG」が1に設定される） X6809 /DTOP=100 TEST;　　（「TOP」が100に設定される）

オプション	E
説　明	マクロ拡張ラインの表示を行います。 つまり、マクロコールを行っている行以下に実際に展開されているソースを表示します。
用　途	マクロ展開中にエラーが発生した場合、実際に展開されている様子を見ないと分かりません。また、マクロのパラメータが正しいのかなどのチェックにも使えます。
例	X6811 /E TEST6811;

オプション	F
説　明	デフォルトでは、アセンブル展開して１ラインが80文字を越える様な時、それ以上の文字を表示しないようにしています。これは、80桁プリンタに合わせての仕様で１行がそれ以上になるとリスト表示が汚くなる為です。 このオプションを指定すると、表示を132文字に切り替えます。
用　途	80桁以上を印字可能なプリンタに出力する場合などには、必ず指定した方が良いでしょう。
例	X8048 /F TEST8048; >PRN　　（プリンタへ出力）

オプション	G
説　明	通常、1つの命令に付き数バイトのコードが生成されますが、擬似命令の中には数十バイトも生成するものがあります。この場合、1ラインではとてもコードを表示することはできませんので数ラインに渡って表示するようにしています。 このオプションを指定すると、数ラインに渡るようなコードを1ライン分だけ表示するようにします。 ただし、これはあくまで表示上だけの問題で省略された分のコードもオブジェクトには正常に展開されています。
用　途	リストを見やすくする為に使用します。
例	X8051 /G TEST8051;

オプション	H
説　明	オブジェクトの出力形式を変更します。 80系のアセンブラはモトローラSフォーマットで、68系のアセンブラはインテルHEXフォーマットでそれぞれ出力するようになります。
用　途	転送する先のハードが、デフォルトのフォーマットでは受け付けない場合に使用します。
例	X8085 /H TEST8085;

オプション	L
説　明	アセンブルリストを画面に表示せずにアセンブルします。
用　途	エラー行のみの表示を行いたい場合や、オブジェクトコードの出力のみを行いたい場合など、あえてアセンブルリストを見なくても良い場合に使用します。
例	XZ8 /L TESTZ8;

オプション	M
説　明	マクロ定義を行っている行やマクロコールを行っている行の表示を行わないようにします。
用　途	Mオプションだけ指定すると、正確なリスト表示が行えません。通常、Eオプションと共に使用します。このようにした場合、本来のアセンブルリストが表示されることになります。
例	XZ80 /EM TESTZ80;　　(Eオプションも付けた場合)

オプション	O
説　明	「ORG,CSEG」など、プログラムアドレスを設定する命令の指示を無効にします。つまり、プログラムは0番地からの通しアドレスになります。ただし「DSEG」などデータアドレスを設定する命令には影響しません。
用　途	デバッグなどの時に使用される可能性があります。
例	X6502 /O A6502;

オプション	P
説　明	ページングを禁止します。 もちろん、ページングを行わないとヘッダータイトルの表示はしません
用　途	プリンタにきれいにアセンブルリストを出力するときは、ページングはかかせませんが、エラー行だけの表示を行いたい場合は、かえってページングを行わない方が良いかもしれません。
例	X6801 /P A6801;

オプション	S
説　明	シンボルテーブルを画面に表示しません。
用　途	これもLオプションなどの様に、シンボルテーブルの表示をしたくない時に使用します。
例	X6805 /S A6805;

オプション	T
説　明	通し番号をアセンブルリストに付けます。
用　途	アセンブルエラーを修正する場合は、デフォルトの行番号が良いのですが、最終リストを出力する場合は、やはり通し番号の方が良いと思います。
例	X6809 /T A6809;

オプション	W
説　明	ワーニングエラーの表示を禁止します。 本アセンブラでは「Value out of range」がその対象になります。
用　途	ワーニングエラーは、 プログラムの誤りを見つけるのにも役立ちますが、場合によっては、そのエラーを無視したいことがあります。
例	X6811 /W A6811;

オプション	X
説　明	クロスリファレンスリストを表示します。
用　途	クロスリファレンスを表示させたい場合に指定します。
例	X8048 /X A8048;

オプション	Y
説　明	アセンブルエラーのチェックのみを行いたい時に使用します。 このオプションは、「/BLS」と指定した場合とまったく同じ動作をします。
用　途	プログラムを作成した後、初めてアセンブルする場合は、このオプションだけ付ければ良いでしょう。一番高速にアセンブル作業が行えます。 ただし、オブジェクトの出力も行いませんので、出力させたい場合は「/LS」と指定してください。
例	X8051 /Y A8051;

オプション	Z
説　明	行番号の表示が5桁に満たない場合、頭の空白の部分を「0」と表示します。
用　途	行の表示を 「　　1」とするよりも 「00001」としたい場合に使用します。
例	X8085 /Z A8085;

２－３　ラベルの最大登録数の拡張

ラベルは、プログラムでのジャンプ先、コール先などのアドレスを現わすもので、シンボルとは、「EQU」などの擬似命令で故意的に数値に名前を付けたものです。
本アセンブラでは、それらの区別を行っていません。その為、シンボルテーブルといってもシンボルだけの出力をするのではなく、ラベルも共に出力されます。
すなわち、ラベルの最大登録数の拡張は、シンボル数の拡張でもあります。

デフォルトでは4000個までラベル(シンボル)が使用できます。ラベル数がそれ以上になった場合、「Symbol table overflow」エラーが発生しアセンブルを中止します。
そのような時、ラベルの最大登録数の拡張を行います。

アセンブル時に「#」に続いて数値(0～9までの値)を指定することで、ラベルの最大登録数の拡張ができます。コマンドラインで「#」を指定する場所はオプションを指定する所と同じ位置です。

「#」の後に指定する数値でシンボル数が決定します。

#0	最大500個	｜
#1	最大1000個	｜ 縮小
#2	最大2000個	｜
#3	最大3000個	｜
#4	最大4000個	(デフォルト値)
#5	最大5000個	｜
#6	最大6000個	｜
#7	最大6500個	｜ 拡大
#8	最大6500個	｜
#9	最大6500個	｜

すなわち、デフォルトの4000個は「#4」と指定したのと同じことになります。

```
例)   A>XZ80 BIGSIM; #9□          (最大値を指定)
        ~~~~~~~~~~~~~~~~
```

もちろん、オプションなども同時に指定できます。

```
例)   A>XZ80 BIGSIM; /BLS #9□
        ~~~~~~~~~~~~~~~~~~~~~
```

●注　意

ラベルの最大登録数を拡張するとそれだけメモリを消費します。
基本的にメモリの消費は「最大登録数×10バイト」です。すなわち、デフォルトの4000個では「4000×10」で約40Kバイト使用されることになります。このエリアはアセンブラ起動時に確保されます。

残りのエリアが、ラベル名やマクロ関係に使用されることになります。つまり、ラベル名の長さやマクロ定義の数などによってそのエリアがどれだけ消費されるかが決まります。
残りエリアが足りなくなりラベル名の登録ができなくなった場合、「Symbol table space full」というエラーを出力しアセンブルを中止します。この場合は最大登録数を減らしてみてください。それでもエラーになる場合は、メモリに常駐するプログラム(フロントエンドプロセッサなど)を取り除いてアセンブルして見てください。それでも駄目な場合はソースを変えなければどうしようもありません（シンボル名の縮小、マクロ定義の減少など)。

また、内部でマクロ登録を行っている際に残りエリアが足りなくなった場合、「Macro table space overflow」というエラーを出力しアセンブルを中止します。この場合も上記のような方法で解決してください。

２－４　フィールド位置の変更

アセンブリソースをアセンブルすると、「ラベル」「オペコード」「オペランド」「コメント」というようにそれぞれのフィールドにセパレートして出力されます。
つまり、

```
START LDS #STACK スタックの設定
 JSR WKINIT ワークエリアの初期化
 JSR MAIN メインプログラムの実行
 BRA * 戻って来たら無限ループ
```

と記述してもアセンブルすると、

```
    1 0000 10CEBFFF     START    LDS    #STACK     スタックの設定
    2 0004 BD1000                JSR    WKINIT     ワークエリアの初期化
    3 0007 BD2000                JSR    MAIN       メインプログラムの実行
    4 000A 20FE                  BRA    *          戻って来たら無限ループ
```

と列ぞろえをして出力されるのです。

それぞれのフィールドの位置は決まっていますが、「オペコード」「オペランド」「コメント」のフィールド位置に限り、コマンドラインで再設定が可能です。

アセンブル時に「%」に続いて数値を指定することで、フィールド位置の設定ができます。「%」を指定する場所はオプションを指定する所と同じ位置です。

「%」の後に指定する数値でフィールド位置が決定します。
数値はラベル位置からのオフセット値で、必ず2桁で「オペコード位置」「オペランド位置」「コメント位置」を続けて指定します。数値が9以下の場合は「09」などと頭に「0」を付けて指定します。

上記のアセンブルリストはデフォルトの位置で、わざわざ指定しなくても良いのですが、あえて指定すると「%112032」というようになります。つまり、「オペコード」はラベル位置から11桁目に、「オペランド」は20桁目に、「コメント」は32桁目に出力するということを意味します。

```
  1 0000 10CEBFFF   START_____LDS_____#STACK_____スタックの設定
                    ~         |       ~          ~~
                    |         |       |          |
                    0         11      20         32
```

```
例)   A>X6809 DEFAULT; %112032□
        ~~~~~~~~~~~~~~~~~~~~~~~~
```

もちろん、オプションなども同時に指定できます。

```
例)   A>XZ80 BIGSYM /B #9 %112032□
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

●注　意

設定値によっては出力リストが乱れる場合があります。

リストファイルの方も有効になります。その為、そのファイルをLISTユーティリティーを使って出力しようとした場合、オペコードの認識ができなくなりますので正常な動作をしません。

２－５　クロスリファレンスについて

クロスリファレンスとは、設定したラベルがどこで使われているかを見る為のものです。ラベル名とラベル定義の行、ラベル参照の行が表示されます。

クロスリファレンスは、以下のようなフォーマットで表示します。

```
SSSSSSSSSS AAAA NNNNN NNNNN NNNNN NNNNN NNNNN NNNNN NNNNN NNNNN NNNNN
~~~~~~~~~~ ~~~~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
|      |        |          |
|      |        |          └──  参照している行番号
|      |        |               (11行以上ある場合は次の行に渡って表示)
|      |        └─────────────  シンボルの値
|      └──────────────────────  シンボルの名前
```

クロスリファレンスの出力には、比較的大きなワーキングバッファが必要です。メモリ上に取るというのも1つの方法ですが、それを行うとマクロやラベルの為に使用するバッファが減ってしまい、大きなプログラムをアセンブルすることができなくなってしまいます。

その為、本アセンブラではワーキングバッファをディスク上に取って行っています。その時のテンポラリファイルは「XREFTBL.$$$」で、デフォルトではカレントディレクトリに作成します。

また、環境変数「ASMTMP」にディレクトリ名を登録しておくと、そこにテンポラリファイルを作成します。ワーキングエリアには、フリーエリアを大きく開けて置き、書き込み可能な状態にしておいてください。

テンポラリファイルの最小サイズは、「登録ラベル数×16」で、もちろん参照する行が多ければ多い程、それだけテンポラリファイルは大きくなります。

大きいプログラムのクロスリファレンスを出力する時、フロッピーディスクをテンポラリとして使用すると、大変時間がかかります。その為、できるだけワーキングには、RAMディスクを使用することをお奨めします。

```
例）ワーキングディレクトリにドライブCを指定する。

A>SET ASMTMP=C:□
  ~~~~~~~~~~~~~~~
```

２－６　プリンタへの出力

本アセンブラはプリンタへアセンブルリストを出力する機能は持っていませんが、MS-DOSのリダイレクション機能を使用することにより、プリンタへ出力することができます。例えば以下の様に行います。

```
例)   A>X6809 TEST; >PRN□
        ~~~~~~~~~~~~~~~~~~
```

上記の例では、「TEST.TXT」を6809のアセンブラでアセンブルし、その結果をプリンタへ出力します。

また、リストファイルを作成し、そのファイルをLISTユーティリティーで出力するという方法もあります。この場合も、リダイレクション機能を使ってプリンタに出力します。

```
例)   A>XZ80 TEST,,TEST;□
        ~~~~~~~~~~~~~~~~~~

      A>LIST TEST >PRN□
        ~~~~~~~~~~~~~~~~
```

上記の例では、「TEST.TXT」をZ80のアセンブラでアセンブルと共にリストファイルを作成します。そして、LISTユーティリティーでリストファイル「TEST.LST」をプリンタに出力します。

２－７　実行例

```
例1)  A>XZ80 TEST.Z80;□
        ~~~~~~~~~~~~~~~~
```

Z80のアセンブラで「TEST.Z80」というソースファイルをアセンブルし、オブジェクトファイル「TEST.HEX」を作成します。同時に画面にアセンブルリストをページングしながら表示し、最後にシンボルテーブルも表示します。

```
例2)  A>XZ80 TEST.Z80,B:¥;□
        ~~~~~~~~~~~~~~~~~~~~
```

例1とほとんど同じ動作をしますが、違う所はオブジェクトをドライブBに作成するという所です。

```
例3)  A>X6502 TEST; /Y□
        ~~~~~~~~~~~~~~~~
```

6502のアセンブラで「TEST.TXT」というソースファイルをアセンブルし、オブジェクトファイルも、リストもシンボルテーブルも表示しません。すなわち、エラーの発生した行だけ表示します。

```
例4)  A>X8048 TEST; /B >PRN□
        ~~~~~~~~~~~~~~~~~~~~
```

8048のアセンブラで「TEST.TXT」というソースファイルをアセンブルし、オブジェクトファイルの出力を禁止し、プリンタへアセンブルリストを出力します。

```
例5)  A>XZ8 TEST,B:OBJ¥; /LS□
        ~~~~~~~~~~~~~~~~~~~~~
```

Z8のアセンブラで「TEST.TXT」というソースファイルをアセンブルし、オブジェクトファイルをドライブBの「OBJ」ディレクトリに作成します。リスト及びシンボルテーブルの表示は行いません。

```
例6)  A>X6809 B:TEST;□
        ~~~~~~~~~~~~~~~
```

6809のアセンブラでドライブBの「TEST.TXT」というソースファイルをアセンブルします。オブジェクトファイルもドライブBに作成します。

第３章　ソースラインの書式

この章では、実際にアセンブルソースを記述する時に必要なソースラインの書き方について説明します。

各々のソースラインは４つのフィールドで構成します。「ラベル」「オペコード」「オペランド」「コメント」で、それぞれのフィールドはスペース（半角でないといけません、全角スペースは区切り文字に使えません)、またはタブにより区切ります。ただし、以下の２つはこの限りではありません。

(1)1列目が「*」や「;」や「:」で始まるライン。すなわちコメントになる行です。
(2)空白またはキャリッジリターンだけのライン。

その他は以下のように記述します。

```
[ラベル(:)]    [オペコード]    [オペランド]    [コメント]
```

どの様な場合でも1ラインは最高135文字以内でないといけません。
^^^

●ソースをどのように記述しても（桁そろえをしなくても)、アセンブルリスト上ではそれぞれのフィールドに振り分け、きれいな表示を行います。

３－１　ラベル・フィールド

```
┌──────────────────────────────────────────────────┐
│ ラベル(:)      ［オペコード］    ［オペランド］    ［コメント］ │
│ ^^^^^^^^                                         │
└──────────────────────────────────────────────────┘
```

　このフィールドは分岐命令の飛び先アドレスやシンボルの定義に使用します。
また、マクロ定義時のマクロ名もここで指定します。

(1)68系ではラベルは必ず1列目から始まらなくてはいけません。80系では「:」でラベルを認識できる為、ラベル名の後ろに「:」を付けた場合は必ずしも1列目から始まる必要はありません。また、同じ名前のラベルを付けてはいけません。ただし、「=」命令などは同名のラベル名を付けても構いません(４－３参照)。
もし、ラベルを付ける必要のない時は、68系では行の最初の文字はスペース、またはタブでなければなりません。80系では必ずしもスペースやタブを入れる必要はなく、いきなりオペコードを書いても構いません。

(2)ラベルは、文字(A～Z, a～z, _, ?)や数字(0～9)で構成します。
(ラベルは大文字、小文字を区別する為、「ABC」と「ABc」では別のラベルとして扱われます。)

(3)すべてのラベルは文字(A～Z, a～z, _, ?)で始まらなくてはいけません。
(「A0001」はOK、「9AAAA」はNG)

(4)ラベルはいかなる長さでも構いませんが、最初の10文字まで認識されます。
("NAGAILABEL"と"NAGAILABELDESU"は同じものと認識されますので二重定義エラーになります。)

(5)ラベルフィールドはスペースまたは「:」、もしくはリターンで終了します。
また、80系の場合ではラベル名にニーモニックや擬似命令と同じ名前を付ける時は、必ず「:」で終了しなければなりません。これがない場合は命令として解釈されます。
(Z80の場合:「LD:　 LD　 A,20H」など)

(6)レジスタ、演算子などと同じ名前は使用できません（第９章「各アセンブラの詳細」参照)。
(6809の場合:「A　　EQU　$0A」など)
(8048の場合:「SHR　EQU　10H」など)

３－２　オペコード・フィールド

```
┌──────────────────────────────────────────────┐
│[ラベル(:)]      オペコード      [オペランド]    [コメント] │
│                 ^^^^^^^^^                    │
└──────────────────────────────────────────────┘
```

　このフィールドはオペコード(ニーモニック)または擬似命令やマクロ名(マクロコール)を表記します。

(1)オペコードは文字(A～Zかa～z)や数字(0～9)で構成します。このフィールドは大文字、小文字の区別をしません。

(2)もし、ラベルフィールドにラベルを記述していない場合、68系では必ずスペースまたはタブを入れなければなりません。80系の場合は1列目から記述しても構いません。

(3)オペコードフィールドはスペース、タブ、またはリターンで終了します。

３－３　オペランド・フィールド

```
┌───────────────────────────────────────────────┐
│［ラベル(:)］　［オペコード］　　オペランド　　［コメント］│
│　　　　　　　　　　　　　　　　 ^^^^^^^^^^　　　　　　　 │
└───────────────────────────────────────────────┘
```

ほとんどのオペコードは、データやアドレスの指定を必要とします。そのデータやアドレスを指定する所がこのオペランドフィールドです。

もちろん、インヘレント命令などオペランドの指定をする必要がない場合は記述する必要はありません。

オペランドは、レジスタ，定数，ラベル，アスキーキャラクタ、および数式などの組み合わせで構成します。

(1)原則として68系のアセンブラは、オペランドフィールドにスペースを含んではいけません。
(ただし、68系の場合でも、「FCC」などで直接アスキーキャラクタを指定するものは構いません。「FCC　’TEST MESSAGE’」など)
80系のアセンブラはスペースを含んでも構いません。
(「DB　2 shl 2」などスペースを必ず入れなければいけないものもあります。)

(2)このフィールドはリターンで終了します。
ただし、この後にコメントを記述したい場合は、68系ではスペースまたはタブを、80系では「;」(Z8は「!」でも良い)を付けます。

レジスタの指定

レジスタ転送命令、スタック退避命令など、レジスタを指定するものがあります。レジスタの区切りは「,」で行います。くわしくは、各CPUの参考書籍類をご覧下さい。

式

＜数値定数＞

数字は以下の基数とそれに続くキャラクタで成り立ちます。
プリフィックスでは、プリフィックスキャラの後に数値を記述します。
サフィックスでは、数値の後にサフィックスキャラを添付します。
プリフィックスとサフィックスは同時には使用できません。どちらか一方のみを使用します。

Z8では他のCPUと表現が違いますので注意して下さい。

基　　数	ﾌﾟﾘﾌｨｯｸｽ	ｻﾌｨｯｸｽ	キャラクタ	
Decimal　(10進)			0 - 9	
Decimal　(10進)		D	0 - 9	
Binary　(2進)	%		0 - 1	
Binary　(2進)		B	0 - 1	
Octal　(8進)	@		0 - 7	
Octal　(8進)		O	0 - 7	
Octal　(8進)		Q	0 - 7	
Hexadecimal(16進)	$		0 - 9, A - F	※
Hexadecimal(16進)		H	0 - 9, A - F	

●注　意

80系のアセンブラでは、プリフィックスの「$」は16進として使用できません。サフィックス「H」のみが使用できます。

その場合、「1234H」などと最初の文字が数字の場合はこれで良いのですが、「A123H」などと最初の文字がアルファベットの場合は、「0A123H」と頭に「0」を付けて下さい。

```
例)        100              10進数で100です。
           100D             10進数で100です。
           %1010            10進数で10です。
           1010B            10進数で10です。
           @77              10進数で115です。
           77O              10進数で115です。
           77Q              10進数で115です。
           $C               10進数で12です。
           0CH              10進数で12です。

           LD   A,100
           LD   B,100D
           LD   C,%1010
           LD   D,1010B
           LD   E,@77
           LD   H,77O
           LD   L,77Q
           LD   IX,0CH
           LDA  #$C
```

Z8アセンブラでは以下の様になります。

基　　数	ﾌﾟﾘﾌｨｯｸｽ	ｻﾌｨｯｸｽ	キャラクタ
Decimal　(10進)			0 - 9
Decimal　(10進)		D	0 - 9
Decimal　(10進)	%(10)		0 - 9
Binary　(2進)		B	0 - 1
Binary　(2進)	%(2)		0 - 1
Octal　(8進)		O	0 - 7
Octal　(8進)		Q	0 - 7
Octal　(8進)	%(8)		0 - 7
Hexadecimal(16進)		H	0 - 9, A - F
Hexadecimal(16進)	%		0 - 9, A - F
Hexadecimal(16進)	%(16)		0 - 9, A - F
n, n<=10　(n進)	%(n)		0 - (n-1)

例)

```
        123             10進数で123です。
        123D            10進数で123です。
        %(10)123        10進数で123です。
        1010B           10進数で10です。
        %(2)1010        10進数で10です。
        77O             10進数で115です。
        77Q             10進数で115です。
        %(8)77          10進数で115です。
        0CH             10進数で12です。
        %(16)C          10進数で12です。

        LD   R0,123
        LD   R1,123D
        LD   R2,%(10)123
        LD   R3,1010B
        LD   R4,%(2)1010
        LD   R5,77O
        LD   R6,77Q
        LD   R7,%(8)77
        LD   R8,0CH
        LD   R9,%(16)C
```

＜ASCII定数＞

アセンブラでは以下の引用符のついたアスキー文字はそれに等しい16進数に変換されます。

	ﾌﾟﾘﾌｨｯｸｽ	ｻﾌｨｯｸｽ	キャラクタ	
ASCII-1	’		<sp> -	<sp>はスペース文字
ASCII-1	’	’	<sp> -	
ASCII-2	″	″	<sp> -	

上記のASCII-1とは、1文字のアスキー文字の定数で、ASCII-2とは複数文字の文字列の定数です。

```
例)     'A              16進で41を発生します。
        'A'             同上。
        "AB"            16進で4142を発生します。

        LD   A,'A
        LD   A,'A'
        LD   HL,"AB"
```

＜ラベルまたはシンボル＞

定数を割り当てたシンボルや、ジャンプ先やコール先のアドレスを設定したラベルは、式の中で自由に使うことができます。

```
例)     JMP  EXIT
        LDB  #EDATA-SDATA
```

＜ＰＣ指示子＞

「ＰＣ指示子」とは現在のアドレス(カレントアドレス)を示します。

68系では「*」を使用します。80系では「$」もしくは「*」を使用します。

```
例)  ABC  EQU  *        ラベルABCは現在のアドレスがセットされます。
     ABC  EQU  $        同上。
```

式演算子
^^^^^^^^

　演算子には、算術演算子、論理演算子、関係演算子、ビット演算子などがあります。アセンブラはこの演算子を認識し、その結果を出力します。

　演算子には記号で表わすものと、文字で表わすものがあります。記号で表わすものはすべてのアセンブラで使用可能です(Z8のみ使用できないものもある)。
　演算子が文字のもの(「MOD」など)は必ず左右に空白を入れなければいけません。演算子名は大文字でも小文字でも構いません。

＜単項演算子＞

　数値の先頭に演算子をおきます。もし、その演算結果が指定したレジスタに入りきれない場合、値の上位バイトを切り捨てます。

●記　号

演算子	意　味	例	例の結果(16進)	備　考
+	正 の 数	+10	000A	
-	負 の 数	-10	FFF6	
!	論理否定	!22	FFE9	Z8　使用不可
^	論理否定	^11	FFF4	
{	上位バイト	{6789H	0067	
}	下位バイト	}6789H	0089	

```
例)       LD    A,+10
          LD    B,-10
          LD    C,!22
          LD    D,^11
          LD    H,}ADRS
          LD    L,{ADRS
```

●文　字

演算子	意　味	例	例の結果(16進)	備　考
NOT	論理否定	NOT 11	FFF4	68系使用不可
LNOT	論理否定	LNOT 11	FFF4	Z8のみ使用可
HIGH	上位バイト	HIGH 6789H	0067	68系使用不可
LOW	下位バイト	LOW 6789H	0089	68系使用不可

```
例)       LD   R0,LNOT 11

          LD   E,NOT 11
          LD   B,HIGH ADRS
          LD   C,LOW ADRS
```

＜算術演算子＞

数値と数値の間に演算子をおきます。もし、加算した結果が指定したレジスタに入りきれない場合、ワーニングエラーを出力します。もし、減算した結果がマイナスになった場合(FFF6など)は、指定したレジスタに入るように上位バイトを切り捨て調節します。

演算子	意　味	例	例の結果(16進)	備　考
+	加　算	120+8	0080	
-	減　算	136-8	0080	
*	乗　算	10*3	001E	
/	除　算	120/50	0002	
¥	除算の剰余	13¥4	0001	
MOD	除算の剰余	13 MOD 4	0001	68系使用不可

```
例)       LDA  #10*3
          LD   A,13 MOD 4
```

＜論理演算子＞

●記　号

演算子	意　味	例	例の結果(16進)	備　考
>>	シフトライト	$AA>>3	0015	
<<	シフトレフト	$AA<<3	0550	
&	論 理 積	$C&$5	0004	
|	論 理 和	$B\|$E	000F	
!	論 理 和	$B!$E	000F	Z8　使用不可
^	排他的論理和	5^3	0006	

```
例)       LDA  #DATA>>3
          LDB  #DATA&3
```

●文　字

演算子	意　味	例	例の結果(16進)	備　考
SHR	シフトライト	55 SHR 3	002A	68系使用不可
SHL	シフトレフト	55 SHL 3	00AA	68系使用不可
AND	論 理 積	0CH AND 5	0004	68系使用不可
LAND	論 理 積	0CH LAND 5	0004	Z8のみ使用可
OR	論 理 和	0BH OR 5	000F	68系使用不可
LOR	論 理 和	0BH LOR 5	000F	Z8のみ使用可
XOR	排他的論理和	5 XOR 3	0005	68系使用不可
LXOR	排他的論理和	5 LXOR 3	0005	Z8のみ使用可

```
例)       LD   A,DATA SHR 3
          LD   A,DATA AND 3
```

＜関係演算子＞

左右の数値が条件を満たしていれば「1」を返し、満たしていない場合は「0」を返します。

●記　号

演算子	意　味	例	例の結果(16進)	備　考
=	等 し い	10=5	0000	
<	小 さ い	26<25	0000	
>	大 き い	22>18	0001	
<=	小さいか等しい	55<=13	0000	
=<	等しいか小さい	55=<75	0001	
>=	大きいか等しい	40>=44	0000	
=>	等しいか大きい	40=>24	0001	
<>	等しくない	55<>12	0001	
><	等しくない	32><32	0000	

```
例)       IF   LAST>0FFFH
```

●文　字

演算子	意　味	例	例の結果(16進)	備　考
EQ	等 し い	10 EQ 5	0000	68系使用不可
LT	小 さ い	26 LT 25	0000	68系使用不可
GT	大 き い	22 GT 18	0001	68系使用不可
LE	等しいか小さい	55 LE 75	0001	68系使用不可
GE	等しいか大きい	40 GE 24	0001	68系使用不可
NE	等しくない	32 NE 32	0000	68系使用不可

```
例)        IF    LAST GT 0FFFH
```

＜演算子の優先順位＞

式の中の演算は優先順位の高いものから計算します。もし、同じ優先順位の演算子の時は左から右の方に計算します。
優先順位は以下の通りです。(1)が一番高くて下に行くほど優先順位が下がります。

(1)　カッコで囲まれた式
(2)　単項演算子
(3)　シフト演算（<<, >>）
(4)　乗算(*)、除算(/)、除算の剰余(¥)
(5)　加算(+)、減算(-)
(6)　関係演算子(<, >)
(7)　論理否定演算(NOT)
(8)　論理積(&)、論理和(|)、排他的論理和(^)
(9)　上位バイト({), 下位バイト(})

３－４　コメント・フィールド

```
┌──────────────────────────────────────────────────┐
│[ラベル(:)]    [オペコード]    [オペランド]      コメント  │
│                                              ^^^^^^^^  │
└──────────────────────────────────────────────────┘
```

　このフィールドはコメントをソースラインに付加する時に使用します。アセンブラはこのフィールドを無視する為、どの様な文字，漢字，数字，記号でも構いません。

(1)80系のアセンブラでは、コメントは「;」で始まります。また、Z8では「!」も使用できます。

(2)68系のアセンブラでは、コメントはオペランドフィールド(インヘレント命令の場合はオペコードフィールド)の次のスペースまたはタブの後より始まります。

(3)このフィールドは特に必要のない限りは、記述する必要はありません。

(4)リターンによりこのフィールドは終了します。

80系の場合

```
┌──────────────────────────────┐
│ABC:  LD    A,2 SHL 2  ;comment  │
│      DAA              ;comment  │
│                       ;comment  │
└──────────────────────────────┘
```

68系の場合

```
┌──────────────────────────────┐
│ABC   LDA   #2<<2       comment  │
│      CLRA             comment  │
└──────────────────────────────┘
```

第４章　擬似命令

各CPUの標準ニーモニックの他にいくつかの擬似命令を持っています。これらはアセンブラがあるオペレーションを実行する為の命令で、多くのものは直接機械語にはなりません(データセット命令は機械語になります)。

擬似命令は、データの定数を設定したり、プログラム及びデータのロケーションアドレスを設定したり、メモリの確保を行うなどプログラムには必要不可欠なものです。

アセンブラによって使える擬似命令などが異なりますので注意してください。
^^

この章では擬似命令を7つのグループに分類して説明しています。

(1)アドレス設定
本アセンブラはアブソリュートアセンブラですから、ソース中でアドレスを設定しなくてはなりません。プログラムやデータのロケーション設定はこれらの命令を使用します。

(2)データ設定・確保
データテーブルの作成、バッファの確保などに使用します。

(3)シンボルの定義
決まった定数に名前を付ける時に使用します。

(4)リストコントロール
故意にページングを行ったり、タイトルを設定したりしてアセンブルリストを見やすくする為に使用します。

(5)他のファイルの呼び出し
コマンドラインでもファイルの連結はできますが、この命令を使うとソースファイルの途中でも他のファイルを呼び出すことができます。

(6)オプション設定
コマンドラインで指定できるオプションは、そのほとんどがソース中で指定できます。

(7)その他
各CPU固有の機能の為の命令です。

４－１　アドレスの設定

●プログラムのロケーション設定

*=,ASEG,CSEG,ORG	ALL(全てのアセンブラで使用可能)
ABS,REL	Z8

【機　能】

プログラムのロケーション(プログラムを置くアドレス)を設定する命令です。

【書　式】

```
*=      アドレス
ASEG    アドレス
CSEG    アドレス
ORG     アドレス
ABS     アドレス
REL     アドレス
```

【説　明】

プログラムや初期化されたデータのアドレスを設定します。ROM化を対象にしている場合、通常このアドレスはROM部になります。
プログラムの最初でこれを使ってアドレスを設定しないとアドレスは0番地になります。
また、アセンブルする時、「0」オプションを付けるとこれらの命令は無視されます。
プログラム中にいくらでも記述して構いません。

【使用例】

```
            ORG     100H        プログラムを100番地に置く

MAIN:       LD      SP,STACK
            CALL    WKINIT
LOOP:       CALL    MAINLOOP
            JP      NZ,LOOP
            JP      $

            ORG     1000H       ここからのプログラムは1000番地に置く

MAINLOOP:   CALL    EXITCHK
```

【注　意】

アドレスをだぶって指定してはいけません。下記のようにロケーションが重複してもエラーにはなりませんので注意してください。
オブジェクトの方には影響ないのですが、ロード時に重なってロードされます。

```
            ORG     100H
            
MAIN:       LD      SP,STACK
            :
            ORG     102H
MAINLOOP:   CALL    EXITCHK
            :
```

●データのロケーション設定

DSEG	ALL

【機　能】

初期化されないデータのロケーションを設定する命令です。

【書　式】

DSEG　　アドレス

【説　明】

初期化されたデータ(「DB」「RZB」など)は対象になりません。初期化されないデータ、つまり、領域確保命令(「DS」「RMB」など)だけが対象になります。

ROM化を対象にしている場合、このアドレスはRAM部になります。

【使用例】

```
            CSEG    100H
                                  ←┐
MAIN:       LD      SP,STACK      │ │
            CALL    WKINIT        │ │
LOOP:       CALL    MAINLOOP      │ │ ここに持ってきても結果は同じ
            JP      NZ,LOOP       │ │
            JP      $             │ │
PRGEND:                           │ │
            DSEG    1000H         │─┘
                                  │
BUFF:       DS      256           │
                                  │
MAINLOOP:   CALL    EXITCHK       │ ここからのプログラムは「PRGEND」から
              :                   │ の続いたアドレスになる
BUFF2:      DS      64            │ 「BUFF2」は「BUFF+256」からの続いた
                                  └ アドレスになる
```

【注　意】

尚、この命令もアドレスが重複してもエラーになりませんので注意してください。

「DSEG」命令を使わなかった場合は、下記のようにすべてプログラムアドレスが採用されます。

```
            CSEG    100H
            
MAIN:       LD      SP,STACK
            CALL    WKINIT
LOOP:       CALL    MAINLOOP
            JP      NZ,LOOP
            JP      $
PRGEND:
BUFF:       DS      256         ここのアドレスは「JP      $」命令
                                の次のアドレスになる
```

４－２　データの設定・確保

●１バイトデータの設定

BYTE,FCB	ALL
DB	6502,80系
DEFB	80系
DFB	6502

【機　能】

1バイトの値をメモリにセットする命令です。

【書　式】

```
BYTE    データ1[,データ2,[データn・・・]]
FCB     データ1[,データ2,[データn・・・]]
DB      データ1[,データ2,[データn・・・]]
DEFB    データ1[,データ2,[データn・・・]]
DFB     データ1[,データ2,[データn・・・]]
```

【説　明】

オペランド部で、指定した数値をカレントアドレスにセットします。
一度に数バイトものデータを記述できます。

【使用例】

```
        ORG     $1000
DATA    FCB     0,1,2,3,4,5          1000番地からオブジェクトコード
                                     「00,01,02,03,04,05」をセット
ABC     FCB     'A','B','C','D','E'  1006番地からオブジェクトコード
                                     「41,42,43,44,45」をセット
ABC2    FCB     'ABCDE'              100B番地からオブジェクトコード
                                     「41,42,43,44,45」をセット
```

●２バイトデータの設定（その１）

DBYTE,FDB	ALL
DD,DEFD	80系
DW	8048,8051

【機　能】

2バイトの値をメモリにセットする命令です。

【書　式】

DBYTE　データ1[,データ2,[データn・・・]]
FDB　　データ1[,データ2,[データn・・・]]
DD　　データ1[,データ2,[データn・・・]]
DEFD　　データ1[,データ2,[データn・・・]]
DW　　データ1[,データ2,[データn・・・]]

【説　明】

2バイトコードの上位バイトをカレントアドレスに、下位バイトをその次のアドレスにそれぞれセットします。

【使用例】

```
          ORG     1000H
DATA1:    FDB     1234H  | 1000番地に12、1001番地に34をセット
DATA2:    DEFD    1234H  | 1002番地に12、1003番地に34をセット
          ORG     2000H
DATA3:    DW      10H    | 2000番地に00、2001番地に10をセット
DATA4:    DW      1000H  | 2002番地に10、2003番地に00をセット
```

●２バイトデータの設定（その２）

WORD	ALL
DEFW	80系
DW	6502,8085,Z80,Z8

【機　能】

2バイトの値をメモリにセットする命令です。

【書　式】

WORD　　データ1[,データ2,[データn・・・]]
DEFW　　データ1[,データ2,[データn・・・]]
DW　　データ1[,データ2,[データn・・・]]

【説　明】

2バイトコードの下位バイトをカレントアドレスに、上位バイトをその次のアドレスにそれぞれセットします。

【使用例】

```
          ORG     1000H
DATA1:    WORD    1234H  1000番地に34、1001番地に12をセット
DATA2:    DEFW    1234H  1002番地に34、1003番地に12をセット

          ORG     2000H
DATA3:    DW      10H    2000番地に10、2001番地に00をセット
DATA4:    DW      1000H  2002番地に00、2003番地に10をセット
```

●文字データの設定（その１）

CHAR,FCC	ALL
DEFM,DM,TEXT	80系
ASC	6502

【機　能】

文字列をメモリにセットする命令です。

【書　式】

CHAR　　文字列データ
FCC　　文字列データ
DEFM　　文字列データ
DM　　文字列データ
TEXT　　文字列データ
ASC　　文字列データ

【説　明】

文字列で指定したデータをそれぞれのアスキーコードに変換してメモリにセットします。
「１バイトデータの設定」の命令でも代用できます。

【使用例】

```
          ORG     $1000
DATA1     FCC     'TEST'      1000番地から「54,45,53,54」をセット
DATA2     FCC     'TEST',0    1004番地から「54,45,53,54,00」をセット
```

●文字データの設定（その２）

FCS	ALL

【機　能】

文字列をメモリにセットする命令です。

【書　式】

FCS　　文字列データ

【説　明】

文字列で指定したデータをそれぞれのアスキーコードに変換してメモリにセットします。

ただし、一番最後のデータ(文字)のビット7をオンにします。文字列の最後ということを示すインジケータとして活用できます。

【使用例】

```
PRINT   LDX     #STRING
LOOP    LDA     ,X+
        BMI     EXIT        ビット7が1だったら終了
        JSR     PUTCHAR
        BRA     LOOP
EXIT    ANDA    #$7F
        JMP     PUTCHAR
        ORG     $1000
STRING  FCS     'MESSAGE'   1000番地から「4D,45,53,53,41,47,C5」をセット
```

【注　意】

漢字やカタカナやグラフ文字は、もともと8ビットコードで表わされている為、この命令が有効になりません。7ビットコードで表わせるアスキー文字の時に有効です。

●文字データの設定（その３）

FCT	ALL
DEFT,DT	80系

【機　能】

文字列で指定したデータをそれぞれのアスキーコードに変換してメモリにセットします。

【書　式】

FCT　　文字列データ
DEFT　　文字列データ
DT　　文字列データ

【説　明】

文字列で指定したデータをそれぞれのアスキーコードに変換してメモリにセットします。
ただし、指定したデータの個数をカレントアドレスにセットし、その次から指定したデータをメモリにセットします。
通常、故意的に行う作業ですが、この命令を使うと自動的に行います。文字列の表示やその他の用途に使用できるでしょう。

【使用例】

```
PRINT:  LD      HL,STRING
        LD      B,(HL)      ;Bレジに文字列のバイト数を入れる
LOOP    INC     HL
        LD      A,(HL)
        CALL    PUTCHAR
        DJNZ    LOOP        ;Bレジのバイト数だけ文字を出力したら終了
        RET
;
        ORG     1000H
STRING  FCT     'MESSAGE'   ;1000番地に文字列の個数「07」をセット
                            ;1001番地から「4D,45,53,53,41,47,45」をセット
```

【補　追】

(1)文字列を指定する時の囲み記号として、次のものが使用できます。

「'」「″」「#」「%」「&」「/」「?」「*」

```
STR1    FCC    'MESSAGE'
STR2    FCC    ″MESSAGE″
STR3    FCC    #MESSAGE#
STR4    FCC    %MESSAGE%
STR5    FCC    &MESSAGE&
STR6    FCC    /MESSAGE/
STR7    FCC    ?MESSAGE?
STR8    FCC    *MESSAGE*
```

(2)文字列の外に1バイトデータも記述できます。

```
STR1    FCC    'STRINGS',0
STR2    FCC    $0C,'CLEAR',0
STR3    FCC    'H',8,'HELP CMD'
```

(3)文字列にはカタカナや漢字が使用できます。
　漢字はMS-JISのコードになります。

```
MSG     FCT    '漢字'
```
漢字は1文字で2バイトになります。

●メモリのクリア確保

RZB,ZERO	ALL

【機　能】
　　メモリ領域を確保しゼロで初期化を行います。

【書　式】
　　RZB　　確保バイト数
　　ZERO　　確保バイト数

【説　明】
　　指定した数値分、カレントアドレスからゼロをセットします。
　　この命令は初期化を行う為、「DSEG」命令のアドレスの対象になりません。

【使用例】

```
          ORG     1000H
BUFF:     RZB     255
```

1000番地から255バイト分、0をセット
(1000番地から10FE番地まで0が埋まる)

【注　意】
　　設定できる数値は、最高255までです。
　　^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
　　それ以上の数を確保したい場合は、何回かに分けて行ってください。

```
          ORG     1000H
BUFF:     RZB     255
          RZB     255
          RZB     255
```

●メモリの確保

RMB	ALL
RES	8085,Z8,68系
DS	6502,80系
BLK,DEFS	80系

【機　能】

メモリ領域を確保する。

【書　式】

RES　　確保バイト数
RMB　　確保バイト数
DS　　確保バイト数
BLK　　確保バイト数
DEFS　　確保バイト数

【説　明】

指定した数値分、カレントアドレスから領域を確保します。
もし、「DSEG」命令を使用している場合、カレントアドレスはデータアドレスになります。使用していなかった場合は、前のアドレスからのものになります。

【使用例】

```
          ORG     1000H
BUFF:     DS      1024      1000番地から1024バイト確保される
                            (ここまで「DSEG」が使用されていない)
          DSEG    2000H     (ここから「DSEG」のアドレスが有効になる)
BUFF2:    DS      256       2000番地から256バイト確保される
DATA:     DB      1,2,3     1400番地(1000番地+1024)からデータ「01,02,
                            03」がセットされる)
BUFF3:    DS      128       2100番地(2000番地+256)から128バイト確保
                            される
```

４－３　シンボルの定義

●シンボル定義

EQU	ALL

【機　能】

シンボルを定義します。

【書　式】

シンボル名　　EQU　　数値

【説　明】

シンボルにオペランドで指定した数値を登録します。数値には演算子やラベルなどが使用できます。ただし、ラベル(シンボル)を使用する場合、そのラベルは既に定義済みのものでないといけません。
登録したシンボルは、一般のニーモニックのオペレーションで使用できます。

【使用例】

```
BASE     EQU     $FC80
CMD      EQU     BASE+0
PARANO   EQU     BASE+1
PARA     EQU     BASE+2
         
SETUP    STA     CMD
         STB     PARANO
         LDU     #PARA
LOOP     LDA     ,X+
         STA     ,U+
         DECB
         BNE     LOOP
         RTS
```

「BASE」は「CMD,PARANO,PARA」より前で定義を行わなければいけない

【注　意】

この命令で定義したシンボルは再定義できません。
再定義が必要な場合は次の命令を使ってください。

●再定義可能なシンボル定義

=	ALL
SET	Z80を除く
DEFL	80系
SETL	8048,8051,Z80
:=	Z8

【機　能】

シンボルを定義します。この命令は再定義可能です。

【書　式】

```
シンボル名      =       数値
シンボル名      SET     数値
シンボル名      DEFL    数値
シンボル名      SETL    数値
シンボル名      :=      数値
```

【説　明】

シンボルにオペランドで指定した数値を登録します。数値には演算子やラベルなどが使用できます。ただし、ラベル(シンボル)を使用する場合、そのラベルは既に定義済みのものでないといけません。
登録したシンボルは、一般のニーモニックのオペレーションで使用できます。
これらの命令で登録したシンボルは再定義できます。

【使用例】

```
BASE      EQU     1000H
          LD      IX,BASE

OFFSET    =       0
          LD      A,(IX+OFFSET)

OFFSET    =       1
          LD      A,(IX+OFFSET)
```

４－４　リストコントロール

●強制改ページ

PAG,PG	ALL
PAGE	68系
*EJECT	80系

【機　能】

改ページを行います。

【書　式】

PAG
PG
PAGE
*EJECT

【説　明】

コード「0C」を出力し改ページ動作を行います。
通常のプリンタは「0C」コードを送ることによって改ページを行います。

【使用例】

```
P1        EQU     $01
P2        EQU     $02
P3        EQU     $03
          
          PAG                 ここでページング
          
START     LDS     #STACK
          JSR     INIT
```

【注　意】

アセンブルオプション「P」を指定した場合は有効になりません。
尚、ページング動作を行った場合、この命令は表示されません。もし、この命令を表示させたい場合はアセンブラオプション「A」を指定してください。

●リスト出力停止・開始

*LIST,LIST,XLIST	80系

【機　能】

リスト出力の停止及び開始を行います。
68系のアセンブラでは「OPT」命令にて行います(４－６参照)。

【書　式】

```
*LIST   ON
*LIST   OFF

LIST
XLIST
```

【説　明】

「*LIST ON」と「*LIST OFF」、「LIST」と「XLIST」は対になっています。
「*LIST OFF」または「XLIST」でアセンブルリストの表示を行わなくなり、
「*LIST ON」または「LIST」でアセンブルリストの表示を行うようになります。

【使用例】

```
          *LIST   OFF          ｜ここからリスト表示を停止
ADRS1     EQU     0A000H       ｜
ADRS2     EQU     0B000H       ｜
ADRS3     EQU     0C000H       ｜
          *LIST   ON           ｜ここからリスト表示を開始
START     CALL    INIT         ｜
          CALL    MAIN         ｜
```

【注　意】

アセンブルオプション「L」を指定した場合、この命令は有効になりません。

●空白行挿入

SPC	ALL

【機　能】
　　空白行を挿入します。

【書　式】
　　SPC

【説　明】
　　ソースで単にリターンを入力した行と同じ表示になります。

【使用例】

```
********************************
*     SPC Sample Program       *
********************************
            SPC                 
START       LDX       #TABLE    
            JMP       MAIN      
                                
            END                 
```

この２つの行はアセンブルリスト上は同じ空白表示になる

【注　意】
　　「SPC」はアセンブルリスト上には表示されません。

●タイトル設定

NAM,TTL	ALL
*HEADING,NAME,TITLE	80系

【機　能】

タイトルを設定します。

【書　式】

```
NAM       タイトル
TTL       タイトル
*HEADING  タイトル
NAME      タイトル
TITLE     タイトル
```

【説　明】

アセンブルリストでページングをする時、第１行目にタイトルを表示します。
タイトルは最高36文字まで設定できます。
タイトルの表示は次のページング時から有効になります。

【使用例】

```
          TTL     Sample Program
          PAG
START:    CALL    INIT_WK
          CALL    ABC
          JP      MAIN
```

```
Sample Program                 1989- 1-26 Z80 Assembler  Page    1
                                                      File:SAMPLE.TXT
    3 0100 CD0010   START:    CALL     INIT_WK
    4 0102 CD3510             CALL     ABC
    5 0104 C30020             JP       MAIN
```

ここに表示される

【注　意】

ページングを禁止すると(オプション「P」など)、この命令は無効になります。
タイトル表示を行った場合、この命令自身は表示されません。もし、表示させたい場合はアセンブルオプション「A」を指定してください。

●サブタイトル設定

STTL	ALL
SUBTTL	80系

【機　能】

サブタイトルを設定します。

【書　式】

```
STTL     サブタイトル
SUBTTL   サブタイトル
```

【説　明】

アセンブルリストでページングをする時、第２行目にサブタイトルを表示します。
サブタイトルは最高53文字まで設定できます。
サブタイトルの表示は次のページング時から有効になります。

【使用例】

```
          TTL     Sample Program
          STTL    for Cross Assembler-MS
          PAG
START:    CALL    INIT_WK
          CALL    ABC
          JP      MAIN
```

```
Sample Program                 1989- 1-26 Z80 Assembler  Page    1
for Cross Assembler-MS                                File:SAMPLE.TXT
                                                                        ここに
                                                                        表示さ
    4 0100 CD0010   START:    CALL     INIT_WK                          れる
    5 0102 CD3510             CALL     ABC
    6 0104 C30020             JP       MAIN
```

【注　意】

ページングを禁止すると(オプション「P」など)、この命令は無効になります。
タイトル表示を行った場合、この命令自身は表示されません。もし、表示させたい場合はアセンブルオプション「A」を指定してください。

４－５　他のファイルの呼び出し

LIB,USE	ALL
*INCLUDE,INCLUDE	80系

【機　能】

他のファイルを挿入します。

【書　式】

```
LIB       ファイル名
USE       ファイル名
*INCLUDE  ファイル名
INCLUDE   ファイル名
```

【説　明】

指定したファイルをその行に挿入します。
ファイル名は通常のファイルを指定するように、ドライブ名，パス名，拡張子を正確に指定しなくてはいけません。ただし、MS-DOSの環境変数に呼び出すドライブ名やパス名などを指定できるようにしていますので、ソースの中ではファイル名と拡張子のみを指定すれば良いでしょう。環境変数名は「ASMLIB」という名前で設定します。もし、この変数が設定されておらず、かつソース中でもドライブの指定をしていなかった場合は、カレントディレクトリが対象になります。

```
A>SET ASMLIB=B:□
  ~~~~~~~~~~~~~~~
```

と環境変数「ASMLIB」を設定すると、ソースの方で

```
        LIB  IOEQU.TXT
```

とした場合、

```
        LIB  B:IOEQU.TXT
```

と指定した時と同じ動作をします。
つまり、ドライブBの「IOEQU.TXT」を呼び出します。

パスも指定したい場合は、

```
A>SET ASMLIB=B:LIB¥□                    (パス名の後ろに「¥」を付ける)
  ~~~~~~~~~~~~~~~~~~
```

と行います。
この場合はドライブBのパス「LIB」の中の「IOEQU.TXT」を呼び出します。

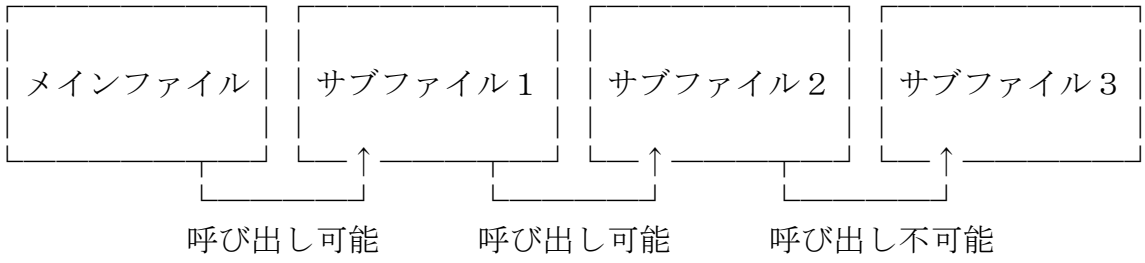
【使用例】

[IOEQU.TXT]ファイル

```
;*******************************
; ＩＯレジの定義ファイルを挿入 *
;          （ＩＯＥＱＵ）      *
;*******************************
;
          LIB       IOEQU.TXT ──┐
INITIO:   LD        A,80H       │
          OUT       PPI_CTRL    │
                                │
PPI_A     EQU       00H         │
PPI_B     EQU       01H         │
PPI_C     EQU       02H         │
PPI_CTRL  EQU       03H         │
                                │
                                └─ この行に挿入される
```

【注　意】

ファイルの呼び出しのネスティングは2つまでです。
「LIB」したファイル(サブファイル1)の中で「LIB」を行っても構いませんが、そのファイル(サブファイル2)で「LIB」を行ってはいけません。



メインファイルや、サブファイル1からは複数個のファイルが呼び出せます。

```
          LIB       IOEQU.TXT
          LIB       BIOS.TXT
          LIB       MAIN.TXT
          LIB       COMMAND.TXT
           :
```

４－６　オプション設定

OPT	ALL

【機　能】

アセンブルオプションをソース中で指定します。

【書　式】

OPT　　オプション

【説　明】

アセンブルオプションはいくつかありますが、そのオプションをソース中に指定することができます。「備考」に何も書かれていないものはソース中で何回でも使用可能なものです。

ｵﾌﾟｼｮﾝ	省略型	機　　能	備　　考
BIN	B	オブジェクトファイルを作成する	ｿｰｽ中でどちらか1回のみ指定できます
NOB	-B	オブジェクトファイルを作成しない	
CON	C	条件付きコードの表示を行う	
NOC	-C	条件付きコードの表示を禁止する	
NOE	-E	マクロ拡張ラインを表示しない	
EXP	E	マクロ拡張ラインを表示する	
NOF	-F	80文字の出力に切り換える	
FRE	F	132文字の出力に切り換える	
GEN	G	複数の行に渡るコードの出力を行う	
NOG	-G	複数の行に渡るコードの出力を禁止する	
LIS	L	アセンブルリストの表示を行う	
NOL	-L	アセンブルリストの表示を行わない	
MAC	M	マクロ定義，コールラインを出力する	
NOM	-M	マクロ定義，コールラインを出力しない	
OSS	O	「ORG」を有効にする	
NOO	-O	「ORG」を無効にする	
PAG	P	ページングする	
NOP	-P	ページングをしない	
SYM	S	シンボルテーブルを出力する	ｿｰｽ中でどちらか1回のみ指定できます
NOS	-S	シンボルテーブルを出力しない	
NOT	-T	通し番号を付けない	
TRU	T	通し番号を付ける	
WRN	W	ワーニングエラーを出力する	
NOW	-W	ワーニングエラーを出力しない	
NOX	-X	クロスリファレンスを出力しない	ｿｰｽ中でどちらか1回のみ指定できます
XRF	X	クロスリファレンスを出力する	

※詳しくは「２－２」を参照してください。

【使用例】

```
┌──────────────────────────────┐
│          OPT     NOL         │ ここからリスト表示を停止
│ADRS1     EQU     $A000       │
│ADRS2     EQU     $B000       │
│ADRS3     EQU     $C000       │
│          OPT     LIS         │ ここからリスト表示を開始
│START     JSR     INIT        │
│          JSR     MAIN        │
└──────────────────────────────┘
```

【注　意】

ソース中で指定したオプションの機能と正反対のオプションをコマンドラインで指定した場合、コマンドラインで指定したものが優先されます。

４－７　その他

●ソースラインの終了

END	ALL

【機　能】

ソースラインの終了を知らせます。また、エントリーアドレスの設定も行えます。
「END」以下にプログラムが記述されている場合でも、そのプログラムは無視されます。

【書　式】

END　　エントリーアドレス

【説　明】

エントリーアドレスとはプログラムの実行アドレスです。
オブジェクト(Sフォーマットやインテル HEX レコード)の最後にこの情報が出力されます。

Sフォーマットの場合は「S9031000EC」
　　~~~~

インテルHEXの場合は「:00100001EF」
　　~~~~

それぞれアンダーラインの所がエントリーアドレスです。

【使用例】

```
          ORG     1000H
          :
START:    LD      SP,STACK
          LD      HL,BUFF
          LD      B,ENDBUF-TOPBUF
            :
            :
          DB      'Data End Mark'
          :
          END     START
```

エントリーアドレスは1000番地に設定されます。

●エラーの発生

ERR	ALL

【機　能】
　　故意的にエラーを発生させます。

【書　式】
　　ERR　　エラーメッセージ

【説　明】
　　強制的にエラーを表示させます。
　　通常、条件付き命令と共に使用します。

【使用例】

```
LAST_ROM  EQU     7FFFH
;
START:    LD      SP,STACK
          :
          RET
ENDPROG:
          IF      ENDPROG>LAST_ROM    ;プログラムがROMエリアを
          ERR     ＲＯＭエリアオーバー  ;オーバーした場合はエラー
          ENDIF                       ;を表示
```

●行の繰り返し

RPT	ALL

【機　能】
　　指定した回数分、次の行を繰り返し展開します。

【書　式】
　　RPT　　展開数

【説　明】
　　次の行を繰り返し展開します。

【使用例】

```
SPACE    MACRO
         RPT     &1      指定回数分、空白を挿入
         SPC
         ENDM
         SPACE   4       空白行を4つ挿入
         RPT     4       「ASLA」命令を4回展開
         ASLA
```

●レジスタ登録

REG	6809

【機　能】

スタック退避命令の為にレジスタのリストを定義します。

【書　式】

REG　　レジスタリスト

【説　明】

「PSHS」「PULS」「PSHU」「PULU」命令のオペレーションで指定するレジスタリストを定義しておく命令です。

これらの命令のポストバイトは次のようにビット対応になっています。

b7	b6	b5	b4	b3	b2	b1	b0
PC	S/U	Y	X	DP	B	A	CC

「REG」で定義すると、指定したレジスタにあたるビットを1に数値化して保持します。

【使用例】

```
ALLREG   REG     A,B,X,Y,U,DP
CMDEXEC  PSHS    #ALLREG        「PSHS  A,B,X,Y,U,DP」とした時と同じ
           :
         PULS    #ALLREG        「PULS  A,B,X,Y,U,DP」とした時と同じ
         RTS
```

●ダイレクトページ設定

SETDP	6809

【機　能】

ダイレクトページアドレスを設定します。

【書　式】

SETDP　　ダイレクトページ

【説　明】

6809にはダイレクトページ・アドレッシングというものがあります。
2バイトのアドレス空間のうち上位1バイトを6809のDPレジスタで示し、下位1バイトだけをオペランドで指定するというものです。
つまり、256バイトの空間に対してこのアドレッシングが使用できるのです。
アセンブラでは、通常DPレジスタは0として考えています。その為、0000番地から00FF番地に対してのオペレーションはすべてダイレクトページ・アドレッシングに展開しています。
DPレジスタの値が変られるように、アセンブラでもこのアドレッシングを対象にする領域(アドレス)を自由に変更できます。「SETDP」命令でその領域の上位1バイト(つまりDPレジスタの設定値)を設定します。

【使用例】

```
          SETDP   $10       1000番地から10FF番地をダイレクトアドレッシングの対象にする

START     LDA     #$10      プログラムの方でDPレジスタを設定しないと正常に動作しないので注意!
          TFR     A,DP

          LDA     $1012     ダイレクトアドレッシング命令に展開される
```

【注　意】

デフォルトでは、ダイレクトページは0になっています。つまり、0000番地から00FF番地をダイレクトページの対象にしています。

SETDPはあくまでアセンブラに対しての定義命令です。「SETDP」を使用した場合、プログラムの方でも必ずDPレジスタを設定しなければ、そのプログラムは正常に動作しません。

●アドレスページング

PAGE	80系

【機　能】
　　カレントアドレスを次のページに変更します。

【書　式】
　　PAGE

【説　明】
　　256バイトを1ページとしています。この命令でカレントアドレスを強制的に
　　次のページに変更できます。
　　例えば、カレントアドレスが0274番地だった場合、この命令を実行すると
　　0300番地に更新されるということです。
　　0274番地から02FF番地の間は空(無コード)になります。

【使用例】

```
          ORG    100H
START:    JMP    MAIN      ここは100番地

          PAGE             アドレスページング

MAIN:     CALL   INIT      ここからは200番地になる
          CALL   GETKEY
```

第５章　条件付きアセンブリ

条件付きアセンブリとは、指定した条件に満たされた所をアセンブリしたり、しなかったりするものです。

(1)書式その1(全アセンブラで使用可能)

```
    IF    〈式〉
          〈式〉が真ならアセンブルされる
    ELSE
          〈式〉が偽ならアセンブルされる
    ENDIF
```

(2)書式その2(Z8のアセンブラのみ使用可能)

```
    IF    〈式〉
          〈式〉が真ならアセンブルされる
    ELSE
          〈式〉が偽ならアセンブルされる
    FI
```

(3)書式その3(80系のアセンブラのみ使用可能)

```
    COND  〈式〉
          〈式〉が0でなければアセンブルされる
    ELSE
          〈式〉が0ならアセンブルされる
    ENDC
```

５－１　式の真偽によるもの

●式が０でなければアセンブル

IF,IFNE,IFT	ALL
COND	80系

【書　式】

```
IF      式
IFNE    式
IFT     式
COND    式
```

【説　明】

式が真(1)ならアセンブルし、偽(0)ならアセンブルしません。
関係演算子も使える為、数値のイコール、大小関係などで条件付きアセンブルが行えます。

【使用例】

```
LAST_ROM EQU    7FFFH
;
START:   LD     SP,STACK
          :
         RET
ENDPROG:
         IF     ENDPROG>LAST_ROM
         ERR    ＲＯＭエリアオーバー
         ENDIF
```

「ENDPROG」が「LAST_ROM」より大きければ、条件が真になる

【注　意】

通常「COND」命令を使った時は、「ENDC」命令で終ります。
ただし、「ENDIF」命令もそのまま使用できます。

これらの命令は、オペレーションを符号なしで処理します。

```
IF   -1<0
```

のオペレーションは、「-1<0」ですが、内部でこれを「FFFF<0」と展開します。その為、この場合の式は偽になるということです。

●式が０ならばアセンブル

IFC,IFEQ,IFF,IFN	ALL

【書　式】

```
IFC     式
IFEQ    式
IFF     式
IFN     式
```

【説　明】

式が偽(0)ならアセンブルし、真(1)ならアセンブルしません。
「IF」とは正反対の動作を行います。

【使用例】

```
SWFLAG   DS     1
;
SET_SW:
         IFN    SWITCH
         LD     A,0          ; シンボル「SWITCH」が0ならこの行を展開
         ELSE
         LD     A,I          ; でなければこの行を展開する
         ENDIF
         LD     (SWFLAG),A
         RET
```

【注　意】

これらの命令は、オペレーションを符号なしで処理します。

```
IFEQ -1<0
```

のオペレーションは、「-1<0」ですが、内部でこれを「FFFF<0」と展開します。その為、この場合の式は偽になるということです。

●式が０より大きいか等しいならアセンブル

IFGE	ALL

【書　式】

IFGE　　式

【説　明】

式が0より大きいか、等しい場合にアセンブルします。

式の計算は符号付きで処理します。
計算結果が0000から7FFFの場合はアセンブルし、8000からFFFFの場合はアセンブルしないということです。

【使用例】

```
          IFGE    LEVEL
          LD      B,LEVEL+1   ｜シンボル「LEVEL」が0以上の値であれば、
LOOP      CALL    LEVELUP     ｜この中身が展開される
          DJNZ    LOOP
          ENDIF
```

●式が０より小さいか等しいならアセンブル

IFLE	ALL

【書　式】

IFLE　　式

【説　明】

式が0より小さいか、等しい場合にアセンブルします。

式の計算は符号付きで処理します。
計算結果が0000もしくは8000からFFFFの場合はアセンブルし、0001から7FFFの場合はアセンブルしないということです。

【使用例】

```
          IFLE    VALUE
          LD      B,VALUE-1   ｜シンボル「VALUE」が0以下の値であれば、
LOOP      CALL    WAIT        ｜この中身が展開される
          DEC     B
          JP      NZ,LOOP
          ENDIF
```

●式が０より大きければアセンブル

IFGT	ALL

【書　式】

IFGT　　式

【説　明】

式が0より大きい場合にアセンブルします。

式の計算は符号付きで処理します。
計算結果が0001から7FFFの場合はアセンブルし、0000もしくは8000からFFFFの場合はアセンブルしないということです。

【使用例】

```
        IFGT    ROMEND-$
        LIB     SPECIAL.TXT
        ENDIF
```

「ROMEND」がカレントアドレスより大きければ「SPECIAL.TXT」ファイルを挿入する

●式が０より小さければアセンブル

IFLT	ALL

【書　式】

IFLT　　式

【説　明】

式が0より小さい場合にアセンブルします。

式の計算は符号付きで処理します。
計算結果が8000からFFFFの場合はアセンブルし、0000から7FFFの場合はアセンブルしないということです。

【使用例】

```
        IFLT    ROMEND-$
                        ; カレントアドレスが「ROMEND」より小さければこの行を展開する
        CALL    SPECIAL
                        ;
        ENDIF
```

５－２　アセンブリパスによるもの

IF1,IFP1	ALL
IF2,IFP2	ALL

【書　式】

IF1
IFP1
IF2
IFP2

【説　明】

アセンブラは、パス1、パス2で最終的にオブジェクトを生成します。
パス1でラベルの登録を行い、パス2でオブジェクトの生成を行います。この擬似命令は、パスによってアセンブルする行をコントロールするものです。

「IF1」「IFP1」は、どちらもパス1の時のみアセンブルします。
「IF2」「IFP2」は、どちらもパス2の時のみアセンブルします。

【使用例】

```
IFP1
LIB     IOEQU.TXT ──┐
ENDIF               │
LDA     KEYDATA     │
                    │
                    └──┐
                       │
KEYDATA   EQU    $FD01 │
IRQMASK   EQU    $FD02 │
IRQCHECK  EQU    $FD03 │
                       │
「IOEQU.TXT」ファイル
```

パス1で「IOEQU.TXT」の呼び出しを行います(シンボルの登録)。パス2ではこのファイルの呼び出しを行いません。
パス2ではコードの生成のみを行う為、シンボルの定義だけを行っている「IOEQU.TXT」は特に呼び出す必要はないわけです。
生成されるコードはどちらも変わりありませんが、アセンブル処理の時間が異なります。
大きなファイルをアセンブルするときに有用でしょう。

【注　意】

この命令を使用するにあたっては、アセンブラの動作をよく把握していないと、パス1とパス2のアドレスのずれが起こり、エラーが多発する恐れがあります。

５－３　ラベルの定義の有無によるもの

IFDEF,IFNDEF	ALL

【書　式】

IFDEF　シンボル
IFNDEF　シンボル

【説　明】

指定したシンボル(ラベル)が定義されているかどうかで、アセンブルする行をコントロールします。

「IFDEF」では指定したシンボルが定義されている場合、アセンブルします。
「IFNDEF」では指定したシンボルが定義されていなかった場合、アセンブルします。

【使用例】

```
ALLCLR   EQU    *          
         IFDEF  DEBUG      | シンボル「DEBUG」が定義されている場合、
         LDX    #MSG       | この中のプログラムを展開する
         JSR    PRINT      
         ENDIF             
         LDX    #BUFF      
```

コマンドラインでシンボルの定義が行える為、ソースの変更なしでデバッグ用プログラム(「DEBUG」を定義したもの)と正規のプログラム(「DEBUG」を定義しないもの)の両方が作成できます。

【注　意】

これらの命令で参照するシンボルは、現時点でのラベルの登録の有無を見ます。
その為、参照するシンボルがこれらの命令より後ろで定義されている場合は未定義扱いとなりますので注意してください。

第６章　マクロ機能

マクロとは、いくつかの命令(ニーモニック)を1つにまとめて定義しておき、それを1つの命令として実行(展開)する為のものです。
マクロを使用するにあたっては、まず、その命令群の定義をしなければなりません。これは必ずマクロをコールする前に行います。

６－１　マクロの定義

マクロの定義は、「MACRO」で始まり、「ENDM」で終わります。
「MACRO」命令のラベル部には、そのマクロの名前(マクロコール名)を書きます。
定義の仕方は68系も80系も共通で次のように記述します。

```
マクロ名     MACRO
               ・
               ・
               ・
             ENDM
```

例えば、6809でDレジスタを左にシフトする場合、

```
             LSLB
             ROLA
```

と記述します。しかし、マクロでこの一連の命令を定義しさえすれば、あたかもその命令が始めからあるもののように使用できるのです。

```
LSLD         MACRO
             LSLB
             ROLA
             ENDM
```

これで定義はできました。

６－２　マクロの呼び出し

先ほど定義したマクロは、引数が何もないのでコールの方は、

```
┌──────────────────────────┐
│          LSLD            │
└──────────────────────────┘
```

というように行います。アセンブラは自動的に定義した通りに

```
┌──────────────────────────┐
│          LSLB            │
│          ROLA            │
└──────────────────────────┘
```

と展開します。

なお、実際にアセンブルリスト上で、このマクロの展開の様子を表示させたい場合は、アセンブル時に、「E」オプションを指定するか、ソース中で「 OPT EXP」と記述してください。

６－３　パラメータの使用法

マクロには引数(パラメータ)が使用でき、最高9つまで設定できます。マクロ定義の部分では、「&1」「&2」と「&」に続いて引数の番号を指定します。

しかし、マクロ定義の中で「MASK&1」と演算子として「&」を使いたい場合があると思います。その場合は、「MASK¥&1」と、「&」の前に「¥」を指定します。

マクロコールでは引数をオペランドフィールドに指定します。引数の句切りには「,」を使用します。

例えば、

```
MOVE        MACRO
            LD      A,&2
            LD      (&1),A
            ENDM
```

と「MOVE」を定義した場合、

```
            MOVE    ADRS,1
```

とコールするとMOVE定義&1に「ADRS」が、&2に「1」が代入され、

```
            LD      A,1
            LD      (ADRS),A
```

と展開されます。

パラメータを3つ以上指定した場合は3つ目以降のパラメータは無視されます。また、1つしか指定されていなかった場合は、展開時に「&2」には何も代入されないのでエラーになります。

パラメータにはスペースを入れることができません。つまり、

```
        STR     TEST MESSAGE
```

とすると、第一パラメータには「TEST」だけが認識され、その次のスペースからはコメントとして認識されてしまいます。
これを解決する為に、「'」や「″」で囲んだ所は1つのパラメータとして認識するようにしています。

```
        STR     ″TEST MESSAGE″
```

この場合、パラメータは「TEST MESSAGE」になり、「″」は含まれません。もし、「″」が必要なものはマクロ定義の所で、

```
STR     MACRO
        FCC     ″&1″
        ENDM
```

と行ってください。

６－４　条件付きマクロ展開

先ほどの「MOVE」では、1バイトのロードしかできません。これを1バイトでも2バイトでも行えるようにするには、条件付きマクロ展開を行うとよいでしょう。その時、必要になる命令が「EXITM」です。

```
MOVE    MACRO
        IF      &2<100H
        LD      A,&2
        LD      (&1),A
        EXITM
        ENDIF
        LD      BC,&2
        LD      (&1),BC
        ENDM
```

パラメータ1が100(16進)より小さい場合は、その次からのマクロ展開を行い、それ以外は、「ENDIF」以下のマクロ展開を行うというものです。
「ENDIF」の上に出てきた「EXITM」ですが、これはここでマクロ展開を終了するというものです。

●パラメータがNULかどうかの判定

パラメータが指定されているかどうかも、「IF」で見ることができます。

```
MOVE    MACRO
        IF      '&2'=''     | 第２パラメータが指定されていなかった場合はこの行
        LD      A,0         | を展開する
        ELSE
        LD      A,&2
        ENDIF
        LD      (&1),A
        ENDM
```

６－５　マクロの使用による注意

(1)マクロコールはコールする以前にそのマクロが定義されていなくてはいけません。

(2)マクロ名は8文字まで認識されます。

(3)使用できるマクロ名はラベル名と同じ文字ですが、大文字と小文字の区別を行いません。

(4)マクロコールはネストできますが最高5レベルまでです。また、マクロ定義はネストできませんので注意してください。

(5)マクロ定義の中のコメントはそのままの形で保存しています。ですから、それだけメモリを使うということを覚えておいてください。シンボルの登録可能数などにも影響を及ぼします。

(6)マクロ定義中は、ある特別の命令以外は認識していません。その為、不法な命令が現われても、この時点ではエラーにはなりません。また、ニーモニックやコメントのセパレートもこのような理由により、きちんとセパレートされない場合がありますので御了承ください。

(7)ローカルラベルはサポートしていません。

(8)マクロネームテーブルは、ニーモニックテーブルよりも先にサーチされます。その為、すでに存在するニーモニックや擬似命令は再定義することが可能です。

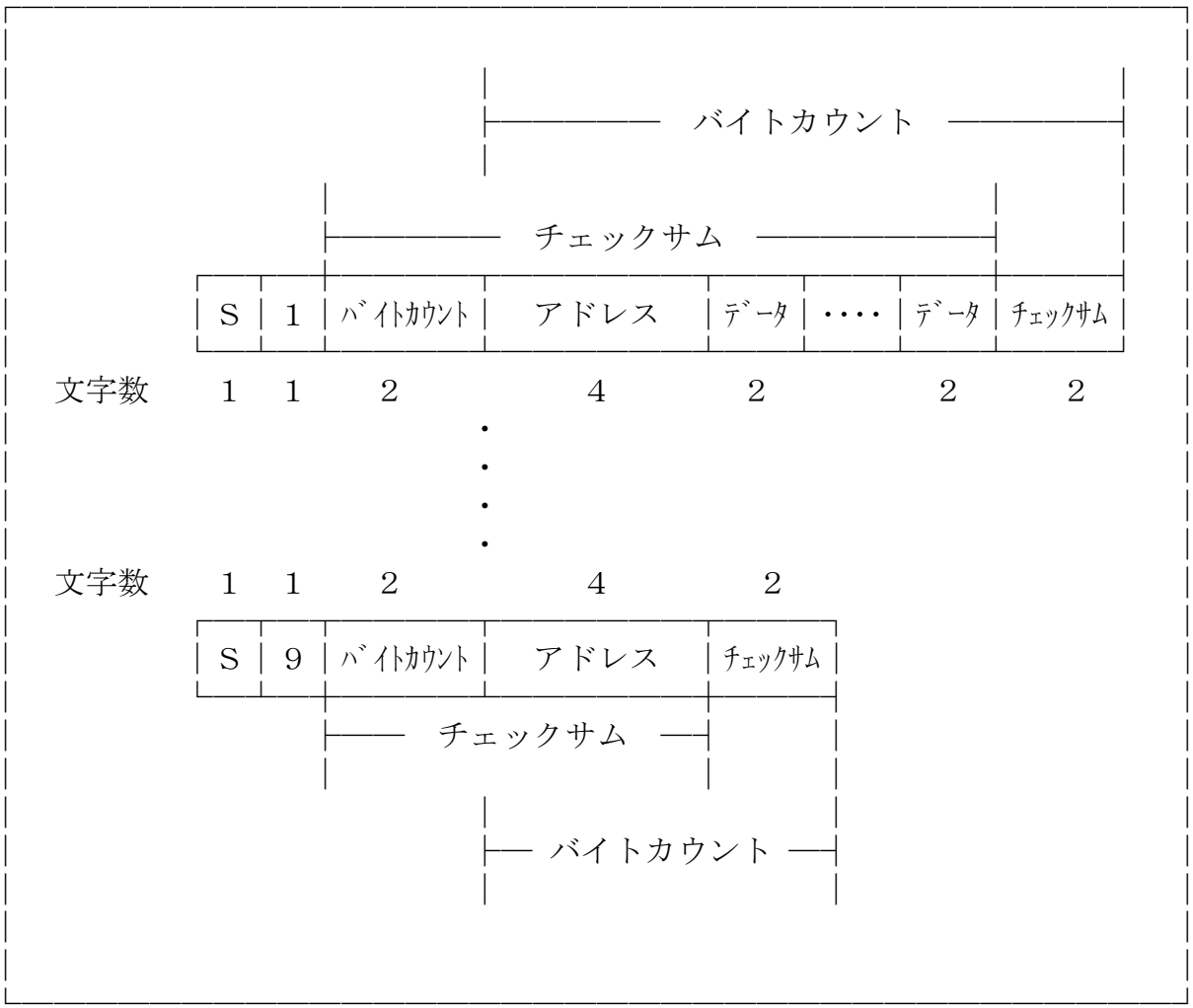
(9)オプション指定で「M」を指定するとマクロ関係の行を表示しなくなりますので、通常は「M」を指定した場合は「E」も指定した方がよいでしょう。

７－１　出力オブジェクトについて

クロスのアセンブラでは、生成されたオブジェクトはそのCPUでは実行できない為、実行可能ファイル(「.EXE」や「.COM」)にする必要がありません。
その為、68系のアセンブラではモトローラSフォーマットファイルを、80系のアセンブラではインテルHEXフォーマットファイルをデフォルトで出力します。
これらはテキストファイルになっていますので、「TYPE」コマンドで見ることができます。

●「MOTOROLA EXORMACS」フォーマット

通常、Sフォーマットと呼んでいます。このフォーマットは以下のように、「S」及び「0」から「9」、「A」から「F」までのキャラクタで構成されています。



「S1」の文字に以下の項目が続きます。

[バイトカウント]　2文字(00～FFまで)を1バイトとし、「アドレス」部から「チェックサム」部までのデータの個数を示します。
必ず、「13」(16進)以下の数値になります。

[アドレス]　先頭の「データ」を配置するアドレスを示します。
「0000」から「FFFF」までの値です。

[データ]　「00」から「FF」までのバイナリデータが続きます。

[チェックサム]　「バイトカウント」から「データ」の最後までの値を加算し、「FF」からその結果の下位8ビットを引いた値(1の補数)が「チェックサム」になります。

「S9」でオブジェクトの終わりを示し、「アドレス」部でエントリアドレスを示します。

```
        ORG     $1234
DATA    FCC     '0123456789'
        FCC     'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        FCC     'abcdefghijklmnopqrstuvwxyz'

        END     $100
```

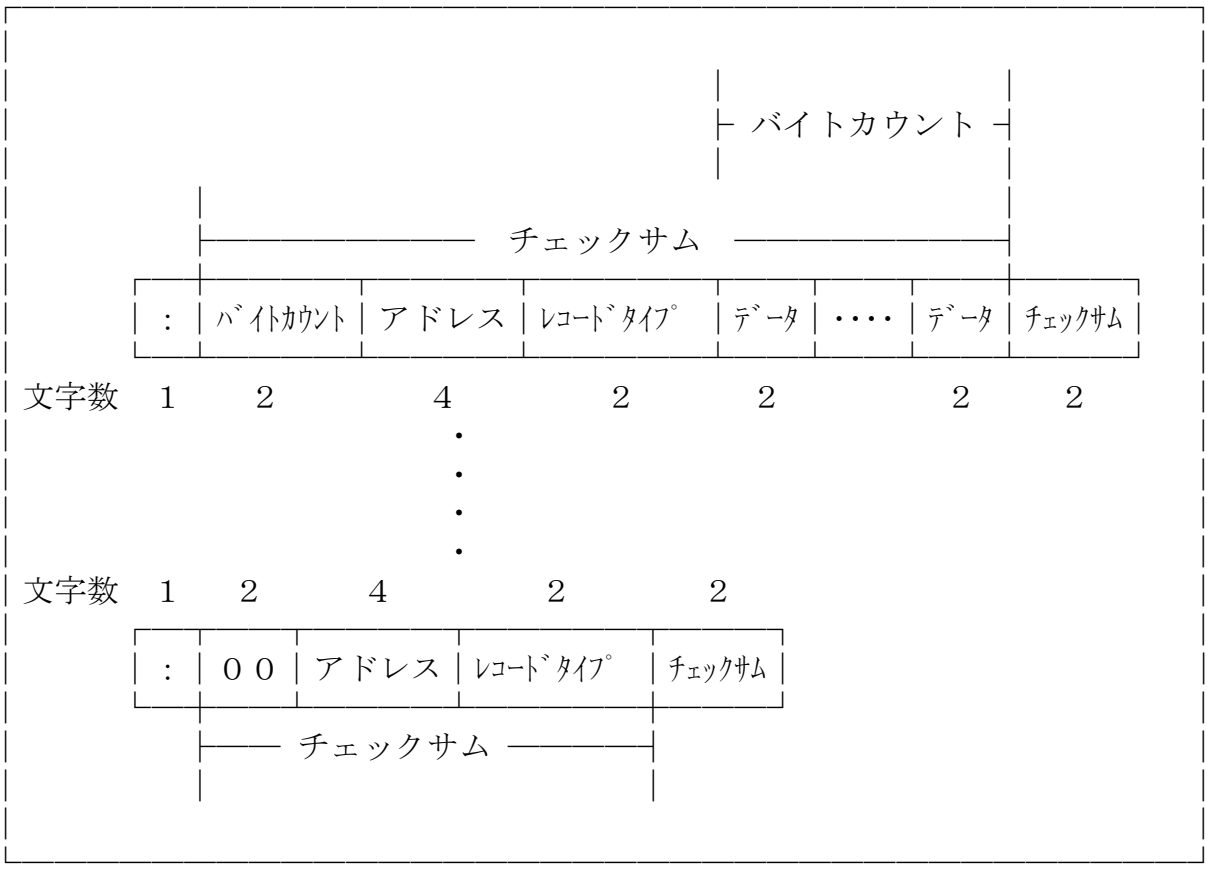
は以下のオブジェクトを出力します。

```
S113123430313233343536373839414243444546 04
   ~~^^^^||||||||||||||||||||||||||||||''
S11312444748494A4B4C4D4E4F50515253545556AE
   ~~^^^^||||||||||||||||||||||||||||||''
S11312545758595A6162636465666768696A6B6C56
   ~~^^^^||||||||||||||||||||||||||||||''
S11112646D6E6F707172737475767778797A27
   ~~^^^^||||||||||||||||||||||||||||''
S9030100FB
   ~~^^^^''
```

~~が「バイトカウント」
^^が「アドレス」
||が「データ」
''が「チェックサム」

●「INTELLEC HEX」フォーマット

通常、インテルヘキサフォーマットと呼んでいます。このフォーマットは以下のように、「:」及び「0」から「9」、「A」から「F」までのキャラクタで構成されています。



「:」の文字に以下の項目が続きます。

[バイトカウント] 2文字(00～FFまで)を1バイトとし、「データ」の個数を示します。
必ず、「10」(16進)以下の数値になります。

[アドレス] 先頭の「データ」を配置するアドレスを示します。
「0000」から「FFFF」までの値です。

[レコードタイプ] データ部(オブジェクトコード)は「00」になります。
終了時は「01」になり、データの終わりを知らせます。
その時の「先頭アドレス」部にはエントリーアドレスが入ります。

[データ] 「00」から「FF」までのバイナリデータが続きます。

[チェックサム] 「バイトカウント」から「データ」の最後までの値を加算し、
「00」からその結果の下位8ビットを引いた値(2の補数)が「チェックサム」になります。

```
        ORG     1234H
;
DATA    DB      '0123456789'
        DB      'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        DB      'abcdefghijklmnopqrstuvwxyz'
;
        END     100H
```

は以下のオブジェクトを出力します。

```
:1012340030313233343536373839414243444546 08
  ~~^^^^""||||||||||||||||||||||||||||||''
:10124400474849 4A4B4C4D4E4F50515253545556B2
  ~~^^^^""||||||||||||||||||||||||||||||''
:10125400575859 5A616263646566676869 6A6B6C5A
  ~~^^^^""||||||||||||||||||||||||||||||''
:101264006D6E6F70717273747576777879 7A2B
  ~~^^^^""||||||||||||||||||||||||||''
:00010001FE
  ~~^^^^""''
```

~~が「バイトカウント」
^^が「アドレス」
||が「データ」
''が「チェックサム」
""が「レコードタイプ」

７－２　ＲＯＭライターへの転送

●汎用タイプのＲＯＭライターの場合

RS232Cで接続する汎用型のROMライターの場合は非常に簡単です。
以下に手順を示します。

(1)ROMライター側の設定
・RS232C関係の設定(ボーレート、ビット長、パリティなど)
・ROMライターのロードアドレスの設定
・受け取るオブジェクトフォーマットの設定

(2)パソコン側の設定
・RS232C関係の設定(ボーレート、ビット長、パリティなど)

(3)ROMライターをオブジェクト受け取りモードにする

(4)パソコンからROMライターにオブジェクトを送る

```
A>COPY オブジェクト名 AUX□
```

または、

```
A>TYPE オブジェクト名 >AUX□
```

または、

各種の通信プログラムで、テキストファイルをそのまま転送する方法で行っても同じです。

●ボード型のROMライターの場合

パソコン本体に装着するボード型のROMライターの場合も比較的簡単です。ただし、その場合はそれに付属しているサポートソフトにも寄ります。
また、物によってはサポートソフトが付いていないものもあります。この場合は自分で作らなければならない為、非常にやっかいです。
サポートソフトの方で、モトローラSフォーマットやインテルHEXフォーマットを直接読んでROMに書いてくれるものでしたら何の手間もなく行えます。
また、ある特定のアドレスの内容をROMに書くようなソフトでしたら、本ソフトウェアに付属の「LOAD」ユーティリティーで実アドレスへのロードができる為、比較的かんたんに行えます。

７－３　他のパソコンへの転送

現在、市場に流れている8ビットパソコンには、

・NEC　　PC-8800シリーズ
・富士通　FM-7シリーズ
・SHARP　X1シリーズ

などがあります。

●汎用OSの場合

これらの8ビット機にもOSが走ります。PC-8800やX1にはCP/M-80、FM-7にはFLEXやOS/9、またZ80カードを装着すればCP/M-80も動作します。
OSが走っていれば、RS232Cを使ってオブジェクトを転送することは比較的かんたんです。
CP/MではPIPコマンドでRS232Cからのデータの受け取りができ、LOADコマンドでインテルHEXフォーマットを実行可能なCOMファイルに変換することができます。
FLEXでは標準ではRS232Cをサポートしていない為、別売のTERM-7(弊社で販売しています)を使うなどしなければなりません。
OS/9ではCOPYコマンドでRS232Cからのデータの受け取りができ、EXBINコマンドでモトローラSフォーマットを実行可能なファイルに変換することができます。

●その他の場合

DISK-BASIC上にオブジェクトを送りたい場合は、やはりそれなりのソフトを作成しなくてはなりません。
ただし、たいていのBASICではRS232Cをサポートしている為、オブジェクトを受け取ることはかんたんにできます。後は、そのオブジェクトのフォーマットを見ながらメモリ上にストアしていけば良いのです。ただし、その作成したソフトとロードしたいオブジェクトのアドレスが重なってはまずいので、オフセットロード機能を付けた方が良いでしょう。

書式に乗っ取った記述を行わないとアセンブラはエラーを出力します。

エラーは、ハード的なものと記述ミスで発生するものとに分けられます。

８－１　致命的なエラー

ハード的なエラーなどで、それ以上アセンブルを続行することは不可能なものです。これらのエラーを表示した後、アセンブルを中止します。

メッセージ	Can't open cross reference file
説　明	クロスリファレンスファイルが書き込みオープンできませんでした。 正確なファイル名を指定し、そのファイルが書き込み可能な状態にしてください。

メッセージ	[file] Can't open file
説　明	「LIB」などで他のファイルの呼び出しを行った時に、指定したファイルが見つかりませんでした。 正確なファイル名を指定してください。

メッセージ	Can't open listting file
説　明	リストファイルが書き込みオープンできませんでした。 正確なファイル名を指定し、そのファイルが書き込み可能な状態にしてください。

メッセージ	Can't open object file
説　明	オブジェクトファイルが書き込みオープンできませんでした。 正確なファイル名を指定し、そのファイルが書き込み可能な状態にしてください。

メッセージ	Can't open parameter file
説　明	指定したパラメータファイルが見つかりませんでした。 正確なファイル名を指定してください。

メッセージ	Can't open source file
説　明	指定したソースファイルが見つかりませんでした。 正確なファイル名を指定してください。

メッセージ	Can't open temporary file
説　明	クロスリファレンスを出力する時はテンポラリファイルを使用します。 そのファイルが書き込みオープンできませんでした。 「XREFTBL.$$$」という名でファイルをオープンしますので、このファイルを書き込み可能な状態にしてください。

メッセージ	Cross reference table space full
説　明	クロスリファレンスで使用するディスクのフリーエリアがなくなりました。 ディスクのフリーエリアを大きく開けてください。

メッセージ	Macro nesting over
説　明	マクロコールのネスティングレベルをオーバーしました。 マクロコールのネスティングは5レベルまでです。プログラムを変更してください。

メッセージ	Macro table space overflow
説　明	マクロを登録するバッファがいっぱいになりました。 マクロ登録を減らしてください。 本体にメモリをフル実装していない場合は、メモリを増やしても良いでしょう。

メッセージ	Option error
説　明	指定したオプションが間違っています。 もう１度、確認してください。

メッセージ	Out of heap space
説　明	アセンブラ起動時に確保するメモリが足りません。最大シンボル数を小さくしてください。 本体にメモリをフル実装していない場合は、メモリを増やしても良いでしょう。

メッセージ	Symbol table overflow
説　明	シンボル（ラベル）の最大登録数をオーバーしました。 最大登録数を広げるか、シンボルやラベルの数を減らしてください。

メッセージ	Symbol table space full
説　明	シンボル（ラベル）名を実際に登録するフリーエリアを使い尽くしました。 全体的にシンボル名を短くするか、シンボル数を減らしてください。 また、マクロを使っている場合は、そのマクロの定義体を減らしても良いでしょう。 本体にメモリをフル実装していない場合は、メモリを増やしても良いでしょう。

８－２　ソースラインのエラー

一般に記述ミスで、これらのエラーを表示した後もアセンブルは続行します。

メッセージ	Can't define macro
説　明	マクロ定義の中で再びマクロ定義を行おうとしています。 マクロ定義中は他のマクロ定義はできません。記述方法を変更してください。

メッセージ	Illegal label
説　明	決められた文字や数字以外はラベル名として使用できません。 定められた文字や数字でラベル名を付けてください。 また、レジスタ名や演算子などと同じ名前は使用できません。

メッセージ	Illegal operator
説　明	そのような命令はありません。

メッセージ	Internal error
説　明	内部エラーが発生しました。状況を正確に判断し弊社までお知らせください。 応急処置として、記述方法を変更するとエラーが発生しなくなる場合があります。

メッセージ	Invalid addressing mode
説　明	この命令では、そのアドレッシングモードは使用できません。

メッセージ	Invalid conditional structure
説　明	条件付きアセンブリの使用法が間違っています。 マニュアル(第５章)でもう１度確認してください。

メッセージ	Invalid forward reference
説　明	違法な前方参照を行っています。 シンボルを定義する時に他のシンボルを使って定義しようとした場合、 そのシンボルはその時点で定義済みでないといけません。

メッセージ	Invalid quoted string
説　明	コーテーション文字の使い方が間違っています。 文字列を指定できない命令でコーテーション文字を使用してはいけません。

メッセージ	Invalid register specification
説　明	その命令では、そのレジスタは使用できません。

メッセージ	Multiply defind symbol
説　明	同じ名前のラベルを再定義しました。 再定義をしたい場合には「=」命令などを使ってください。 二重定義を行っていないにもかかわらずこのエラーが発生した場合は、 パス1とパス2によるアドレスのずれが起こったことを意味します。

メッセージ	Phasing error detected
説　明	パス1、パス2によるアドレスのずれが生じました。 通常は、エラーの発生した行より前にその原因となるものがあります。

メッセージ	Relative branch to long
説　明	相対ジャンプの分岐先が遠すぎます。

メッセージ	Syntax error
説　明	文法が間違っています。

メッセージ	Unbalanced clause
説　明	括弧の左右の個数が一致しません。

メッセージ	Undefined symbol
説　明	未定義のシンボルを使用しました。

メッセージ	Unrecognizable mnemonic or macro
説　明	間違ったニーモニックの使用をしています。

メッセージ	Value out of range (warning)
説　明	指定した値が大きすぎます。 記述方法を変更すればエラーは取れますが、「W」オプションを付ける とこのエラーは表示されなくなります。

第９章　各アセンブラの詳細

この章では各アセンブラの特有の機能について説明します。
ただし、各ニーモニックの説明などは致しませんので、それらは各種の参考書籍をご覧ください。
ディスクの中にそれぞれテストプログラムが入っていますので、そちらの方も参考にしてください。

各アセンブラは、代表CPUの命令をベースにして、他のCPUの追加命令を付加した形で作られています。
つまり、「XZ80」では名こそ「Z80」ですが、実際はHD64180の追加命令を付加したアセンブラになっています。
その為、Z80のプログラムを書いたつもりで、誤ってHD64180の命令である「MLT」を使ってしまってもエラーにはなりません(もちろん、このプログラムは実行しても正常に動作しません)。

これを防ぐには、これから開発しようとしているCPUで、実行できない命令をマクロを使って強制的にエラーにしてしまうことで免れます。
ディスクの中に「Axxxx.TXT」というファイルが入っています(拡張命令のないアセンブラには入っていません)。
これは、拡張命令をエラーにするマクロ定義ファイルです。もし、HD64180の命令を使っていないかを調べたい場合は、

```
XZ80 /DXZ80 AZ80+TESTZ80; /Y
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

とすれば、その拡張命令を使った所がエラーになります。

６５０２編

●レジスタ名

A	X	Y

上記の名前はレジスタ名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	=	ASC	ASEG	BYTE	CHAR	CSEG	DB
DBYTE	DFB	DS	DSEG	DW	ELSE	END	ENDC
ENDIF	ENDM	EQU	ERR	EXITM	FCB	FCC	FCS
FCT	FDB	IF	IF1	IF2	IFC	IFDEF	IFEQ
IFF	IFGE	IFGT	IFLE	IFLT	IFN	IFNDEF	IFNE
IFP1	IFP2	IFT	LIB	MACRO	NAM	OPT	ORG
PAG	PAGE	PG	RES	RMB	RPT	RZB	SET
SPC	STTL	TTL	USE	WORD	ZERO		

●基本命令

ADC	AND	ASL	※ASLA	BCC	BCS	BEQ	※BHS
BIT	※BLO	BMI	BNE	BPL	BRK	BVC	BVS
CLC	CLD	CLI	CLV	CMP	CPX	CPY	DEC
DEX	DEY	EOR	INC	INX	INY	JMP	JSR
LDA	LDX	LDY	LSR	※LSRA	NOP	ORA	PHA
PHP	PLA	PLP	ROL	※ROLA	ROR	※RORA	RTI
RTS	SBC	SEC	SED	SEI	STA	STX	STY
TAX	TAY	TSX	TXA	TXS	TYA		

※印のついている命令は特別追加命令です。

・特別追加命令

BHS　　「BCC」と同じコードを発生 (Branch if higher or same)
BLO　　「BCS」と同じコードを発生 (Branch if lower)
ASLA　　「ASL A」と同様
LSRA　　「LSR A」と同様
ROLA　　「ROL A」と同様
RORA　　「ROR A」と同様

●絶対番地とゼロページ

6502にはゼロページというものがあります。これは、0番地からFF番地までのアドレスに対して1バイトで表現しようというものです。

アセンブラでは、指定した値(アドレス)により自動的にどちらにするかを決定します。

その値がシンボル(ラベル)の場合は、現時点でそのシンボルが定義されていれば、そのシンボルの値で決定します。定義されていない場合は、絶対番地(16ビット値)形式で行います。

例えば、

```
┌──────────────────────────┐
│DIRECT    EQU     $20     │        コード
│                          │      ┌─────────┐
│          LDA     DIRECT  │ ＝＞ │A5 20    │
│          LDA     DIRECT,X│      │B5 20    │
└──────────────────────────┘      └─────────┘
```

のように、シンボルを「LDA」命令より前で定義した場合はこの値が見れる為、ゼロページで展開できるもの($00～$FF)であればゼロページモードにします。

また、

```
┌──────────────────────────┐
│          LDA     DIRECT  │        コード
│          LDA     DIRECT,X│      ┌─────────┐
│                          │ ＝＞ │AD 20 00 │
│DIRECT    EQU     $20     │      │BD 20 00 │
└──────────────────────────┘      └─────────┘
```

のように、シンボルを「LDA」命令より後で定義した場合はこの値が見れない為、ゼロページで展開できるものでも絶対番地モードにします。

しかし、

```
┌──────────────────────────┐
│          LDA     <DIRECT │        コード
│          LDA     <DIRECT,X      ┌─────────┐
│                          │ ＝＞ │A5 20    │
│DIRECT    EQU     $20     │      │B5 20    │
└──────────────────────────┘      └─────────┘
```

のように、シンボル指定の前に「<」を付けると、いかなる場合でもゼロページモードで展開します。つまり、このシンボルの値が$FFより大きい場合でも、下位8ビットに切捨て強制的にゼロページモードで展開するということです。

また、これとは逆に強制的に絶対番地モードにするものもあります。以下のようにシンボル指定の前に「>」を付けるとこのモードになります。

```
┌──────────────────────────┐
│DIRECT    EQU     $20     │        コード
│                          │      ┌─────────┐
│          LDA     >DIRECT │ ＝＞ │AD 20 00 │
│          LDA     >DIRECT,X      │BD 20 00 │
└──────────────────────────┘      └─────────┘
```

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
アセンブリ・プログラム 6502編	倍風館
6502マイクロコンピュータのプログラミング	日刊工業新聞社

６８０１編

●レジスタ名

A	B	D	X

上記の名前はレジスタ名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	=	ASEG	BYTE	CHAR	CSEG	DBYTE	DSEG
ELSE	END	ENDC	ENDIF	ENDM	EQU	ERR	EXITM
FCB	FCC	FCS	FCT	FDB	IF	IF1	IF2
IFC	IFDEF	IFEQ	IFF	IFGE	IFGT	IFLE	IFLT
IFN	IFNDEF	IFNE	IFP1	IFP2	IFT	LIB	MACRO
NAM	OPT	ORG	PAG	PAGE	PG	RES	RMB
RPT	RZB	SET	SPC	STTL	TTL	USE	WORD
ZERO							

●6800/2の基本命令

ABA	ADCA	ADCB	ADDA	ADDB	ANDA	ANDB	ASL
ASLA	ASLB	ASR	ASRA	ASRB	BCC	BCS	BEQ
BGE	BGT	BHI	※BHS	BITA	BITB	BLE	※BLO
BLS	BLT	BMI	BNE	BPL	BRA	BSR	BVC
BVS	CBA	CLC	CLI	CLR	CLRA	CLRB	CLV
CMPA	CMPB	COM	COMA	COMB	CPX	DAA	DEC
DECA	DECB	DES	DEX	EORA	EORB	INC	INCA
INCB	INS	INX	JMP	JSR	LDAA	LDAB	LDS
LDX	※LSL	※LSLA	※LSLB	LSR	LSRA	LSRB	NEG
NEGA	NEGB	NOP	ORAA	ORAB	PSHA	PSHB	PULA
PULB	ROL	ROLA	ROLB	ROR	RORA	RORB	RTI
RTS	SBA	SBCA	SBCB	SEC	SEI	SEV	STAA
STAB	STS	STX	SUBA	SUBB	SWI	TAB	TAP
TBA	TPA	TST	TSTA	TSTB	TSX	TXS	WAI

※印のついている命令は特別追加命令です。

・特別追加命令

BHS　　「BCC」と同じコードを発生 (Branch if higher or same)
BLO　　「BCS」と同じコードを発生 (Branch if lower)
LSL　　「ASL」と同様
LSLA　「ASLA」と同様
LSLB　「ASLB」と同様

●6801/3の追加基本命令

ABX	ADDD	ASLD	BRN	※LDAD	LDD	※LSLD	LSRD
MUL	PSHX	PULX	※STAD	STD	SUBD		

6801/3は、6800/2の基本命令に加え上記の命令が追加されています。

・特別追加命令

LDAD　　「LDD」と同様
LSLD　　「ASLD」と同様
STAD　　「STD」と同様

●6301/3の追加基本命令

AIM	※BCLR	※BSET	※BTGL	※BTST	EIM	OIM	SLP
TIM	XGDX						

6301/3は、6801/3の基本命令に加え上記の命令が追加されています。

・特別追加命令

BCLR　　「AIM」の変形命令で指定ビットをクリアするものです。

構文(1)　BCLR　<BIT>,<MEM>
構文(2)　BCLR　<BIT>,<DISP>,X

構文(1)は<MEM>のダイレクトアドレスの<BIT>のビットをクリアしてそのアドレスに格納します。
構文(2)はXレジスタのアドレスに<DISP>のオフセット値を加えたアドレスの<BIT>のビットをクリアしてそのアドレスに格納します。

BSET　「OIM」の変形命令で指定ビットをセットするものです。

構文(1)　BSET　<BIT>,<MEM>
構文(2)　BSET　<BIT>,<DISP>,X

構文(1)は<MEM>のダイレクトアドレスの<BIT>のビットをセットしてそのアドレスに格納します。
構文(2)はXレジスタのアドレスに<DISP>のオフセット値を加えたアドレスの<BIT>のビットをセットしてそのアドレスに格納します。

BTGL　「EIM」の変形命令で指定ビットを反転するものです。

構文(1)　BTGL　<BIT>,<MEM>
構文(2)　BTGL　<BIT>,<DISP>,X

構文(1)は<MEM>のダイレクトアドレスの<BIT>のビットを反転してそのアドレスに格納します。
構文(2)はXレジスタのアドレスに<DISP>のオフセット値を加えたアドレスの<BIT>のビットを反転してそのアドレスに格納します。

BTST　「TIM」の変形命令で指定ビットをテストするものです。

構文(1)　BTST　<BIT>,<MEM>
構文(2)　BTST　<BIT>,<DISP>,X

構文(1)は<MEM>のダイレクトアドレスの<BIT>のビットをテストしてフラグを変化させます。
構文(2)はXレジスタのアドレスに<DISP>のオフセット値を加えたアドレスの<BIT>のビットをテストしてフラグを変化させます。

<BIT>は0から7までの数値でビット位置を表わします。
<MEM>は8ビットのダイレクト値(0番地からFF番地)を指定します。
<DISP>はXレジスタの8ビットオフセット値を指定します。

●エクステンディッドとダイレクト

6801にはダイレクトページというものがあります。これは、0番地からFF番地までのアドレスに対して1バイトで表現しようというものです。
アセンブラでは、指定した値(アドレス)により自動的にどちらにするかを決定します。
その値がシンボル(ラベル)の場合は、現時点でそのシンボルが定義されていれば、そのシンボルの値で決定します。定義されていない場合は、絶対番地(16ビット値)形式で行います。
例えば、

```
┌──────────────────────────┐
│DIRECT    EQU     $20     │       コード
│                          │     ┌─────────┐
│          LDAA    DIRECT  │ ＝＞ │96 20    │
└──────────────────────────┘     └─────────┘
```

のように、シンボルを「LDAA」命令より前で定義した場合はこの値が見れる為、ダイレクトアドレッシングで展開できるもの($00～$FF)であればダイレクトアドレッシングにします。
また、

```
┌──────────────────────────┐
│          LDAA    DIRECT  │       コード
│                          │     ┌─────────┐
│DIRECT    EQU     $20     │ ＝＞ │B6 00 20 │
└──────────────────────────┘     └─────────┘
```

のように、シンボルを「LDAA」命令より後で定義した場合はこの値が見れない為、ダイレクトアドレッシングで展開できるものでもエクステンディッドアドレッシングにします。
しかし、

```
┌──────────────────────────┐
│          LDAA    <DIRECT │       コード
│                          │     ┌─────────┐
│DIRECT    EQU     $20     │ ＝＞ │96 20    │
└──────────────────────────┘     └─────────┘
```

のように、シンボル指定の前に「<」を付けると、いかなる場合でもダイレクトアドレッシングで展開します。つまり、このシンボルの値が$FFより大きい場合でも、下位8ビットに切捨て強制的にダイレクトアドレッシングで展開するということです。
また、これとは逆に強制的に絶対番地モードにするものもあります。以下のようにシンボル指定の前に「>」を付けるとこのモードになります。

```
┌──────────────────────────┐
│DIRECT    EQU     $20     │       コード
│                          │     ┌─────────┐
│          LDAA    >DIRECT │ ＝＞ │B6 00 20 │
└──────────────────────────┘     └─────────┘
```

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
日立マイクロコンピュータ	日立製作所
マイクロコンピュータソフトウェア基礎技術	ラジオ技術社
6800/6809による図解マイコンアセンブラ入門	オーム社

６８０５編

●レジスタ名

X

上記の名前はレジスタ名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	=	ASEG	BYTE	CHAR	CSEG	DBYTE	DSEG
ELSE	END	ENDC	ENDIF	ENDM	EQU	ERR	EXITM
FCB	FCC	FCS	FCT	FDB	IF	IF1	IF2
IFC	IFDEF	IFEQ	IFF	IFGE	IFGT	IFLE	IFLT
IFN	IFNDEF	IFNE	IFP1	IFP2	IFT	LIB	MACRO
NAM	OPT	ORG	PAG	PAGE	PG	RES	RMB
RPT	RZB	SET	SPC	STTL	TTL	USE	WORD
ZERO							

●6805の基本命令

ADC	ADD	AND	ASL	ASLA	ASLX	ASR	ASRA
ASRX	BCC	BCLR	BCS	BEQ	BHCC	BHCS	BHI
※BHS	BIH	BIL	BIT	※BLO	BLS	BMC	BMI
BMS	BNE	BPL	BRA	BRCLR	BRN	BRSET	BSET
BSR	CLC	CLI	CLR	CLRA	CLRX	CMP	COM
COMA	COMX	CPX	DEC	DECA	DECX	EOR	INC
INCA	INCX	JMP	JSR	LDA	LDX	LSL	LSLA
LSLX	LSR	LSRA	LSRX	NEG	NEGA	NEGX	NOP
ORA	ROL	ROLA	ROLX	ROR	RORA	RORX	RSP
RTI	RTS	SBC	SEC	SEI	STA	STX	SUB
SWI	TAX	TST	TSTA	TSTX	TXA		

※印のついている命令は特別追加命令です。

・特別追加命令

BHS　　「BCC」と同じコードを発生 (Branch if higher or same)
BLO　　「BCS」と同じコードを発生 (Branch if lower)

●6305の追加基本命令

```
DAA     STOP    WAIT
```

6305は、6805の基本命令に加え上記の命令が追加されています。

●63705の追加基本命令

```
MUL
```

63705は、6305の基本命令に加え上記の命令が追加されています。

●エクステンディッドとダイレクト

6805にはダイレクトページというものがあります。これは、0番地からFF番地までのアドレスに対して1バイトで表現しようというものです。
アセンブラでは、指定した値(アドレス)により自動的にどちらにするかを決定します。
その値がシンボル(ラベル)の場合は、現時点でそのシンボルが定義されていれば、そのシンボルの値で決定します。定義されていない場合は、絶対番地(16ビット値)形式で行います。
また、インデックスドアドレッシング時のオフセットも同じように処理します。

例えば、

```
DIRECT    EQU     $20          |        コード
                               |     ┌─────────┐
          LDA     DIRECT       | ＝> | B6 20   |
          LDA     DIRECT,X     |     | E6 20   |
                               |     └─────────┘
```

のように、シンボルを「LDA」命令より前で定義した場合はこの値が見れる為、ダイレクトアドレッシングで展開できるもの($00～$FF)であればダイレクトアドレッシングにします。
また、

```
          LDA     DIRECT       |        コード
          LDA     DIRECT,X     |     ┌─────────┐
                               | ＝> | C6 00 20|
DIRECT    EQU     $20          |     | D6 00 20|
                               |     └─────────┘
```

のように、シンボルを「LDA」命令より後で定義した場合はこの値が見れない為、ダイレクトアドレッシングで展開できるものでもエクステンディッドアドレッシングにします。

しかし、

```
┌──────────────────────────┐
│          LDA     <DIRECT   │      コード
│          LDA     <DIRECT,X │   ┌──────┐
│                            │ ＝＞ │B6 20   │
│DIRECT    EQU     $20       │   │E6 20   │
└──────────────────────────┘   └──────┘
```

のように、シンボル指定の前に「<」を付けると、いかなる場合でもダイレクトアドレッシングで展開します。つまり、このシンボルの値が$FFより大きい場合でも、下位8ビットに切捨て強制的にダイレクトアドレッシングで展開するということです。

また、これとは逆に強制的に絶対番地モードにするものもあります。以下のようにシンボル指定の前に「>」を付けるとこのモードになります。

```
┌──────────────────────────┐
│DIRECT    EQU     $20       │      コード
│                            │   ┌──────┐
│          LDA     >DIRECT   │ ＝＞ │C6 00 20│
│          LDA     >DIRECT,X │   │D6 00 20│
└──────────────────────────┘   └──────┘
```

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
日立マイクロコンピュータ	日立製作所

６８０９編

●レジスタ名

A	B	D	S	U	X	Y	Z ~
CC	PC	DP					

上記の名前はレジスタ名となる為、単独ではシンボル名として使用できません。
※「Z」はアセンブラ内部で使用されているものです。

●擬似命令

*=	=	ASEG	BYTE	CHAR	CSEG	DBYTE	DSEG
ELSE	END	ENDC	ENDIF	ENDM	EQU	ERR	EXITM
FCB	FCC	FCS	FCT	FDB	IF	IF1	IF2
IFC	IFDEF	IFEQ	IFF	IFGE	IFGT	IFLE	IFLT
IFN	IFNDEF	IFNE	IFP1	IFP2	IFT	LIB	MACRO
NAM	OPT	ORG	PAG	PAGE	PG	REG	RES
RMB	RPT	RZB	SET	SETDP	SPC	STTL	TTL
USE	WORD	ZERO					

●6809の基本命令

※ABA	ABX	ADCA	ADCB	ADDA	ADDB	ADDD	ANDA
ANDB	ANDCC	ASL	ASLA	ASLB	※ASLD	ASR	ASRA
ASRB	BCC	BCS	※BEC	BEQ	※BES	BGE	BGT
BHI	BHS	BITA	BITB	BLE	BLO	BLS	BLT
BMI	BNE	BPL	BRA	BRN	BSR	BVC	BVS
※CBA	※CLC	※CLF	※CLI	CLR	CLRA	CLRB	※CLRD
※CLV	※CLZ	CMPA	CMPB	CMPD	CMPS	CMPU	CMPX
CMPY	COM	COMA	COMB	※CPX	CWAI	DAA	DEC
DECA	DECB	※DES	※DEX	EORA	EORB	EXG	INC
INCA	INCB	※INS	※INX	JMP	JSR	LBCC	LBCS
※LBEC	LBEQ	※LBES	LBGE	LBGT	LBHI	LBHS	LBLE
LBLO	LBLS	LBLT	LBMI	LBNE	LBPL	LBRA	LBRN
LBSR	LBVC	LBVS	LDA	※LDAA	※LDAB	※LDAD	LDB
LDD	LDS	LDU	LDX	LDY	LEAS	LEAU	LEAX
LEAY	LSL	LSLA	LSLB	LSR	LSRA	LSRB	※LSRD
MUL	NEG	NEGA	NEGB	※NEGD	NOP	ORA	※ORAA
※ORAB	ORB	ORCC	※PSHA	※PSHB	PSHS	PSHU	※PSHX
※PULA	※PULB	PULS	PULU	※PULX	※RESET	※RHF	ROL
ROLA	ROLB	ROR	RORA	RORB	RTI	RTS	※SBA
SBCA	SBCB	※SEC	※SEF	※SEI	※SEV	SEX	※SEZ
STA	※STAA	※STAB	※STAD	STB	STD	STS	STU
STX	STY	SUBA	SUBB	SUBD	SWI	SWI2	SWI3

SYNC	※TAB	※TAP	※TBA	TFR	※TPA	TST	TSTA
TSTB	※TSTD	※TSX	※TXS	※WAI			

※印のついている命令は特別追加命令です。

●6809変換機能(特別追加命令)

6809アセンブラには、6800や6801の命令を6809の命令に変換するトランスレート機能があります。つまり、6800/1で作成したソースがそのまま6809のコードに変換できるということです。
また、それ以外に便利な命令を数個追加しています。

・6800/1の命令

命　令	変換後の命令
ABA	PSHS B ADDA ,S+
CBA	PSHS B CMPA ,S+
CLC	ANDCC #$FE
CLI	ANDCC #$BF
CLV	ANDCC #$FD
CPX xxx	CMPX xxx
DES	LEAS -1,S
DEX	LEAX -1,X
INS	LEAS 1,S
INX	LEAX 1,X
LDAA xxx	LDA xxx
LDAB xxx	LDB xxx
ORAA xxx	ORA xxx
ORAA xxx	ORB xxx

命　令	変換後の命令
PSHA	PSHS A
PSHB	PSHS B
PULA	PULS A
PULB	PULS B
SBA	PSHS B SUBA ,S+
SEC	ORCC #$01
SEI	ORCC #$10
SEV	ORCC #$02
STAA xxx	STA xxx
STAB xxx	STB xxx
TAB	TFR A,B TSTA
TAP	TFR A,CC
TBA	TFR B,A TSTA
TPA	TFR CC,A

命　令	変換後の命令
TSTD	STD -2,S
TSX	TFR S,X
TXS	TFR X,S
WAI	CWAI #$FF

・6801の命令

命　令	変換後の命令
ASLD	ASLB ROLA
LDAD xxx	LDD　xxx

命　令	変換後の命令
PSHX	PSHS　X
PULX	PULS　X
STAD xxx	STD　xxx

・その他の命令

命　令	変換後の命令	命令の読み方(意味)
BEC　xxx	BCC　xxx	Branch short if error clear
BES	BCS	Branch short if error set
CLF	ANDCC #$BF	Clear FIRQ interrupt mask
CLRD	CLRA CLRB	Clear D register
CLZ	ANDCC #$FB	Clear zero condition code bit
LBEC xxx	LBCC　xxx	Branch long if error clear
LBES xxx	LBCS　xxx	Branch long if error set
NEGD	COMA COMB SUBD　#$FFFF	Negate D register
SEF　xxx	ORCC　#$40	Set FIRQ interrupt mask
SEZ　xxx	ORCC　#$04	Set zero condition code bit
TSTD	STD　-2,S	Test D register

●エクステンディッドとダイレクト

6809にはダイレクトページというものがあります。これは、DPレジスタの値を上位アドレスとし、下位アドレスのみを指定して1バイトで表現しようというものです。
アセンブラでは、DPレジスタの内容を見ることはできない為、「SETDP」命令で故意的に指定します(指定しなかった場合は、0と指定したことになる)。

ダイレクトとエクステンディッドは、指定した値(アドレス)により自動的にどちらにするかを決定します。その値がシンボル(ラベル)の場合は、現時点でそのシンボルが定義されていれば、そのシンボルの値で決定します。定義されていない場合は、絶対番地(16ビット値、つまりエクステンディッド)形式で行います。
また、インデックスドアドレッシング時のオフセットも同じように処理します。
例えば、

```
┌──────────────────────────┐
│DIRECT   EQU     $20      │        コード
│                          │      ┌─────────┐
│         LDA     DIRECT   │ ＝＞ │96 20    │
│         LDA     DIRECT,X │      │A6 88 20 │
└──────────────────────────┘      └─────────┘
```

のように、シンボルを「LDA」命令より前で定義した場合はこの値が見れる為、ダイレクトアドレッシングで展開できるもの($00～$FF)であればダイレクトアドレッシングやインデックス8(5または0)ビットオフセットにします。
また、

```
┌──────────────────────────┐
│         LDA     DIRECT   │        コード
│         LDA     DIRECT,X │      ┌──────────────┐
│                          │ ＝＞ │B6 00 20      │
│DIRECT   EQU     $20      │      │A6 89 00 20   │
└──────────────────────────┘      └──────────────┘
```

のように、シンボルを「LDA」命令より後で定義した場合はこの値が見れない為、ダイレクトアドレッシングやインデックス8ビットオフセットで展開できるものでもエクステンディッドアドレッシングやインデックス16ビットオフセットにします。
しかし、

```
┌──────────────────────────┐
│         LDA     <DIRECT  │        コード
│         LDA     <DIRECT,X│      ┌─────────┐
│                          │ ＝＞ │96 20    │
│DIRECT   EQU     $20      │      │A6 88 20 │
└──────────────────────────┘      └─────────┘
```

のように、シンボル指定の前に「<」を付けると、いかなる場合でもダイレクトアドレッシングや8ビットオフセットのインデックスで展開します。つまり、このシンボルの値が$FFより大きい場合でも、下位8ビットに切捨て強制的にダイレクトアドレッシング8ビットオフセットのインデックスで展開するということです。
また、これとは逆に強制的に絶対番地モードにするものもあります。以下のようにシンボル指定の前に「>」を付けるとこのモードになります。

```
┌──────────────────────────┐
│DIRECT   EQU     $20      │        コード
│                          │      ┌──────────────┐
│         LDA     >DIRECT  │ ＝＞ │B6 00 20      │
│         LDA     >DIRECT,X│      │A6 89 00 20   │
└──────────────────────────┘      └──────────────┘
```

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
MC6809-MC6809E　マイクロプロセッサ プログラミング　マニュアル	ＣＱ出版社
マイクロコンピュータの内部構造と機械語 6809CPU プログラミング　マニュアル	ＣＱ出版社
6809CPU プログラミング入門	ＣＱ出版社
マイコンピュータ No.6 「完全理解6809のすべて」	ＣＱ出版社
6809ハンドブック	アスキー出版
日立マイクロコンピュータ	日立製作所
マイクロコンピュータ　ソフトウェア基礎技術	ラジオ技術社
インターフェース 1981年9月号「6809システムの徹底研究」	ＣＱ出版社

６８ＨＣ１１編

●レジスタ名

A	B	D	X	Y

上記の名前はレジスタ名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	=	ASEG	BYTE	CHAR	CSEG	DBYTE	DSEG
ELSE	END	ENDC	ENDIF	ENDM	EQU	ERR	EXITM
FCB	FCC	FCS	FCT	FDB	IF	IF1	IF2
IFC	IFDEF	IFEQ	IFF	IFGE	IFGT	IFLE	IFLT
IFN	IFNDEF	IFNE	IFP1	IFP2	IFT	LIB	MACRO
NAM	OPT	ORG	PAG	PAGE	PG	RES	RMB
RPT	RZB	SET	SPC	STTL	TTL	USE	WORD
ZERO							

●6800/2の基本命令

ABA	ABX	ABY	ADCA	ADCB	ADDA	ADDB	ADDD
ANDA	ANDB	ASL	ASLA	ASLB	ASLD	ASR	ASRA
ASRB	BCC	BCLR	BCS	BEQ	BGE	BGT	BHI
BHS	BITA	BITB	BLE	BLO	BLS	BLT	BMI
BNE	BPL	BRA	BRCLR	BRN	BRSET	BSET	BSR
BVC	BVS	CBA	CLC	CLI	CLR	CLRA	CLRB
CLV	CMPA	CMPB	COM	COMA	COMB	CPD	CPX
CPY	DAA	DEC	DECA	DECB	DES	DEX	DEY
EORA	EORB	FDIV	IDIV	INC	INCA	INCB	INS
INX	INY	JMP	JSR	LDAA	LDAB	LDD	LDS
LDX	LDY	LSL	LSLA	LSLB	LSLD	LSR	LSRA
LSRB	LSRD	MUL	NEG	NEGA	NEGB	NOP	ORAA
ORAB	PSHA	PSHB	PSHX	PSHY	PULA	PULB	PULX
PULY	ROL	ROLA	ROLB	ROR	RORA	RORB	RTI
RTS	SBA	SBCA	SBCB	SEC	SEI	SEV	STAA
STAB	STD	STOP	STS	STX	STY	SUBA	SUBB
SUBD	SWI	TAB	TAP	TBA	TEST	TPA	TST
TSTA	TSTB	TSX	TSY	TXS	TYS	WAI	XGDX
XGDY							

●エクステンディッドとダイレクト

68HC11にはダイレクトページというものがあります。これは、0番地からFF番地までのアドレスに対して1バイトで表現しようというものです。
アセンブラでは、指定した値(アドレス)により自動的にどちらにするかを決定します。
その値がシンボル(ラベル)の場合は、現時点でそのシンボルが定義されていれば、そのシンボルの値で決定します。定義されていない場合は、絶対番地(16ビット値)形式で行います。
例えば、

```
┌──────────────────────────┐
│DIRECT    EQU     $20     │       コード
│                          │     ┌─────────┐
│          LDAA    DIRECT  │ ＝＞ │96 20    │
└──────────────────────────┘     └─────────┘
```

のように、シンボルを「LDAA」命令より前で定義した場合はこの値が見れる為、ダイレクトアドレッシングで展開できるもの($00～$FF)であればダイレクトアドレッシングにします。
また、

```
┌──────────────────────────┐
│          LDAA    DIRECT  │       コード
│                          │     ┌─────────┐
│DIRECT    EQU     $20     │ ＝＞ │B6 00 20 │
└──────────────────────────┘     └─────────┘
```

のように、シンボルを「LDAA」命令より後で定義した場合はこの値が見れない為、ダイレクトアドレッシングで展開できるものでもエクステンディッドアドレッシングにします。
しかし、

```
┌──────────────────────────┐
│          LDAA    <DIRECT │       コード
│                          │     ┌─────────┐
│DIRECT    EQU     $20     │ ＝＞ │96 20    │
└──────────────────────────┘     └─────────┘
```

のように、シンボル指定の前に「<」を付けると、いかなる場合でもダイレクトアドレッシングで展開します。つまり、このシンボルの値が$FFより大きい場合でも、下位8ビットに切捨て強制的にダイレクトアドレッシングで展開するということです。
また、これとは逆に強制的に絶対番地モードにするものもあります。以下のようにシンボル指定の前に「>」を付けるとこのモードになります。

```
┌──────────────────────────┐
│DIRECT    EQU     $20     │       コード
│                          │     ┌─────────┐
│          LDAA    >DIRECT │ ＝＞ │B6 00 20 │
└──────────────────────────┘     └─────────┘
```

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
M68HC11 HCMOS Single-Chip Microcomputer Programmer's Reference Manual	モトローラ

８０４８編

●レジスタ・IO名

A	AN0	AN1	BUS	C	CLK	CNT	DBB
DMA	F0	F1	FLAGS	I	MB0	MB1	P0
P1	P2	P3	P4	P5	P6	P7	PSW
R0	R1	R2	R3	R4	R5	R6	R7
RB0	RB1	STS	T	TCNT	TCNTI		

上記の名前はレジスタやIO名となる為、単独ではシンボル名として使用できません。

●演算子名

AND	EQ	GE	GT	HIGH	LE	LOW	LT
MOD	NE	NOT	OR	SHL	SHR	XOR	

上記の名前は演算子名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	*EJECT	*HEADING	*INCLUDE	*LIST	=	ASEG	BLK
BYTE	CHAR	COND	CSEG	DB	DBYTE	DD	DEFB
DEFD	DEFL	DEFM	DEFS	DEFT	DEFW	DM	DS
DSEG	DT	DW	ELSE	END	ENDC	ENDIF	ENDM
EQU	ERR	EXITM	FCB	FCC	FCS	FCT	FDB
IF	IF1	IF2	IFC	IFDEF	IFEQ	IFGE	IFGT
IFLE	IFLT	IFN	IFNDEF	IFNE	IFP1	IFP2	IFT
INCLUDE	LIB	LIST	MACRO	NAM	NAME	OPT	ORG
PAG	PAGE	PG	RMB	RPT	RZB	SET	SETL
SPC	STTL	SUBTTL	TEXT	TITLE	TTL	USE	WORD
XLIST	ZERO						

擬似命令や基本命令をシンボル名として使用したい場合は、名前の後ろに「:」が必要です。

●8048の基本命令

ADD	ADDC	ANL	ANLD	CALL	CLR	CPL	DA
DEC	DIS	DJNZ	EN	ENT0	IN	INC	INS
JB0	JB1	JB2	JB3	JB4	JB5	JB6	JB7
JC	JF0	JF1	JMP	JMPP	JNC	JNI	JNT0
JNT1	JNZ	JT0	JT1	JTF	JZ	MOV	MOVD
MOVP	MOVP3	MOVX	NOP	ORL	ORLD	OUTL	RET
RETR	RL	RLC	RR	RRC	SEL	STOP	STRT
SWAP	XCH	XCHD	XRL				

●8041の基本命令

ADD	ADDC	ANL	ANLD	CALL	CLR	CPL	DA
DEC	DIS	DJNZ	EN	IN	INC	JB0	JB1
JB2	JB3	JB4	JB5	JB6	JB7	JC	JF0
JF1	JMP	JMPP	JNC	JNIBF	JNT0	JNT1	JNZ
JOBF	JT0	JT1	JTF	JZ	MOV	MOVD	MOVP
MOVP3	NOP	ORL	ORLD	OUT	OUTL	RET	RETR
RL	RLC	RR	RRC	SEL	STOP	STRT	SWAP
XCH	XCHD	XRL					

●8022の基本命令

ADD	ADDC	ANL	ANLD	CALL	CLR	CPL	DA
DEC	DIS	DJNZ	EN	IN	INC	JC	JMP
JMPP	JNC	JNT0	JNT1	JNZ	JT0	JT1	JTF
JZ	MOV	MOVD	MOVP	NOP	ORL	ORLD	OUTL
RAD	RET	RETI	RL	RLC	RR	RRC	SEL
STOP	STRT	SWAP	XCH	XCHD	XRL		

●8021の基本命令

ADD	ADDC	ANL	ANLD	CALL	CLR	CPL	DA
DEC	DJNZ	IN	INC	JC	JMP	JMPP	JNC
JNT1	JNZ	JT1	JTF	JZ	MOV	MOVD	MOVP
NOP	ORL	ORLD	OUTL	RET	RL	RLC	RR
RRC	STOP	STRT	SWAP	XCH	XCHD	XRL	

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
MSC-48/UPI-41 アセンブリ言語マニュアル	インテルジャパン

８０５１編

●レジスタ・IO名

A	AB	AC	ACC	B	C	CY	DPH
DPL	DPTR	EA	ES	ET0	ET1	EX0	EX1
EXTI0	EXTI1	F0	IE	IE0	IE1	IP	INT0
INT1	IT0	IT1	OV	P	P0	P1	P2
P3	PC	PS	PSW	PT0	PT1	PX0	PX1
R0	R1	R2	R3	R4	R5	R6	R7
RB8	RD	REN	RESET	RI	RS0	RS1	RXD
SBUF	SCON	SINT	SM0	SM1	SM2	T0	T1
SP	TB8	TCON	TF0	TF1	TH0	TH1	TI
TIMER0	TIMER1	TL0	TL1	TMOD	TR0	TR1	TXD
WR							

上記の名前はレジスタやIO名となる為、単独ではシンボル名として使用できません。

●演算子名

AND	EQ	GE	GT	HIGH	LE	LOW	LT
MOD	NE	NOT	OR	SHL	SHR	XOR	

上記の名前は演算子名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	*EJECT	*HEADING	*INCLUDE	*LIST	=	ASEG	BLK
BYTE	CHAR	COND	CSEG	DB	DBYTE	DD	DEFB
DEFD	DEFL	DEFM	DEFS	DEFT	DEFW	DM	DS
DSEG	DT	DW	ELSE	END	ENDC	ENDIF	ENDM
EQU	ERR	EXITM	FCB	FCC	FCS	FCT	FDB
IF	IF1	IF2	IFC	IFDEF	IFEQ	IFGE	IFGT
IFLE	IFLT	IFN	IFNDEF	IFNE	IFP1	IFP2	IFT
INCLUDE	LIB	LIST	MACRO	NAM	NAME	OPT	ORG
PAG	PAGE	PG	RMB	RPT	RZB	SET	SETL
SPC	STTL	SUBTTL	TEXT	TITLE	TTL	USE	WORD
XLIST	ZERO						

擬似命令や基本命令をシンボル名として使用したい場合は、名前の後ろに「:」が必要です。

●基本命令

ACALL	ADD	ADDC	AJMP	ANL	※CALL	CJNE	CLR
CPL	DA	DEC	DIV	DJNZ	INC	JB	JBC
JC	JMP	JNB	JNC	JNZ	JZ	LCALL	LJMP
MOV	MOVC	MOVX	MUL	NOP	ORL	POP	PUSH
RET	RETI	RL	RLC	RR	RRC	SETB	SJMP
SUBB	SWAP	XCH	XCHD	XRL			

※印のついている命令は特別追加命令です。

・特別追加命令

CALL　指定したアドレスがカレントアドレスから2Kバイト内にある場合、この命令は「ACALL」に展開されます。そうでなければ「LCALL」に展開されます。
ただし、アドレスにラベルを指定し、かつそのラベルがその時点で定義済みでなかった場合は最終的に2Kバイト内であっても「LCALL」に展開されます。

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
MSS-51　マクロアセンブリ言語	インテルジャパン

８０８５編

●レジスタ・IO名

A	B	C	D	E	H	L	M
PSW	SP						

上記の名前はレジスタ名となる為、単独ではシンボル名として使用できません。

●演算子名

AND	EQ	GE	GT	HIGH	LE	LOW	LT
MOD	NE	NOT	OR	SHL	SHR	XOR	

上記の名前は演算子名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	*EJECT	*HEADING	*INCLUDE	*LIST	=	ASEG	BLK
BYTE	CHAR	COND	CSEG	DB	DBYTE	DD	DEFB
DEFD	DEFL	DEFM	DEFS	DEFT	DEFW	DM	DS
DSEG	DT	DW	ELSE	END	ENDC	ENDIF	ENDM
EQU	ERR	EXITM	FCB	FCC	FCS	FCT	FDB
IF	IF1	IF2	IFC	IFDEF	IFEQ	IFGE	IFGT
IFLE	IFLT	IFN	IFNDEF	IFNE	IFP1	IFP2	IFT
INCLUDE	LIB	LIST	MACRO	NAM	NAME	OPT	ORG
PAG	PAGE	PG	RES	RMB	RPT	RZB	SET
SPC	STTL	SUBTTL	TEXT	TITLE	TTL	USE	WORD
XLIST	ZERO						

擬似命令や基本命令をシンボル名として使用したい場合は、名前の後ろに「:」が必要です。

●8080の基本命令

ACL	ADC	ADD	ADI	ANA	ANI	CALL	CC
CM	CMA	CMC	CMP	CNC	CNZ	CP	CPE
CPI	CPO	CZ	DAA	DAD	DCR	DCX	DI
EI	HLT	IN	INR	INX	JC	JM	JMP
JNC	JNZ	JP	JPE	JPO	JZ	LDA	LDAX
LHLD	LXI	MOV	MVI	NOP	ORA	ORI	OUT
PCHL	POP	PUSH	RAL	RAR	RC	RET	RLC
RM	RNC	RNZ	RP	RPE	RPO	RRC	RST
RZ	SBB	SBI	SHLD	SPHL	STA	STAX	STC
SUB	SUI	XCHG	XRA	XRI	XTHL		

●8085の追加命令

RIM	SIM

8085は、8080の基本命令に加え上記の命令が追加されています。

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
マイクロコンピュータ　ソフトウェア基礎技術	ラジオ技術社

Ｚ８編

●レジスタ・IO名

C	DIND	DTC	EQ	F	FALSE	FLAGS	GE
GT	IMR	IPR	IRQ	LC	LE	LT	MI
MIC	MIV	NC	NE	NOV	NZ	OV	P0
P01M	P1	P1M	P2	P2M	P3	P3M	PL
PRE0	PRE1	R0	R1	R2	R3	R4	R5
R6	R7	R8	R9	R10	R11	R12	R13
R14	R15	RP	RR0	RR1	RR2	RR3	RR4
RR5	RR6	RR7	RR8	RR9	RR10	RR11	RR12
RR13	RR14	RR15	SIO	SP	SPH	SPL	T
T0	T1	TMR	TRUE	UGE	UGT	ULE	ULT
Z							

上記の名前はレジスタやIO名となる為、単独ではシンボル名として使用できません。

●演算子名

AND	EQ	GE	GT	HIGH	LAND	LE	LNOT
LOR	LOW	LT	LXOR	MOD	NE	NOT	OR
SHL	SHR	XOR					

上記の名前は演算子名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	*EJECT	*HEADING	*INCLUDE	*LIST	:=	=	ABS
ASEG	BLK	BYTE	CHAR	COND	CSEG	DB	DBYTE
DD	DEFB	DEFD	DEFL	DEFM	DEFS	DEFT	DEFW
DM	DS	DSEG	DT	DW	ELSE	END	ENDC
ENDIF	ENDM	EQU	ERR	EXITM	FCB	FCC	FCS
FCT	FDB	FI	IF	IF1	IF2	IFC	IFDEF
IFEQ	IFGE	IFGT	IFLE	IFLT	IFN	IFNDEF	IFNE
IFP1	IFP2	IFT	INCLUDE	LIB	LIST	LOAD	MACRO
NAM	NAME	OPT	ORG	PAG	PAGE	PG	REL
RES	RMB	RPT	RZB	SET	SPC	STTL	SUBTTL
TEXT	TITLE	TTL	USE	WORD	XLIST	ZERO	

擬似命令や基本命令をシンボル名として使用したい場合は、名前の後ろに「:」が必要です。

●基本命令

ADC	ADD	AND	CALL	CCF	CLR	COM	CP
DA	DEC	DECW	DI	DJNZ	EI	INC	INCW
IRET	JP	JR	LD	LDC	LDCI	LDE	LDEI
NOP	OR	POP	PUSH	RCF	RET	RL	RLC
RR	RRC	SBC	SCF	SRA	SRP	SUB	SWAP
TCM	TM	XOR					

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
マイクロコンピュータ　ソフトウェア基礎技術	ラジオ技術社
Z8/UPC　プログラミングマニュアル　アセンブリ語	シャープ

●レジスタ・IO名

A	AF	B	BC	C	D	DE	E
H	HL	I	IX	IY	L	M	NC
NZ	P	PE	PO	R	SP	Z	

上記の名前はレジスタやフラグ名となる為、単独ではシンボル名として使用できません。

●演算子名

AND	EQ	GE	GT	HIGH	LE	LOW	LT
MOD	NE	NOT	OR	SHL	SHR	XOR	

上記の名前は演算子名となる為、単独ではシンボル名として使用できません。

●擬似命令

*=	*EJECT	*HEADING	*INCLUDE	*LIST	=	ASEG	BLK
BYTE	CHAR	COND	CSEG	DB	DBYTE	DD	DEFB
DEFD	DEFL	DEFM	DEFS	DEFT	DEFW	DM	DS
DSEG	DT	DW	ELSE	END	ENDC	ENDIF	ENDM
EQU	ERR	EXITM	FCB	FCC	FCS	FCT	FDB
IF	IF1	IF2	IFC	IFDEF	IFEQ	IFGE	IFGT
IFLE	IFLT	IFN	IFNDEF	IFNE	IFP1	IFP2	IFT
INCLUDE	LIB	LIST	MACRO	NAM	NAME	OPT	ORG
PAG	PAGE	PG	RMB	RPT	RZB	SETL	SPC
STTL	SUBTTL	TEXT	TITLE	TTL	USE	WORD	XLIST
ZERO							

擬似命令や基本命令をシンボル名として使用したい場合は、名前の後ろに「:」が必要です。

●Z80の基本命令

ADC	ADD	AND	BIT	CALL	CCF	CP	CPD
CPDR	CPI	CPIR	CPL	DAA	DEC	DI	DJNZ
EI	EX	EXX	HALT	IM	IN	INC	IND
INDR	INI	INIR	JP	JR	LD	LDD	LDDR
LDI	LDIR	NEG	NOP	OR	OTDR	OTIR	OUT
OUTD	OUTI	POP	PUSH	RES	RET	RETI	RETN
RL	RLA	RLC	RLCA	RLD	RR	RRA	RRC
RRCA	RRD	RST	SBC	SCF	SET	SLA	SRA
SRL	SUB	XOR					

●HD64180の追加基本命令

IN0	MLT	OTDM	OTDMR	OTIM	OTIMR	OUT0	SLP
TST	TSTIO						

HD64180は、Z80の基本命令に加え上記の命令が追加されています。

●参考書籍

ニーモニックや命令の動作などは、以下の書籍を参考にして下さい。

タ　イ　ト　ル	出　版　会　社
日立8ビット・16ビット　マイクロプロセッサ	日立製作所
Z80プログラミングマニュアル	シャープ
マイクロコンピュータ　ソフトウェア基礎技術	ラジオ技術社

付　録

付録Ａ：対応ＣＰＵ一覧表

68系のアセンブラ

アセンブラ	対応CPU
X6502	6502
	6503
X6801	6800
	6801
	6802
	6803
	6808
	6301
	6303
X6805	6805
	146805
	68HC05
	6305
	63705
X6809	6809
	6800（注
	6801（注
X6811	6800
	6801
	6802
	6803
	6808
	68HC11

80系のアセンブラ

アセンブラ	対応CPU
X8048	8020
	8021
	8022
	8035
	80C35
	8039
	80C39
	8040
	8041
	8041A
	8048
	8049
	80C49
	8050
	8748
	8749
X8051	8031
	8051
	8751
X8085	8080
	8085
XZ8	Z8
XZ80	Z80
	HD64180

（注　6800/1の命令を6809の命令に相当するように変換します。

付録Ｂ：疑似命令一覧表

命　令	6502	6801	6805	6809	6811	8048	8051	8085	Z8	Z80	頁
*=	○	○	○	○	○	○	○	○	○	○	37
*EJECT						○	○	○	○	○	52
*HEADING						○	○	○	○	○	55
*INCLUDE						○	○	○	○	○	57
*LIST ON						○	○	○	○	○	53
*LIST OFF						○	○	○	○	○	53
:=									○		51
=	○	○	○	○	○	○	○	○	○	○	31
ABS									○		37
ASC	○										44
ASEG	○	○	○	○	○	○	○	○	○	○	37
BLK						○	○	○	○	○	49
BYTE	○	○	○	○	○	○	○	○	○	○	41
CHAR	○	○	○	○	○	○	○	○	○	○	44
COND						○	○	○	○	○	68
CSEG	○	○	○	○	○	○	○	○	○	○	37
DB	○					○	○	○	○	○	41
DBYTE	○	○	○	○	○	○	○	○	○	○	42
DD						○	○	○	○	○	42
DEFB						○	○	○	○	○	41
DEFD						○	○	○	○	○	42
DEFL						○	○	○	○	○	51
DEFM						○	○	○	○	○	44
DEFS						○	○	○	○	○	49
DEFT						○	○	○	○	○	46
DEFW						○	○	○	○	○	43
DFB	○										41
DM						○	○	○	○	○	44
DS	○					○	○	○	○	○	49
DSEG	○	○	○	○	○	○	○	○	○	○	39
DT						○	○	○	○	○	46
DW						○	○				42
DW	○							○	○	○	43
ELSE	○	○	○	○	○	○	○	○	○	○	67
END	○	○	○	○	○	○	○	○	○	○	61
ENDC	○	○	○	○	○	○	○	○	○	○	67
ENDIF	○	○	○	○	○	○	○	○	○	○	67
ENDM	○	○	○	○	○	○	○	○	○	○	76
EQU	○	○	○	○	○	○	○	○	○	○	50
ERR	○	○	○	○	○	○	○	○	○	○	62
EXITM	○	○	○	○	○	○	○	○	○	○	80

命　令	6502	6801	6805	6809	6811	8048	8051	8085	Z8	Z80	頁
FCB	○	○	○	○	○	○	○	○	○	○	41
FCC	○	○	○	○	○	○	○	○	○	○	44
FCS	○	○	○	○	○	○	○	○	○	○	45
FCT	○	○	○	○	○	○	○	○	○	○	46
FDB	○	○	○	○	○	○	○	○	○	○	42
FI									○		67
IF	○	○	○	○	○	○	○	○	○	○	68
IF1	○	○	○	○	○	○	○	○	○	○	74
IF2	○	○	○	○	○	○	○	○	○	○	74
IFC	○	○	○	○	○	○	○	○	○	○	67
IFDEF	○	○	○	○	○	○	○	○	○	○	75
IFEQ	○	○	○	○	○	○	○	○	○	○	69
IFF	○	○	○	○	○	○	○	○	○	○	69
IFGE	○	○	○	○	○	○	○	○	○	○	70
IFGT	○	○	○	○	○	○	○	○	○	○	72
IFLE	○	○	○	○	○	○	○	○	○	○	71
IFLT	○	○	○	○	○	○	○	○	○	○	73
IFN	○	○	○	○	○	○	○	○	○	○	69
IFNDEF	○	○	○	○	○	○	○	○	○	○	75
IFNE	○	○	○	○	○	○	○	○	○	○	68
IFP1	○	○	○	○	○	○	○	○	○	○	74
IFP2	○	○	○	○	○	○	○	○	○	○	74
IFT	○	○	○	○	○	○	○	○	○	○	68
INCLUDE						○	○	○	○	○	57
LIB	○	○	○	○	○	○	○	○	○	○	57
LIST						○	○	○	○	○	53
MACRO	○	○	○	○	○	○	○	○	○	○	76
NAM	○	○	○	○	○	○	○	○	○	○	55
NAME						○	○	○	○	○	55
OPT	○	○	○	○	○	○	○	○	○	○	59
ORG	○	○	○	○	○	○	○	○	○	○	37
PAG	○	○	○	○	○	○	○	○	○	○	52
PAGE	○	○	○	○	○						52
PAGE						○	○	○	○	○	66
PG	○	○	○	○	○	○	○	○	○	○	52
REG			○								64
REL									○		37
RES	○	○	○	○	○			○	○		49
RMB	○	○	○	○	○	○	○	○	○	○	49
RZB	○	○	○	○	○	○	○	○	○	○	48
RPT	○	○	○	○	○	○	○	○	○	○	63
SET	○	○	○	○	○	○	○	○	○		51

命　令	6502	6801	6805	6809	6811	8048	8051	8085	Z8	Z80	頁
SETDP				○							65
SETL						○	○			○	51
SPC	○	○	○	○	○	○	○	○	○	○	54
STTL	○	○	○	○	○	○	○	○	○	○	56
SUBTTL						○	○	○	○	○	56
TEXT						○	○	○	○	○	44
TITLE						○	○	○	○	○	55
TTL	○	○	○	○	○	○	○	○	○	○	55
WORD	○	○	○	○	○	○	○	○	○	○	43
USE	○	○	○	○	○	○	○	○	○	○	57
XLIST						○	○	○	○	○	53
ZERO	○	○	○	○	○	○	○	○	○	○	48

付録Ｃ：ソフトウェア開発によるアドバイス

●「初期化されたデータ」の問題（ＲＯＭ化を対象とした場合）

```
COUNT:    LD    HL,TIMER
          DEC   (HL)          ; TIMER の値を1減少させる
          RET

TIMER:    DB    128
```

ROM化をする場合、このようなプログラムは正常な動作をしません。もちろん、パソコン上で動作させる場合は問題なく正常に動きます。
「COUNT」ルーチンをROM部、「TIMER」をRAMに持って行けばよいのですが、RAM部は初期化できないのです。初期化できる所はROM部だけです。
また、「TIMER」をROM部に持って行くと、初期化はされますが今度は値の変更ができず、いつまでたっても「TIMER」は128の内容を保持したままになってしまいます。
通常、こういうプログラムは次のように記述します。

```
INIT:     LD    HL,TIMER
          LD    (HL),128      ; TIMER の値を128に初期化する
          RET

COUNT:    LD    HL,TIMER
          DEC   (HL)          ; TIMER の値を1減少させる
          RET

TIMER:    DS    1
```

まず、最初に「INIT」を1度だけコールします。これで、RAM部の「TIMER」が128になりますので、後はそのまま「COUNT」をコールするだけで、「TIMER」は順次減っていきます。

このようなことはミスは単純なようですが、初期化されたデータはうっかり見逃してしまいますので注意してください。
また、RAM部では、「DS」「RMB」などのデータ確保命令を使った方が良いでしょう。

付録Ｄ：ユーティリティープログラム

●ＬＯＡＤ（メモリへのオブジェクトのロード）

このユーティリティーは、出力したオブジェクト(Sフォーマットやインテル HEXフォーマット)をメモリ上にロードするものです。

```
┌──────────────────────────────────────┐
│  LOAD ファイル名 物理アドレス［オフセット］ │
└──────────────────────────────────────┘
```

ファイル名　　Sフォーマットかインテル HEXファイルです。
自動的にフォーマットを認識します。
拡張子を省略した場合は、「.S」または「.HEX」を自動的に付けます。

物理アドレス　　何番地からロードするかを指定します。
セグメントアドレス、オフセットアドレスを合わせて物理アドレスで指定します。
例えば、セグメントB000Hで、オフセット0の場合は、「B0000」と指定します。
ロード範囲の対象は、「B0000」～「BFFFF」になります。

オフセット　　アセンブラでのアドレスにオフセット値を与えたい場合に指定します。
マイナスのオフセットも「-100」のように指定できます。
この場合は、「FF00」を指定したのと同じです。

```
┌──────────────────────────────────────┐
│例)   A>LOAD TEST B0000 -100□          │
│        ~~~~~~~~~~~~~~~~~~~~~          │
└──────────────────────────────────────┘
```

「TEST」プログラムが、100番地から1FF番地のものだとすると、メモリのB0000～B00FFまでのエリアにロードされます。

★このユーティリティーは、直接メモリにロードしますので、MS-DOSなどで使用しているアドレスを指定すると、システムを破壊する可能性があります。アドレスの指定には十分注意してください。

●ＬＩＳＴ（ＬＳＴファイルのプリントアウト）

このユーティリティーは、出力したリストファイルをプリンタなどに印刷するものです(標準では画面に表示される為、プリンタへの出力にはリダイレクションを使用します)。

```
┌──────────────────────────────────────┐
│  LIST ファイル名 [オプション]                │
└──────────────────────────────────────┘
```

ファイル名　　リストファイルを指定します。
　　拡張子を省略した場合は、「.LST」を自動的に付けます。

オプション
　　<n> ‥‥ 整数値を指定(10進数)
　　<x> ‥‥ アドレスを指定(16進数)
　　アドレスは、シンボルを使っても指定できます。
　　シンボルを使う場合は、「.」に続いて指定します。
　　(例) -a.START
　　もちろん、シンボルファイルがなければシンボルは使用できません。

-g<m>="PAGE"	ページ番号を表示する位置とページ名を指定 (mは何列目に表示するかの指定)
-hx<m>="TITLE"	ヘッダータイトルの設定 (mは何列目に表示するかの指定)
-d="NAME"	表示しない行のニーモニック名を指定
-x	アセンブル行を通し番号に直して表示
-s	リスト中の「TTL」「STTL」などをそのままヘッダーとして使用
-k	「PAG」命令でページングを行う。
-l<n>	1ページの行数を指定
-w<n>	横幅の桁数を指定
-r<n>	ページの頭の所で指定行数改行する
-p<n>	指定ページより表示開始
-a<x>	指定アドレスより表示開始
-n<n>	指定行数より表示開始
-m<map-frame>	シンボル(マップ)ファイルの指定 デフォルトでは、リストファイル名の拡張子を「MAP」に変えたもの
-ep<n>	指定ページに達したら表示を終了
-ea<x>	指定アドレスに達したら表示を終了
-en<n>	指定行数に達したら表示を終了

オプションは、一度にいくつも指定して構いませんが、同じオプションは一度しか指定してはいけません。ただし、「-h」は5回まで指定できます(つまり、タイトルを5行まで設定できる)。また、「-d」は50回まで指定できます(50個まで未表示命令を設定できる)。

```
例)   A>LIST TEST -s >PRN□
        ~~~~~~~~~~~~~~~~~~
```

「TEST.LST」を読み、ソース中の「TTL」「STTL」などのタイトルをそのままヘッダーとして採用します。
リダイレクション機能を使って、プリンターに印刷します。

```
例)   A>LIST TEST -p5 -ep10□
        ~~~~~~~~~~~~~~~~~~~~
```

「TEST.LST」の5ページから10ページまでを表示します。

★「-k」「-s」「-d」の各オプションは、ソース中のオペコード部を見て機能します。
その為、アセンブル時に「%」を使って、オペコード部の表示位置を変更した場合は、正常に機能しません。

●ＨＥＸＡＤＲ（オブジェクトのアドレス確認）

このプログラムは、出力したオブジェクトのロードアドレスを表示するものです。
プログラム及び初期化されたデータのアドレスを詳細に表示します。ROM化を対象としている場合は、必ず表示されたアドレスはROM部でないといけません。

```
┌──────────────────────────────────────┐
│  HEXADR ファイル名                   │
└──────────────────────────────────────┘
```

ファイル名　　Sフォーマットかインテル HEXファイルです。
自動的にフォーマットを認識します。
拡張子を省略した場合は、「.S」または「.HEX」を自動的に付けます。

```
┌──────────────────────────────────────┐
│例)  A>HEXADR TEST□                   │
│        ~~~~~~~~~~~~                  │
└──────────────────────────────────────┘
```

「TEST」のロードアドレスを確認します。

［スタートアドレス］-［エンドアドレス］

という具合いに連続して表示します。

Ｃｒｏｓｓ　Ａｓｓｅｍｂｌｅｒ－ＭＳ　ユーザーズ・マニュアル

初　版　１９８６年　４月
第２版　１９８７年　９月
第３版　１９８８年　７月
第４版　１９８９年　３月

発行責任　ソフトマート株式会社