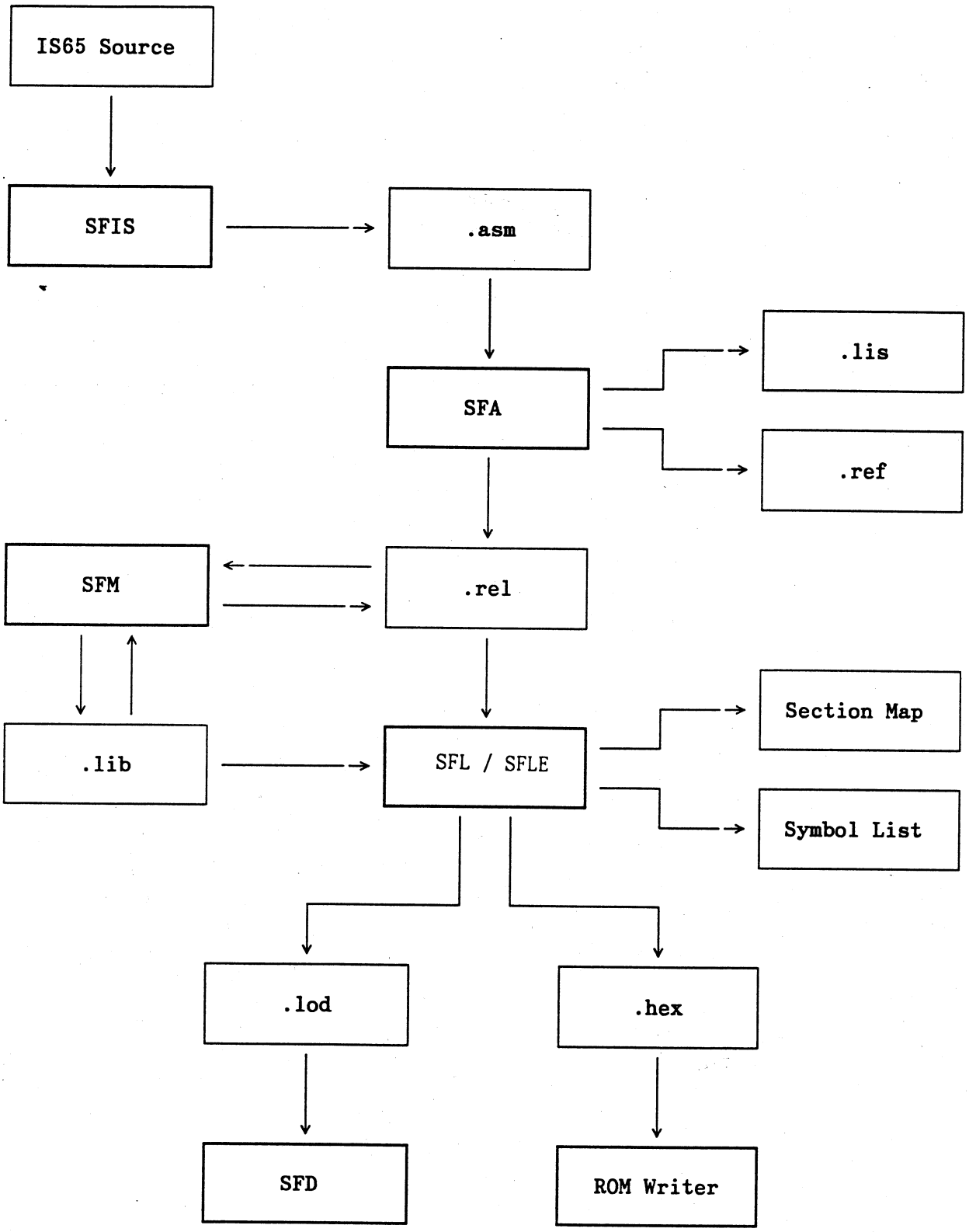
スーパーファミコン用アセンブラシステム
ユーザーマニュアル

株式会社リコー
電子デバイス事業部

システム相関図


IS65 Source
SFIS
.asm
SFA
.lis
.ref
SFM
.rel
.lib
SFL / SFLE
Section Map
Symbol List
.lod
.hex
SFD
ROM Writer

クロスアセンブラ
ユーザーマニュアル

１９９２年１１月　作成

目　　次

1　アセンブラの操作方法

1.1　アセンブラの起動

コマンド行からの入力で、アセンブラを起動します。
コマンドの書式は、次のとおりです:

sfa ［オプション指定］ ソース･ファイル名

アセンブラは、指定されたオプションに従ってソース･ファイルのアセンブルを行ない、オブジェクト･ファイル、アセンブル･リスト･ファイル、相互参照･リスト･ファイル、エラー等のメッセージを出力します。

ソース･ファイルは、一つだけ指定可能です。オプションの位置は、ソース･ファイル名の前後どちらでもかまいません。

カレント･ディレクトリ以外のソース･ファイルを指定してもオブジェクト･ファイルなどは全てカレント･ディレクトリに作成されます。

ソース･ファイルには ".asm" という拡張子が必要です。コマンド行でのソース･ファイル名では ".asm" を省略できます。

アセンブラが出力するファイルについては、「１.２　出力ファイル」で説明します。
オプションの指定については、「１.３　オプションの指定」で説明します。

例:

```
sfa foo.asm
```
オブジェクト･ファイル "foo.rel" が作成されます。

```
sfa foo
```
ソース･ファイル "foo.asm" がアセンブルされオブジェクト･ファイル "foo.rel" が作成されます。

```
sfa -n foo.obj foo.asm
```
オブジェクト･ファイル "foo.obj" が作成されます。

```
sfa -l foo.asm
```
オブジェクト･ファイル "foo.rel" と、アセンブル･リスト･ファイル "foo.lis" が作成されます。

```
sfa -lx foo.asm
```
オブジェクト･ファイル "foo.rel" と、アセンブル･リスト･ファイル "foo.lis" と、相互参照･リスト･ファイル "foo.ref" が作成されます。

１．２　出力ファイル

アセンブラは、つぎの出力ファイルを作成します。

オブジェクト･ファイル:

アセンブラによって作成されるオブジェクトのファイルです。

ソース･ファイル名から拡張子を取りさったものに 拡張子 ".rel" をつけたファイル名になります。

オブジェクト･ファイル名は -n オプションで指定することもできます。

オブジェクト･ファイルは、とくにオプションの指定をしなくても作成されます。

ただし、アセンブル中にエラーが検出されたときには、オブジェクト･ファイルは作成されません。

オブジェクト･ファイルは -r オプションが指定されたときには、作成されません。

アセンブル･リスト･ファイル:

アセンブラによって作成されるアセンブル･リストのファイルです。

ソース･ファイル名から拡張子を取りさったものに 拡張子 ".lis" をつけたファイル名になります。

アセンブル･リスト･ファイルは、-l オプションが指定されたときに作成されます。

相互参照･リスト･ファイル:

アセンブラによって作成される相互参照･リストのファイルです。

ソース･ファイル名から拡張子を取りさったものに 拡張子 ".ref" をつけたファイル名になります。

相互参照･リスト･ファイルは、-x オプションが指定されたときに作成されます。

１.３　オプションの指定

コマンド行には、次のオプションが指定できます。
オプションの文字は大文字、小文字のどちらでもかまいません。

-a

マクロ展開部および インクルード･ファイルの内容をすべて含んだアセンブル･リスト･ファイルを作成します。

-d　シンボル［＝数値］

下線部には余分の空白を入れることはできません。
指定されたシンボルをアセンブル時に参照可能なシンボルとして定義します。数値が省略されたときには、０ とみなします。

-e

エラーメッセージを標準出力（画面）に出力しません。

-g

デフォルトのセクションのグループ化を禁止します。

-h

アドレス、シンボル値を１６進でプリントします。（デフォルト）

-l

アセンブル･リスト･ファイルを作成します。

-m

アセンブル･リスト･ファイル、相互参照･リスト･ファイルの全データを LSB から MSB へプリントします。デフォルトは
「 MSB から LSB 」です。

-n　ファイル名

オブジェクト･ファイル名を指定します。

-o

アドレス、シンボル値を８進でプリントします。（デフォルトは１６進）

-r

オブジェクト･ファイルを作成しません。

-s

シンボル一覧表を付加したアセンブル･リスト･ファイルを作成します。

-u

未定義シンボルを外部シンボルとみなします。

-v

アセンブラのバージョン番号、エラーやワーニングの数、作成したファイル（オブジェクト･ファイル、アセンブル･リスト･ファイル、相互参照･リスト･ファイル）の名前の表示などを行ないます。

-W

ワーニング･メッセージを標準出力（画面）に出力しません。

-X

相互参照･リスト･ファイルを作成します。

-Z

ローカル･シンボルの情報をオブジェクト･ファイルに出力しません。

2　ソース プログラムの書式

入力ソース行は長さ１３２文字まで許されます。大文字と小文字は別々の文字セットとして処理されます。１つのシンボル内で大文字と小文字を混ぜるのはかまいませんが、命令ニーモニック、疑似命令、その他の予約語は、大文字、小文字のいずれかで書かなくてはなりません。

2.1　ステートメント

ステートメントの書式は、次のとおりです:

［ラベル］［命令コード　［オペランド］］［注釈］

ステートメントを構成する ラベル、命令コード、オペランド、注釈 の各フィールドは、空白文字(スペースまたは水平タブ)で区切ります。

ステートメントには、次の種類があります。

- 空行(改行コードだけの行)
- 注釈行
- 実行可能な命令
- アセンブラ擬似命令
- マクロ命令の呼び出し

ソース･プログラムの終わりには、END 擬似命令を置いてください。

2.1.1　ラベル

ラベルは、次の形式で記述します。

シンボル:

行の先頭から始めるときは、":"（コロン）を省略できます。

シンボルそのものについての説明は、「２.２.１　シンボル」を参照してください。

命令ニーモニック、擬似命令、その他の予約語、マクロ名は、ラベルに使用できません。

2.1.2　命令コード

命令コード･フィールドには、命令ニーモニック、擬似命令、マクロ名を書くことができます。命令ニーモニック、疑似命令をマクロ名にすることはできませんので、３つのうちどれであるかは、一意に決まります。

2.1.3　オペランド

オペランド･フィールドは、命令の操作の対象を記述します。　オペランドが複数の要素からなる場合は、","（カンマ）で区切って並べます。

2.1.4　注釈

注釈は、次の形式で記述します

; 任意の文字

";"（セミコロン）で始まる行は注釈行です。

２.２　式

２.２.１　シンボル

シンボルに使える文字は、つぎのとおりです。　ただし、シンボルは英字およびアンダーバー記号”＿”で開始しなければなりません。

.(ピリオド)　_(下線記号)　$(ドル記号)　?(疑問符)　@(アットマーク)
A から Z の英大文字　　a から z の英小文字
0 から 9 の数字

アセンブラは、デフォルトではシンボルの大文字と小文字を別の文字として扱います。この動作は、アセンブラの CASEON/CASEOFF 擬似命令により変更することができます。

シンボルの長さには 制限はありませんが、アセンブラは、シンボルの先頭から32文字までを認識します。

"$" は、現在のプログラム カウンタの値をあらわします。

２.２.２　テンポラリーシンボル

テンポラリシンボルは特殊なタイプのシンボルです。＄で終わるシンボル（シンボルがコロンで終わる場合は、最後から２番目の文字が＄）はすべてテンポラリーシンボルとして処理されます。テンポラリーシンボルは、次に通常のシンボルが定義されるまで（マクロ定義を跨がない範囲、セクションの開始、終了を跨がない範囲、マクロ内で使用した場合はそのマクロの中）が有効範囲となります。従って範囲を超えたところでは、同じ名前のテンポラリシンボルを違うシンボル値として使用することができます。有効範囲内では通常のシンボルと同等の扱いができます。またテンポラリシンボルは相互参照･リスト･ファイルなどに出力されません。

２.２.３　数値

アセンブラでは、16進数、10進数、8進数、2進数 の数値を使用できます。

数値	表記方法
2進数	0 と 1 の数字の列の後に B または b をつけたもの
8進数	0 から 7 の数字の列の後に O または o をつけたもの
10進数	0 から 9 の数字の列、または 0 から 9 の数字の列の後に D を付けたもの
16進数	0 から 9 の数字と A から F、a から f の文字の列の後に H または h を付けたもの

例:

２進数　0101B
８進数　1230, 123o
１０進数　123, 123D
１６進数　123H, 0F00h

文字 A-F または a-f で始まる１６進数は先頭にゼロ（０）を付けなくてはなりません。

２．２．４　文字列

文字列は、引用符(')で囲んだ任意の文字の列です。　文字列が引用符そのものを含む場合には、引用符を二重にかさねて表わします。

例:

```
'You are a girl'
'I''m a boy'
```

式の中に文字列を書いた場合には、文字列は次のように評価されます。

1) 空の文字列の値は、0 になります。
2) 文字列の左から８ビット単位でワードの上位に向かって詰められます。

例:
```
''    = 0
'A'   = 41H
'AB'  = 4142H
''''  = 27H
```

２．２．５　演算子

式には、次の演算子が使用できます。

単項演算子:

－	２の補数
￣	１の補数
LOW	オペランドの下位バイト（ビット０－７）
HIGH	オペランドの中位バイト（ビット８－１５）
BANK	オペランドの上位バイト（ビット１６－２３）
OFFSET	オペランドの下位ワード（ビット０－１５）

算術演算子:

＋	加算
－	減算
＊	乗算
／	除算
％	剰余

ビット毎の論理演算子:

＆	論理積
｜	論理和
＾	排他的論理和

シフト演算子:

＞＞	右シフト
＜＜	左シフト

関係演算子:

＝＝	等しい
！＝	等しくない
＞	より大きい
＞＝	等しいかより大きい
＜	より小さい
＜＝	等しいかより小さい

□ 演算子の優先順位

演算子には、つぎの１０段階の優先順位があります。

(単項演算子)－　(単項演算子)￣　LOW HIGH BANK OFFSET	順位が高い
＊　／　％	↓
＋　－	↓
＜＜　＞＞	↓
＜　＜＝　＞　＞＝	↓
＝＝　！＝	↓
＆	↓
＾	↓
｜	順位が低い

同じ優先順位をもつ演算子どうしでは、左から順に評価されます。　演算の順番を変えたいときには、（）(括弧)を使用します。

2.2.6 演算規則

相対アドレス値と外部参照値に関する演算は、つぎにあげるものだけが有効です。

相対アドレス値は　R　で表します。
外部参照値　　は　X　で表します。
絶対値　　は　A　で表します。

R＋A	（結果はR）	
A＋R	（結果はR）	
R－A	（結果はR）	
R－R	（結果はA）	（両オペランドは同一セクション）
LOW	R	
HIGH	R	
BANK	R	
OFFSET	R	

X＋A	（結果はX）
A＋X	（結果はX）
X－A	（結果はX）
LOW	X
HIGH	X
BANK	X
OFFSET	X

LOW、HIGH、BANK、OFFSETの結果に演算を施すことはできません。

```
LOW  R        ・・・  OK
LOW  R+A      ・・・  error
```

３．アドレッシングモード

イミディエート・アドレッシング
 #式

例:
```
  LDA   #1234h          ;LOAD ACCUMULATOR IMMEDIATE
  ADC   #VALUE
```

アブソリュート・アドレッシング
 !式

例:
```
  LDA   !1234h          ;LOAD ACCUMULATOR ABSOLUTE
  ADC   !VALUE
```

アブソリュート・ロング・アドレッシング
 >式

例:
```
  LDA   >123456h        ;LOAD ACCUMULATOR ABSOLUTE LONG
  ADC   >VALUE
```

ダイレクト・アドレッシング
 <式

例:
```
  LDA   <12h            ;LOAD ACCUMULATOR DIRECT
  ADC   <VALUE
```

アキュームレータ・アドレッシング
 A

例:
```
  ASL   A               ;ARITHMERTIC SHIFT LEFT ACCUMULATOR
  DEC   A
```

インプライド・アドレッシング

例:
```
  CLC
  SEC
```

ダイレクト・インダイレクト・インデックスド・Ｙ・アドレッシング
 （<式），Ｙ

例:
```
  LDA   (<12h),Y
  ADC   (<VALUE),Y
```

ダイレクト・インダイレクト・ロング・インデックスド・Y・アドレッシング
［<式］，Y

例:
```
LDA   [<12h],Y
ADC   [<VALUE],Y
```

ダイレクト・インデックスド・インダイレクト・X・アドレッシング
（<式，X）

例:
```
LDA   (<12h,X)
ADC   (<VALUE,X)
```

ダイレクト・X・アドレッシング
<式，X

例:
```
LDA   <12h,X
ADC   <VALUE,X
```

ダイレクト・Y・アドレッシング
<式，Y

例:
```
LDX   <12h,Y
LDX   <VALUE,Y
```

アブソリュート・X・アドレッシング
！式，X

例:
```
LDA   !1234h,X
ADC   !VALUE,X
```

アブソリュート・ロング・X・アドレッシング
>式，X

例:
```
LDA   >123456h,X
ADC   >VALUE,X
```

アブソリュート・Y・アドレッシング
！式，Y

例:
```
LDA   !1234h,Y
ADC   !VALUE,Y
```

リラティブ・アドレッシング
式

例:
```
BCC   12h
BCC   VALUE
```

リラティブ・ロング・アドレッシング
式

例:
```
BRL   1234h
BRL   VALUE
```

アブソリュート・インダイレクト・アドレッシング
(! 式)

例:
```
JML   (!1234h)
JML   (!VALUE)
```

ダイレクト・インダイレクト・アドレッシング
(< 式)

例:
```
LDA   (<12h)
ADC   (<VALUE)
```

ダイレクト・インダイレクト・ロング・アドレッシング
[< 式]

例:
```
LDA   [<12h]
ADC   [<VALUE]
```

アブソリュート・インデックスド・インダイレクト・アドレッシング
(! 式, X)

例:
```
JMP   (!1234h,X)
JMP   (!VALUE,X)
```

スタック・アドレッシング

命令にオペランドがあるか否かによって、２つの形式が使用されます。

１バイト命令

例:

```
PHD
PLX
BRK
```

命令にオペランドあり

式

例:

```
PEA   1234h
PEI   12h
PER   34h
COP   78h
BRK   03h
```

スタック・リラティブ・アドレッシング

＜式，Ｓ

例:

```
LDA   <12h,S
ADC   <VALUE,S
```

スタック・リラティブ・インダイレクト・インデックスド・アドレッシング

（＜式，Ｓ），Ｙ

例:

```
LDA   (<12h,S),Y
ADC   (<VALUE,S),Y
```

ブロック・ムーブ

＃式，＃式

例:

```
MVN   #12h,#34h
MVP   #VALUE,#VALUE_2
```

ＧＲＯＵＰ　グループ定義

【書式】

グループ名　GROUP　セクション名［、セクション名．．．］

【説明】

グループを構成するセクションを定義します。
オペランドのセクションはソース･プログラム中で定義されていなければなりません。

一つのセクションは、複数のグループに所属することはできません。また、アブソリュートセクションは、グループに所属することはできません。

グループにまとめられたセクションは、（リンカにより）連続した領域に配置されます。

ソース・ファイル名 "src.asm" の場合、デフォルトとしてアセンブラ内部で
”ＰＲＯＧ　　ＧＲＯＵＰ　　Ｐｓｒｃ”
”ＤＡＴＡ　　ＧＲＯＵＰ　　Ｄｓｒｃ”
と同じ処理がされます。（これはオプションによって禁止できます。）

ＳＥＣＴ　セクションの定義／再開

【書式】

セクション名　ＳＥＣＴ　［属性［，ベースアドレス］］...(1)
セクション名　ＳＥＣＴ　...(2)

【説明】

(1)の書式でセクションの定義を行います。
(2)の書式ですでに定義されているセクションを再開します。
どちらの場合にも、次ＳＥＣＴ疑似命令が現れるまで現ＳＥＣＴ疑似命令で指定されたセクションが有効です。

□　属性

属性には、次のものが指定できます。

・ＡＢＳ　（アブソリュート・セクション）
このセクションに属するコードは、すべて絶対アドレスに配置されます。

・ＲＥＬ　（リロケータブル・セクション）
リロケータブル・セクションは、再配置可能なセクションです。複数ファイルに分けられた同名のセクションは、リンカによって結合されて連続した領域に配置

されます

・ＯＶＲ （オーバーレイ・セクション）
オーバーレイ・セクションは、リロケータブル・セクションと同じく再配置可能なセクションです。複数ファイルに分けられた同名のセクションは、リンカによって（先頭をそろえて）重ね合わされます・

属性が省略された場合には、アブソリュート・セクションになります。

□　ベースアドレス

ベースアドレスは次の形式で指定します。

ＢＡＳＥ＝定数式

ベースアドレスは、属性にＡＢＳを指定したときのみ有効です。
式は、定数式でなければならず、前方参照シンボルを含むこともできません。
ベースアドレスが省略された場合にはゼロになります。

□　デフォルト・セクション

セクションの定義を省略した場合には、ソース・ファイル名 "src.asm" では、”Ｐｓｒｃ”という名前のリロケータブル・セクションをアセンブラが自動的に定義します。

【注意事項】
複数のソース・ファイルで同名のセクションを定義する場合は、属性、ベースアドレスはすべて同一のセクションでなければなりません。

ＯＲＧ	セクションの開始

【書式】

```
ORG     式
```

【説明】
アブソリュート・セクションを開始します。

式でアブソリュート・セクションの先頭アドレスを指定します。式には、値を確認できないシンボルを含むことはできません。

ソース・ファイル名 "src.asm" で ORG 10h , ORG 20h という記述があった場合、アセンブラ内部で

```
”Ａ１ｓｒｃ　　ＳＥＣＴ　　ＡＢＳ，ＢＡＳＥ＝１０Ｈ”
”Ａ２ｓｒｃ　　ＳＥＣＴ　　ＡＢＳ，ＢＡＳＥ＝２０Ｈ”
```

と同じ処理をします。

ＰＲＯＧ　　セクションの開始

【書式】

PROG

【説明】

プログラム・セクションを開始します。

ソース・ファイル内の、任意個のＰＲＯＧ内のコードはプログラム・セクションとして、１つに連結されます。

各ソース・ファイルのプログラム・セクションは、リンカによって、１つに連結し、任意のアドレスに配置することができます。（連結せず、個々に任意のアドレスに配置することもできます。）

ソース・ファイル名 "src.asm" で PROG という記述があった場合、アセンブラ内部で

”Ｐｓｒｃ　　ＳＥＣＴ　　ＲＥＬ”

と同じ処理をします。

ＤＡＴＡ　　セクションの開始

【書式】

DATA

【説明】

データ・セクションを開始します。

ソース・ファイル内の、任意個のＤＡＴＡ内のコードは、データ・セクションとして、１つに連結されます。

各ソース・ファイルのデータ・セクションは、リンカによって、１つに連結し、任意のアドレスに配置することができます。（連結せず、個々に任意のアドレスに配置することもできます。）

ソース・ファイル名 "src.asm" で DATA という記述があった場合、アセンブラ内部で

”Ｄｓｒｃ　　ＳＥＣＴ　　ＲＥＬ”

と同じ処理をします。

ＣＯＭＮ　　セクションの開始

【書式】

```
COMN
```

【説明】

コモン・セクションを開始します。

ソース・ファイル内の任意個のＣＯＭＮ内のコードはコモン・セクションとして１つに連結されます。

各ソース・ファイルのコモン・セクションは、リンカによって、１つに重ね合わせ任意のアドレスに配置することができます。

ソース・ファイル名 "src.asm" で COMN という記述があった場合、アセンブラ内部で

”ＣＯＭＮ　　ＳＥＣＴ　　ＯＶＬ”

と同じ処理をします。

ＥＱＵ　　シンボルの定義

【書式】

```
シンボル        EQU     式
```

【説明】

シンボルを定義します。　シンボルの値は、引数で与えられた式の値になります。　式には、外部参照シンボルまたは未定義シンボルを含んでいてはいけません。

例:

```
FOO     EQU     100H
BAR     EQU     FOO+20H
```

ＳＥＴ　　変数の定義

【書式】

```
シンボル        SET     式
```

【説明】

シンボルに、引数であたえた値をセットします。　式には、外部参照または未定義シンボルを含んでいてはなりません。

SET 擬似命令で定義したシンボルには、何回でも値をセットできますが、外部シンボルの宣言はできません。

例:

```
FOO     SET     10        ; FOO＝10
FOO     SET     FOO＋10   ; FOO＝20
```

ＥＸＴ／ＥＸＴＥＲＮ／ＥＸＴＥＲＮＡＬ　　外部参照シンボルの宣言

【書式】

```
EXT       シンボル[, シンボル .. ]
EXTERN    シンボル[, シンボル .. ]
EXTERNAL  シンボル[, シンボル .. ]
```

【説明】

引数で指定したシンボルが、外部参照シンボルであることを宣言します。　外部参照シンボルには値を与えることができません。(多重定義エラーになります)

宣言したシンボルを使用しなくてもかまいません。

ＧＬＢ／ＧＬＯＢＡＬ／ＰＵＢＬＩＣ　　外部シンボルの宣言

【書式】

```
GLB     シンボル[, シンボル .. ]
GLOBAL  シンボル[, シンボル .. ]
PUBLIC  シンボル[, シンボル .. ]
```

【説明】

引数で指定したシンボルが、外部から参照されることを宣言します。　GLOBAL で宣言されたシンボルは、他のファイルから参照することができます。

GLOBAL 宣言を行なうシンボルは、そのファイルの中で定義されている必要があります。

ＤＢ／ＢＹＴＥ　　バイト定義

【書式】

```
DB      式 [,式...]
BYTE    式 [,式...]
```

【説明】

この命令は、引数に与えられた式および文字列を、ロケーション・カウンタの現在値で開始する連続的なメモリ・ロケーションに記憶します。

引数が式のときは、値は１バイトでなくてはなりません。さもないと警告メッセージが出され、下位の１バイトが採用されます。

引数が文字列のときは、文字列内の文字は現れた順に記憶されます。各１バイトの値です。

ＡＳＣＩＩ／ＡＳＣ　　文字列の定義

【書式】

```
ASCII   文字列 [,文字列...]
ASC     文字列 [,文字列...]
```

【説明】

この命令は、文字列内の文字を、ロケーション・カウンタの現在値で開始する連続的なメモリ・ロケーションに記憶します。文字列内の文字は現れた順に記憶されます。各１バイトの値です。ASC のアーギュメントが NULL の場合は、エラーが出されます。

ＤＳ／ＲＭＢ　　スペースの定義

【書式】

```
DS      式
RMB     式
```

【説明】

この命令は、メモリ領域を確保します。式の値は、割り振るべきバイト数です。式内で使用されるシンボルはすべて前もって定義されていなくてはなりません。さもないと、エラーが発生します。

ＤＷ／ＷＯＲＤ　　ワード定義

【書式】

```
DW      式 [,式 ...]
WORD    式 [,式 ...]
```

【説明】

この命令は、式の値を、ロケーション・カウンタの現在値で開始する連続的なメモリ・ロケーションに記憶します。式は２バイト値として表現されます。２バイトより長い場合は、警告メッセージが出され、下位の２バイトが採用されます。

ＤＬ／ＬＷＯＲＤ　　ロング・ワード定義

【書式】

```
DL      式 [,式 ...]
LWORD   式 [,式 ...]
```

【説明】

この命令は、式の値を、ロケーション・カウンタの現在値で開始する連続的なメモリ・ロケーションに記憶します。式は３バイト値として表現されます。３バイトより長い場合は、警告メッセージが出され、下位の３バイトが採用されます。

ＤＤ　　ワード定義

【書式】

```
DD      式 [,式 ...]
```

【説明】

この命令は、式の値を、ロケーション・カウンタの現在値で開始する連続的なメモリ・ロケーションに記憶します。式は２バイト値として表現されます。値はＤＷ／ＷＯＲＤ擬似命令とは逆に、上位バイト、下位バイトの順に記憶されます。２バイトより長い場合は、警告メッセージが出され、下位の２バイトが採用されます。

ＢＩＮ／ＢＩＮＡＲＹ　　バイト定義

【書式】

```
BIN     式 [,式 ...]
BINARY  式 [,式 ...]
```

【説明】

この命令は、式の値を、ロケーション・カウンタの現在値で開始する連続的なメモリー・ロケーションに記憶します。式は１バイト値として評価されます。１バイトより長い場合は、警告メッセージが出され、下位１バイトが採用されます。式には２進データ（０または１）しか使えないことに注意してください。２進値の後にＢまたはｂを使用するかどうかはオプションです。

ＨＥＸ　　バイト定義

【書式】

```
HEX             式 [,式 ...]
```

【説明】

この命令は、式の値を、ロケーション・カウンタの現在値で開始する連続的なメモリー・ロケーションに記憶します。式は１バイト値として評価されます。１バイトより長い場合は、警告メッセージが出され、下位１バイトが採用されます。式には１６進データ（[0-9,a-f,A-F]）しか使えないことに注意してください。１６進値の後にＨまたはｈを使用するかどうかはオプションです。

ＯＣＴ／ＯＣＴＡＬ　　バイト定義

【書式】

```
OCT     式 [,式 ...]
OCTAL   式 [,式 ...]
```

【説明】

この命令は、式の値を、ロケーション・カウンタの現在値で開始する連続的なメモリー・ロケーションに記憶します。式は１バイト値として評価されます。１バイトより長い場合は、警告メッセージが出され、下位１バイトが採用されます。式には８進データ（０－７）しか使えないことに注意してください。８進値の後にＯまたはｏを使用するかどうかはオプションです。

ＩＮＣＬＵＤＥ　　ファイルの取り込み

【書式】

INCLUDE　　ファイル名

【説明】

INCLUDE 擬似命令は、指定されたファイルの内容を INCLUDE 擬似命令がある位置に挿入してアセンブルを行ないます。

INCLUDE 擬似命令は、最高32レベルまでネストすることができますが、使用するシステムで同時に OPEN できるファイル数の制限をうけます。

例:

```
INCLUDE        HEADER.ASM
```

ＴＩＴＬＥ　　タイトルの指定

【書式】

TITLE　タイトル

【説明】

アセンブル・リストの各ページの先頭に表示するタイトルを指定します。タイトルは１回しか指定できません。

ＮＡＴＩＶＥ　　ＣＰＵモードの選択

【書式】

NATIVE

【説明】

イミディエート・アドレッシングモードでコマンド生成を行うばあいに、２バイトもしくは１バイトのオペランドをＭＥＭ８、ＭＥＭ１６、ＩＤＸ８、ＩＤＸ１６の疑似命令に応じて生成します。

デフォルトのＣＰＵモードは NATIVE となります。

ＭＥＭ８　　オペランドサイズの選択

【書式】

MEM8

【説明】

アキュームレータセレクトビット（Ｍ）＝１の状態を示します。１バイトオペランドを生成します。

ＭＥＭ１６　　オペランドサイズの選択

【書式】

MEM16

【説明】

アキュームレータセレクトビット（Ｍ）＝０の状態を示します。２バイトオペランドを生成します。

この疑似命令をエミュレーション・モードで使用するとエラーになります。

ＩＤＸ８　　オペランドサイズの選択

【書式】

IDX8

【説明】

インデックスセレクトビット（Ｘ）＝１の状態を示します。１バイトオペランドを生成します。

ＩＤＸ１６　　オペランドサイズの選択

【書式】

IDX16

【説明】

インデックスセレクトビット（Ｘ）＝０の状態を示します。２バイトオペランドを生成します。

この疑似命令をエミュレーション・モードで使用するとエラーになります。

EMULATION　　CPUモードの選択

【書式】

EMULATION

【説明】

イミディエートアドレッシングモードでコマンド生成を行う場合は、１バイトオペランドを生成します。

EXTEND　　アドレッシング・モードの選択

【書式】

EXTEND

【説明】

アブソリュート・アドレス指定モードでは通常、シンボル”！”の使用が必要ですが EXTEND 疑似命令により、シンボル”！”の使用はオプションになります。

LONG　　アドレッシング・モードの選択

【書式】

LONG

【説明】

アブソリュート・ロング・アドレス指定モードでは通常、シンボル”＞”の使用が必要ですが EXTEND 疑似命令により、シンボル”＞”の使用はオプションになります。

DIRECT　　アドレッシング・モードの選択

【書式】

DIRECT

【説明】

ダイレクト・アドレス指定モードでは通常、シンボル”＜”の使用が必要ですが EXTEND 疑似命令により、シンボル”＜”の使用はオプションになります。

AUTO	アドレッシング・モードの選択

【書式】

AUTO

【説明】

この疑似命令は、ＡＵＴＯモードを設定します。

このモードでは、複数のアドレッシングが可能な命令において、オペランド値を評価することで最適なアドレッシングが自動的に採用されます。
オペランドに前方参照が含まれる場合はＥＸＴＥＮＤモードと同じになります。
ＡＵＴＯモードはＤＩＲＥＣＴ、ＥＸＴＥＮＤ、ＬＯＮＧ疑似命令がくるまで有効で、デフォルトのモードになっています。
（注）外部シンボルとリロケータブルシンボルについては、値がアセンブル時には確定しないため、ＡＵＴＯ疑似命令の対象として扱うことはできません。よって、アドレッシングの接頭辞を必ずつけなければなりません。（つけなければエラーとなります。）

ＡＵＴＯモードの時、複数のアドレッシングが可能な命令において、以下の順でオペランドの評価が行われ、アドレッシングが決定されます。

ＪＳＲ、ＪＭＰ以外のオペコードの場合

１．アドレッシングの接頭辞があるものはそれに従う。

２．ＤＰＡＧＥで定義した値をオペランドより引き１バイトになれば、その値を用い、ダイレクト（Ｘ or Ｙ）アドレッシングとする。

３．ＤＢＡＮＫで定義した値と、オペランドのバンクバイトが等しい場合は、下位２バイトをアブソリュート（Ｘ or Ｙ）アドレッシングとして採用する。
ただし、ＤＢＡＮＫの値が０～３Ｆまたは８０～ＢＦで、オペランドの値が０～７ＦＦＦのときは（オペランドのバンクバイトがＤＢＡＮＫで設定したものと同じであるとし）この場合もアブソリュート（Ｘ or Ｙ）アドレッシングを採用する。

４．アブソリュートロング（Ｘ or Ｙ）アドレッシングが可能な命令の場合、これを採用する。

ＪＳＲ、ＪＭＰの場合

１．アドレッシングの接頭辞があるものはそれに従う。

２．実行時のＰＢＲの値と、オペランドのバンクバイトが等しい場合は、下位２バイトをアブソリュートアドレッシングとして採用する。
（ＰＲＯＧなどのリロケータブル・セクションにおいては、実行時のＰＢＲの値が確定しないため、この時点でエラーとします。）

３．アブソリュートロングアドレッシングを採用する。

ＡＵＴＯ疑似命令の対象となる命令とアドレッシングについてを列挙します。

アブソリュートアドレスでＤＢＲが使われる命令（ＪＳＲ、ＪＭＰ以外）

１．ADC AND CMP EOR LDA ORA SBC STA
```
        d       a       al
        d,X     a,X     al,X
```

２．ASL LSR ROL ROR BIT DBC INC LDY STZ
```
        d       a
        d,X     a,X
```

３．CPX CPY STX STY TRB TSB
```
        d       a
```

アブソリュートアドレスでＰＢＲが使われる命令（ＪＳＲ・ＪＭＰ）

４．JMP
```
        a       al
```

５．JSR
```
        a       al
```

ＤＰＡＧＥ　　ダイレクト・アドレスの指定

【書式】

DPAGE　　[式]

【説明】

これはダイレクト・アドレス指定モードのオペランドから式の値を減じます。ダイレクト・アドレス・モードではオペランドにダイレクトレジスタの値を加えたものが実行アドレスになるので、ダイレクトレジスタの値と同じものを DPAGE に設定することで、実行アドレスを容易に設定できます。式を指定しないと、デフォルト値ゼロが想定されます。（ＡＵＴＯを参照）

ＤＢＡＮＫ　　バンク・アドレスの指定

【書式】

DBANK　　[式]

【説明】

これはアブソリュート・ロング・アドレス指定モードにのみ適用されます。アブソリュート・ロング・アドレス指定モードで使用されるオペランド・データの BANK バイトから式の値が減じられます。式を指定しないと、デフォルト値ゼロが想定されます。（ＡＵＴＯを参照）

ＣＡＳＥＯＮ／ＣＡＳＥＯＦＦ　　大文字／小文字の区別

【書式】

CASEON
CASEOFF

【説明】

CASEON 命令は、シンボルについて大文字と小文字の区別を行なうことを指定します。

CASEOFF 命令は、シンボルについて大文字と小文字の区別を行なわないことを指定します。

アセンブラが起動されたときの初期値は、CASEON です。

ＬＩＳＴ　　リスト出力の再開

【書式】

LIST

【説明】

NLIST 擬似命令によって禁止されたアセンブル･リストの出力を再開します。
リストオプションが有効でないと、この疑似命令は無視されます。

ＮＬＩＳＴ／ＮＯＬＩＳＴ　　リスト出力の禁止

【書式】

NLIST
NOLIST

【説明】

ＬＩＳＴ疑似命令が出現するまでアセンブル･リストの出力を禁止します。
リストオプションが有効でないと、この疑似命令は無視されます。

ＬＡＬＬ／ＳＡＬＬ／ＸＡＬＬ　　リスト出力の制御

【書式】

LALL
SALL
XALL

【説明】

LALL 命令は、以降のマクロ展開および INCLUDE ファイルの内容を、すべてリストします。

SALL 命令は、以降のマクロ展開および INCLUDE ファイルの内容を、リストしません。

XALL 命令は、以降のマクロ展開および INCLUDE ファイルの内容を、コードが実際に生成された行についてのみリストします、

アセンブラが起動されたときの初期値は、XALL です。

ＰＡＧＥ／ＥＪＥＣＴ／ＳＫＩＰ　　リストの改ページ

【書式】

PAGE
EJECT
SKIP

【説明】

アセンブル･リストの改ページを行ないます。
リストオプションが有効でないと、この疑似命令は無視されます。

ＳＰＣ　　ブランク行のプリント

【書式】

SPC　　式

【説明】

リストファイルに、式行だけブランク行をプリントします。
リスト・オプションが有効でないと、この疑似命令は無視されます。

ＥＮＤ　　ソースプログラムの終わり

【書式】

END　　[式]

【説明】

この命令は、現在のアセンブリ言語プログラムの終りを指定します。式があれば、これはプログラムの実行開始アドレスを指定します。式がシンボルの場合はリロケータブル、数値の場合はアブソリュートのアドレス指定になります。リンク時に複数のリロケータブルファイルで実行開始アドレスが指定されていると、エラーになるので注意してください。

【書式】

```
IF      式

IFDEF   シンボル
IFNDEF  シンボル

IF16
IFMEM8
IFMEM16
IFIDX8
IFIDX16

IF1
IF2
```

【説明】

IF 疑似命令は、式が０でない場合は、真。

IFDEF 疑似命令は、シンボルが定義されるか、外部シンボルと宣言されていれば、真。

IFNDEF 疑似命令は、シンボルが定義されておらず、外部シンボルと宣言されていなければ、真。

IF16 疑似命令は、アセンブラが MEM16 または IDX16 のいずれかのモードであれば、真。

IFMEM8 疑似命令は、アセンブラが MEM8 のモードであれば、真。

IFMEM16 疑似命令は、アセンブラが MEM16 のモードであれば、真。

IFIDX8 疑似命令は、アセンブラが IDX8 のモードであれば、真。

IFIDX16 疑似命令は、アセンブラが IDX16 のモードであれば、真。

IF1 疑似命令は、アセンブラがパス１を実行中であれば、真。

IF2 疑似命令は、アセンブラがパス２を実行中であれば、真。

すべての条件で次の形式が使用されます。

```
IFxx    [ 引数 ]
ステートメント
ステートメント
.
.
[ ELSE
ステートメント
.
.
. ]
ENDIF
```

IFxx ステートメントが真（ゼロでない）と評価されると、ELSE ステートメントまでの全ステートメントが、アセンブルされます。

ELSE がない場合は、ENDIF キーワードまでの全ステートメントがアセンブルされます。IFxx が偽（ゼロ）と評価されると、ELSE ステートメント（もしあれば）の後から ENDIF までの全ステートメントが、アセンブルされます。if 条件疑似命令ではアブソリュート式を使用しなくてはなりません。さもないと、エラーが発生します。

MACRO／LOCAL／ENDM　　マクロの定義

【書式】

```
マクロ名        MACRO   [仮引数,仮引数 ... ]
                LOCAL シンボル [,シンボル ...]
                マクロ定義の内容 ..
                ENDM
```

【説明】

MACRO 擬似命令は、MACRO 擬似命令のある行から対応する ENDM 擬似命令のある行までをマクロ登録します。

仮引数は、最大８個まででそれ以上は無視されます。

□ マクロの呼び出し

マクロは、次のようにして呼び出します。

書式:

```
[ラベル]        マクロ名        [実引数,実引数 .. ]
```

マクロが呼び出されると、パラメータとしてあたえた実引数が仮引数に受け渡され、マクロ定義に含まれる仮引数が置き換えられて展開されます。

□ 実引数の受け渡し

実引数から仮引数へのパラメータの受け渡しは、つぎのようにして行なわれます。

１）　仮引数の数が実引数の数よりも少ない時（仮引数の数　＜　実引数の数）余った実引数は無視されます。

２）　実引数の数が仮引数の数よりも少ない時（仮引数の数　＞　実引数の数）余った仮引数は "^&" になります。

３）　引用符(" または ')で囲まれた文字列は、ひとつの引数として扱います。

例:

```
'FOO,BAR'   →   'FOO,BAR'
```

４）　山形括弧（＜ ＞）で囲まれた文字列は、ひとつの引数として扱います。引数の両端の山形括弧は取り除かれます。

例:

```
<FOO,BAR>   →   FOO,BAR
```

□ 仮引数の置き換え

マクロ定義内の仮引数シンボルは、実引数と置き換えられます。

"&" （アンパサント記号）は、仮引数の置き換えのためにシンボルを分割する機能を持ちます。文字としての＆記号が必要なときには、＆記号を二つ続けて書いて下さい。

例

```
FOO     MACRO   P
        LDA     FOO&P
        LDA     FOO&Q
        LDA     FOO&&R
        ENDM
        FOO     X
```

これは次のように展開されます。

```
        LDA     FOOX
        LDA     FOOQ
        LDA     FOO&R
```

□ マクロ内でのラベル

マクロ内で局所的なラベルを使用する場合は、ラベル名をＬＯＣＡＬ宣言して下さい。

ＬＯＣＡＬ宣言されたシンボルは、マクロ呼び出し毎に異なった名前が割り当てられます。

ＬＯＣＡＬ宣言されたシンボルには
　　＿＿００００　～　＿＿９９９９
という名前を使用していますので、一般のシンボルは＿＿で始めないで下さい。

例

```
JR      MACRO   AB,CD
        LOCAL   LAB,LAB2
LAB     LDA     #AB
LAB2    STA     >CD
        BCC     LAB
        ENDM
;
        JR      2,3
;
        JR      1
        END
```

これは次のように展開されます。

```
__0000  LDA     #2
__0001  STA     >3
        BCC     __0000
;
__0002  LDA     #1
__0003  STA     >^&
        BCC     __0002
        END
```

ＲＥＰＴ　　繰り返しステートメント

【書式】

```
REPT            式
ステートメント
...
ENDM
```

【説明】

REPTとENDMの間のステートメントは式の回数繰り返されます。式は評価の結果、アブソリュート数にならなくてはなりません。さもないとエラー・メッセージが出されます。繰り返しステートメントは最大１０までネストできます。

ＩＲＰ　　無限繰り返しステートメント

【書式】

```
IRP             仮引数,<実引数リスト>
ステートメント
...
ENDM
```

【説明】

実引数リスト内の各々に対し、IRPとENDMの間のステートメントを繰り返します。実引数リストが空の場合は、ステートメントのリストを一度だけ繰り返し、仮引数の置き換えはしません。IRPステートメントは最大１０までネストできます。

例:

```
        IRP     X,<1,2>
        DB      X
        ENDM;
        ;
        IRP     Y,<>
        LDA     #Y
        STA     >LABEL
        ENDM
```

これは以下のように展開されます。

```
        DB      1
        DB      2
        ;
        LDA     #
        STA     >LABEL
```

マクロのアーギュメントをIRPのパラメータリストに渡すことはできません。

例:

```
FOO:    MACRO   X
        IRP     Y,<X>･･･ILLEGAL
        DB      Y
        ENDM
        ENDM
```

この場合はうまく処理されませんので注意して下さい。

5　出力ファイルの書式

5.1　アセンブル･リストの書式

リストファイルの出力形式は、以下の通りです。

```
File <input filename>   65C816 Assembler Version 1.0   <month>/<date>/<year>
<time>  Page   X
```

リストファイルの各ページの先頭には、上のフォーマットでプリントされます。

< input filename > － アセンブリファイル名
< month > － 月　　< date > － 日　　< year > － 年
< time > － hh:mm:ss で表します。

Page X － リストさせているページ数X

TITLE 疑似命令をソース・アセンブリ・ファイル内で使ったとき、以下のように現れます。

```
**********************************************************
                        < title text >
**********************************************************
```

リストファイル内の続きは以下のようになります。

```
                ****        < error massege >
   < line number > < location counter > < opcode > < oprand > <source code >
```

< error massege > － 行番号覧に＊記号を埋めた特別な行として出力されます。エラーメッセージは、エラーが発生した行の上に表示されます。

< line number > － ソース・ファイル（.asm）の行番号がプリントされます。

< location counter > － ４つのロケーション・カウンタがあり、それぞれセクションの１つを用います。ロケーション・カウンタの先頭の１文字で表します。有効な文字は以下の通りです。

A － アブソリュート・セクション・カウンタ
P － プログラム・セクション・カウンタ
D － データ・セクション・カウンタ
C － コモン・セクション・カウンタ

プログラム・カウンタの値は１６進か８進のいずれかでプリントされ、３２ビット数に相当します。リストされるフォーマットのデフォルトは１６進数で、アセンブリ内で -o スイッチを使うことにより、８進数になります。

< opcode > － カレント・インストラクションのオペコード値です。１６進法または、８進法でリストすることができます。

< oprand > － カレント・インストラクションのオペランド値です。オペ

ランドは１６進法または、８進法でリストすることができます。
オペランドのリスト指定が可能です。（上位ビット先頭また下位ビット先頭。リスト形式のデフォルトは上位ビット先頭です。）

< source code >　－　ソース・ファイルの内容そのものです。

５.２　クロスリファレンスファイル

クロスリファレンスファイルにはシンボル情報がプリントされます。このファイルは、アセンブリ時に -x オプションを使用した場合にのみ作成されます。リストファイルについて述べたのと同じように、シンボルファイルの先頭には同じ見出し情報がプリントされます。シンボル情報は、すべて次の見出しの下にリストされます。

```
  DEFINED  SYMBOL NAME  SYMBOL VALUE  TYPE  GLOBAL  REFERENCE
```

DEFINED　－　シンボルが定義されるソース・ファイルの行番号。
SYMBOL NAME　－　シンボル名
SYMBOL VALUE　－　シンボル値（１６進法または８進法でプリントできます）
TYPE　－　シンボルのモード定義
有効なモードは以下の通りです。
ABS　－　アブソリュート・セクション・シンボル
CODE　－　プログラム・セクション・シンボル
DATA　－　データ・セクション・シンボル
COMN　－　コモン・セクション・シンボル
EXTRN　－　外部シンボル
UNDEF　－　無定義シンボル　……　エラー
GLOBAL　－　シンボルがグローバルで定義されていると、このヘディングの下に’GLB’が現れます。

6　予約語一覧表

ニーモニック:

ADC	AND	ASL	BCC	BCS	BEQ	BIT	BMI	BNE
BPL	BRA	BRK	BRL	BVC	BVS	CLC	CLD	CLI
CLV	CMP	COP	CPX	CPY	DEC	DEX	DEY	EOR
INC	INX	INY	JML	JMP	JSL	JSR	LDA	LDX
LDY	LSR	MVN	MVP	NOP	ORA	PEA	PEI	PER
PHA	PHB	PHD	PHK	PHP	PHX	PHY	PLA	PLB
PLD	PLP	PLX	PLY	REP	ROL	ROR	RTI	RTL
RTS	SBC	SEC	SED	SEI	SEP	STA	STP	STX
STY	STZ	TAX	TAY	TCD	TCS	TDC	TRB	TSB
TSC	TSX	TXA	TXS	TXY	TYA	TYX	WAI	WDM
XBA	XCE							
adc	and	asl	bcc	bcs	beq	bit	bmi	bne
bpl	bra	brk	brl	bvc	bvs	clc	cld	cli
clv	cmp	cop	cpx	cpy	dec	dex	dey	eor
inc	inx	iny	jml	jmp	jsl	jsr	lda	ldx
ldy	lsr	mvn	mvp	nop	ora	pea	pei	per
pha	phb	phd	phk	php	phx	phy	pla	plb
pld	plp	plx	ply	rep	rol	ror	rti	rtl
rts	sbc	sec	sed	sei	sep	sta	stp	stx
sty	stz	tax	tay	tcd	tcs	tdc	trb	tsb
tsc	tsx	txa	txs	txy	tya	tyx	wai	wdm
xba	xce							

演算子:

+	−	—	*	/	%	>>	<<	==
!=	>	>=	<	<=	&	^	|	
LOW	HIGH	BANK	low	high	bank			

擬似命令:

ORG	PROG	DATA	COMN	EQU	SET	EXT	EXTERN	EXTERNAL
GLB	GLOBAL	PUBLIC	DB	BYTE	ASCII	ASC	DS	RMB
DW	WORD	DL	LWORD	DD	BIN	BINARY	HEX	OCT
OCTAL	INCLUDE	TITLE	NATIVE	MEM8	MEM16	IDX8	IDX16	EXTEND
LONG	DIRECT	AUTO	LOW	HIGH	BANK	DPAGE	DBANK	LIST
NLIST	NOLIST	PAGE	EJECT	SKIP	SPC	END	IF	IFDEF
IFNDEF	IF16	IFMEM8	IFMEM16	IFIDX8	IFIDX16	IF1	IF2	MACRO
ENDM	REPT	IRP	LALL	SALL	XALL	OFFSET	EMULATION	
org	prog	data	comn	equ	set	ext	extern	external
glb	global	public	db	byte	ascii	asc	ds	rmb
dw	word	dl	lword	dd	bin	binary	hex	oct
octal	include	title	native	mem8	mem16	idx8	idx16	extend
long	direct	auto	low	high	bank	dpage	dbank	list
nlist	nolist	page	eject	skip	spc	end	if	ifdef
ifndef	if16	ifmem8	ifmem16	ifidx8	ifidx16	if1	if2	macro
endm	rept	irp	lall	sall	xall	offset	emulation	

レジスタ名:

A	X	Y	S
a	x	y	s

その他

$	!	(	)	[	]	,	;	#

7　エラーメッセージ一覧表

アセンブラが生成するエラーメッセージは、次の４つに分類されます。

1．警告メッセージ

　警告メッセージは、警告のみを行ないます。これは、アセンブルを妨げません。

　警告メッセージは、以下の形式で表示されます。

ファイル名（行番号）：warning: メッセージ

2．アセンブル時エラーメッセージ

　アセンブル時エラーメッセージは、実際のプログラムのエラーを示します。このエラーが起こった時は、このプログラムのオブジェクト･ファイルは生成されません。

　アセンブル時エラーメッセージは、以下の形式で表示されます。

ファイル名（行番号）：error: メッセージ

3．致命的エラーメッセージ

　致命的エラーメッセージは、重大な問題がありアセンブルを継続できない事を示します。致命的エラーが発生した場合アセンブラは、致命的エラーメッセージを出力した後オブジェクト･ファイルを生成せずにアセンブルを中止します。

　致命的エラーメッセージは、以下の形式で表示されます。

ファイル名（行番号）：fatal : メッセージ

4．アセンブラ内部エラーメッセージ

　アセンブラ内部エラーメッセージは、ソースプログラムがどの様なものであろうと、通常は表示されません。もし表示された場合は、弊社にお知らせ下さい。

　アセンブラ内部エラーは、以下の形式で表示されます。

ファイル名（行番号）：　assembler error: メッセージ

IF less ELSE
　　対応するＩＦのないＥＬＳＥがあります。

IF less ENDIF
　　対応するＩＦのないＥＮＤＩＦがあります。

IF nesting error
　　ＩＦのネストがおかしくなっています。

IF nesting too deep
　　ＩＦのネストが深すぎます。

IF or MACRO nesting error
　　ＩＦもしくはＭＡＣＲＯのネストがおかしくなっています。

MACRO or INCLUDE nesting error
　　ＩＦもしくはＩＮＣＬＵＤＥのネストがおかしくなっています。

AUTO directive cannot use relocatable section
　　リロケータブルセクション内でＡＵＴＯ擬似命令が使われています。

absolute section: XXX
　　グループ定義の中にアブソリュートセクションの記述があります。

address must be grater than base address
　　ＯＲＧ命令で指定するアドレスは領域のベースアドレスより大きくなければなりません。

backward branch out of bounds line
　　ショート分岐命令では１２６バイトまでの後方分岐が有効です。ロング分岐命令では６４Ｋメモリ・バンクまでが有効ですが、その範囲を越えています。

cannot create file: XXX
　　ファイルが作成できません。

cannot open file: XXX
　　ファイルがオープンできません。

cannot use relocatable data
　　リロケータブルシンボルを扱えない命令のオペランドにリロケータブルシンボルの記述があります。

directive cannot have label
　　ラベルを付けることができない擬似命令にラベルを付けています。

divide by zero
　　ゼロで除算しています。

extended addressing mode assumed at forward reference
　　前方参照ではアブソリュート・アドレス・モードが仮定されます。

expression syntax error
　　式中に構文誤りがあります。

file write error
　　ファイル書き込みに失敗しました。

forward branch out of bounds line
　　ショート分岐命令では１２９バイト前方までの分岐が有効です。ロング分岐命令
　　では６４バンク・メモリまでが有効ですが、その値を越えています。

illegal addresing mode
　　アドレッシングモードが誤っています。

illegal address type
　　現領域以外の相対値を使用しています。

illegal comment delimiter
　　行末に不正な文字があります。

illegal external reference: XXX
　　パス２でのみ参照された外部シンボルがあります。

illegal forward reference or phase error
　　前方参照をしているかもしくはフェーズエラーが発生しました。

illegal label name
　　ラベル名に誤りがあります。

illegal operand syntax
　　オペランド記述に構文誤りがあります。

illegal operand type
　　不正な演算を行なっています。

illegal operand value
　　オペランド値に誤りがあります。

illegall section attribute
　　ＳＥＣＴ擬似命令の属性に誤りがあります。

illegal use of external
　　外部シンボルの使い方が間違っています。

input nesting too deep
　　ＭＡＣＲＯまたはＩＮＣＬＵＤＥのネストが深すぎます。

instruction is lot legal in the emulation mode
　　NATIVE モードでしか正しくない命令が EMULATION モードで使用されています。

invalid addresing mode
　　無効なアドレッシングモードが記述されています。

invalid character in number
　　数値定数に誤りがあります。

invalid index register
　　インデックスレジスタに誤りがあります。

invalid macro name
　　マクロ名に使えないシンボルが記述されています。

invalid op-code
　　オペコードに誤りがあります。

invalid op-code: XXX
　　オペコードに誤りがあります。

list data buffer overflow
　　リストの１行が長すぎます。

long word value out of range
　　オペランドの値が３バイトの範囲を越えています。

memory allocation fail
　　メモリー割当に失敗しました。

missing END
　　ＥＮＤがありません。

missing ENDM
　　ＥＮＤＭがありません。

missing XXX in parameter
　　引数でＸＸＸが足りません。

missing label
　　ラベルが必要です。

missing name
　　名前が必要です。

missing operand
　　オペランドに誤りがあります。

missing quote
　　引用符が足りません。

missing ')'
　　') ' が足りません。

missing ','
　　' , ' が足りません。

missing ']'
　　'] ' が足りません。

modulo by zero
　　ゼロで剰余算しています。

multiply define GROUP: XXX
　　グループ名が多重定義されています。

multiply defined MODULE: XXX
　　モジュール名が多重定義されています。

multiply defined title
　　タイトルが多重定義されています。

multiply defined: XXX
　　同一名のシンボルが多重に定義されます。

must be in MACRO definetion
　　マクロ定義ブロックの中になければなりません。

operand expected
　　オペランドがありません。

operand value must be absolute
　　オペランドの値は絶対値でなければなりません。

operand value out of range
　　オペランドの値が許容範囲を越えています。

out of address range
　　アドレスが許容範囲を越えています。

out of address space
　　メモリ空間の上限を越えました。

PBR was in agreement with bank-byte of operand
　　ＰＢＲの値とオペランドのバンク・バイトの値が一致しません。

phase error
　　フェーズエラーが発生しました。

phase error: XXX
　　フェーズエラーが発生しました。

relative branch out of range
　　相対ジャンプの許容範囲を越えています。

SECTION XXX out of limit
　　セクション内の限界値を越えました

storage size out of range
　　正の数値しか扱えません。

string data too long
　　式中の文字列定数が長すぎます。

symbol already different kind: XXX
　　シンボルとして使用することはできません。

symbol is not permanent: XXX
　　シンボルに外部属性を付けることはできません。

syntax error
　　構文誤りがあります。

temporary label is not parmanent
　　テンポラリーラベルに外部属性を与えることはできません。

too few operand
　　オペランドが足りません。

too long MACRO line
　　展開した１行が長すぎます。

too many MACRO parameter
　　マクロ引数が多すぎます。

too many local symbol
　　ローカルシンボルが多すぎます。

too many open files
　　オープンされたファイルが多すぎます。

too many operand
　　オペランドが多すぎます。

unbalanced ENDM
　　ＥＮＤＭの対応が間違っています。

undefined MACRO: XXX
　　マクロが定義されていません。

undefined section: XXX
　　セクション名が定義されていません。

undefined: XXX
　　シンボルが定義されていません。

unexpected XXX in parameter
　　引数に誤りがあります。

word value out of range
　　値が１６ビットの範囲を越えています。

リンカ

ユーザーマニュアル

１９９２年１０月　作成

はじめに

この説明書では、読者がリロケータブルアセンブラ、リンカの基本的な概念について理解していることを前提に、リンカの操作方法について説明します。

目　　次

1. リンカの概要

リンカは、アセンブラが出力した複数のオブジェクトモジュールとライブラリをひとつに結合します。外部シンボルの参照を解決し、リロケータブルセクションを再配置して、実行可能なロードモジュールを作成します。

- 複数のオブジェクトモジュールを結合する

  リロケータブルアセンブラでは、ソースプログラムを複数のファイルに分割して開発を行うことができます。リンカは、これらの別々に記述されアセンブルされたオブジェクトファイルを結合します。

- 外部参照を解決する

  分割されたモジュールから他のモジュールの情報を外部シンボルとして参照します。リンカは、外部シンボルの参照と定義をつきあわせて、外部参照を対応する外部定義に結び付けます。

- リロケータブルセクションを再配置する

  各モジュールのリロケータブルセクションは、同名のセクションごとに一つにまとめられて、リンカのコマンドで指定されたアドレスに再配置されます。

- ライブラリを結合する

  リンカは、ライブラリに含まれるモジュールを探索して、必要なモジュールのみをロードモジュールに結合します。

ライブラリの作成・保守の方法については、ライブラリアンの説明書を参照して下さい。

２．　リンカの操作

２．１　コマンド行

リンカは次のコマンド行で起動します。

書式：

```
ｓｆｌ　［オプション］　入力ファイル ...
ｓｆｌｅ　［オプション］　入力ファイル ...
```

入力ファイルには、アセンブラが出力したオブジェクトファイルまたはライブラリファイルを並べて指定します。リンカはコマンド行で指定された入力ファイルを左から右の順に結合します。
出力ファイル名は、最初の入力ファイル名の拡張子を ".lod" に変えたものになります。ただし、ＨＥＸフォーマットで出力する場合は拡張子は ".hex" になります。
出力ファイル名は、-ｏオプションで指定することもできます。

ｓｆｌｅは内部のワークエリアとしてＥＭＳ規格の拡張メモリを使用します。これによりリンクすることが出来るプログラムが大きくなります（速度は若干低下します）。
大きなプログラムのリンクなどで "memory allocation fail" エラーが発生した場合は、ｓｆｌｅを使用するとよいでしょう。

リンカのオプションを次に示します：

-Dシンボル[＝１６進数値]

外部シンボルを定義します。シンボル名は大文字小文字を区別するので注意してください。下線部分には余分な空白を入れることはできません。シンボルの値は“＝”に続いて指定した１６進数値に設定されます。１６進数値を省略したときの値は０です。
例：

```
-Ｄｓｐ＿ｉｎｉｔ＝ＦＦＦＦ
```

-Uシンボル

未定義の外部参照シンボルをリンカのシンボルテーブルに登録します。シンボル名は大文字小文字を区別するので注意してください。下線部分には余分な空白を入れることはできません。この機能はライブラリに含まれるモジュールを強制的に引っ張りだしてくる目的に使用します。
例：

```
-Ｕｌｉｂｎａｍｅ
```

-o　出力ファイル名

リンカの出力ファイル名を指定します。
例：

```
-ｏ　ａ.ｏｕｔ
```

-p　入力ファイルを結合しますが、リロケータブルセクションの再配置を行いません。このオプションを指定した場合には－ｒオプションは意味がありません。出力されるモジュールは再びリンカの入力ファイルにすることができます。このオプションは、複数のオブジェクトモジュールを結合してひとまとめにする目的に使用します。

-r　名前[＝１６進数値][,名前[＝１６進数値]] ...
リロケータブルセクションを再配置するアドレスを指定します。名前にはグループ名またはセクション名を指定します。グループ名、セクション名は大文字小文字を区別するので注意してください。下線部分には余分な空白を入れることはできません。このオプションは必要なだけ指定できます。複数のオプションがある場合には、２番目以降は直前の指定の続きとして扱います。詳細は２.２「リロケータブルセクションの再配置」を参照して下さい。

-s　ロードモジュールにローカルシンボル情報を出力しません。リンク後のロードモジュールのファイルの大きさを節約したい場合に使用します。ただし、このオプションを指定するとローカルシンボルが出力ファイルから削除されるので、デバッガでローカルシンボルを使用できなくなります。

-v　バージョン番号を標準エラー出力に表示します。

-w　ワーニングメッセージの出力を抑止します。

-ｌｓ[v]　ファイル名
指定ファイルにセクション・マップを出力します。ｖオプションが指定された場合はセクションの先頭アドレスの小さいものから出力されます。ｖオプションが指定されなければセクションの名前順に出力されます。ファイル名に「－」が指定された場合には標準出力に出力されます。

-ｌｇ[l][m][v]　ファイル名
指定ファイルにシンボル・リストを出力します。ｌオプションが指定された場合はローカルシンボルも出力されます。ｍオプションが指定された場合にはモジュール毎に出力されます。ｖオプションが指定された場合にはシンボルの値の小さいものから出力されます。ｖオプションが指定されなければシンボルの名前順に出力されます。ファイル名に「－」が指定された場合には標準出力に出力されます。

-ｌｃ　ファイル名
指定ファイルにクロス・リファレンス・リストを出力します。ファイル間の相互参照のみリストされます。ファイル名に「－」が指定された場合には標準出力に出力されます。クロス・リファレンス・リストは、シンボルの名前順に出力されます。

-I　　インテルＨＥＸフォーマットで出力します。（デフォルトはバイナリフォーマットで出力します。）

-Z　　モトローラＨＥＸフォーマット（ＭＨＥＸ）で出力します。

+O　１６進数値
-O　１６進数値
　　１６進数値で指定されたオフセットを出力するＨＥＸファイルのアドレスに加算（＋）または減算（－）します。ＨＥＸフォーマットで出力するときのみ有効です。

-e　グループ名
-e　セクション名
　　指定されたグループ、または、セクションのオブジェクトを出力します。 -eオプションの指定が複数あるときにはＯＲ条件となります。 -Rオプションとの組合せではＡＮＤ条件になります。

-R　先頭アドレス：終了アドレス
　　指定されたアドレスの範囲内のオブジェクトを出力します。ＨＥＸフォーマットで出力するときのみ有効です。アドレスは１６進数値で指定します。下線部分には余分な空白を入れることはできません。 -Rオプションの指定が複数あるときにはＯＲ条件となります。 -eオプションとの組合せではＡＮＤ条件になります。

-d　ファイル名
　　コマンド行の「-d ファイル名」の形式の引数は、ファイルに格納されたコマンド引数の並びに置き換えられます。リンカ起動時のオプション指定、入力ファイル指定がたくさんある場合や、同じ指定で繰り返しリンクをする場合に使用します。ファイルに格納する引数は、空白文字、水平タブ、改行文字で区切ります。また、「＃」文字から行末までを注釈として扱います。
　　例：

```
sfl -d linkfile
```

出力フォーマットの指定がなければ、バイナリフォーマットで出力します。
-e、-Rオプションの指定がなければ、全てのオブジェクトを出力します。

２.２　リロケータブルセクションの再配置

リンカは、-ｒオプションの指定にしたがって、リロケータブルセクションの再配置を行います（注意：アブソリュートセクションは再配置できません）。指定がない場合には、リンカは０番地から上位アドレス方向にリロケータブルセクションを再配置していきます。
次の書式でセクションの配置を指定します。

書式：

-ｒ　名前[=１６進数値][,名前[=１６進数値]]...

名前にはグループ名またはセクション名を指定します。指定された名前がグループであるかセクションであるかの判定はリンカが自動的に行います。１６進数値でグループ／セクションを再配置するメモリ領域の先頭アドレスをあたえます。先頭アドレスの指定は、「，」で区切っていくつでも並べることができます。先頭アドレスを省略すると、直前に割り付けられたグループ／セクションの後ろに続けて再配置します。指定されなかったリロケータブルセクションは、指定によって割付を行った最後のグループ／セクションの後ろに続けて再配置されます。
以下に指定例を示します。

-ｒ　ｐｒｏｇ=１０００,ｄａｔａ=８０００

グループ／セクションｐｒｏｇを１０００Ｈ番地に、グループ／セクションｄａｔａを８０００Ｈ番地にそれぞれ再配置します。

-ｒ　ｐｒｏｇ=１０００,ｄａｔａ

グループ／セクションｐｒｏｇを１０００Ｈ番地に再配置し、その後ろに続けてグループ／セクションｄａｔａを再配置します。仮にｐｒｏｇの大きさを４００Ｈとすると、ｄａｔａは１４００Ｈ番地に再配置されることになります。

２．３　出力条件の指定

-e、-Rオプションで条件を指定してロードモジュールを出力することができます。出力条件は、グループ、セクション、アドレス範囲で指定できます。同じオプションを複数指定した場合にはＯＲ条件に、異なるオプションの組合せではＡＮＤ条件になります。

２．３．１　グループ／セクション指定

-eオプションで、指定したグループまたはセクションのオブジェクトを取り出すことができます。

書式：

```
-e  グループ名
-e  セクション名
```

グループ名、セクション名は大文字小文字を区別するので注意してください。
指定された名前がグループ／セクションのどちらであるかの判定は、リンカが自動的に行います。

例：

```
-e  ＡＡＡ                          ....(1)
-e  ＢＢＢ  -e  ＣＣＣ              ....(2)
```

(1)は、「ＡＡＡ」という名前のグループ／セクションのオブジェクトを取り出します。
(2)は、「ＢＢＢ」、「ＣＣＣ」両方のグループ／セクションのオブジェクトを取り出します。

２．３．２　アドレス範囲指定

-Rオプションで、指定されたアドレスの範囲内のオブジェクトを出力します。範囲指定はＨＥＸフォーマットで出力するときのみ有効です。

書式：

-R　先頭アドレス：終了アドレス

アドレスは１６進数値で指定します。下線部分には余分な空白を入れることはできません。アドレスはバイト単位で指定できます。

例：

```
-R  0：1FFF                              ....(1)
-R  0：FFF  -R  8000：8FFF               ....(2)
-e  AAA  -R  2000：3FFF                  ....(3)
```

(1)は、０～１ＦＦＦＨ番地までのオブジェクトを取り出します。

(2)は、０～ＦＦＦＨ番地、８０００Ｈ～８ＦＦＦＨ番地のオブジェクトを取り出します。

(3)は、“ＡＡＡ”という名前のグループ／セクションの２０００Ｈ～３ＦＦＦＨ番地のオブジェクトを取り出します。

２．４　ライブラリの探索

リンカは、コマンド行に指定された入力ファイルを左から右の順に処理します。したがって、通常はコマンド行の右端にライブラリファイルを指定します。ライブラリとオブジェクトモジュールは自動的に識別されます。

例：

```
ｓｆｌ　モジュール１　モジュール２ ...　ライブラリファイル ...
```

ライブラリファイルを２つ以上指定することもできます。この場合ライブラリは左から右の順に探索されることになります。

リンカは、モジュールの探索をＮパスでおこなっています。このため、ライブラリ中のモジュールの並びに関係なく必要なモジュールが結合されます。

３．　リンカ出力リストファイル

指定ファイルに各種リストを出力することが出来ます。ファイル名に「－」が指定された場合には標準出力に出力します。

３.１　セクション・マップ

-ｌｓオプションでセクション・マップを出力します。

書式：

```
-ｌｓ、-ｌｓｖ
```

出力形式：

```
ｎａｍｅ　　ｓｔａｒｔ－ｅｎｄ　　ｓｉｚｅ　　[ｇｒｏｕｐ]
   ･･･
```

ｖオプションが指定された場合にはセクションの先頭アドレスの小さいものから出力されます。ｖオプションが指定されなければセクションの名前順に出力されます。

３.２　シンボル・リスト

-ｌｇオプションでシンボル・リストを出力します。

書式：

```
-ｌｇ、-ｌｇｌ、-ｌｇｖ、-ｌｇｌｖ
```

出力形式：

```
ｎａｍｅ　　ｖａｌｕｅ　　[*][ｓｅｃｔｉｏｎ]
   ･･･
```

書式：

```
-ｌｇｍ、-ｌｇｍｖ、-ｌｇｌｍ、-ｌｇｌｍｖ
```

出力形式：

```
ｍｏｄｕｌｅ：
ｎａｍｅ　　ｖａｌｕｅ　　[*][ｓｅｃｔｉｏｎ]
   ･･･
```

ｌオプションが指定された場合はローカルシンボルも出力されます。ｍオプションが指定された場合にはモジュール毎に出力されます。ｖオプションが指定された場合にはシンボルの値の小さいものから出力されます。ｖオプションが指定されなければシンボルの名前順に出力されます。
コモンシンボルには「＊」印がつきます。

3.3 クロス・リファレンス・リスト

-ｌｃオプションでクロス・リファレンスを出力します。

書式：

```
-ｌｃ　　ファイル名
```

出力形式：

```
name  file  value
      file:  address  address ...
      ...
...
```

クロス・リファレンス・リストはシンボルの名前順に出力されます。
ファイル間の相互参照のみリストされます。

４　ＨＥＸファイルフォーマット

４.１　ＨＥＸファイル

これは、リンクした結果のメモリーイメージをインテルＨＥＸ形式と似たフォーマットで表現したものです。
インテルＨＥＸ形式との違いは２４ビットアドレスを表現するために、拡張アドレスでの扱いがやや異なるという点です。アドレス、データなどは全て１６進数で表現されます。基本的なフォーマットを下に示します。

<start_charactor><byte_count><address><record_type><data><sumcheck>

・start_charactor
これはレコードの先頭を示すもので、コロン”：”を使用します。

・byte_count
一つのレコードにあるデータ部の長さをバイト単位で示したもので、２文字（２digits）であらわします。

・address
データが入る先頭アドレスです。４文字（４digits）です。

・record_type
レコードの種類を示すコードで２文字です。４種類ありますが詳細は後述します。

・data
格納されるデータは１バイト単位ではいります。ここでの文字数はbyte_countの値によってきまります。例えば、byte_countが１０であれば１０進数で１６になるので、１６バイト＝３２文字分がデータとなります。

・sumcheck
レコードのサムチェックを表すもので２文字です。byte_countからデータ部までを１バイト毎に加え、合計の下位１バイトの２の補数がこれに相当します。

レコードタイプには以下の４種類があります。

00	データレコード
01	エンドレコード
02	拡張アドレスレコード
03	スタートレコード

00のデータレコードの場合、データ部には通常のデータがはいります。
例：

:10000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00

この例では１６バイトのデータが指定アドレスから連続して入ります。

01のエンドレコードでは、これがレコードの最後である事を示します。
例：

:00000001FF

このレコードを使う場合は、BYTE_COUNT=00,ADDRESS=0000になります。

02の拡張アドレスレコードでは、２４ビットアドレスの上位２０ビットのオフセットアドレスをデータ部に示します。ただし、データ部での表現は３バイト、つまり２４ビット分のアドレスになりますが、最上位の４ビット分は無視します。実アドレスは拡張表現のオフセットアドレスとレコードアドレスとの和になります。
例：

:030000020020000B 拡張アドレス（オフセットアドレス）
:01800000017E

オフセットアドレス	02000	（上位２０ビット）
＋レコードアドレス	8000	（下位１６ビット）
＝実アドレス	028000	（２４ビット）

１行目はレコードタイプが02なのでデータ部の02000がオフセットアドレスになります。従って２行目のデータ01は、28000のアドレスにデータ01が入ります。

03のスタートレコードは、メモリイメージがプログラムコードを含んでいる場合、実行すべきプログラムコードがどれかを示すためのものです（実行開始アドレス）。このアドレスはアセンブリファイルのＥＮＤ疑似命令で指定したものが入ります。
例：

:05000003002000800058

データ部は１０バイトで表現されていますが、このうち最初の６バイトがオフセットアドレス（上位２０ビット）で残りの４バイトがレコードアドレス（下位１６ビット）になります。従ってオフセットアドレス＝02000、レコードアドレス＝8000より実アドレス28000が、スタートアドレスになります。

４.２　ＭＨＥＸファイル

ＭＨＥＸファイルは、ＨＥＸファイルと同様の内容ですがモトローラＳフォーマットで表現されています。以下に基本的なモトローラＳフォーマットを示します。

<type><record_length><address><code/data><sumcheck>

・type

タイプは後に続く内容が何であるかを示します。

- Ｓ０　－－　ブロックの開始及びコメントデータ。
- Ｓ２　－－　３バイト長のアドレスとイメージデータ。
- Ｓ５　－－　ブロック内にあるレコードの数をデータとする。
- Ｓ８　－－　ブロックの終了と３バイトの実行開始アドレス。

・record_length

コード全体（address+code/data+sumcheck）に含まれるバイト数。

・address

データの開始アドレス。Ｓ０では常に0000、Ｓ５ではレコード数。

・code/data

メモリーイメージデータを１バイトごとに表現します。Ｓ０ではコメント、Ｓ５、Ｓ８ではこのフィールドはありません。

・sumcheck

record_length, address, code/dataの３つのフィールドのチェックサム値。１の補数で表現します。

データは全て１６進数で表現されます。またＳ８での実行開始アドレスはＥＮＤ疑似命令で指定されたアドレスが用いられますが、指定されていない場合は、プログラムセクションの開始アドレスがはいります。

例：

```
S00700000077656469
S20A001006A9341218181 8A9
S804001000EC
```

付録Ａ

エラーメッセージ一覧表

<XXX> grouping section
　-rで指定されたセクションがグループに所属しています。

<XXX> has different base address
　同一セクションに、ファイル間で異なるベースアドレスが与えられています。

<XXX> has incompatible type
　同一セクションに、ファイル間で異なる属性が与えられています。

<XXX> multiply assign group
　一つのセクションが多重にグループに割り付けられました。

<XXX> not defined
　-r、-b、-Bで指定されたグループ／セクションが定義されていません。

<XXX> out of address sapce
　セクションを再配置した結果、メモリ空間の上限を越えました。

<XXX> out of limit
　セクションを結合した結果、許容サイズの上限を越えました。

<XXX> out of page
　セクションを再配置した結果、ページの限界を越えました。

<XXX> out of range
　セクションを再配置した結果、許容アドレス範囲を越えました。

XXX: cannot open
　ファイルがオープンできません。

XXX: cannot open in pass 2
　パス２でファイルがオープンできません。

XXX: not an object format
　オブジェクトファイルでもライブラリファイルでもありません。

XXX: not an object format: XXX
　ライブラリ中にオブジェクト以外のモジュールがあります。

bad library index
　ライブラリのインデックスが正しくありません。

calculate stack overflow
calculate stack underflow
　リロケーションデータに誤りがあります。

cannot create file: XXX
　ファイルが作成できません。

cannot relocate: section <XXX> is ABS
　アブソリュートセクションは再配置できません。

cannot take group/section base
　セクションに所属しないシンボルのグループ／セクションベースアドレスが参照されました。

cannot take option value(1)
cannot take option value(2)
　セクションに所属しないシンボルのオプション値が参照されました。

cannot take section base
　セクションに所属しないシンボルのセクションベースアドレスが参照されました。

cannot take section offset
　セクションに所属しないシンボルのセクション相対アドレスが参照されました。

divide by zero
　リロケーションデータに誤りがあります。

duplicate group/section name <XXX>
　グループ名とセクション名に同じ名前は使えません。

entry address not defined
　エントリアドレスが定義されていません。

external symbol section miss match: XXX
　シンボルの所属するセクションが定義と宣言で異なっています。

format error: group id out of range: XX
format error: no current section
format error: section id out of range: XX
format error: symbol id out of range: XX
format error: unknown tag code: 0xXX
　ファイルフォーマットに誤りがあります。

group table overflow
　グループテーブルがオーバーフローしました。

illegal common symbol: XXX
　コモンシンボルと同名のＥＱＵシンボルがあります。

library index too big
　ライブラリのインデックスが大きすぎます。

macro id out of range: XX
macro nesting too deep
　リロケーションデータに誤りがあります。

memory allocation fail
　メモリ割当てに失敗しました。

message syntax error
　メッセージファイル内に不正な行があります。

message table overflow
　メッセージが多すぎます。

modulo by zero
　リロケーションデータに誤りがあります。

multiply defined common: XXX
　コモンシンボルを格納するセクションが複数定義されました。

multiply defined entry address
　エントリアドレスが複数定義されました。

multiply defined: XXX
　シンボルが複数定義されました。

not support OM/LM format
　ファイルフォーマットに誤りがあります。

overlapped section XXX/XXX
　セクションの配置アドレスが重なりました。

reserved message number: XX
　予約された番号のメッセージがあります。

section table overflow
　セクションテーブルがオーバーフローしました。

symbol link table overflow
　　シンボル操作のためのテーブルがオーバーフローしました。

table of contents for library is out of date
　　ライブラリインデックスが最新のものではありません。

target name no match
　　ターゲット名が異なるオブジェクトをリンクしました。

undefined: XXX
　　未定義のシンボルがあります。

unknown OM/LM version
　　ファイルフォーマットに誤りがあります。

unknown operator: XX
working variable id out of range: XX
　　リロケーションデータに誤りがあります。

write error
　　ファイル書き込みに失敗しました。

ライブラリアン
ユーザーマニュアル

1992年　7月　作成

はじめに

この説明書ではライブラリアンの操作方法について説明します。ライブラリアンの操作方法はＵＮＩＸのａｒコマンドと互換性があります。

目　　次

１．　ライブラリアンの概要

ライブラリアンは複数のオブジェクトモジュールを単一のライブラリファイルにまとめて管理するためのツールです。リンカには、ライブラリを探索し必要なモジュールをロードモジュールに結合する機能がありますので、頻繁に使用するサブルーチンやデータをモジュール単位でライブラリに格納しておくと便利です。

ライブラリに対して次のような操作ができます：

- ライブラリを新しく作る
- モジュールを追加する
- モジュールを削除する
- モジュールを更新する
- モジュールを移動する
- モジュールを取り出す

具体的な操作例を３.「操作例」にあげてあるので参考にして下さい。

ライブラリアンはモジュールをモジュール名で識別します。モジュール名には、そのモジュールが格納されている（されていた）ファイル名を使用します。ＭＳ－ＤＯＳではファイル名の大文字・小文字を区別しないので注意が必要です。同じ綴りのモジュール名は大文字でも小文字でも同じ名前になります。

2. ライブラリアンの操作

2.1 コマンド行

ライブラリアンは次のコマンド行で起動します。
書式：

ｓｆｍ　操作指定　［位置名］　ライブラリファイル　モジュール名 ...

操作指定は、操作の種類を示す英文字“ｄｒｑｔｐｍｘ”のどれか一文字（コマンド）とオプションの英文字“ｖｕａｉｂｃｌ”の組合せで指定します。オプションは複数指定できます。
これらの英文字の意味は下のとおりです：

ｄ　削除コマンド。指定されたモジュールをライブラリファイルから削除します。

ｒ　更新コマンド。ライブラリファイルに格納されている指定モジュールの更新を行います。ｕオプションが同時に指定されたときには、変更日付がライブラリのものよりも新しいモジュールだけを更新します。位置指定のオプション“ａｉｂ”を指定したときにはコマンド行に位置名が必要です。ライブラリアンは、ライブラリに登録されていないモジュールを位置名で示されるモジュールの後ろ（ａ）に追加またはその前（ｉまたはｂ）に挿入します。オプション“ａｉｂ”の指定がない場合には、ライブラリの最後に未登録のモジュールを追加します。

ｑ　追加コマンド。ライブラリファイルの最後に指定されたモジュールを追加します。位置指定のオプション“ａｉｂ”は使用できません。このコマンドでは追加するモジュールが既にライブラリファイルに格納されているかどうかを調べません。作業ファイルを使用せずに直接ライブラリファイルにモジュールを追加するので高速です。

ｔ　表示コマンド。ライブラリファイルに格納されているモジュールの一覧表を標準出力に出力します。モジュール名が指定されていない場合には、ライブラリに格納されている全てのモジュールを表示します。モジュール名が指定されている場合には、指定されたものについてのみ表示を行います。

ｐ　出力コマンド。指定されたモジュールを標準出力に書き出します。

ｍ　移動コマンド。指定されたモジュールをライブラリファイルの最後に移動します。位置指定のオプション“ａｉｂ”を指定したときには、コマンド行に位置名が必要です。この場合には、ｒコマンドと同様に、指定された位置へモジュールを移動します。

ｘ　抽出コマンド。指定されたモジュールをライブラリファイルから取り出します。モジュール名を省略した場合には全てのモジュールがライブラリファイルから取り出されます。

ｖ　　冗舌オプション。このオプションが指定されると、ライブラリアンがひとつのモジュールを処理するごとに標準出力にその旨が表示されます。ｔコマンドと同時に指定した場合には、モジュールの詳細な情報を表示します。ｐコマンドと同時に指定した場合には、書き出すモジュールの先頭にモジュール名を出力します。また、ライブラリアン起動時にバージョン番号などを標準エラー出力に表示します。

ｃ　　新規作成オプション。指定されたライブラリファイルがないときに限り、ライブラリアンはライブラリファイルを新規作成します。このオプションはライブラリファイルが新規に作成されるときに出力されるメッセージを抑止します。

ｌ　　ローカル作業ファイルのオプション。ライブラリアンは作業ファイルを「／ｔｍｐ」ディレクトリ（ＭＳ－ＤＯＳでは環境変数ＴＭＰで指定されたディレクトリ）に作成します。このオプションは、作業ファイルをコマンドを起動した時点のローカルディレクトリに作成することを指示します。

ライブラリアンのコマンド指定の具体例が３.「操作例」にあります。これもあわせて参照して下さい。

２.２　ファイル引数

モジュール名がたくさんあって、ライブラリアンのコマンド行に書ききれない場合や同じ指定を繰り返す場合には、ファイルによるコマンドの指定ができます。

コマンド行の「-F ファイル名」の形をした引数は、そのファイルに格納されている引数のリストに置き換えられます。ファイルに格納する引数は、空白文字、水平タブ、改行文字で区切ります。また、「＃」文字から行末までを注釈として扱います。

例：

```
ｓｆｍ　-F ｌｉｂｃｍｄ
```

２.３　作業ファイル

ライブラリアンはライブラリを変更するときに作業ファイルを使用します。ライブラリファイルを不注意な操作で破壊しないために、ライブラリアンは新しいライブラリのイメージを作業ファイルに作成し、エラーがなければ元のライブラリファイルに書き戻します。

したがって、ライブラリの変更を行うときには、変更しようとしているライブラリの大きさの２倍程度のディスク領域が必要です。

３．　操作例

この項ではライブラリアンの操作例を示します。
説明のために次のライブラリ“ＭＹＬＩＢ”を使用します。

```
sfm  t  MYLIB          ... ＭＹＬＩＢの一覧表を表示
AAA
BBB
CCC
```

下線を引いた行はライブラリアンのコマンドで、それ以外の行はライブラリアンが出力したメッセージです（ただしライブラリアンが起動時に出力するバージョン表示などは省略します）。“...”に続く文章は説明のための注釈です。

３.１　ライブラリを新しく作る

ライブラリを新しく作成するときには、ｑコマンドまたはｒコマンドを使用します。ｖオプションを指定してライブラリアンの動きを確認することができます。ｃオプションを指定するとライブラリを新規に作成したときに表示されるメッセージを抑止することができます。
ＭＹＬＩＢを新しく作る：

```
sfm qv MYLIB AAA BBB CCC  ... qコマンドのとき
sfm:creating MYLIB
q - AAA
q - BBB
q - CCC
```

３.２　モジュールを追加する

ライブラリにモジュールを追加する場合には、ｑコマンドまたはｒコマンドを使用します。ｑコマンドは無条件にライブラリの最後にモジュールを追加したい場合に使用し、ｒコマンドは指定した位置にモジュールを挿入したい場合に使用します。位置指定は“ａｉｂ”オプションで指定します。
ＭＹＬＩＢの最後にモジュールＤＤＤを追加する：

```
sfm qv MYLIB DDD       ... ＤＤＤを追加
q - DDD
sfm t MYLIB            ... リストしてみる
AAA
BBB
CCC
DDD  ... 最後に追加されている
```

ＭＹＬＩＢのモジュールＢＢＢの前にモジュールＸＸＸを挿入する：

```
sfm rvi BBB MYLIB XXX    ... XXXを挿入
a - XXX
sfm t MYLIB ... リストしてみる
AAA
XXX ... BBBの前に挿入されている
BBB
CCC
```

ＭＹＬＩＢのモジュールＢＢＢの後ろにモジュールＸＸＸを追加する：

```
sfm rva BBB MYLIB XXX    ... XXXを追加
a - XXX
sfm t MYLIB ... リストしてみる
AAA
BBB
XXX ... BBBの後ろに追加されている
CCC
```

3.3 モジュールを削除する

ライブラリからモジュールを削除する場合にはｄコマンドを使用します。引数で指定したモジュールが削除されます。

ＭＹＬＩＢからモジュールＣＣＣを削除する：

```
sfm dv MYLIB CCC      ... CCCを削除
d - CCC
sfm t MYLIB ... リストしてみる
AAA
BBB ... 確かにCCCが削除されている
```

3.4 モジュールを更新する

ライブラリのモジュールを更新する場合にはｒコマンドを使用します。ｕオプションと組み合わせて指定すれば、ファイルの更新日時を比較して、変更されているモジュールのみを更新することもできます。

ＭＹＬＩＢのモジュールＡＡＡを更新します：

```
sfm rv MYLIB AAA      ... AAAを更新
r - AAA ... AAAは更新されました。
```

３．５　モジュールを移動する

ライブラリのモジュールを移動する場合にはｍコマンドを使用します。位置指定オプション“ａｉｂ”を指定すれば指定した場所に、そうでないときにはライブラリの最後に、それぞれ指定されたモジュールを移動します。

ＭＹＬＩＢのモジュールＡＡＡを最後へ移動：

```
sfm mv MYLIB AAA      ... ＡＡＡを移動
m - AAA
sfm t MYLIB ... リストしてみる
ＢＢＢ
ＣＣＣ
ＡＡＡ ... ＡＡＡが最後に移動している
```

ＭＹＬＩＢのモジュールＣＣＣをＢＢＢの前へ移動：

```
sfm mvb BBB MYLIB CCC   ... ＣＣＣを移動
m - CCC
sfm t MYLIB ... リストしてみる
ＡＡＡ
ＣＣＣ ... ＣＣＣがＢＢＢの前に移動している
ＢＢＢ
```

ＭＹＬＩＢのモジュールＡＡＡをＢＢＢの後ろに移動：

```
sfm mva BBB MYLIB AAA     ... ＡＡＡを移動
m - AAA
sfm t MYLIB ... リストしてみる
ＢＢＢ
ＡＡＡ ... ＡＡＡがＢＢＢの後ろに移動している
ＣＣＣ
```

３．６　モジュールを取り出す

ライブラリに格納されているモジュールを取り出す場合にはｘコマンドを使用します。指定したモジュールの内容を独立したファイルとして取り出すことができます。

ＭＹＬＩＢからモジュールＣＣＣを取り出す：

```
sfm xv MYLIB CCC      ... ＣＣＣを取り出す
x - CCC ... ＣＣＣを取り出した
```

付録Ａ

エラーメッセージ一覧表

XXX cannot create
　ファイルが作成できません。

XXX cannot open
　ファイルがｏｐｅｎできません。

XXX does not exist
　指定されたライブラリ・ファイルが存在しません。

XXX not found
　指定されたモジュールがライブラリ中にありません。

XXX not in library format
　ライブラリではありません。

abi not allowed with q
　ｑコマンドにはａｂｉオプションを指定できません。

bad option 'X'
　コマンド・ラインのオプションの指定が間違っています。

cannot create XXX
　ファイルを作成することができません。

cannot create scond temp
cannot create temp file
cannot create third temp
　作業ファイルを作成することができません。

cannot open XXX
　引数ファイルをｏｐｅｎすることができません。

memory allocateion fail
　メモリ割当ができません。

missing file name after -F
　引数ファイルの指定がありません。

malformed library (at XX)
　ライブラリに格納されているメンバーのヘッダー情報が破壊されています。

one of [mrxtdpq] must be specified
　コマンド文字の指定が抜けています。

only one of [mrxtdpq] allowed
　コマンド文字が複数指定されています。

phase error on XXX
　ファイルが作業途中で変更されています。

write error
　書き込みエラーが発生しました。

ＩＳソース・コンバータ
ユーザーマニュアル

１９９２年　７月　作成

はじめに

この説明書ではＩＳソース・コンバータの操作方法について説明します。

目　　次

1. ＩＳソース・コンバータの概要

ＩＳソース・コンバータは、インテリジェント・システムズの６５０２／６５８１６リロケータブルアセンブラ（ＩＳ６５）用に書かれたソースを、ＳＦＡ用のソースに変換するツールです。

2. ＩＳソース・コンバータの操作

2.1 コマンド行

ＩＳソース・コンバータは次のコマンド行で起動します。

書式：

ｓｆｉｓ [-v] ソースファイル...

-ｖオプションが指定された場合には、ＩＳソース・コンバータのバージョン番号を標準エラー出力に表示します。

ソースファイルはＩＳ６５用のアセンブラソースファイルを指定します。複数の指定が可能です。
ソースファイルの拡張子が ".x65" の場合は、拡張子を省略できます。
ソースファイルに拡張子が ".asm" のファイルは指定できません。

2.2 出力ファイル

コンバータが出力するファイルは、ソースファイル名から拡張子を取りさったものに拡張子 ".asm" をつけたファイル名になります。
カレント・ディレクトリ以外のソースファイルを指定しても、コンバータが出力するファイルはカレント・ディレクトリに作成されます。

３．　制限事項

ＩＳ６５とＳＦＡの仕様の違いからソースコンバータには幾つかの制限事項があります。これらについて次に説明します。

□マクロ引き数の展開

ＩＳ６５ではマクロの引き数に％ｌａｂｅｌと書くことによりｌａｂｅｌの実際の値を文字列として渡すことができますが、ＳＦＡにはこの機能はありません。

□グルーピング

ＩＳ６５ではＧＲＯＵＰ疑似命令により、ソース上でバンク値を指定してバンク単位でのグルーピングを行います。ＳＦＡではＳＥＣＴ疑似命令でセクションを定義し、ＧＲＯＵＰ疑似命令でセクションの集合を定義します。アドレスの割付はリンク時にグループまたは、セクション（単一セクションに限る）に対して指定します。このことからソースコンバータはＩＳ６５のＧＲＯＵＰ疑似命令をＳＦＡのＳＥＣＴ疑似命令に変換します。そしてそのセクションを単一セクション（どのグループにも属さない）としてリンク時にバンク値に対応したアドレスを指定します。

□疑似命令

ＩＳ６５にある次の疑似命令はＳＦＡにはありません。使用しないで下さい。

- ＤＳＰＩＦ／ＮＤＳＰＩＦ　　条件式部分のリスト出力／抑止制御
- ＰＵＢＡＬＬ／ＯＮＰＵＢ／ＯＦＦＰＵＢ　シンボルのパブリック宣言の制御
- ＥＲＲＯＲ／ＮＥＲＲＯＲ　　エラー情報のリスト出力／抑止制御
- ＯＮＭＥＭ／ＯＦＦＭＥＭ　　オブジェクト出力／抑止制御
- ＩＦＢ／ＩＦＮＢ　　マクロ内の条件アセンブル制御
- ＥＲＲＴＡＧ　　エラー情報ファイルの出力制御
- ＦＩＮＡＬ　　エラー発生時のエディタ起動制御
- ＦＩＬＥ　　ソースファイルの切り替え
- ＫＡＮＪＩ　　リストファイルへの漢字コード出力許可

付録Ａ

エラーメッセージ一覧表

cannot create: XXX
　ファイルが作成できません。

cannot open: XXX
　ソースファイルがオープンできません。

memory allocation fail
　メモリ割当てに失敗しました。

unknown flag 'X'
　コマンド行で指定されたオプションに誤りがあります。

write error
　ファイル書き込みに失敗しました。

warning: not supported directive 'XXX'
　擬似命令 XXX に対応する機能がＳＦＡに存在しません。

スーパーファミコン用アセンブラシステム　ユーザーマニュアル