

ADJ-702-138C

日立マイクロコンピュータ開発環境システム SH シリーズ クロスアセンブラ ユーザーズマニュアル

HS0700ASCU3S

発行年月日	平成 4 年 12 月 第 1 版 平成 8 年 6 月 第 4 版
発行	株式会社 日立製作所 電子統括営業本部
編集	株式会社日立マイコンシステム 技術情報センタ

©株式会社 日立製作所 1992

はじめに

本マニュアルは、「SuperH RISC engine アセンブラ」(以下、アセンブラと称します)について解説しています。

本アセンブラは、日立 SuperH RISC engine ファミリ(以下、マイコンと称します)のプログラムを開発するためのソフトウェアです。

本マニュアルの構成は、次のようになっています。

概説	アセンブラの機能概要を説明しています。
言語編	アセンブリ言語の文法とプログラミングの手順を説明しています。
操作編	アセンブラの起動方法とコマンドライン・オプションを説明しています。
付録	制限事項やエラーメッセージについて説明しています。

本アセンブラをご使用になる前に、本マニュアルをよくお読みになり、ご理解のうえご使用ください。また、関連するハードウェア、ソフトウェアについても該当マニュアルをよくお読みになり、ご理解のうえご使用ください。

備考

本マニュアル内で、次の表記は特別の意味をもちます。

項目	指定項目
	空白またはタブ
%	プロンプト（入力待ち表示）
(RET)	リターン入力
...	繰り返し可能
[]	省略可能

本マニュアル内で、数値は次のように表しています。

2進数	数値の前に、B' を記述
8進数	数値の前に、Q' を記述
10進数	数値の前に、D' を記述
16進数	数値の前に、H' を記述

特に断らない場合、前になにもつかない数値は10進数です。

Windows®95 および Windows®NT は、米国 Microsoft Corporation の登録商標です。

UNIX は、X/Open 社により管理されているオペレーティングシステムの名称です。

SPARC は、米国 SPARC International 社の登録商標です。

HP9000/700 シリーズは、米国 Hewlett Packard 社の商標です。

PC-9800 シリーズは、日本電気株式会社の商標です。

IBM PC は、米国 IBM 社の商標です。

目次

概説

1. アセンブラの機能概要	3
2. ソフトウェア開発サポートツールの相互関係	7

言語編

1. プログラムの要素	13
1.1 ソースステートメント	15
1.1.1 ソースステートメントの構成	15
1.1.2 ソースステートメントの書き方	16
1.1.3 複数行にわたるソースステートメントの書き方	18
1.2 予約語	19
1.3 シンボル	20
1.3.1 シンボルの役割	20
1.3.2 シンボルの名づけ方	22
1.4 定数	23
1.4.1 整数定数	23
1.4.2 文字定数	24
1.4.3 浮動小数点定数	25
1.4.4 固定小数点定数	31
1.5 ロケーションカウンタ	34
1.6 式	36
1.6.1 式の要素	36
1.6.2 演算の順序	38
1.6.3 演算の詳細	41
1.6.4 式に関する注意事項	44
1.7 文字列	45

1.8	ローカルラベル	46
1.8.1	ローカルラベルの役割	46
1.8.2	ローカルラベルの名付け方	47
1.8.3	ローカルラベルの有効範囲	47
2.	プログラミングの基礎知識	49
2.1	セクション	51
2.1.1	セクションの用途別の種類	51
2.1.2	絶対アドレスセクションと相対アドレスセクション	55
2.2	絶対値と相対値	57
2.2.1	絶対値	57
2.2.2	相対値	57
2.3	シンボルの定義と参照	58
2.3.1	シンボルの定義	58
2.3.2	シンボルの参照	60
2.4	分割プログラミング	62
2.4.1	分割プログラミングとは	62
2.4.2	外部定義シンボルと外部参照シンボルの宣言	64
3.	実行命令	65
3.1	実行命令の概要	67
3.2	実行命令に関する注意事項	74
3.2.1	オペレーションサイズに関する注意	74
3.2.2	遅延分岐命令に関する注意	91
3.2.3	アドレス計算に関する注意	93
4.	DSP 命令	97
4.1	プログラムの要素	99
4.1.1	ソースステートメント	99
4.1.2	並列演算命令	100
4.1.3	データ転送命令	102
4.1.4	複数行にわたるソースステートメントの書き方	103
4.2	DSP 命令	104
4.2.1	DSP 演算命令	104
4.2.2	データ転送命令	108
5.	アセンブラ制御命令	113
5.1	アセンブラ制御命令の概要	115
5.2	アセンブラ制御命令リファレンス	117
5.2.1	CPU に関するアセンブラ制御命令	117
	CPU CPU 種別を指定する	118
5.2.2	セクションまたはロケーションカウンタに関するアセンブラ制御命令	120

	.SECTION セクションを宣言する.....	121
	.ORG ロケーションカウンタ値を設定する.....	125
	.ALIGN ロケーションカウンタ値を補正する.....	127
5.2.3	シンボルに関するアセンブラ制御命令.....	129
	.EQU シンボルに値を設定する（再設定は不可能）.....	130
	.ASSIGN シンボルに値を設定する（再設定が可能）.....	131
	.REG レジスタ別名を定義する.....	133
	.FREG 浮動小数点レジスタ名を定義する.....	135
5.2.4	データまたはデータ領域を確保するアセンブラ制御命令.....	137
	.DATA 整数データを確保する.....	138
	.DATAB 整数データブロックを確保する.....	140
	.SDATA 文字列データを確保する.....	143
	.SDATAB 文字列データブロックを確保する.....	145
	.SDATAC 計数付き文字列データを確保する.....	147
	.SDATAZ ゼロ終端文字列データを確保する.....	149
	.FDATA 浮動小数点データを確保する.....	151
	.FDATAB 浮動小数点データブロックを確保する.....	153
	.XDATA 固定小数点データを確保する.....	155
	.RES データ領域を確保する.....	157
	.SRES 文字列データ領域を確保する.....	159
	.SRESC 計数付き文字列データ領域を確保する.....	161
	.SRESZ ゼロ終端文字列データ領域を確保する.....	163
	.FRES 浮動小数点データ領域を確保する.....	165
5.2.5	外部定義または外部参照に関するアセンブラ制御命令.....	167
	.EXPORT 外部定義シンボルを宣言する.....	168
	.IMPORT 外部参照シンボルを宣言する.....	170
	.GLOBAL 外部定義シンボルまたは外部参照シンボルを宣言する.....	172
5.2.6	オブジェクトモジュールに関するアセンブラ制御命令.....	174
	.OUTPUT オブジェクトモジュールの出力を制御する.....	175
	.DEBUG シンボルデバッグ情報の部分出力を制御する.....	178
	.ENDIAN エンディアン種別を指定する.....	180
	.LINE 行番号を変更する.....	183
5.2.7	アセンブルリストに関するアセンブラ制御命令.....	185
	.PRINT アセンブルリストの出力を制御する.....	186
	.LIST ソースプログラム・リストの部分出力を制御する.....	188
	.FORM アセンブルリストの行数と桁数を設定する.....	192
	.HEADING ソースプログラム・リストのヘッダを設定する.....	194
	.PAGE ソースプログラム・リストを改ページする.....	196

.SPACE	ソースプログラム・リストに空行を出力する	198
5.2.8	その他のアセンブラ制御命令	200
PROGRAM	オブジェクトモジュール名を設定する	201
.RADIX	基数のない整数定数を何進数とするかを指定する	203
.END	ソースプログラムの終わりと実行開始アドレスを宣言する	205
6.	ファイルインクルード機能	207
.INCLUDE	指定のインクルードファイルを取り込む	210
7.	条件つきアセンブリ機能	213
7.1	条件つきアセンブリ機能の概要	215
7.1.1	プリプロセッサ変数	215
7.1.2	置換シンボル	216
7.1.3	条件つきアセンブル	216
7.1.4	繰り返し展開	219
7.1.5	条件つき繰り返し展開	220
7.2	条件つきアセンブリ機能に関する制御文	221
.ASSIGNA	整数型プリプロセッサ変数を定義する（再定義が可能）	222
.ASSIGNC	文字型プリプロセッサ変数を定義する（再定義が可能）	225
.DEFINE	プリプロセッサ置換文字列を定義する	227
.AIF, .AELIF, .AELSE, .AENDI	比較型条件つきアセンブル	229
.AIFDEF, .AELSE, .AENDI	定義型条件つきアセンブル	231
.AREPEAT, .AENDR	繰り返し展開	233
.AWHILE, .AENDW	条件つき繰り返し展開	235
.AERROR	プリプロセッサ展開時のエラー処理をする	237
.EXITM	展開の中断終了	239
.ALIMIT	プリプロセッサでの.AWHILE の展開の上限値を設定する	241
8.	マクロ機能	243
8.1	マクロ機能の概要	245
8.2	マクロ機能に関する制御文	248
.MACRO, .ENDM	マクロ命令を定義する	249
.EXITM	マクロ展開の中断終了	252
8.3	マクロ本体	253
8.4	マクロコール	257
8.5	文字列操作関数	259
.LEN	文字列の長さを返す文字列操作関数	260
.INSTR	文字列の検索を行なう文字列操作関数	261
.SUBSTR	文字列の切り出しを行なう文字列操作関数	262
9.	リテラルプール自動生成機能	263
9.1	リテラルプール自動生成機能の概要	265

9.2	リテラルプール自動生成機能に関する拡張命令.....	266
9.3	リテラルプール自動生成機能のサイズモード.....	267
9.4	リテラルプールの出力.....	269
9.4.1	無条件分岐を利用したリテラルプール出力.....	270
9.4.2	.POOL の位置へのリテラルプール出力.....	271
9.5	リテラルの共有.....	272
9.6	リテラルプール出力の抑止.....	273
9.7	リテラルプール自動生成に関する注意事項.....	274
10.	リピートループ命令自動生成機能.....	277
10.1	リピートループ命令自動生成機能の概要.....	279
10.2	リピートループ命令自動生成機能に関する拡張命令.....	280
10.3	REPEAT の記述方法.....	280
10.4	コーディング例.....	281
10.5	REPEAT 拡張命令に関する注意事項.....	284

操作編

1.	起動.....	289
1.1	コマンドラインの形式.....	291
1.2	ファイルの指定形式.....	292
1.3	SHCPU 環境変数.....	293
2.	コマンドライン・オプション.....	295
2.1	コマンドライン・オプションの概要.....	297
2.2	コマンドライン・オプション リファレンス.....	299
2.2.1	CPU に関するコマンドライン・オプション.....	299
	-CPU CPU の種別を指定する.....	300
2.2.2	オブジェクトモジュールに関するコマンドライン・オプション.....	301
	-OBJECT, -NOBJECT オブジェクトモジュールの出力を制御する.....	302
	-DEBUG, -NODEBUG デバッグ情報の出力を制御する.....	303
	-ENDIAN エンディアン種別を指定する.....	304
2.2.3	アセンブルリストに関するコマンドライン・オプション.....	305
	-LIST, -NOLIST アセンブルリストの出力を制御する.....	306
	-SOURCE, -NOSOURCE ソースプログラム・リストの出力を制御する.....	307
	-CROSS_REFERENCE, -NOCROSS_REFERENCE クロスリファレンス・リストの出力を制御する.....	308
	-SECTION, -NOSECTION セクション情報リストの出力を制御する.....	309
	-SHOW, -NOSHOW ソースプログラム・リストの部分出力を制御する.....	310
	-LINES アセンブルリストの行数を設定する.....	312

-COLUMNS	アセンブルリストの桁数を設定する.....	313
2.2.4	ファイルインクルード機能に関するコマンドライン・オプション..	314
-INCLUDE	インクルードファイルの取り込み先を指定する.....	315
2.2.5	条件つきアセンブリ機能に関するコマンドライン・オプション.....	316
-ASSIGNA	整数型のプリプロセッサ変数を定義する.....	317
-ASSIGNC	文字型のプリプロセッサ変数を定義する.....	318
-DEFINE	文字列の置き換えを定義する.....	320
2.2.6	アセンブラの実行に関するコマンドライン・オプション.....	321
-EXPAND	プリプロセッサの展開結果を出力する.....	322
-ABORT	アセンブラが異常終了するエラーのレベルを変更する....	323
2.2.7	ソースファイル内の日本語記述に関するコマンドライン・オプション.....	324
-SJIS	ソースファイル内の漢字コードを シフト JIS コードとして解釈する.....	325
-EUC	ソースファイル内の漢字コードを EUC コードとして解釈する.....	326
-OUTCODE	オブジェクトファイルへ出力する漢字コードを設定する....	327
2.2.8	リテラルプール自動生成機能に関するコマンドライン・オプション.....	328
-AUTO_LITERAL	リテラルプール自動生成機能の サイズモードを設定する.....	329
2.2.9	コマンドラインの指定方法に関するコマンドライン・オプション..	330
-SUBCOMMAND	コマンドラインをファイルから入力する.....	331
2.2.10	浮動小数点データに関するコマンドライン・オプション.....	332
-ROUND	浮動小数点データの丸め方法を指定する.....	333
-DENORMALIZE	浮動小数点データの非正規化数の扱いを指定する.....	334

付録

付録 A. プログラム作成に関する制限と注意事項.....	337
付録 B. サンプルプログラム.....	338
B.1 サンプルプログラム仕様.....	338
B.2 コーディング例.....	340
付録 C. アセンブルリスト出力例.....	342
C.1 ソースプログラム・リスト.....	343
C.2 クロスリファレンス・リスト.....	345
C.3 セクション情報リスト.....	346
付録 D. エラーメッセージ.....	347
D.1 エラーの種類.....	347
D.2 エラーメッセージ一覧.....	350
付録 E. 旧バージョンとの相違.....	370
E.1 CPU.....	370
E.2 オブジェクトフォーマット.....	370
E.3 定数.....	371
E.4 アセンブラ制御命令の変更.....	371
E.5 コマンドライン・オプションの追加.....	371
付録 F. ASCII コード表.....	372

図表目次

概説

図 1.1	アセンブラの機能概要	5
図 2.1	ソフトウェア開発サポートツールの相互関係	9

言語編

図 2.1	コモンセクションのメモリへの配置	52
図 2.2	データ構造の記述例（ダミーセクションの使用例）	54
図 2.3	「前方」「後方」の意味	60
図 2.4	「外部」の意味	60
図 2.5	ソースプログラムの修正範囲と再アセンブルすべき範囲の関係	62
図 3.1	アドレス計算の例（通常）	93
図 3.2	アドレス計算の例（分岐によって PC が変化する場合）	94
図 3.3	アドレス計算の例（マイコンが PC を補正する場合）	95
図 3.4	アドレス計算の例（マイコンが PC を補正しない場合）	95
表 1.1	浮動小数点値のデータタイプ	26
表 1.2	データ形式と数値範囲（絶対値）	27
表 1.3	データ形式のサイズ	28
表 1.4	データタイプの浮動小数点表現	29
表 1.5	演算子一覧	37
表 1.6	演算子の優先順位と結合規則	39
表 1.7	偶奇演算子の演算内容	43
表 3.1	アドレス形式一覧	68
表 3.2	ディスプレイメントとして許される値	70
表 3.3	イミディエイトとして許される値	72
表 3.4	実行命令とオペレーションサイズの組み合わせ（SH-1）	74
表 3.5	実行命令とオペレーションサイズの組み合わせ（SH-2）	80
表 3.6	実行命令とオペレーションサイズの組み合わせ（SH-2E）	81
表 3.7	実行命令とオペレーションサイズの組み合わせ（SH-3）	82
表 3.8	実行命令とオペレーションサイズの組み合わせ（SH-3E）	84
表 3.9	実行命令とオペレーションサイズの組み合わせ（SH-4）	86
表 3.10	実行命令とオペレーションサイズの組み合わせ（SH-DSP、SH3-DSP）	89
表 3.11	遅延分岐命令とディレイスロット命令の関係	91

表 4.1	DSP 演算命令ニーモニックの分類.....	104
表 4.2	DSP 演算命令のアドレス形式.....	104
表 4.3	DSP レジスタ直接に指定可能なレジスタ.....	105
表 4.4	イミディエイトデータの値の範囲.....	105
表 4.5	DSP 演算命令一覧.....	107
表 4.6	データ転送命令のニーモニックの一覧.....	108
表 4.7	データ転送命令のアドレス形式.....	109
表 4.8	データ転送命令のアドレス形式に指定可能なレジスタ.....	110
表 4.9	データ転送命令一覧.....	111
表 5.1	アセンブラ制御命令一覧.....	115
表 9.1	拡張命令の種類と展開結果.....	266
表 9.2	データ転送命令とサイズモードの関係.....	267
表 9.3	サイズ選択モードで選択される命令.....	268
表 10.1	リピートループ間の命令数とアドレスの設定値.....	279
表 10.2	拡張命令の種類と展開結果.....	280

操作編

表 2.1	コマンドライン・オプション一覧.....	297
-------	----------------------	-----

付録

図 C.1	ソースプログラム・リスト出力例.....	343
図 C.2	クロスリファレンス・リスト出力例.....	345
図 C.3	セクション情報リスト出力例.....	346
表 A.1	プログラム作成に関する制限と注意事項.....	337
表 D.1	起動時のエラーに対するメッセージ.....	350
表 D.2	ソースプログラムのエラーに対するメッセージ.....	351
表 D.3	フェイタルエラーに対するメッセージ.....	368
表 E.1	制御命令の変更.....	371
表 E.2	コマンドライン・オプションの追加.....	371
表 F.1	ASCII コード表.....	372

概説

目次

1.	アセンブラの機能概要.....	3
2.	ソフトウェア開発サポートツールの相互関係.....	7

1. アセンブラの機能概要

アセンブラは、アセンブリ言語で書かれたソースプログラムをマイコンが理解できる形式に変換し、オブジェクトモジュールとして出力します。また、アセンブル処理の結果を、アセンブルリストとして出力します。

アセンブラは次のような機能を持ち、効率の良いプログラム開発を支援します。

(1) アセンブラ制御命令

アセンブラに各種の指示を与えます。

(2) ファイルインクルード機能

ソースファイルの中に他のファイルを取り込みます。

(3) 条件つきアセンブリ機能

条件によってアセンブル内容を切り替えたり、繰り返しアセンブルしたりします。

(4) マクロ機能

一連の処理に名前をつけ、1つの命令として定義します。

(5) リテラルプール自動生成機能

SuperH RISC engine ファミリにはないデータ転送命令 (MOV.W #imm、MOV.L #imm、MOVA #imm) を拡張命令として解釈し、マイコンの実行命令と定数データ (リテラル) に展開します。

(6) リピートループ命令自動生成機能

SH-DSP の固有動作にリピートループがあります。SH-DSP にはない命令 REPEAT を拡張命令として解釈し、SH-DSP のリピートループ命令 (LDRS、LDRE、SETRC) に展開します。

図 1.1 に、アセンブラの機能概要を示します。

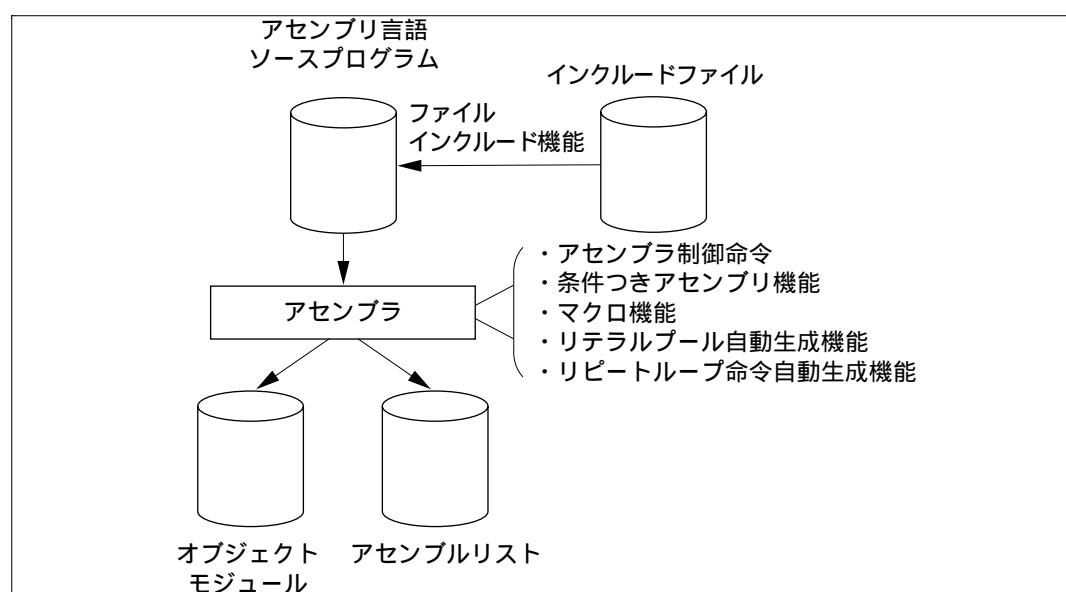


図 1.1 アセンブラの機能概要

2. ソフトウェア開発 サポートツールの相互関係

SuperH RISC engine ファミリのソフトウェア開発サポートツールには、アセンブラの他に C/C++ コンパイラ、リンケージエディタ、ライブラリアン、オブジェクトコンバータ、シミュレータ・デバッガがあります。

これらのツールは、ソフトウェア開発を効率的に進める手助けをします。

図 2.1 に、ソフトウェア開発サポートツールの相互関係を示します。

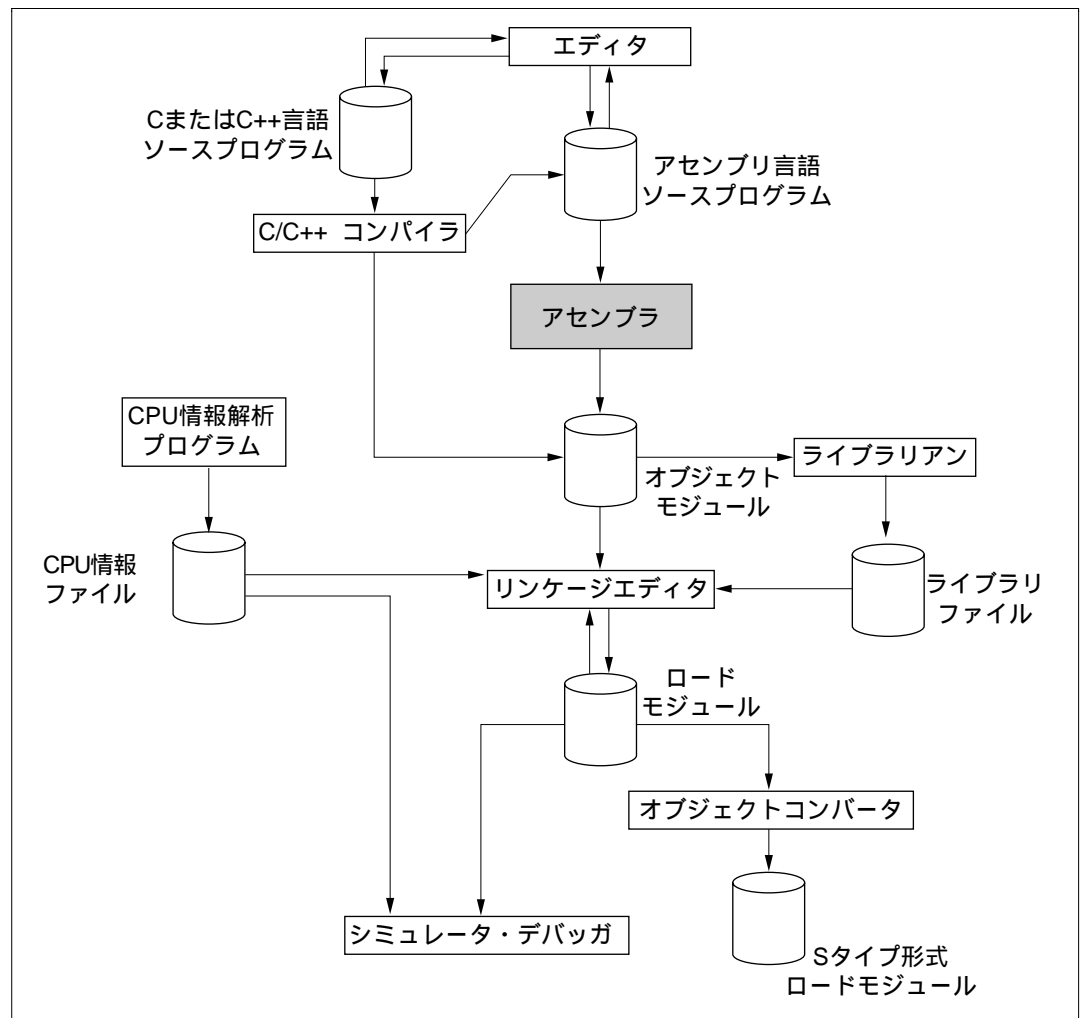


図 2.1 ソフトウェア開発サポートツールの相互関係

【補足】

- (1) ソースプログラムの編集には、汎用のエディタを使ってください。
- (2) C/C++コンパイラは、C または C++言語で書かれたソースプログラムを変換し、オブジェクトモジュールまたはアセンブリ言語ソースプログラムを出力します。

- (3) ライブラリアンは、複数のオブジェクトモジュールまたはリロケータブルロードモジュールをまとめてライブラリファイルを作成します。多数のプログラムに共通する処理は、ライブラリファイル化することをおすすめします（モジュールを容易に管理できるなどの利点があります）。
- (4) リンケージエディタは、オブジェクトモジュールやライブラリファイルをリンケージ（結合）して、ロードモジュール（実行形式のプログラム）を出力します。
- (5) オブジェクトコンバータは、ロードモジュールを S タイプ形式（ロードモジュールの標準的な形式）に変換します。
- (6) シミュレータ・デバッガは、マイコン用ソフトウェアのデバッグを手助けします。
- (7) できあがったロードモジュールは、各種のエミュレータ（マイコンシステムのハードウェアとソフトウェアをデバッグする装置）へ入力できます。
また、S タイプ形式ロードモジュールは、各種の EPROM ライタ（メモリヘデータを書き込む装置）へ入力できます。

言語編

目 次

1.	プログラムの要素.....	13
2.	プログラミングの基礎知識	49
3.	実行命令	65
4.	DSP 命令.....	97
5.	アセンブラ制御命令	113
6.	ファイルインクルード機能	207
7.	条件つきアセンブリ機能.....	213
8.	マクロ機能.....	243
9.	リテラルプール自動生成機能	263
10.	リピートループ命令自動生成機能	277

1. プログラムの要素

1.1 ソースステートメント

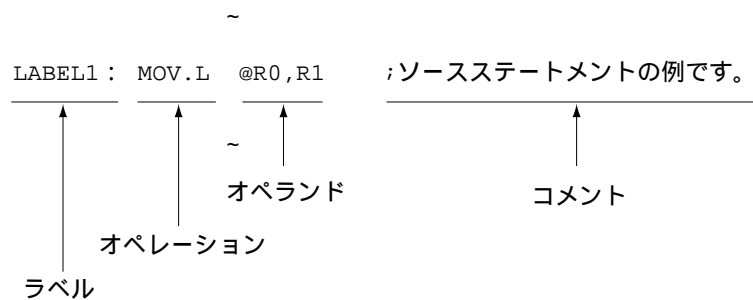
ソースプログラムを「文章」に例えると、1つの「文」にあたるのがソースステートメントです。そのソースステートメントを構成する「単語」として、予約語やシンボルなどがあります。

1.1.1 ソースステートメントの構成

ソースステートメントの構成を、以下に示します。

[ラベル] [オペレーション [オペランド]] [コメント]

コーディング例



(1) ラベル

ソースステートメントにつける名札として、シンボルまたはローカルシンボルを書きます。

シンボルとは、プログラマが定義する名前です。

(2) オペレーション

実行命令、拡張命令、DSP 命令、アセンブラ制御命令、各種制御文のニーモニック（綴り）を書きます。

実行命令とは、マイコンの命令です。

拡張命令とは、実行命令と定数データ（リテラル）または複数の実行命令に展開される命令です（言語編「9. リテラルプール自動生成機能」および「10. リピートループ命令自動生成機能」で詳しく説明しています）。

DSP 命令とは、SH-DSP マイコンの DSP を制御する命令です（言語編「4. DSP 命令」で詳しく説明しています）。

アセンブラ制御命令とは、アセンブラに指示を与える命令です。

制御文には、ファイルインクルード機能に関するもの、条件つきアセンブル機能に関するもの、マクロ機能に関するものがあります（各々の制御文については、言語編「6. ファイルインクルード機能」、「7. 条件つきアセンブリ機能」、「8. マクロ機能」で詳しく説明しています）。

（3）オペランド

オペレーションの実行対象となるものを書きます。

オペランドの個数と種類は、オペレーションによって決まります。オペランドを必要としないオペレーションもあります。

（4）コメント

プログラムを分かりやすくするための注釈を書きます。

1.1.2 ソースステートメントの書き方

ソースステートメントは、ASCII 文字で記述します。ただし、文字列またはコメントの中にはかな・漢字（シフト JIS コード、EUC コード）を記述できます。

基本的に、1 つのソースステートメントは 1 行に納まるように書いてください。1 行の最大長は、255 バイトです。

（1）ラベルの書き方

ラベルは、次のように書きます。

- ・ 1 カラム目から書き始める。

または

- ・ ラベル名の直後にコロン（:）をつける。

良い例

```
LABEL1          ; 1 カラム目から書き始めています。  
LABEL2:         ; コロン（:）で終わっています。
```

悪い例

```
LABEL3         ; 1 カラム目から書き始めず、コロン（:）をつけてもいけないので、  
                ; アセンブラはラベルと見なしません。
```

(2) オペレーションの書き方

オペレーションは、次のように書きます。

- ・ラベルを記述していない場合
2 カラム目以降から書き始める。
- ・ラベルを記述している場合
ラベルとの間に、1 つ以上の空白またはタブを置いて書き始める。

コーディング例

```
ADD R0,R1 ; ラベルを記述していない場合です。  
LABEL1: ADD R1,R2 ; ラベルを記述してある場合です。
```

【注意】 空白もタブも ASCII 文字ですので、1 バイト分の領域を必要とします。

(3) オペランドの書き方

オペランドは、オペレーションとの間に 1 つ以上の空白またはタブを置いて書き始めます。

コーディング例

```
ADD R0,R1 ; ADD 命令のオペランドは、2 つです。  
SHAL R1 ; SHAL 命令のオペランドは、1 つです。
```

(4) コメントの書き方

セミコロン (;) の後に続けて書きます。

アセンブラは、セミコロンから行末までをコメントと見なします。

コーディング例

```
ADD R0,R1 ; R1 に R0 を加えます。
```

1.1.3 複数行にわたるソースステートメントの書き方

次のようなときには、プログラムを見やすくするため、1つのソースステートメントを複数の行に分けて書くことができます。

- ・ソースステートメントが長くなる場合
- ・オペランドの1つ1つにコメントをつけたい場合

複数行にわたるソースステートメントは、次の(a)～(c)の手順で記述してください。

(a) オペランドとオペランドを区切るカンマ(,)を切れ目として改行します。

(b) すぐ次の行の1カラム目にプラス(+)を書きます。

(c) そのプラスの後ろに、ソースステートメントの続きを書きます。

プラスの後ろに、空白またはタブを入れても構いません。

コーディング例1

```
.DATA.L      H'FFFF0000,
+            H'FF00FF00,
+            H'FFFFFFFF
; 1つのソースステートメントを3行にわたって書いた例です。
```

各行の最後に、コメントを書くことができます。

コーディング例2

```
.DATA.L      H'FFFF0000,    ; 初期値 1
+            H'FF00FF00,    ; 初期値 2
+            H'FFFFFFFF     ; 初期値 3
; オペランド1つ1つに、コメントをつけた例です。
```

1.2 予約語

予約語は、特別な意味を持つ語としてアセンブラが用意している名前です。

予約語の種類には、レジスタ名、演算子、ロケーションカウンタがあります。レジスタ名はCPU 種別ごとに異なりますので、「プログラミングマニュアル」を参照してください。

予約語は、シンボルとしては使用できません。

・レジスタ名

R0 ~ R15、FR0 ~ FR15、DR0 ~ DR14 (偶数値のみ)、XD0 ~ XD14 (偶数値のみ)、FV0 ~ FV12 (4 の倍数値のみ)、R0_BANK ~ R7_BANK、SP*、SR、GBR、VBR、MACH、MACL、PR、PC、SSR、SPC、FPUL、FPSCR、MOD、RE、RS、DSR、A0、A0G、A1、A1G、M0、M1、X0、X1、Y0、Y1、XMTRX、DBR、SGR

・演算子

STARTOF、SIZEOF、HIGH、LOW、HWORD、LWORD、\$EVEN、\$ODD、\$EVEN2、\$ODD2

・ロケーションカウンタ

\$

【注】 * R15 と SP は、同じレジスタを表します。

【参照】 演算子	言語編「1.6.1 式の要素」
ロケーションカウンタ	言語編「1.5 ロケーションカウンタ」
CPU 種別	言語編「5.2.1 CPU に関するアセンブラ制御命令」.CPU 操作編「1.3 SHCPU 環境変数」 操作編「2.2.1 CPU に関するコマンドライン・オプション」.CPU
シンボル	言語編「1.3 シンボル」

1.3 シンボル

1.3.1 シンボルの役割

シンボルは、プログラマが定義する名前であり、次の役割を果たします。

- ・アドレスシンボル：データの格納場所、分岐先などのアドレスを表します。
- ・定数シンボル：定数を表します。
- ・レジスタ別名：汎用レジスタおよび浮動小数点レジスタを表します。
- ・セクション名：セクション*の名前を表します。

【注】 * セクションとは、プログラムの1区切りであり、リンケージ処理を実行する単位です。

シンボルの使用例を、以下に示します。

コーディング例 1

```
~  
BRA SUB1 ;BRA は、分岐命令です。  
;SUB1 は、分岐先のアドレスシンボルを表します。  
~  
SUB1:  
~
```

コーディング例 2

```
~  
MAX: .EQU 100 ;.EQU は、シンボルに値を設定するアセンブラ制御命令  
;です。  
MOV.B #MAX,R0 ;MAX は、値 100 を表します。  
~
```

コーディング例 3

```
~  
MIN: .REG R0 ;.REG は、レジスタ別名を定義するアセンブラ制御命令  
;です。  
MOV.B #100,MIN ;MIN は、R0 の別名になります。  
~
```

コーディング例 4

~

```
.SECTION CD,CODE,ALIGN=4
```

; .SECTION は、セクションを宣言するアセンブラ制御命令です。

; CD は、このセクションの名前となります。

~

1.3.2 シンボルの名づけ方

(1) シンボルに使用できる文字

次の ASCII 文字を使用できます。

- ・英大文字、英小文字 (A ~ Z, a ~ z)
- ・数字 (0 ~ 9)
- ・アンダースコア (_)
- ・ドル (\$)

アセンブラは、シンボル中の英大文字と英小文字を区別します。

(2) 先頭の文字

次のいずれかに限ります。

- ・英大文字、英小文字 (A ~ Z, a ~ z)
- ・アンダースコア (_)
- ・ドル (\$)

【注意】 ドル (\$) 1 文字は、ロケーションカウンタを表す予約語です。

【参照】 予約語 言語編「1.2 予約語」

(3) 最大文字数

251 文字です。

アセンブラは、251 文字を超える部分を無視します。

(4) シンボルとして使用できない名前

予約語は、シンボルとして使用できません。また、以下に示すような名前は、アセンブラが内部シンボル*として使用します。プログラマが同じ名前を使うことはできません。

`_$nnnnn` (n は数字 (0 ~ 9) です)

【注】 * 内部シンボルとは、アセンブラの内部処理のため必要なシンボルです。内部シンボルは、アセンブルリストやオブジェクトモジュールには出力されません。

1.4 定数

1.4.1 整数定数

整数定数は、基数をつけて表現します。

基数とは、その整数定数が何進数であるかを示す記述です。

- ・ 2 進数 …… 基数 B' と、2 進表現の数値で記述
- ・ 8 進数 …… 基数 Q' と、8 進表現の数値で記述
- ・ 10 進数 …… 基数 D' と、10 進表現の数値で記述
- ・ 16 進数 …… 基数 H' と、16 進表現の数値で記述

アセンブラは、基数の英大文字と英小文字を区別しません。

基数と数値は、間を空けずに続けて書いてください。

コーディング例

```
.DATA.B B'10001000 ;  
.DATA.B Q'210      ;これらのソースステートメントは、全く同じ内容  
.DATA.B D'136      ;を表しています。  
.DATA.B H'88       ;
```

基数は省略しても構いません。基数のない整数定数は、(通常は)10進数として扱われます(.RADIX アセンブラ制御命令によって、何進数にするかを指定できます)。

【参照】 基数のない整数定数の解釈

言語編「5.2.8 その他のアセンブラ制御命令」.RADIX

【補足】

B' : BINARY (2 進) の意味です。

Q' : OCTAL (8 進) の意味です。O は数字のゼロと紛らわしいので Q を使います。

D' : DECIMAL (10 進) の意味です。

H' : HEXADECIMAL (16 進) の意味です。

1.4.2 文字定数

文字定数は、ASCII 文字のコードを定数と考えるものです。

4 文字以内の ASCII 文字を、ダブルコーテーション (") で囲んで記述してください。

文字定数には、次の ASCII 文字を使用できます。

ASCII コード H'09 (タブ)
 H'20 (空白) ~ H'7E (~)

コーディング例 1

```
.DATA.L "ABC"      ; .DATA.L H'00414243   同じ意味です。  
.DATA.W "AB"       ; .DATA.W H'4142       同じ意味です。  
.DATA.B "A"        ; .DATA.B H'41         同じ意味です。  
                                ; A の ASCII コード... H'41  
                                ; B の ASCII コード... H'42  
                                ; C の ASCII コード... H'43
```

また、シフト JIS コード、もしくは EUC コードのかな・漢字を記述することができます。
文字として、かな・漢字を記述した場合、必ずシフト JIS コードのときは SJIS コマンド
ライン・オプションを、また、EUC コードのときは EUC コマンドライン・オプションを
指定してください。ただし、ソースプログラム内でシフト JIS コードと EUC コードを混在
して使うことはできません。

定数を表す文字としてダブルコーテーションを使う場合は、ダブルコーテーションを 2
つ続けて書いてください。

コーディング例 2

```
.DATA.B "_"        ; ダブルコーテーション 1 文字の文字定数です。  
.DATA.L "漢字"    ; 漢字
```

【参照】 SJIS コマンドライン・オプション 操作編「2.2.7 -SJIS」
 EUC コマンドライン・オプション 操作編「2.2.7 -EUC」

1.4.3 浮動小数点定数

浮動小数点定数は、浮動小数点定数確保のアセンブラ制御命令のオペランドで指定することができます。

(1) 浮動小数点定数の書き方

浮動小数点定数の表記には、10進表記と16進表記の2種類があります。

(a) 10進表記

$$F' [\{\pm\}] \left\{ \begin{matrix} n [. m] \\ .m \end{matrix} \right\} [t [\{\pm\}] xx]$$

F' : 10進浮動小数点定数であることを示します。省略はできません。

$[\{\pm\}] \left\{ \begin{matrix} n [. m] \\ .m \end{matrix} \right\}$: n には整数部を10進で指定します。 m には小数部を10進で指定します。整数部と小数部の値は、どちらか一方を省略できます。また、 \pm を省略すると $+$ を仮定します。

t : 精度のコードを指定します。
次の2種類があります。
・S 単精度
・D 倍精度
省略すると、アセンブラ制御命令のオペレーションサイズに従います。

$[\{\pm\}]xx$: 指数部(10のべき乗)の値を10進で指定します。
省略すると0乗を仮定します。また、 \pm を省略すると $+$ を仮定します。

例

$$F' 0.5S - 2 = 0.5 \times 10^{-2} = 0.005 = H' 3BA3D70A$$

$$F' .123D3 = 0.123 \times 10^3 = 123 = H' 405EC00000000000$$

(b) 16 進表記

H' xxxx [.t]

- H' : 16 進であることを示します。省略はできません。
- xxxx : 浮動小数点定数のビットパターンを 16 進で指定します。データ長より指定が短い場合は右づめに確保し、左側は必要分の 0 をつめます。長い場合は指定の右側の有効長分のデータを確保します。
- t : 精度のコードを指定します。
- 次の 2 種類があります。
- ・ S 単精度
 - ・ D 倍精度
- 省略すると、アセンブラ制御命令のオペレーションサイズに従います。

この形式は、精度幅の 0 や無限大表示など、10 進表記の書き方では表記しにくいデータを記述するもので、確保すべき浮動小数点定数のビットパターンを直接指定します。

例

H' 0123456789ABCDEF.S H' 89ABCDEF
H' FFFF.D H' 000000000000FFFF

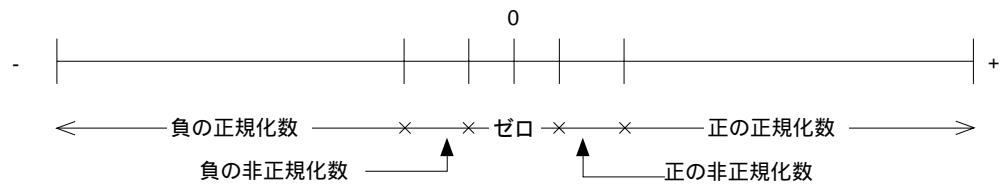
(2) 浮動小数点値の範囲

表 1.1 に、浮動小数点値のデータタイプを示します。

表 1.1 浮動小数点値のデータタイプ

データタイプ	内容
正規化数 (Normalized Number)	絶対値がアンダフロー境界以上でオーバーフロー境界以下の値
非正規化数 (Denormalized Number)	絶対値が 0 より大きくアンダフロー境界未満の値
ゼロ	絶対値が 0 の値
無限大	絶対値がオーバーフローより大きい値
非数 (Not a Number : NAN)	数値でない値 sNAN (signaling NAN) と qNAN (quiet NAN) があります

これらを数直線上に表すと次のようになります。ただし、非数は数値として扱わないため、数直線上には表せません。



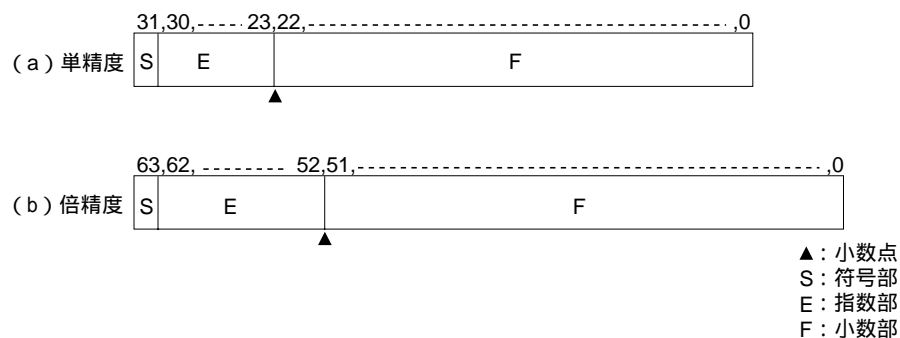
次にアセンブラで記述できる数値範囲を表 1.2 に示します。

表 1.2 データ形式と数値範囲（絶対値）

データ形式		単精度	倍精度
正規化数	最大値	3.40×10^{38}	1.79×10^{308}
	最小値	1.18×10^{-38}	2.23×10^{-308}
非正規化数	最大値	1.17×10^{-38}	2.22×10^{-308}
	最小値	1.40×10^{-45}	4.94×10^{-324}

(3) 浮動小数点データ形式

FPU の浮動小数点データ形式を示します。



- ・ 符号部 (S)

数値の符号を表現します。0 で正数、1 で負数です。

- ・ 指数部 (E)

指数を表現します。データフォーマット中の指数部の値からある一定のバイアス値 (bias) を引いた値が実際の指数になります。

- ・ 小数部 (F)

小数部は各ビットごとに重みをもっており、先頭ビットから 2^{-1} , 2^{-2} , \dots , 2^{-n} (n は小数部のビット長) に対応しています。

次にデータ形式のサイズを表 1.3 に示します。

表 1.3 データ形式のサイズ

パラメータ	単精度	倍精度
ビット長	32 ビット	64 ビット
符号ビット (S)	1 ビット	1 ビット
指数部 (E)	8 ビット	11 ビット
小数部 (F)	23 ビット	52 ビット
指数のバイアス (bias)	127	1023

浮動小数点の数値は、表 1.3 の記号を用いると、次のように表わせます。

$$2^{E - \text{bias}} \cdot (-1)^S \begin{cases} (1.F) & \text{正規化数} \\ (0.F) & \text{非正規化数} \end{cases}$$

$$(1.F) = 1 + b_0 \times 2^{-1} + b_1 \times 2^{-2} + \cdots + b_{n-1} \times 2^{-n} \quad (b: \text{小数部のビット位置、} \\ n: \text{小数部のビット長})$$

$$(0.F) = b_0 \times 2^{-1} + b_1 \times 2^{-2} + \cdots + b_{n-1} \times 2^{-n}$$

次にデータタイプごとの浮動小数点表現を示すと表 1.4 のようになります。非数は数値でないため表わせません。

表 1.4 データタイプの浮動小数点表現

データタイプ	単精度	倍精度
正規化数	$(-1)^S \cdot 2^{E-127} \cdot (1.F)$	$(-1)^S \cdot 2^{E-1023} \cdot (1.F)$
非正規化数	$(-1)^S \cdot 2^{-126} \cdot (0.F)$	$(-1)^S \cdot 2^{-1022} \cdot (0.F)$
ゼロ	$(-1)^S \cdot 0$	
無限大	$(-1)^S \cdot \infty$	
非数	quiet NaN , signaling NaN	

(4) 浮動小数点定数の有効数字

浮動小数点定数確保の制御命令において、本アセンブラがオブジェクトコードを生成するとき、以下 2 通りの丸め方をサポートし、有効数字を設定します。

- (a) Round to Nearest even (RN) : オブジェクトコード最下位ビットを最近値に丸めます。最近値が 2 つのとき、最下位ビットがゼロになるように丸めます。
- (b) Round to Zero (RZ) : オブジェクトコード最下位ビットを切り捨てて、0 にします。

例

.FDATA.S F' 1S - 1 のオブジェクトコード

RN の場合 : H' 3DCCCCCD

RZ の場合 : H' 3DCCCCC

【参照】 丸め方法の指定 操作編「2.2.10 浮動小数点データに関するコマンド
ライン・オプション」 - ROUND

(5) 非正規化数の扱い

非正規化数の扱いは、CPU 種別により異なります。非正規化数を扱わない CPU の場合、非正規化数の範囲の値は、ウォーニング 841 とし、0 のオブジェクトコードを出力します。非正規化数を扱う CPU の場合は、非正規化数の範囲の値は、ウォーニング 842 とし、非正規化数のオブジェクトコードを出力します。非正規化数の扱いは、コマンドライン・オプションにて変更することができます。

例

・非正規化数を扱わない CPU の場合

.FDATA.S F' 1S - 40 : ウォーニング 841、オブジェクトコード H' 00000000

・非正規化数を扱う CPU の場合

.FDATA.S F' 1S - 40 : ウォーニング 842、オブジェクトコード H' 000116C2

【参照】 非正規化数の扱い 操作編「2.2.10 浮動小数点データに関するコマンド
ライン・オプション」 - DENORMALIZE

1.4.4 固定小数点定数

固定小数点定数は、固定小数点定数確保のアセンブラ制御命令のオペランドで指定することができます。

(1) 固定小数点定数の書き方

固定小数点定数は、- 1.0 ~ 1.0 までの範囲の実数データが扱え、10 進数で記述します。
固定小数点定数には、ワードサイズと、ロングワードサイズの 2 種類があります。

(a) ワードサイズ固定小数点定数

2 バイトの符号付き整数で - 1.0 ~ 1.0 の範囲の実数を表現します。

2 バイトの符号付き整数の値 x (- 32,768 ≤ x ≤ 32,767) が表現する実数値は、 $x/32768$ となります。

例

固定小数点定数	ワードデータの表現
- 1.0	H'8000
- 0.5	H'C000
0.0	H'0000
0.5	H'4000
1.0	H'7FFF

(b) ロングワードサイズ固定小数点定数

4 バイトの符号付き整数で - 1.0 ~ 1.0 の範囲の実数を表現します。

4 バイトの符号付き整数の値 x (- 2,147,483,648 ≤ x ≤ 2,147,483,647) が表現する実数値は、 $x/2147483648$ となります。

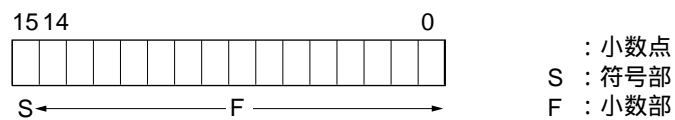
例

固定小数点定数	ロングワードデータの表現
- 1.0	H'80000000
- 0.5	H'C0000000
0.0	H'00000000
0.5	H'40000000
1.0	H'7FFFFFFF

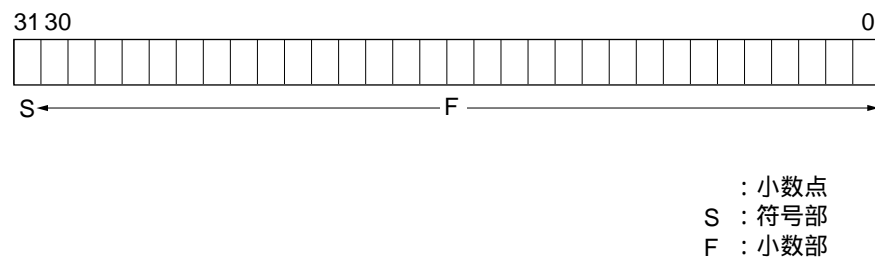
(2) 固定小数点データ形式

固定小数点定数のデータ形式は、ワードサイズの場合、符号 1 ビットと数値 15 ビット、ロングワードサイズの場合、符号 1 ビットと数値 31 ビットからなるもので、小数点は、常に符号部の右側にあるものと考えます。

(a) ワードサイズ



(b) ロングワードサイズ



・ 符号部 (S)

数値の符号を表します。0 で正数、1 で負数です。

・ 小数部 (F)

小数部は、各ビットごとに重みをもっており、先頭ビットから 2^{-1} , 2^{-2} , \dots , 2^{-31} に対応します。

(3) 固定小数点定数の有効数字

ロングワードサイズの場合、31 ビットで表現できる値は、10 進で 9 桁となりますが、アセンブラは、有効数字を 10 進 10 桁の小数で扱い、35 ビット目を RN (絶対値が 0 に近い方に丸める) で丸め、計算結果の上位 31 ビットを固定小数点データとします。

【注意】 実際の固定小数点データの範囲は、- 1.0 ~ 0.9999999999 ですが、1.0 は 0.9999999999 と仮定し、H'7FFFFFFF とします。

1.5 ロケーションカウンタ

ロケーションカウンタは、オブジェクトコード（実行命令やデータをコンピュータが理解できる形式に変換したコード）を配置するアドレス（ロケーション）を指し示します。

ロケーションカウンタの値（以下、ロケーションカウンタ値と称します）は、オブジェクトコードの出力に応じて自動的に変化します。

また、アセンブラ制御命令により、意図的にロケーションカウンタ値を変えることもできます。

コーディング例

```
~
.ORG      H'00001000 ;ロケーションカウンタに、H'00001000 を設定しています。
.DATA.W   H'FF       ;このアセンブラ制御命令のオブジェクトコードは、長さ 2 バイトです。
                        ;ロケーションカウンタ値は、H'00001002 に変わります。
.DATA.W   H'F0       ;このアセンブラ制御命令のオブジェクトコードは、長さ 2 バイトです。
                        ;ロケーションカウンタ値は、H'00001004 に変わります。
.DATA.W   H'10       ;このアセンブラ制御命令のオブジェクトコードは、長さ 2 バイトです。
                        ;ロケーションカウンタ値は、H'00001006 に変わります。
.ALIGN    4          ;ロケーションカウンタ値を 4 の倍数に補正しています。
                        ;ロケーションカウンタ値は、H'00001008 に変わります。
.DATA.L   H'FFFFFFFF ;このアセンブラ制御命令のオブジェクトコードは、長さ 4 バイトです。
                        ;ロケーションカウンタ値は、H'0000100C に変わります。

;      .ORG は、ロケーションカウンタ値を設定するアセンブラ制御命令です。
;      .ALIGN は、ロケーションカウンタ値を補正するアセンブラ制御命令です。
;      .DATA は、データをメモリ上に確保するアセンブラ制御命令です。
;      .W は、データをワード（=2 バイト）単位で扱うとの指定です。
;      .L は、データをロングワード（=4 バイト）単位で扱うとの指定です。
~
```

【参照】 ロケーションカウンタ値の設定

言語編「5.2.2 セクションまたはロケーションカウンタに関するアセンブラ制御命令」.ORG

ロケーションカウンタ値の補正

言語編「5.2.2 セクションまたはロケーションカウンタに関するアセンブラ制御命令」.ALIGN

ロケーションカウンタは、ドル (\$) で参照できます。

コーディング例

```
LABEL1: .EQU $           ;LABEL1 というシンボルに、この時点でのロケーション  
                           ;カウンタ値を設定しています。
```

; .EQU は、シンボルに値を設定するアセンブラ制御命令です。

1.6 式

式は、定数やシンボルと演算子を組み合わせて演算結果を求めるものであり、実行命令やアセンブラ制御命令のオペランドに使用します。

1.6.1 式の要素

式は、項、演算子、カッコから構成されます。

(1) 項

項には、次のものがあります。

- ・ 定数
- ・ ロケーションカウンタ (\$)
- ・ シンボル (レジスタ別名を除く)
- ・ 上記の項と演算子による演算結果

単独の項も、式の種類です。

(2) 演算子

表 1.5 に、演算子の一覧を示します。

表 1.5 演算子一覧

演算区分	演算子	演算内容	書き方
算術演算	+	単項プラス	+ 項
	-	単項マイナス	- 項
	+	加算	項 1 + 項 2
	-	減算	項 1 - 項 2
	*	乗算	項 1 * 項 2
	/	除算	項 1 / 項 2
論理演算	~ ^	単項否定	~ 項
	&	論理積	項 1 & 項 2
		論理和	項 1 項 2
	~ ^	排他的論理和	項 1 ~ 項 2
シフト演算	<<	算術左シフト	項 1 << 項 2
	>>	算術右シフト	項 1 >> 項 2
セクション集合演算*	STARTOF	セクション集合の先頭アドレスを求める	STARTOF セクション名
	SIZEOF	セクション集合のサイズをバイト単位で求める	SIZEOF セクション名
偶奇演算	\$EVEN	2 の倍数の時 1、そうでない時 0	\$EVEN シンボル
	\$ODD	2 の倍数の時 0、そうでない時 1	\$ODD シンボル
	\$EVEN2	4 の倍数の時 1、そうでない時 0	\$EVEN2 シンボル
	\$ODD2	4 の倍数の時 0、そうでない時 1	\$ODD2 シンボル
抽出演算	HIGH	上位バイト抽出	HIGH 項
	LOW	下位バイト抽出	LOW 項
	HWORD	上位ワード抽出	HWORD 項
	LWORD	下位ワード抽出	LWORD 項

【注】* 次ページ【補足】を参照

【補足】 本アセンブリ言語では、プログラムを「セクション」という単位に区切って考えます。

セクションは、リンケージ処理を実行する単位です。

プログラム中に同名で同種類のセクションが複数ある場合、リンケージエディタはそれらを1つの「セクション集合」に結合します。

【参照】 セクション 言語編「2.1 セクション」

(3) カッコ

丸カッコ()によって、演算の優先順位を変更できます。

カッコの使い方については、次項「1.6.2 演算の順序」で説明します。

1.6.2 演算の順序

1つの式の中に複数の演算が含まれる場合、演算子の優先順位とカッコ指定によって、演算を処理する順序が決まります。

アセンブラは、次の規則にしたがって演算を処理します。

・規則 1

カッコでくくられた演算から処理する。

カッコが多重になっているときは、より内側のカッコでくくられた演算を優先する。

・規則 2

演算子の優先順位が高いものから処理する。

・規則 3

演算子の優先順位が同じであるときは、演算子の結合規則の向きに処理する。

表 1.6 に、演算子の優先順位と結合規則を示します。

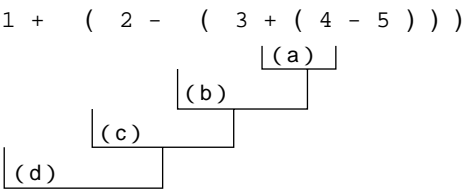
表 1.6 演算子の優先順位と結合規則

優先順位	演算子	結合規則
1 (高)	+ - ~ ^STARTOF SIZEOF \$EVEN \$ODD \$EVEN2 \$ODD2 HIGH LOW HWORD LWORD *	右から左の順に演算を処理する。
2	* /	左から右の順に演算を処理する。
3	+ -	左から右の順に演算を処理する。
4	<< >>	左から右の順に演算を処理する。
5	&	左から右の順に演算を処理する。
6 (低)	~ ^	左から右の順に演算を処理する。

【注】* 優先順位 1 の演算子は単項演算子です。

式の記述例を、以下に示します。

コーディング例 1

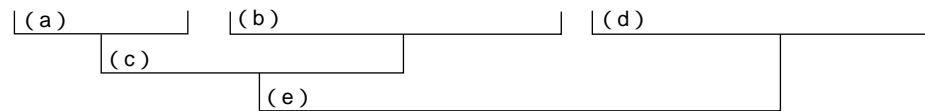


アセンブラは、(a) ~ (d) の順に計算します。

(a) の結果	- 1	} 最終的な結果は、1になります。
(b) の結果	2	
(c) の結果	0	
(d) の結果	1	

コーディング例2

- H'FFFFFFFF1 + H'000000F0 * H'00000010 | H'000000F0 & H'0000FFFF

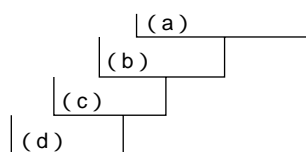


アセンブラは、(a) ~ (e) の順に計算します。

(a) の結果	H' 0000000F	} 最終的な結果は、H' 0000FFFFになります。
(b) の結果	H' 0000F00	
(c) の結果	H' 0000F0F	
(d) の結果	H' 000000F0	
(e) の結果	H' 0000FFF	

コーディング例3

- ~ - ~ H'0000000F



アセンブラは、(a) ~ (d) の順に計算します。

(a) の結果	H' FFFFFFFF0	} 最終的な結果は、H' 00000011になります。
(b) の結果	H' 00000010	
(c) の結果	H' FFFFFFFEF	
(d) の結果	H' 00000011	

1.6.3 演算の詳細

(1) STARTOF 演算

セクション集合の先頭アドレスを求めます。指定したセクションがリンカージェディタで連結された後のセクション先頭アドレスを求めます。

(2) SIZEOF 演算

セクション集合のサイズを求めます。指定したセクションがリンカージェディタで連結された後のセクションサイズを求めます。

コーディング例

```

        .CPU          SH1
        .SECTION      INIT__RAM, DATA, ALIGN=4
        .RES.B        H'100
        .SECTION      INIT__DATA, DATA, ALIGN=4
INIT__BGN .DATA.L      (STARTOF INIT__RAM)                ; [1]
INIT__END .DATA.L      (STARTOF INIT__RAM) + (SIZEOF INIT__RAM) ; [2]
;
;
        .SECTION      MAIN, CODE, ALIGN=4
INITIAL:
        MOV.L         DATA1, R6
        MOV           #0, R5
        MOV.L         DATA1+4, R3
        BRA           LOOP2
        MOV.L         @R3, R4
LOOP1:
        MOV.L         R5, @R4
        ADD           #4, R4
LOOP2:
        MOV.L         @R6, R3
        CMP/Hi        R3, R4
        BF            LOOP1
        RTS
        NOP
DATA1:
        .DATA.L       INIT__END
        .DATA.L       INIT__BGN
        .END

```

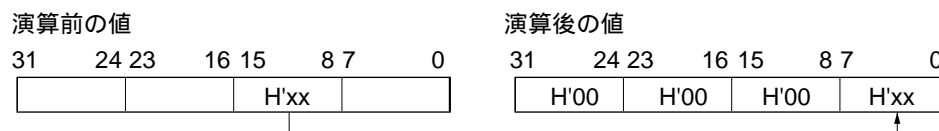
セクション (INIT_RAM) の
データ領域をゼロで初期化
します。

[1]: セクション (INIT_RAM) の先頭アドレスを求めます。

[2]: セクション (INIT_RAM) の最終アドレスを求めます。

(3) HIGH 演算

4 バイト値の下位 2 バイトの上位バイトを抽出します。



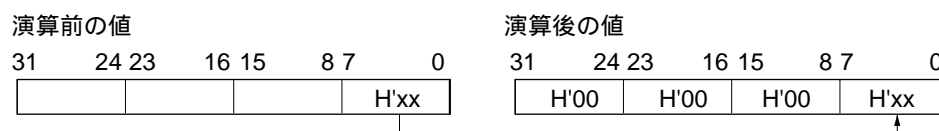
コーディング例

```

LABEL      .EQU      H'00007FFF
            .DATA      HIGH LABEL      ; メモリ上に整数データ
                                           ; H'0000007F を確保します。
  
```

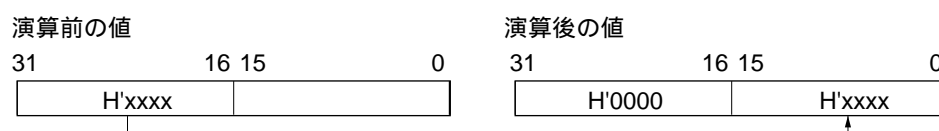
(4) LOW 演算

4 バイト値の下位バイトを抽出します。



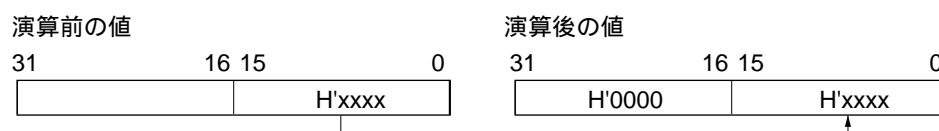
(5) HWORD 演算

4 バイト値の上位 2 バイトを抽出します。



(6) LWORD 演算

4 バイト値の下位 2 バイトを抽出します。



(7) 偶奇演算

アドレスシンボルの値が2の倍数または4の倍数かを判定します。

表 1.7 に、偶奇演算子の演算内容を示します。

表 1.7 偶奇演算子の演算内容

演算子	演算内容
\$EVEN	2 の倍数の時 1、そうでない時 0
\$ODD	2 の倍数の時 0、そうでない時 1
\$EVEN2	4 の倍数の時 1、そうでない時 0
\$ODD2	4 の倍数の時 0、そうでない時 1

コーディング例

\$ODD2 演算子を使用して現在のプログラムカウンタを求める例です。

LAB:

```
MOVA    @(0,PC),R0
ADD     #-4+2*$ODD2 LAB,R1    ;$ODD2 は、LAB が
                                ;4 の倍数ならば 0 とし、
                                ;4 の倍数でないなら 1 とします。
```

1.6.4 式に関する注意事項

(1) 内部処理

アセンブラは、式の値を 32 ビット符号付きで処理します。

コーディング例

```
~H'F0
```

アセンブラは H'F0 を H'000000F0 と解釈します。

したがって、~H'F0 は H'FFFFFFF0F であり、H'0000000F ではありません。

(2) 算術演算

アセンブラ値が確定しなければならない箇所では、乗算、除算で、相対値（リンケージを終えるまで確定しない値）を項とすることはできません。

コーディング例

```
.IMPORT    SYM
.DATA      SYM/10          ; 正常となる
.ORG       SYM/10          ; エラーとなる
```

また、除算で 0 を除数とすることはできません。

(3) 論理演算

論理演算で、相対値を項とすることはできません。

【参照】 相対値 言語編「2.2 絶対値と相対値」

1.7 文字列

文字列は、一連の文字をデータとして考えます。

文字列には、次の ASCII 文字を使用できます。

ASCII コード

- └ H'09 (タブ)
- └ H'20 (空白) ~ H'7E (~)

文字列中の 1 文字は、その文字の ASCII コードを値としてもつ、バイトサイズ of データを表します。また、シフト JIS コード、もしくは EUC コードのかな・漢字を記述することができます。

文字として、かな・漢字を記述した場合は、シフト JIS コードのときは SJIS コマンドライン・オプションを、また、EUC コードのときは EUC コマンドライン・オプションを指定してください。指定しない場合は、ホストマシンに依存する日本語コードとします。

文字列は、文字の並びをダブルコーテーション (") で囲んで記述してください。

データを表す文字としてダブルコーテーションを使う場合は、ダブルコーテーションを 2 つ続けて書いてください。

コーディング例

```
.SDATA    "Hello!"           ;文字列データ Hello!を確保しています。
.SDATA    "アセンブラ"       ;文字列データアセンブラを確保しています。
.SDATA    "\"Hello!\""       ;文字列データ"Hello!"を確保しています。
```

; .SDATA は、文字列データをメモリ上に確保するアセンブラ制御命令です。

【補足】 文字定数と文字列の違い

文字定数は、数値です。データのサイズは、1 バイト、2 バイト、4 バイトのいずれかになります。

文字列は、数値として扱えません。データのサイズは、1 バイト以上 255 バイト以下です。

【参照】	SJIS コマンドライン・オプション	操作編	「2.2.7 -SJIS」
	EUC コマンドライン・オプション	操作編	「2.2.7 -EUC」

1.8 ローカルラベル

1.8.1 ローカルラベルの役割

ローカルラベルは、アドレスシンボル間で局所的に有効なラベルです。ローカルラベルは、有効範囲外の他のシンボルと衝突することがありませんので、他のシンボル名を意識しないで、局所的な制御ができます。

ローカルラベルは、通常のアドレスシンボルと同様に、ラベルに記述することによって定義でき、オペランド内で参照できます。

ローカルラベルの使用例を、以下に示します。

コーディング例

```
LABEL1:                                ; ローカルブロック 1 の開始
?0001:
    ~
    CMP/EQ    R1,R2
    BT        ?0002    ; ローカルブロック 1 の?0002 に分岐します。
    BRA       ?0001    ; ローカルブロック 1 の?0001 に分岐します。
?0002:
    ~

LABEL2:                                ; ローカルブロック 2 の開始
?0001:
    ~
    CMP/GE    R1,R2
    BT        ?0002    ; ローカルブロック 2 の?0002 に分岐します。
    BRA       ?0001    ; ローカルブロック 2 の?0001 に分岐します。
?0002:
LABEL3:                                ; ローカルブロック 3 の開始
```

【注意】 ローカルラベルは、デバッグ時には参照できません。

ローカルラベルは、以下の位置には指定できません。

- ・マクロ名
- ・セクション名
- ・オブジェクトモジュール名
- ・.ASSIGNA, .ASSIGNC, .EQU, .ASSIGN, .REG, .DEFINE のラベル
- ・.EXPORT, .IMPORT, .GLOBAL のオペランド

1.8.2 ローカルラベルの名付け方

(1) 文字の先頭

ローカルラベルは、先頭が"?"で始まる文字列です。

(2) ローカルラベルに使用できる文字

ローカルラベルは、先頭以外の文字が以下の ASCII 文字からなる文字列です。

- ・英大文字、英小文字 (A~Z, a~z)
- ・数字 (0~9)
- ・アンダースコア (_)
- ・ドル (\$)

アセンブラは、英大文字と英小文字を区別しています。

(3) 最大文字数

2 文字以上 16 文字以内です。

17 文字以上記述するとローカルシンボルとして扱われません。

1.8.3 ローカルラベルの有効範囲

ローカルラベルの有効範囲をローカルブロックといいます。ローカルブロックの区切りは、アドレスシンボル、または .SECTION アセンブラ制御命令です。

このローカルブロック内で定義されたローカルラベルは、当該ローカルブロック内で参照できます。

異なるローカルブロックに属するローカルラベルは、同じ名前であっても別のラベルと解釈し、エラーとはなりません。

【注意】 .EQU アセンブラ制御命令または .ASSIGN アセンブラ制御命令で定義したアドレスシンボルは、有効範囲の区切りとはみなしません。

2. プログラミングの基礎知識

2.1 セクション

ソースプログラムを「文章」に例えると、1つの「章」にあたるものがセクションです。セクションは、リンケージエディタがオブジェクトモジュールをリンケージするときの処理単位です。

2.1.1 セクションの用途別の種類

セクションには、用途に応じて次の種類があります。

- ・コードセクション
- ・データセクション
- ・コモンセクション
- ・スタックセクション
- ・ダミーセクション

(1) コードセクション

コードセクションには、次のものを記述してください。

- ・実行命令
- ・拡張命令
- ・初期値をもつデータを確保するアセンブラ制御命令

コーディング例

```
.SECTION    CD, CODE, ALIGN=4    ;CD という名前のコードセクションを
                                   ;宣言しています。

MOV.L      X, R1                  ;実行命令です。
MOV        R1, R2

~

.ALIGN     4

X: .DATA.L  H'FFFFFFFF            ;初期値をもつデータを確保しています。

~
```

(2) データセクション

データセクションには、次のものを記述してください。

- ・初期値をもつデータを確保するアセンブラ制御命令
- ・初期値をもたないデータ領域を確保するアセンブラ制御命令

コーディング例1

```
.SECTION    DT1, DATA, ALIGN=4 ;DT1 という名前のデータセクション
                                ;を宣言しています。

.DATA.W     H'FF00              ;初期値をもつデータを確保しています。
.DATA.B     H'FF
~
```

コーディング例2

```
.SECTION    DT2, DATA, ALIGN=4 ;DT2 という名前のデータセクション
                                ;を宣言しています。

.RES.W      10                  ;初期値をもたないデータ領域を確保し
.RES.B      10                  ;ています。
~
```

(3) コモンセクション

ソースプログラムが複数のファイルで構成される場合に、ファイル間で共通に使用するデータを置くためのセクションを、コモンセクションといいます。

コモンセクションには、次のものを記述してください。

- ・初期値をもつデータを確保するアセンブラ制御命令
- ・初期値をもたないデータ領域を確保するアセンブラ制御命令

【補足】 リンケージエディタは、同名のコモンセクションを、メモリ上の同一領域に配置します。

図 2.1 に示した例では、ファイル A で宣言しているコモンセクション CM と、ファイル B で宣言しているコモンセクション CM を、同じ領域に割りつけています。

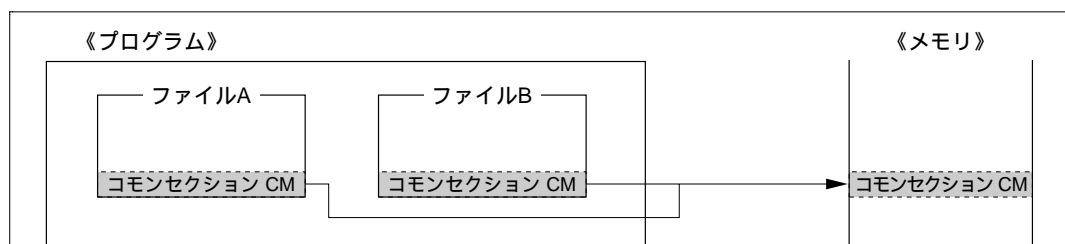


図 2.1 コモンセクションのメモリへの配置

(4) スタックセクション

マイコンがスタック領域(データを一時的に記憶する領域)として使うセクションを、スタックセクションといいます。

スタックセクションには、次のものを記述してください。

- ・初期値をもたないデータ領域を確保するアセンブラ制御命令

コーディング例

```
.SECTION  ST,STACK,ALIGN=4      ;ST という名前のスタックセクション
                                   ;を宣言しています。

.RES.B    1024                  ;1024 バイトのスタック領域を確保しています。
STK:
```

(5) ダミーセクション

ダミーセクションは、データ構造を記述する仮想的なセクションです。アセンブラは、ダミーセクションをオブジェクトモジュールに出力しません。

ダミーセクションには、次のものを記述してください。

- ・初期値をもたないデータ領域を確保するアセンブラ制御命令

コーディング例

```
.SECTION  DM,DUMMY              ;DM という名前のダミーセクション
                                   ;を宣言しています。

.RES.B    1                    ;アセンブラは、セクション DM をオブジェクト
A: .RES.B    1                  ;モジュールに出力しません。
B: .RES.B    2
~
```

データ構造を記述する具体的な方法を、次ページの【補足】で説明します。

【補足】 図 2.2 に示すように、ダミーセクションのアドレスシンボルを使って、メモリ上の領域にアクセスできます。

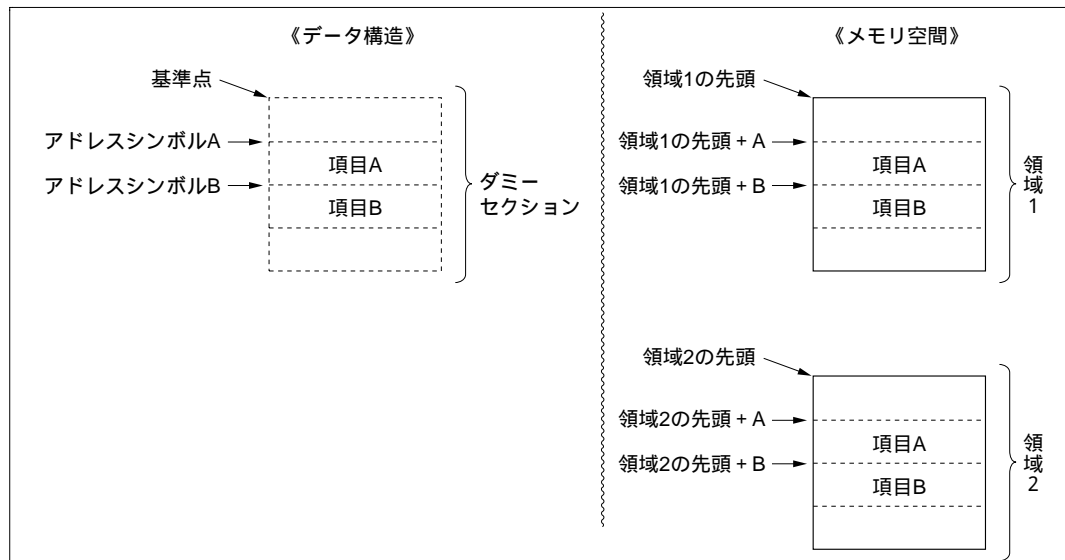


図 2.2 データ構造の記述例（ダミーセクションの使用例）

コーディング例

図 2.2 の例で、R1 に領域 1 の先頭アドレス、R2 に領域 2 の先頭アドレスが格納されているとします。

```
MOV.L    @(B,R1),R0        ;領域 1 の項目 B の内容を、R0 に転送しています。
MOV.L    R0,@(B,R2)        ;R0 の内容を、領域 2 の項目 B に転送しています。
~
```

【注意】 1. スタックセクションとダミーセクションには、次のものを記述できません。

- ・実行命令
- ・拡張命令
- ・初期値をもつデータを確保するアセンブラ制御命令
(.DATA, .DATAB, .SDATA, .SDATAB, .SDATAC, .SDATAZ, .FDATA,
.FATAB, .XDATA)

2. データセクションとコモンセクションは ROM 上に置くか RAM 上に置くかを意識して、セクションを分けてください。

2.1.2 絶対アドレスセクションと相対アドレスセクション

メモリへの配置方法の違いからセクションを考えると、絶対アドレスセクションと相対アドレスセクションとに大別できます。

(1) 絶対アドレスセクション

絶対アドレスセクションのメモリ上での位置は、ソースプログラムで指定し、リンケージのときには指定を変更できません。

本アセンブリ言語では、絶対アドレスセクション内のロケーションを、絶対アドレス(メモリ上の位置そのものを示すアドレス)で表現します。

コーディング例

```
.SECTION ABS,DATA,LOCATE=H'0000F000 ;ABS は、絶対アドレスセクション
;です。ABS の先頭位置は、絶対
;アドレスの H'0000F000 番地に
;なります。

.DATA.W H'1111 ;定数 H'1111 を、絶対アドレスの
;H'0000F000 番地に確保して
;います。

.DATA.W H'2222 ;定数 H'2222 を、絶対アドレスの
;H'0000F002 番地に確保して
;います。
```

(2) 相対アドレスセクション

相対アドレスセクションのメモリ上での位置は、ソースプログラムでは指定せず、リンケージのときに指定します。

本アセンブリ言語では、相対アドレスセクション内のロケーションを、相対アドレス(セクション先頭からの相対距離を示すアドレス)で表現します。

コーディング例

```
.SECTION REL,DATA,ALIGN=4 ;REL は、相対アドレスセクションです。REL の
;先頭位置は、リンケージ後に確定します。

.DATA.W H'1111 ;定数 H'1111 を、相対アドレスの
;H'00000000 番地に確保しています。

.DATA.W H'2222 ;定数 H'2222 を、相対アドレスの
;H'00000002 番地に確保しています。
```

【補足】 ダミーセクションは、絶対アドレスセクションか、相対アドレスセクションかの分類には当てはまりません。

2.2 絶対値と相対値

アセンブルを終えた時点で内容が確定する値を、絶対値といいます。
リンケージを終えなければ内容が確定しない値を、相対値といいます。

2.2.1 絶対値

絶対値として、次のものがあります。

(1) 定数値

- ・ 整数定数
- ・ 文字定数
- ・ 上記の値を表すシンボル (以下、定数シンボルと称します)

(2) 絶対アドレス値

- ・ 絶対アドレスセクション内で参照するロケーションカウンタ
- ・ ダミーセクション内で参照するロケーションカウンタ
- ・ 上記の値を表すシンボル (以下、絶対アドレスシンボルと称します)

(3) その他

アセンブルを終えた時点で内容が確定する式

2.2.2 相対値

相対値として、次のものがあります。

(1) 相対アドレス値

- ・ 相対アドレスセクション内で参照するロケーションカウンタ
- ・ 上記の値を表すシンボル (以下、相対アドレスシンボルと称します)

(2) 外部参照値

他のファイルを参照するシンボル (以下、外部参照シンボルと称します)

(3) その他

リンケージを終えるまで内容が確定しない式

2.3 シンボルの定義と参照

2.3.1 シンボルの定義

(1) 通常の設定

ソースステートメントのラベルとしてシンボルを記述し、その時点でのロケーションカウンタ値をシンボルに与える定義です。

コーディング例

```
.SECTION DT1,DATA,LOCATE=H'0000F000 ;絶対アドレスセクションを
;宣言しています。
X1: .DATA.W H'1111 ;X1の値はH'0000F000になります。
X2: .DATA.W H'2222 ;X2の値はH'0000F002になります。
~
```

```
.SECTION DT2,DATA,ALIGN=4 ;相対アドレスセクションを
;宣言しています。
Y1: .DATA.W H'1111 ;Y1の値は、リンケージ後に確定
;し、セクション先頭アドレス値
;になります。
Y2: .DATA.W H'2222 ;Y2の値は、リンケージ後に確定
;し、セクション先頭アドレス値
;+2 になります。
```

(2) アセンブラ制御命令による定義

任意の値や特別の意味を、アセンブラ制御命令によってシンボルに与える定義です。

コーディング例

```
.SECTION DT1,DATA,ALIGN=4 ;DT1はセクション名です。
;セクション名もシンボルの一種(セクションの先頭
;アドレスを表すシンボル)です。
;ただし、アドレスシンボルとセクション名とは、
;文法上の扱いが異なります。
```

X: .EQU 100

*i*X の値は 100 になります。

*i*X を再定義することはできません。

Y: .ASSIGN 10

*i*Y の値は 10 になります。

*i*Y を再定義することもできます。

Z: .REG R1

*i*Z は汎用レジスタ R1 の別名になります。

*i*Z を再定義することはできません。

2.3.2 シンボルの参照

シンボルの参照には、次の形態があります。

- ・前方参照
- ・後方参照
- ・外部参照

【補足】 本マニュアルでは、「前方」「後方」という用語を、図 2.3 に示す意味で使っています。

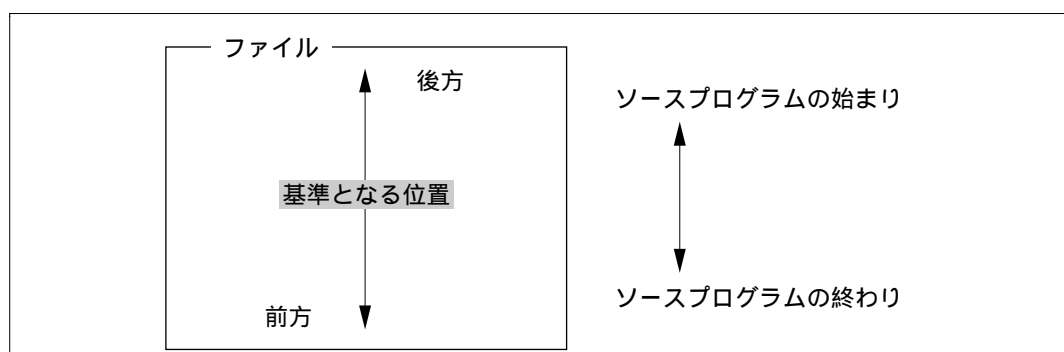


図 2.3 「前方」「後方」の意味

また、「外部」という用語を、図 2.4 に示す意味で使っています。

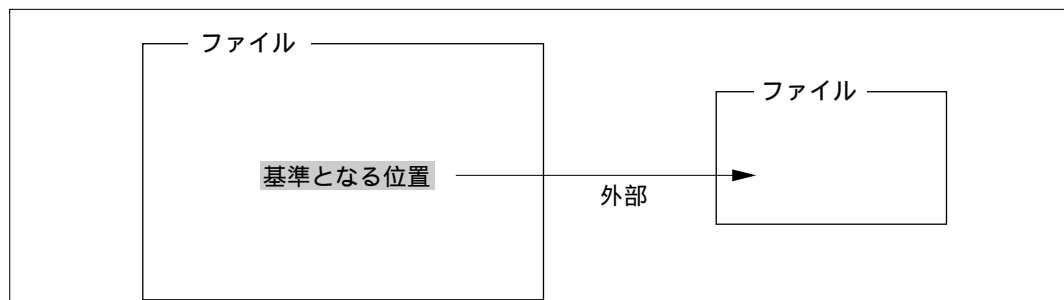


図 2.4 「外部」の意味

(1) 前方参照

前方参照とは、前方で定義しているシンボルを参照することです。

コーディング例

```

~
BRA      FORWARD    ;BRA は、分岐命令です。
                                ;前方のシンボル FORWARD を参照しています。
~
FORWARD :
~

```

(2) 後方参照

後方参照とは、後方で定義しているシンボルを参照することです。

コーディング例

```
~  
  
BACK :  
~  
BRA BACK ;BRA は、分岐命令です。  
;後方のシンボル BACK を参照しています。  
~
```

(3) 外部参照

ソースプログラムが複数のファイルで構成される場合に、他のファイルで定義しているシンボルを参照することです。

外部参照については、次節「2.4 分割プログラミング」で詳しく説明します。

2.4 分割プログラミング

2.4.1 分割プログラミングとは

複数のファイルに分割してソースプログラムを作成し、リンカージェディタによって最終的に1つのロードモジュールに結合する手法を、分割プログラミングといいます。

ソフトウェア開発の過程では、ソースプログラムの修正とアセンブル処理とを繰り返すことがよくあります。その場合、ソースプログラムが分割してあれば、修正したファイルだけをアセンブルし直せばよいので、ソフトウェア完成までに要する時間を短縮することができます。

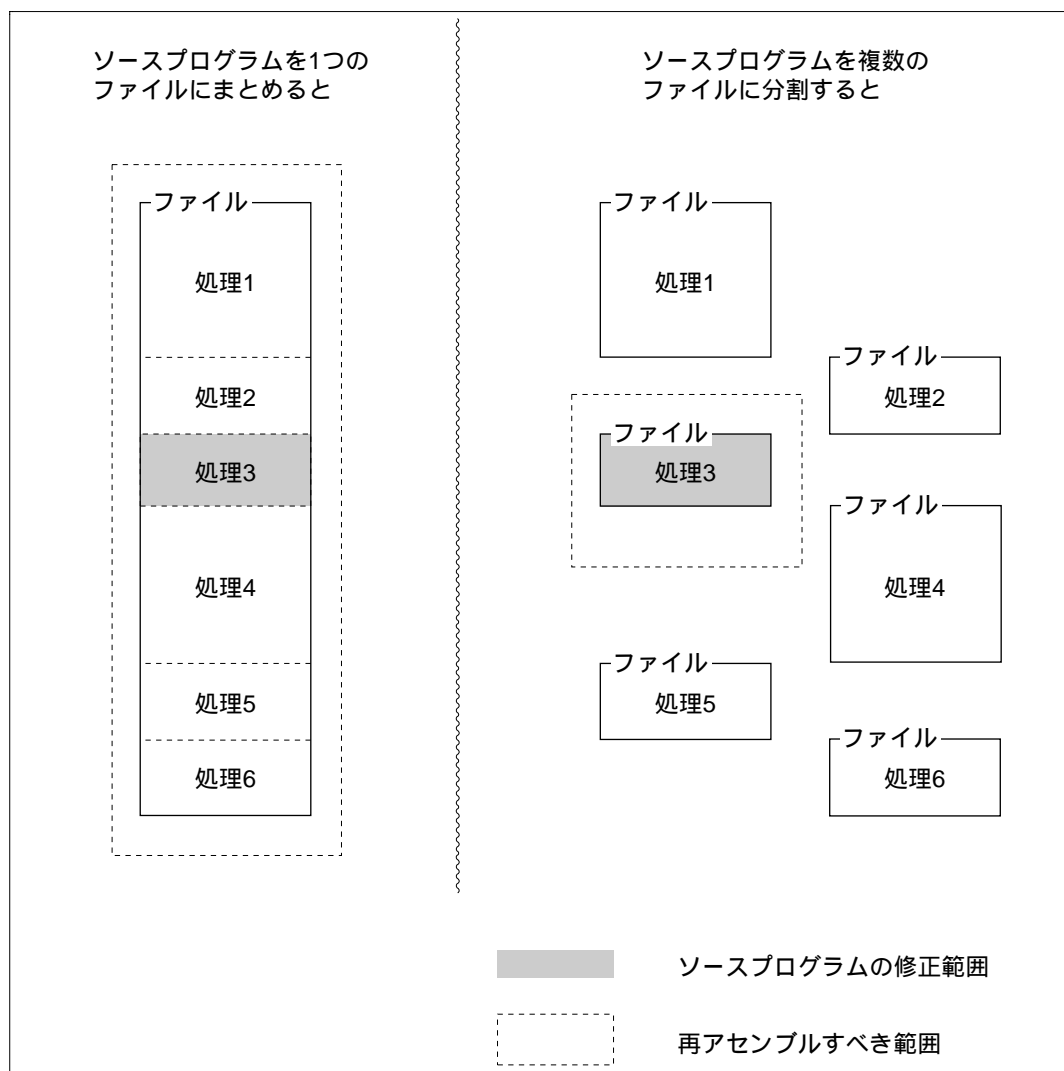


図 2.5 ソースプログラムの修正範囲と再アセンブルすべき範囲の関係

分割プログラミングの作業手順を、(a) ~ (d) に示します。

(a) ソースプログラムの分割方法を検討します。

機能別に分割するのが一般的です。

セクションをメモリ上に配置するときのことも、考慮してください。

(b) 複数のファイルに分けて、ソースプログラムを編集します。

(c) 個々のソースプログラムをアセンブルします。

(d) 個々のオブジェクトモジュールを1つにリンケージします。

2.4.2 外部定義シンボルと外部参照シンボルの宣言

ソースプログラムが複数のファイルで構成される場合、あるファイルで定義しているシンボルを他のファイルから参照することを、外部参照といいます。シンボルを外部参照するには、あらかじめ、「このシンボルは複数のファイル間で共通に使用する」と、宣言しておく必要があります。

(1) 外部定義シンボルの宣言

シンボルの定義を他のファイルでも有効とするための宣言です。

.EXPORT または .GLOBAL アセンブラ制御命令で宣言します。

(2) 外部参照シンボルの宣言

他のファイルで定義しているシンボルを参照するための宣言です。

.IMPORT または .GLOBAL アセンブラ制御命令で宣言します。

コーディング例

ファイル A で定義しているシンボル MAX をファイル B で参照する例です。

・ファイル A

```
~  
      .EXPORT  MAX      ;MAX を外部定義シンボルとして宣言しています。  
MAX :  .EQU      100     ;MAX を定義しています。  
~
```

・ファイル B

```
~  
      .IMPORT  MAX      ;MAX を外部参照シンボルとして宣言しています。  
MOV     #MAX, R0 ;MAX を参照しています。  
~
```

【参照】 シンボルの外部定義、外部参照

言語編「5.2.5 外部定義または外部参照に関するアセンブラ制御命令」

.EXPORT .IMPORT .GLOBAL

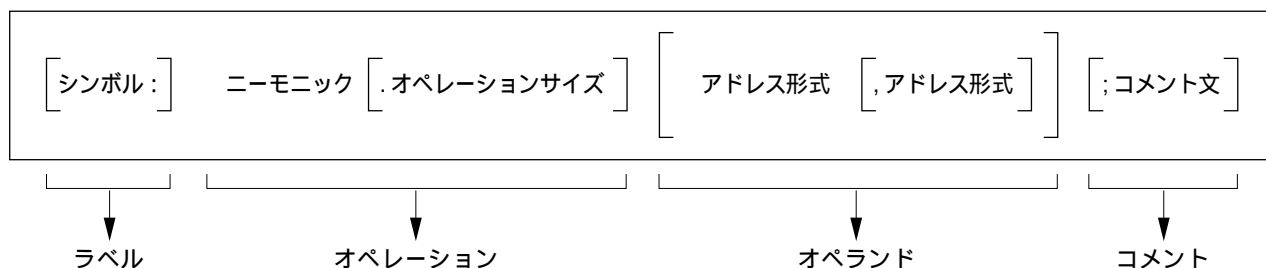
3. 実行命令

3.1 実行命令の概要

実行命令は、マイコンの命令です。

マイコンは、メモリ上にある実行命令のオブジェクトコードを解読して実行します。

実行命令のソースステートメントは、基本的に次のように記述します。



本節では、ニーモニック、オペレーションサイズ、アドレス形式について説明します。
その他の要素に関しては、言語編「1. プログラムの要素」で詳しく説明しています。

(1) ニーモニック

ニーモニックは、実行命令を表します。処理内容を連想させる英文字の名前が、ニーモニックとして用意されています。

アセンブラは、ニーモニック中の英大文字と英小文字を区別しません。

(2) オペレーションサイズ

オペレーションサイズは、データを操作する単位です。

指定できるオペレーションサイズは、実行命令ごとに異なります。

アセンブラは、オペレーションサイズの英大文字と英小文字を区別しません。

指定内容	データのサイズ
B	バイト
W	ワード(2バイト)
L	ロングワード(4バイト)
S	単精度(4バイト)
D	倍精度(8バイト)

(3) アドレス形式

アクセスの対象となるデータ領域や、分岐先アドレスなどを表します。

指定できるアドレス形式は、実行命令ごとに異なります。

表 3.1 に、アドレス形式の一覧を示します。

表 3.1 アドレス形式一覧 (その 1)

アドレス形式	名 称	解 説
Rn	レジスタ直接	レジスタ上の領域です。
@Rn	レジスタ間接	メモリ上の領域です。Rn の値が、領域の先頭アドレスを表します。
@Rn+	ポストインクリメント レジスタ間接	メモリ上の領域です。インクリメント* ¹ する前の Rn の値が、領域の先頭アドレスを表します。 マイコンは、Rn を先に参照し、後からインクリメントします。
@-Rn	プリデクリメント レジスタ間接	メモリ上の領域です。デクリメント* ² した後の Rn の値が、領域の先頭アドレスを表します。マイコンは、Rn を先にデクリメントし、後から参照します。
@(disp,Rn)	ディスプレースメント 付きレジスタ間接* ³	メモリ上の領域です。領域の先頭アドレスは、 <u>Rn の値 + ディスプレースメント (disp)</u> です。 Rn の内容は変わりません。
@(R0,Rn)	インデックス付きレジ スタ間接	メモリ上の領域です。領域の先頭アドレスは、 <u>Rn の値 + R0 の値</u> です。 Rn および R0 の内容は変わりません。
@(disp,GBR)	ディスプレースメント 付き GBR 間接	メモリ上の領域です。領域の先頭アドレスは、 <u>GBR の値 + ディスプレースメント (disp)</u> です。 GBR の内容は変わりません。
@(R0, GBR)	インデックス付き GBR 間接	メモリ上の領域です。領域の先頭アドレスは、 <u>GBR の値 + R0 の値</u> です。 GBR および R0 の内容は変わりません。
@(disp,PC)	ディスプレースメント 付き PC 相対	メモリ上の領域です。領域の先頭アドレスは、 <u>PC の値 + ディスプレースメント (disp)</u> です。

*1 ~ *3 次ページを参照

表 3.1 アドレス形式一覧（その2）

アドレス形式	名 称	解 説
symbol	シンボル指定による PC 相対	<p>【分岐命令のオペランドである場合】</p> <p>symbol は、分岐先のアドレスを表します。アセンブラは、symbol と PC の値からディスプレースメント（disp）を逆算します。<u>disp=symbol-PC</u> です。</p> <p>【データ転送命令のオペランドである場合】</p> <p>メモリ上の領域です。symbol は、領域の先頭アドレスを表します。</p> <p>アセンブラは、symbol と PC の値からディスプレースメント（disp）を逆算します。<u>disp=symbol-PC</u> です。</p> <p>【RS,RE レジスタを指定する命令(LDRS,LDRE)のオペランドである場合】</p> <p>メモリ上の領域です。symbol は、領域の先頭アドレス^{*4}を表します。</p> <p>アセンブラは、symbol と PC の値からディスプレースメント（disp）を逆算します。<u>disp=symbol-PC</u> です。</p>
#imm	イミディエイト	定数を表します。

【注】 *1 インクリメント

オペレーションサイズがバイトのとき 1 を、ワード（2 バイト）のとき 2 を、ロングワード（4 バイト）のとき 4 を加えることです。

*2 デクリメント

オペレーションサイズがバイトのとき 1 を、ワード（2 バイト）のとき 2 を、ロングワード（4 バイト）のとき 4 を減じることです。

*3 ディスプレースメント

2 点間の距離です。本アセンブリ言語ではバイト単位で表現します。

*4 先頭アドレス

本項目の解説に関しては、言語編「10.1 リピートループ命令自動生成機能の概要」を参照してください。

ディスプレースメントとして許される値は、オペランドのアドレス形式や、オペレーションサイズによって異なります。

表 3.2 ディスプレースメントとして許される値（その 1）

アドレス形式	ディスプレースメント〔単位はバイト（ ）内は 10 進表現〕
@(disp,Rn)	<p>オペレーションサイズがバイト（B）のとき H'00000000 ~ H'0000000F（0 ~ 15）</p> <p>オペレーションサイズがワード（W）のとき H'00000000 ~ H'0000001E（0 ~ 30）</p> <p>オペレーションサイズがロングワード（L）のとき H'00000000 ~ H'0000003C（0 ~ 60）</p>
@(disp,GBR)	<p>オペレーションサイズがバイト（B）のとき H'00000000 ~ H'000000FF（0 ~ 255）</p> <p>オペレーションサイズがワード（W）のとき H'00000000 ~ H'000001FE（0 ~ 510）</p> <p>オペレーションサイズがロングワード（L）のとき H'00000000 ~ H'000003FC（0 ~ 1020）</p>
@(disp,PC)	<p>【データ転送命令のオペランドである場合】</p> <p>オペレーションサイズがワード（W）のとき H'00000000 ~ H'000001FE（0 ~ 510）</p> <p>オペレーションサイズがロングワード（L）のとき H'00000000 ~ H'000003FC（0 ~ 1020）</p> <p>【RS、RE レジスタを設定する命令（LDRS, LDRE）のオペランドである場合】 H'FFFFFF00 ~ H'000000FE（-256 ~ 254）</p>
symbol	<p>【分岐命令のオペランドである場合】</p> <p>条件付きの分岐命令（BT, BF, BF/S, BT/S）のオペランドであるとき H'00000000 ~ H'000000FF（0 ~ 255） H'FFFFFF00 ~ H'FFFFFFF（-256 ~ -1）</p> <p>無条件の分岐命令（BRA, BSR）のオペランドであるとき H'00000000 ~ H'00000FFF（0 ~ 4095） H'FFFFFF00 ~ H'FFFFFFF（-4096 ~ -1）</p>

表 3.2 ディスプレースメントとして許される値（その 2）

アドレス形式	ディスプレースメント〔単位はバイト（ ）内は 10 進表現〕
symbol	<p>【データ転送命令のオペランドである場合】</p> <p>オペレーションサイズがワード（W）のとき</p> <p>H'00000000 ~ H'000001FE（0 ~ 510）</p> <p>オペレーションサイズがロングワード（L）のとき</p> <p>H'00000000 ~ H'000003FC（0 ~ 1020）</p> <p>【RS、RE レジスタを設定する命令（LDRS, LDRE）のオペランドである場合】</p> <p>H'FFFFFF00 ~ H'000000FE（-256 ~ 254）</p>

イミディエイトとして許される値は、実行命令によって異なります。

表 3.3 イミディエイトとして許される値

実行命令	イミディエイト
TST, AND, OR, XOR	H'00000000 ~ H'000000FF (0 ~ 255)
MOV	H'00000000 ~ H'000000FF (0 ~ 255) H'FFFFFF80 ~ H'FFFFFFF (-128 ~ -1) * ¹
ADD, CMP/EQ	H'00000000 ~ H'000000FF (0 ~ 255) H'FFFFFF80 ~ H'FFFFFFF (-128 ~ -1) * ¹
TRAPA	H'00000000 ~ H'000000FF (0 ~ 255)
SETRC	H'00000001 ~ H'000000FF (1 ~ 255) * ²

【注】 *¹ H'FFFFFF80 ~ H'FFFFFFF の範囲を、正の 10 進数で記述しても構いません。

*² SETRC 命令のイミディエイトデータに 0 を設定した場合、ウォーニング 835 とし、オブジェクトコードには 0 を設定します。この場合、リピーター範囲は 1 回実行されます。

また、SETRC 命令のイミディエイト値に外部参照シンボルを指定した場合、リンケージデータは、H'00000000 ~ H'000000FF (0 ~ 255) の範囲のチェックとなります。

【注意】 アセンブラは、条件に応じてディスプレースメントを補正します。

条件

オペレーションサイズがワードで
ディスプレースメントが 2 の倍数でない

オペレーションサイズがロングワードで
ディスプレースメントが 4 の倍数でない

分岐命令で
ディスプレースメントが 2 の倍数でない

補正内容

ディスプレースメントの下位 1 ビット
を切り捨て、2 の倍数に補正

ディスプレースメントの下位 2 ビット
を切り捨て、4 の倍数に補正

ディスプレースメントの下位 1 ビット
を切り捨て、2 の倍数に補正

オペランドとして @(disp, Rn)、@(disp, GBR)、@(disp, PC) を記述する場合には、アセンブラによるディスプレースメントの補正を考慮してください。

コーディング例

```
MOV.L  @( 63,PC),R0
```

アセンブラは、ディスプレースメントを 63 から 60 に補正して、MOV.L @(60,PC), R0 と同じオブジェクトコードを生成します。また、ウォーニング 870 を通知します。

3.2 実行命令に関する注意事項

3.2.1 オペレーションサイズに関する注意

ニーモニックとアドレス形式の組み合わせにより、指定できるオペレーションサイズが異なります。

(1) SH-1 の実行命令とオペレーションサイズの組み合わせ

表 3.4 に、SH-1 の実行命令とオペレーションサイズの組み合わせを示します。

表 3.4 実行命令とオペレーションサイズの組み合わせ (その1)

1.データ転送命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
MOV	#imm,Rn				B
MOV	@(disp,PC),Rn	×			L
MOV	symbol,Rn	×			L
MOV	Rn,Rm	×	×		L
MOV	Rn,@Rm				L
MOV	@Rn,Rm				L
MOV	Rn, @-Rm				L
MOV	@Rn+, Rm				L
MOV	R0, @(disp,Rn)				L
MOV	Rn, @(disp,Rm)	×	×		L
MOV	@(disp,Rn), R0				L
MOV	@(disp,Rn), Rm	×	×		L
MOV	Rn, @(R0,Rm)				L
MOV	@(R0,Rn), Rm				L
MOV	R0, @(disp,GBR)				L
MOV	@(disp,GBR), R0				L
MOVA	#imm, R0	×	×		L
MOVA	@(disp,PC), R0	×	×		L
MOVA	symbol,R0	×	×		L
MOVT	Rn	×	×		L
SWAP	Rn,Rm			×	W
XTRCT	Rn,Rm	×	×		L

【注】*1～*3 次ページ【注】を参照

【記号の意味】

Rn、Rm : 汎用レジスタ (R0 ~ R15)
R0 : 汎用レジスタ (R0 固定)
SR : ステータスレジスタ
GBR : グローバル・ベースレジスタ
VBR : ベクタ・ベースレジスタ
MACH、MACL : 積和レジスタ
PR : プロシージャレジスタ
PC : プログラムカウンタ
imm : イミディエイト
disp : ディスプレースメント
symbol : シンボル

B : バイト
W : ワード (2 バイト)
L : ロングワード (4 バイト)

: 指定は有効

× : 指定は無効 オペレーションサイズの指定を省略したのと同じ結果になる
: アセンブラは拡張命令として解釈する

【注】 *1 サイズ選択モードの場合、アセンブラが imm の値によりオペレーションサイズを決めます。

*2 この場合の Rn は、R1 ~ R15

*3 この場合の Rm は、R1 ~ R15

【参照】 拡張命令 言語編「9.2 リテラルプール自動生成機能に関する拡張命令」

サイズ選択モード 言語編「9.3 リテラルプール自動生成機能のサイズモード」

表 3.4 実行命令とオペレーションサイズの組み合わせ（その2）

2.算術演算命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
ADD	Rn,Rm	×	×		L
ADD	#imm,Rn	×	×		L
ADDC	Rn,Rm	×	×		L
ADDV	Rn,Rm	×	×		L
CMP/EQ	#imm,R0	×	×		L
CMP/EQ	Rn,Rm	×	×		L
CMP/HS	Rn,Rm	×	×		L
CMP/GE	Rn,Rm	×	×		L
CMP/HI	Rn,Rm	×	×		L
CMP/GT	Rn,Rm	×	×		L
CMP/PZ	Rn	×	×		L
CMP/PL	Rn	×	×		L
CMP/STR	Rn,Rm	×	×		L
DIV1	Rn,Rm	×	×		L
DIV0S	Rn,Rm	×	×		L
DIV0U	(オペランドなし)	×	×	×	-
EXTS	Rn,Rm			×	W
EXTU	Rn,Rm			×	W
MAC	@Rn+,@Rm+	×		×	W
MULS	Rn,Rm	×			L *
MULU	Rn,Rm	×			L *
NEG	Rn,Rm	×	×		L
NEGC	Rn,Rm	×	×		L
SUB	Rn,Rm	×	×		L
SUBC	Rn,Rm	×	×		L
SUBV	Rn,Rm	×	×		L

【注】* WとLは、同じオブジェクトコードとなります。

表 3.4 実行命令とオペレーションサイズの組み合わせ（その 3）

3.論理演算命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
AND	Rn,Rm	x	x		L
AND	#imm,R0	x	x		L
AND	#imm,@(R0,GBR)		x	x	B
NOT	Rn,Rm	x	x		L
OR	Rn,Rm	x	x		L
OR	#imm,R0	x	x		L
OR	#imm,@(R0,GBR)		x	x	B
TAS	@Rn		x	x	B
TST	Rn,Rm	x	x		L
TST	#imm,R0	x	x		L
TST	#imm,@(R0,GBR)		x	x	B
XOR	Rn,Rm	x	x		L
XOR	#imm,R0	x	x		L
XOR	#imm,@(R0,GBR)		x	x	B

表 3.4 実行命令とオペレーションサイズの組み合わせ（その 4）

4.シフト命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
ROTL	Rn	x	x		L
ROTR	Rn	x	x		L
ROTCL	Rn	x	x		L
ROTCR	Rn	x	x		L
SHAL	Rn	x	x		L
SHAR	Rn	x	x		L
SHLL	Rn	x	x		L
SHLR	Rn	x	x		L
SHLL2	Rn	x	x		L
SHLR2	Rn	x	x		L
SHLL8	Rn	x	x		L
SHLR8	Rn	x	x		L
SHLL16	Rn	x	x		L
SHLR16	Rn	x	x		L

表 3.4 実行命令とオペレーションサイズの組み合わせ（その 5）

5.分岐命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
BF	symbol	x	x	x	-
BT	symbol	x	x	x	-
BRA	symbol	x	x	x	-
BSR	symbol	x	x	x	-
JMP	@Rn	x	x	x	-
JSR	@Rn	x	x	x	-
RTS	(オペランドなし)	x	x	x	-

表 3.4 実行命令とオペレーションサイズの組み合わせ（その6）

6.システム制御命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
CLRT	（オペランドなし）	x	x	x	-
CLRMAC	（オペランドなし）	x	x	x	-
LDC	Rn,SR	x	x		L
LDC	Rn,GBR	x	x		L
LDC	Rn,VBR	x	x		L
LDC	@Rn+,SR	x	x		L
LDC	@Rn+,GBR	x	x		L
LDC	@Rn+,VBR	x	x		L
LDS	Rn,MACH	x	x		L
LDS	Rn,MACL	x	x		L
LDS	Rn,PR	x	x		L
LDS	@Rn+,MACH	x	x		L
LDS	@Rn+,MACL	x	x		L
LDS	@Rn+,PR	x	x		L
NOP	（オペランドなし）	x	x	x	-
RTE	（オペランドなし）	x	x	x	-
SETT	（オペランドなし）	x	x	x	-
SLEEP	（オペランドなし）	x	x	x	-
STC	SR,Rn	x	x		L
STC	GBR,Rn	x	x		L
STC	VBR,Rn	x	x		L
STC	SR,@-Rn	x	x		L
STC	GBR,@-Rn	x	x		L
STC	VBR,@-Rn	x	x		L
STS	MACH,Rn	x	x		L
STS	MACL,Rn	x	x		L
STS	PR,Rn	x	x		L
STS	MACH,@-Rn	x	x		L
STS	MACL,@-Rn	x	x		L
STS	PR,@-Rn	x	x		L
TRAPA	#imm	x	x		L

(2) SH-2 の実行命令とオペレーションサイズの組み合わせ

表 3.5 に、SH-1 に対して SH-2 で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 3.5 実行命令とオペレーションサイズの組み合わせ (その 1)

1. 算術演算命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
MAC	@Rn+, @Rm+	×			W
MUL	Rn, Rm	×	×		L
DMULS	Rn, Rm	×	×		L
DMULU	Rn, Rm	×	×		L
DT	Rn	×	×	×	-

表 3.5 実行命令とオペレーションサイズの組み合わせ (その 2)

2. 分岐命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
BF/S	symbol	×	×	×	-
BT/S	symbol	×	×	×	-
BRAF	Rn	×	×	×	-
BSRF	Rn	×	×	×	-

(3) SH-2E の実行命令とオペレーションサイズの組み合わせ

表 3.6 に、SH-2 に対して SH-2E で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 3.6 実行命令とオペレーションサイズの組み合わせ (その1)

1.データ転送命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略すると
FLDI0	FRn	x	x	x		S
FLDI1	FRn	x	x	x		S
FMOV	@Rm,FRn	x	x	x		S
FMOV	FRn,@Rm	x	x	x		S
FMOV	@Rm+,FRn	x	x	x		S
FMOV	FRn,@-Rm	x	x	x		S
FMOV	@(R0,Rm),FRn	x	x	x		S
FMOV	FRm,@(R0,Rm)	x	x	x		S
FMOV	FRm,FRn	x	x	x		S

表 3.6 実行命令とオペレーションサイズの組み合わせ (その2)

2.算術演算命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略すると
FABS	FRn	x	x	x		S
FADD	FRm,FRn	x	x	x		S
FCMP/EQ	FRm,FRn	x	x	x		S
FCMP/GT	FRm,FRn	x	x	x		S
FDIV	FRm,FRn	x	x	x		S
FMAC	FR0,FRm,FRn	x	x	x		S
FMUL	FRm,FRn	x	x	x		S
FNEG	FRn	x	x	x		S
FSUB	FRm,FRn	x	x	x		S

表 3.6 実行命令とオペレーションサイズの組み合わせ（その 3）

3.システム制御命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略すると
FLDS	FRm,FPUL	x	x	x		S
FLOAT	FPUL,FRn	x	x	x		S
FSTS	FPUL,FRn	x	x	x		S
FTRC	FRm,FPUL	x	x	x		S
LDS	Rm,FPUL	x	x		x	L
LDS	@Rm+,FPUL	x	x		x	L
LDS	Rm,FPSCR	x	x		x	L
LDS	@Rm+,FPSCR	x	x		x	L
STS	FPUL,Rn	x	x		x	L
STS	FPUL,@-Rn	x	x		x	L
STS	FPSCR,Rn	x	x		x	L
STS	FPSCR,@-Rn	x	x		x	L

【記号の意味】

FRm, FRn : 浮動小数点レジスタ

FR0 : 浮動小数点レジスタ（FR0 固定）

FPUL : FPU・ローレジスタ

FPSCR : FPU・ステータスコントロールレジスタ

S : 単精度（4 バイト）

（4）SH-3 の実行命令とオペレーションサイズの組み合わせ

表 3.7 に、SH-2 に対して SH-3 で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 3.7 実行命令とオペレーションサイズの組み合わせ（その 1）

1.データ転送命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
PREF	@Rn	x	x	x	-

表 3.7 実行命令とオペレーションサイズの組み合わせ（その 2）

2.シフト演算命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
SHAD	Rn, Rm	x	x		L
SHLD	Rn, Rm	x	x		L

表 3.7 実行命令とオペレーションサイズの組み合わせ（その3）

3.システム制御命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
CLRS	(オペランドなし)	×	×	×	-
SETS	(オペランドなし)	×	×	×	-
LDC	Rm, SSR	×	×		L
LDC	Rm, SPC	×	×		L
LDC	Rm, Rn_BANK	×	×		L
LDC	@Rm+, SSR	×	×		L
LDC	@Rm+, SPC	×	×		L
LDC	@Rm+, Rn_BANK	×	×		L
STC	SSR, Rn	×	×		L
STC	SPC, Rn	×	×		L
STC	Rm_BANK, Rn	×	×		L
STC	SSR, @-Rn	×	×		L
STC	SPC, @-Rm	×	×		L
STC	Rm_BANK, @-Rn	×	×		L
LDTLB	(オペランドなし)	×	×	×	-

【記号の意味】

Rn_BANK : バンク汎用レジスタ

SSR : セーブ・ステータスレジスタ

SPC : セーブ・プログラムカウンタ

(5) SH-3E の実行命令とオペレーションサイズの組み合わせ

表 3.8 に、SH-3 に対して SH-3E で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 3.8 実行命令とオペレーションサイズの組み合わせ (その 1)

1.データ転送命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略すると
FLDI0	FRn	×	×	×		S
FLDI1	FRn	×	×	×		S
FMOV	@Rm, FRn	×	×	×		S
FMOV	FRn, @Rm	×	×	×		S
FMOV	@Rm+, FRn	×	×	×		S
FMOV	FRn, @-Rm	×	×	×		S
FMOV	@(R0, Rm), FRn	×	×	×		S
FMOV	FRn, @(R0, Rm)	×	×	×		S
FMOV	FRm, FRn	×	×	×		S

表 3.8 実行命令とオペレーションサイズの組み合わせ (その 2)

2.算術演算命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略すると
FABS	FRn	×	×	×		S
FADD	FRm, FRn	×	×	×		S
FCMP/EQ	FRm, FRn	×	×	×		S
FCMP/GT	FRm, FRn	×	×	×		S
FDIV	FRm, FRn	×	×	×		S
FMAC	FR0, FRm, FRn	×	×	×		S
FMUL	FRm, FRn	×	×	×		S
FNEG	FRn	×	×	×		S
FSQRT	FRn	×	×	×		S
FSUB	FRm, FRn	×	×	×		S

表 3.8 実行命令とオペレーションサイズの組み合わせ（その 3）

3.システム制御命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略すると
FLDS	FRm, FPUL	×	×	×		S
FLOAT	FPUL, FRn	×	×	×		S
FSTS	FPUL, FRn	×	×	×		S
FTRC	FRm, FPUL	×	×	×		S
LDS	Rm, FPUL	×	×		×	L
LDS	@Rm+, FPUL	×	×		×	L
LDS	Rm, FPSCR	×	×		×	L
LDS	@Rm+, FPSCR	×	×		×	L
STS	FPUL, Rn	×	×		×	L
STS	FPUL, @-Rn	×	×		×	L
STS	FPSCR, Rn	×	×		×	L
STS	FPSCR, @- Rn	×	×		×	L

【記号の意味】

FRm, FRn : 浮動小数点レジスタ

FR0 : 浮動小数点レジスタ (FR0 固定)

FPUL : FPU・ローレジスタ

FPSCR : FPU・ステータスコントロールレジスタ

S : 単精度 (4 バイト)

(6) SH-4 の実行命令とオペレーションサイズの組み合わせ

表 3.9 に、SH-3E に対して SH-4 で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 3.9 実行命令とオペレーションサイズの組み合わせ (その 1)

1.データ転送命令		オペレーションサイズ					
ニーモニック	アドレス形式	B	W	L	S	D	省略すると
FMOV	DRm,DRn	x	x	x	x		D
FMOV	DRm,@Rn	x	x	x	x		D
FMOV	DRm@-Rn	x	x	x	x		D
FMOV	DRm,@(R0,Rn)	x	x	x	x		D
FMOV	@Rm,DRn	x	x	x	x		D
FMOV	@Rm+,DRn	x	x	x	x		D
FMOV	@(R0,Rm),DRn	x	x	x	x		D
FMOV	DRm,XDn	x	x	x	x		D
FMOV	XDm,DRn	x	x	x	x		D
FMOV	XDm,XDn	x	x	x	x		D
FMOV	XDm,@Rn	x	x	x	x		D
FMOV	XDm,@-Rn	x	x	x	x		D
FMOV	XDm,@(R0,Rn)	x	x	x	x		D
FMOV	@Rm,XDn	x	x	x	x		D
FMOV	@Rm+,XDn	x	x	x	x		D
FMOV	@(R0,Rm),XDn	x	x	x	x		D

表 3.9 実行命令とオペレーションサイズの組み合わせ（その2）

2.算術演算命令		オペレーションサイズ					
ニーモニック	アドレス形式	B	W	L	S	D	省略すると
FABS	DRn	×	×	×	×		D
FADD	DRm,DRn	×	×	×	×		D
FCMP/EQ	DRm,DRn	×	×	×	×		D
FCMP/GT	DRm,DRn	×	×	×	×		D
FDIV	DRm,DRn	×	×	×	×		D
FIPR	FVm,FVn	×	×	×		×	S
FMUL	DRm,DRn	×	×	×	×		D
FNEG	DRn	×	×	×	×		D
FSQRT	DRn	×	×	×	×		D
FSUB	DRm,DRn	×	×	×	×		D
FTRV	XMTRX,FVn	×	×	×		×	S

表 3.9 実行命令とオペレーションサイズの組み合わせ（その3）

3.システム制御命令		オペレーションサイズ					
ニーモニック	アドレス形式	B	W	L	S	D	省略すると
FCNVDS	DRm,FPUL	×	×	×	×		D
FCNVSD	FPUL,DRn	×	×	×	×		D
FLOAT	FPUL,DRn	×	×	×	×		D
FRCHG	(オペランドなし)	×	×	×	×	×	
FSCHG	(オペランドなし)	×	×	×	×	×	
FTRC	DRm,FPUL	×	×	×	×		D
LDC	Rm,DBR	×	×		×	×	L
LDC	@Rm+,DBR	×	×		×	×	L
OCBI	@Rn	×	×	×	×	×	
OCBP	@Rn	×	×	×	×	×	
OCBWB	@Rn	×	×	×	×	×	
STC	DBR,Rn	×	×		×	×	L
STC	DBR,@-Rn	×	×		×	×	L
STC	SGR,Rn	×	×		×	×	L
STC	SGR,@-Rn	×	×		×	×	L

【記号の意味】

DRm, DRn	:	倍精度浮動小数点レジスタ
XDm, XDn	:	拡張倍精度浮動小数点レジスタ
FVm, FVn	:	単精度浮動小数点ベクトルレジスタ
XMTRX	:	単精度浮動小数点拡張レジスタ行列
DBR	:	デバッグベクタベースレジスタ
SGR	:	退避ジェネラルレジスタ 15
D	:	倍精度（8 バイト）

(7) SH-DSP、SH3-DSP の実行命令とオペレーションサイズの組み合わせ

表 3.10 に、SH-2 に対して SH-DSP、SH-3 に対して SH3-DSP で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 3.10 実行命令とオペレーションサイズの組み合わせ (その 1)

1. リピート制御命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
LDRS	@(disp, PC)	×	×		L
LDRS	symbol	×	×		L
LDRE	@(disp, PC)	×	×		L
LDRE	symbol	×	×		L
SETRC	Rn	×	×	×	-
SETRC	#imm	×	×	×	-

表 3.10 実行命令とオペレーションサイズの組み合わせ（その2）

2.システム制御命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略すると
LDC	Rn, MOD	x	x		L
LDC	Rn, RS	x	x		L
LDC	Rn, RE	x	x		L
LDC	@Rn+, MOD	x	x		L
LDC	@Rn+, RS	x	x		L
LDC	@Rn+, RE	x	x		L
LDS	Rn, DSR	x	x		L
LDS	Rn, A0	x	x		L
LDS	@Rn+, DSR	x	x		L
LDS	@Rn+, A0	x	x		L
STC	MOD, Rn	x	x		L
STC	RS, Rn	x	x		L
STC	RE, Rn	x	x		L
STC	MOD, @-Rn	x	x		L
STC	RS, @-Rn	x	x		L
STC	RE, @-Rn	x	x		L
STS	DSR, Rn	x	x		L
STS	A0, Rn	x	x		L
STS	DSR, @-Rn	x	x		L
STS	A0, @-Rn	x	x		L

【記号の意味】

MOD : モジュロレジスタ

RS : リピート・スタートレジスタ

RE : リピート・エンドレジスタ

DSR : DSP・ステータスレジスタ

A0 : DSP・データレジスタ（A0 以外に、X0、X1、Y0、Y1 が指定できます）

3.2.2 遅延分岐命令に関する注意

無条件の分岐命令は、遅延分岐命令です。

マイコンは、遅延分岐命令の実行に先だってディレイスロット命令（メモリ上で、遅延分岐命令の直後に位置する命令）を実行します。

ディレイスロット命令が不当なものである場合、アセンブラはエラー150または151を通知します。

遅延分岐命令とディレイスロット命令との関係を、表3.11に示します。

表 3.11 遅延分岐命令とディレイスロット命令の関係

ディレイスロット		遅延分岐命令									
		BF/S	BT/S	BRAF	BSRF	BRA	BSR	JMP	JSR	RTS	RTE
BF		×	×	×	×	×	×	×	×	×	×
BT		×	×	×	×	×	×	×	×	×	×
BF/S		×	×	×	×	×	×	×	×	×	×
BT/S		×	×	×	×	×	×	×	×	×	×
BRAF		×	×	×	×	×	×	×	×	×	×
BSRF		×	×	×	×	×	×	×	×	×	×
BRA		×	×	×	×	×	×	×	×	×	×
BSR		×	×	×	×	×	×	×	×	×	×
JMP		×	×	×	×	×	×	×	×	×	×
JSR		×	×	×	×	×	×	×	×	×	×
RTS		×	×	×	×	×	×	×	×	×	×
RTE		×	×	×	×	×	×	×	×	×	×
TRAPA		×	×	×	×	×	×	×	×	×	×
LDC	Rn,SR	*1	*1	*1	*1	*1	*1	*1	*1	*1	*1
	@Rn+,SR	*1	*1	*1	*1	*1	*1	*1	*1	*1	*1
MOV	@(disp,PC),Rn	*2	*2	*2	*2	*2	*2	*2	*2	*2	*2
	symbol,Rn	*2	*2	×	×	*2	*2	×	×	×	×
MOVA	@(disp,PC),R0	*2	*2	*2	*2	*2	*2	*2	*2	*2	*2
	symbol,R0	*2	*2	×	×	*2	*2	×	×	×	×
拡張 命令	MOV.L #imm,Rn	×	×	×	×	×	×	×	×	×	×
	MOV.W #imm,Rn	×	×	×	×	×	×	×	×	×	×
	MOVA #imm,R0	×	×	×	×	×	×	×	×	×	×
上記以外の命令											

【記号の意味】

- : 正常 アセンブラは指定どおりにオブジェクトコードを生成
- x : エラー150 または 151
ディレイスロット命令が不当
アセンブラは NOP 命令のオブジェクトコード (H'0009) を生成
- *1 : CPU 種別が、SH-1、SH-2、SH-2E、SH-DSP のとき正常
上記以外るときエラー150 または 151
- *2 : CPU 種別が、SH-4 のときエラー150 または 151
上記以外るときウォーニング 871
PC の値に注意 $PC = \text{遅延分岐命令による分岐先アドレス} + 2$
アセンブラは指定どおりにオブジェクトコードを生成

【注意】 遅延分岐命令とディレイスロット命令が別々のセクションに属する場合には、アセンブラはディレイスロット命令の正当性をチェックしません。

【参照】 拡張命令

言語編「9.2 リテラルプール自動生成機能に関する拡張命令」

3.2.3 アドレス計算に関する注意

オペランドのアドレス形式がディスプレースメント付き PC 相対@ (disp, PC) である場合、PC の値を考慮してプログラミングする必要があります。

PC の値がどうなるかは、状況によって異なります。

(1) 通常

PC の値は、実行中の命令の先頭アドレス+4 バイトです。

例

絶対アドレス H'00001000 の MOV 命令を実行している最中と考えてください。

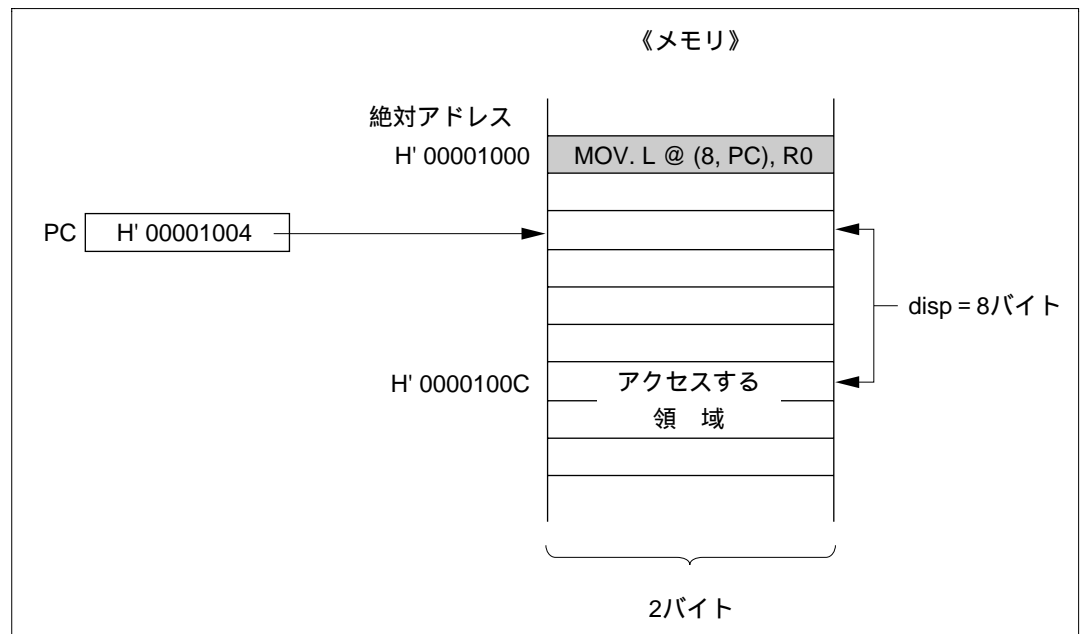


図 3.1 アドレス計算の例（通常）

(2) ディレイスロット命令を実行中

PC の値は、遅延分岐命令による分岐先アドレス+2 バイトとなります。

例

絶対アドレス H'00001000 の MOV 命令を実行している最中と考えてください

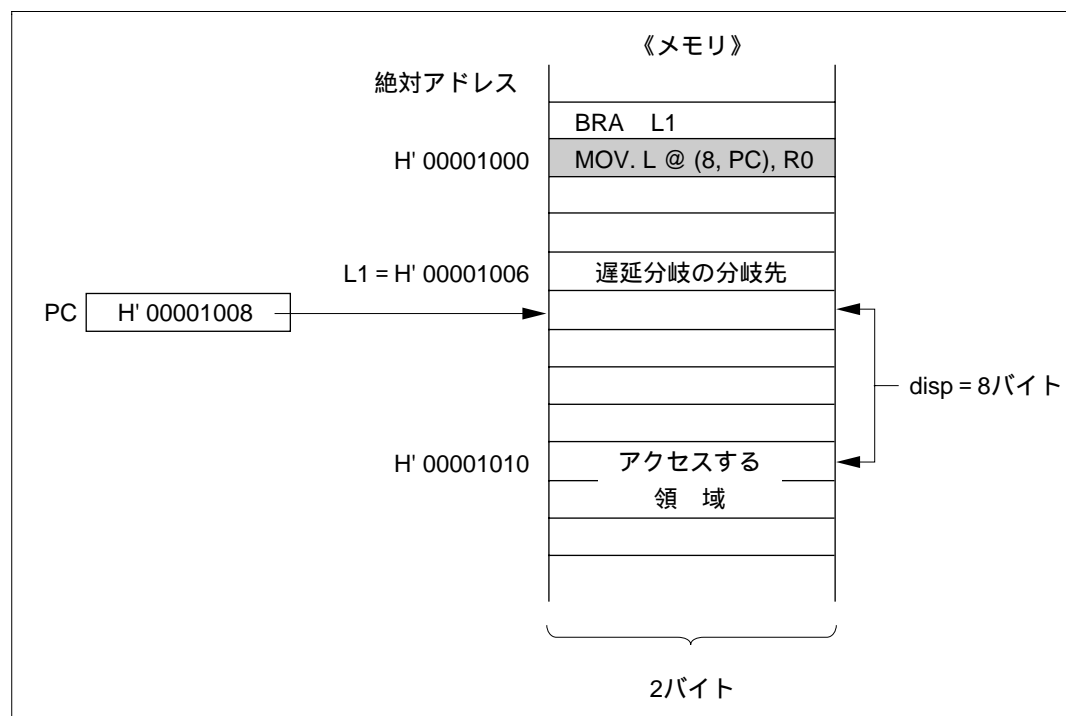


図 3.2 アドレス計算の例（分岐によって PC が変化する場合）

【補足】 オペランドがシンボル指定による PC 相対 (symbol) である場合には、アセンブラは PC の値を考慮した上でディスプレースメントを逆算し、オブジェクトコードを生成します。

(3) MOV.L @(disp,PC),Rn、および MOVA @(disp,PC),R0 のどちらかを実行中

マイコンは、PC の値が 4 の倍数でないとき、下位 2 ビットを切り捨てて 4 の倍数に補正し、アドレスを計算します。

例1 マイコンが PC を補正する場合

H'00001002 番地の MOV 命令を実行している最中と考えてください。

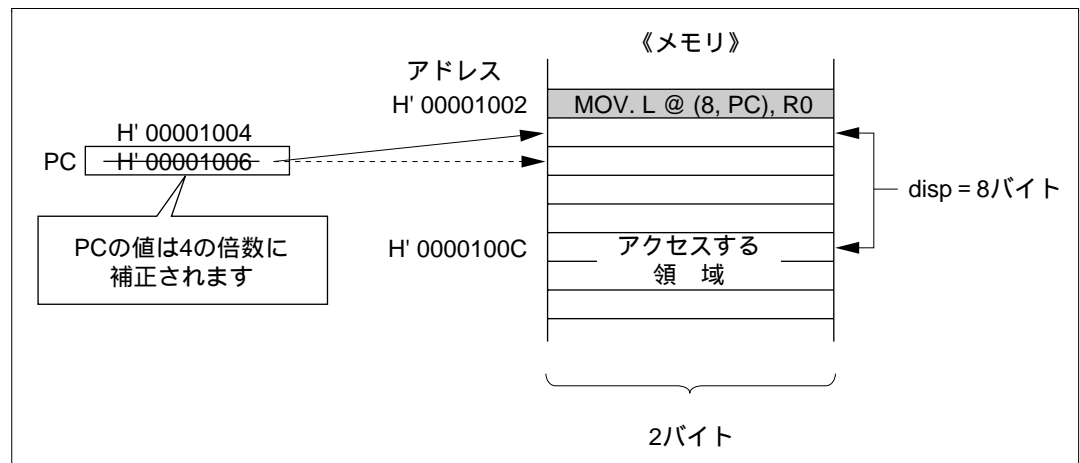


図 3.3 アドレス計算の例（マイコンが PC を補正する場合）

例2 マイコンが PC を補正しない場合

H'00001000 番地の MOV 命令を実行している最中と考えてください。

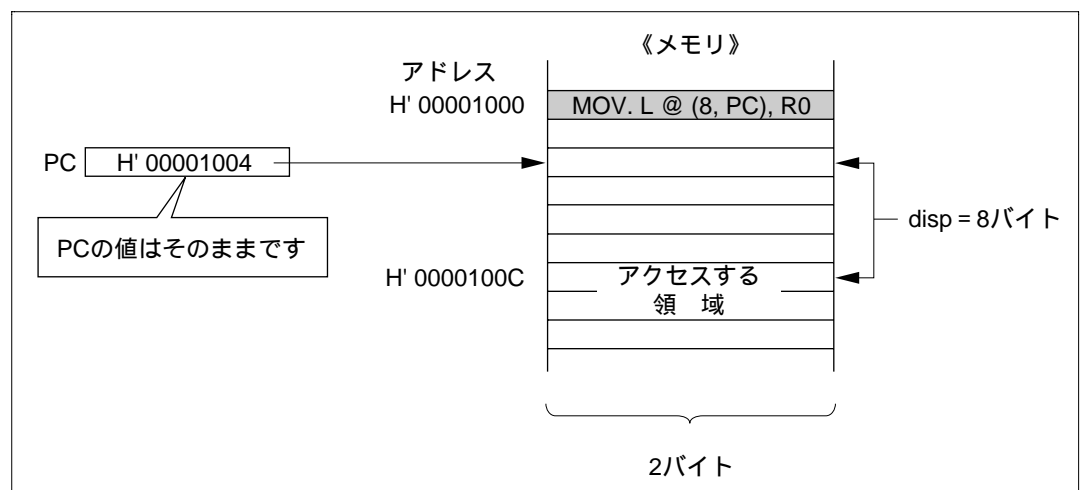


図 3.4 アドレス計算の例（マイコンが PC を補正しない場合）

【補足】 オペランドがシンボル指定による PC 相対（symbol）である場合には、アセンブラは PC の値を考慮した上でディスプレースメントを逆算し、オブジェクトコードを生成します。

4. DSP 命令

4.1 プログラムの要素

4.1.1 ソースステートメント

SH-DSP の命令は、実行命令と DSP 命令に分類できます。DSP 命令は、DSP レジスタを操作する命令です。DSP 命令は、実行命令とは異なる命令体系を持ち、記述方法も異なります。

DSP 命令では、1 ステートメント内に複数のオペレーションを記述することができます。DSP 命令のオペレーションには、以下の種類があります。

(1) DSP 演算オペレーション

DSP レジスタ間の演算を指定するオペレーションで、以下のものがあります。

PABS、PADD、PADDC、PAND、PCLR、PCMP、PCOPY、PDEC、PDMSB、PINC、PLDS、PMULS、PNEG、POR、PRND、PSHA、PSHL、PSTS、PSUB、PSUBC、PXOR

(2) X データ転送オペレーション

DSP レジスタと X データメモリ間のデータ転送を指定するオペレーションで、以下のものがあります。

MOVX、NOPX

(3) Y データ転送オペレーション

DSP レジスタと Y データメモリ間のデータ転送を指定するオペレーションで、以下のものがあります。

MOVY、NOPY

(4) シングルデータ転送オペレーション

X データメモリ、Y データメモリに限定されない一般のメモリと DSP レジスタ間のデータ転送を指定するオペレーションで、以下のものがあります。

MOVS

【参照】 実行命令 言語編「3. 実行命令」

4.1.2 並列演算命令

並列演算命令は、DSP 演算と同時に DSP レジスタと X データメモリ、Y データメモリ 間のデータ転送を行なうことを指定します。命令のサイズは 32 ビットになります。

並列演算命令の構成を、以下に示します。

[ラベル][DSP 演算部][データ転送部][コメント]

(1) DSP 演算部の記述方法

DSP 演算部の構成を、以下に示します。

[コンディション] DSP 演算オペレーション オペランド [DSP 演算オペレーション オペランド]

(a) コンディション

コンディションは、並列動作命令を実行する条件を指定します。

コンディションには、次のものがあります。

DCT、DCF

DCT は、DC ビットが 1 の時に命令を実行することを指定します。

DCF は、DC ビットが 0 の時に命令を実行することを指定します。

(b) DSP 演算オペレーション

DSP 演算を指定します。DSP 演算オペレーションを 2 個指定できるのは、PADD と PMULS、PSUB と PMULS の組み合わせを指定するときだけです。

(2) データ転送部の記述方法

データ転送部の構成を、以下に示します。

[Xデータ転送オペレーション[オペランド]][Yデータ転送オペレーション[オペランド]]

X データメモリの転送と、Y データメモリの転送をこの順に指定します。

データ転送のオペレーションがNOPX またはNOPY の場合は、省略することができます。

コーディング例

<u>LABEL1:</u>	<u>PADD A0,M0,A0</u>	<u>PMULS X0,Y0,M0</u>	<u>MOVX.W @R4+,X0</u>	<u>MOVY.W @R6+,Y0</u>	<u>; DSP 命令</u>
ラベル	DSP 演算部		データ転送部		コメント

<u>DCT PINC X1,A1</u>	<u>MOVX.W @R4,X0</u>	<u>MOVY.W @R6+,Y0</u>
DSP 演算部	データ転送部	

<u>PCMP X1,M0</u>	<u>MOVX.W @R4,X0</u>	<u>; Y メモリ転送を省略</u>
DSP 演算部	データ転送部	コメント

4.1.3 データ転送命令

データ転送命令は、X データメモリの転送と Y データメモリの転送の組み合わせ、またはシングルデータ転送を指定します。

(1) X データメモリの転送と Y データメモリの転送の組み合わせ

X データメモリの転送と Y データメモリの転送の組み合わせの構成を、以下に示します。

[ラベル] X データ転送オペレーション[オペランド][Y データ転送オペレーション[オペランド][コメント]

X データメモリの転送と、Y データメモリの転送をこの順に指定します。データ転送のオペレーションが NOPX または NOPY の場合は、省略することができます。ただし、並列演算命令の場合と異なり、X データメモリの転送と Y データメモリの転送の両方を省略することはできません。

X データメモリの転送命令と Y データメモリの転送命令の記述例を、以下に示します。

コーディング例

```
LABEL2: MOVX.W @R4,X0          ; データ転送命令(Y データメモリ転送を省略)
        MOVX.W @R4,X0  MOVY.W @R6+,Y0
```

(2) シングルデータ転送命令

シングルデータ転送命令の構成を、以下に示します。

[ラベル][シングルデータ転送オペレーション オペランド][コメント]

MOVS 命令を指定します。

シングルデータ転送命令の記述例を、以下に示します。

コーディング例

```
LABEL3: MOVS.W @-R2,A0 ; シングルデータ転送
```

4.1.4 複数行にわたるソースステートメントの書き方

DSP 命令は、1 行に複数のオペレーションを記述することができるため、ソースステートメントが長くなり、プログラムが見にくくなります。そこで、DSP 命令では、オペランドを区切るカンマ (,) 以外にもオペランドとオペレーションの間に区切ることができます。

複数行にわたるソースステートメントは、次の (a) ~ (c) の手順で記述してください。

- (a) オペランドとオペレーションを切れ目として改行します。
- (b) すぐ次の行の 1 カラム目にプラス (+) を書きます。
- (c) そのプラスの後ろに、ソースステートメントの続きを書きます。

プラスの後ろに、空白またはタブを入れてもかまいません。

コーディング例

```

          PADD          A0 , M0 , X0
+          PMULS        A1 , Y1 , M0
+          MOVX          @R4 , X0
+          MOVS         @R6 , Y1

```

; 1 つのソースステートメントを 4 行にわたって書いた例です。

4.2 DSP 命令

4.2.1 DSP 演算命令

表 4.1 に DSP 演算命令のニーモニックを示します。

表 4.1 DSP 演算命令ニーモニックの分類

分類	ニーモニック
DSP 算術演算命令	PADD, PSUB, PCOPY, PDMSB, PINC, PNEG, PMULS, PADDC, PSUBC, PCMP, PDEC, PABS, PRND, PCLR, PLDS, PSTS
DSP 論理演算命令	POR, PAND, PXOR
DSP シフト演算命令	PSHA, PSHL

(1) オペレーションサイズ

DSP 演算命令にオペレーションサイズは指定できません。

(2) アドレス形式

表 4.2 に DSP 演算命令のアドレス形式を示します。

表 4.2 DSP 演算命令のアドレス形式

アドレス形式	表記法
DSP レジスタ直接	Dp (DSP レジスタ名)
イミディエイトデータ	#imm

(a) DSP レジスタ直接

表 4.3 に、DSP レジスタ直接に指定できるレジスタを示します。

表の中で、Sx、Sy、Dz、Du、Se、Sf、Dg は「表 4.5 DSP 演算命令形式一覧」を参照してください。

表 4.3 DSP レジスタ直接に指定可能なレジスタ

		DSP レジスタ							
		A0	A1	M0	M1	X0	X1	Y0	Y1
Dp	Sx								
	Sy								
	Dz								
	Du								
	Se								
	Sf								
	Dg								

(b) イミディエイトデータ

イミディエイトデータは PSHA、PSHL 命令の第 1 オペランドに指定できる場合だけです。以下の値が指定できます。

・ 値の種類

定数、シンボル、または式を指定することができます。

・ シンボルの種類

イミディエイトデータには、相対シンボルや外部参照シンボルを含めて、任意のシンボルが指定できます。*

・ 値の範囲

表 4.4 に指定できる値の範囲を示します。

表 4.4 イミディエイトデータの値の範囲

命令	数値の範囲
PSHA 命令	H'FFFFFFE0 ~ H'00000020 (-32 ~ 32)
PSHL 命令	H'FFFFFFF0 ~ H'00000010 (-16 ~ 16)

【注】 * イミディエイトデータに相対シンボルや外部参照シンボルを指定した場合、リンクエディタでは、H'FFFFFFC0 ~ H'0000003F (-64 ~ 63) の範囲チェックとなります。

(3) 複数の DSP 演算命令の組み合わせ

PADD 命令と PMULS 命令、または PSUB 命令と PMULS 命令の組み合わせを指定することができます。この命令の組み合わせは、本来一つの DSP 演算命令ですが、プログラムを読み易くするために、PADD 命令または PSUB 命令のオペランドと PMULS 命令のオペランドを分割して記述できるようにしているものです。

複数の DSP 演算命令の組み合わせの例を、以下に示します。

コーディング例

```
PADD A0,M0,A0 PMULS X0,Y0,M0 NOPX MOVY.W @R6+,Y0
PSUB A1,M1,A1 PMULS X1,Y1,M1 MOVX @R4+,X0 NOPY
```

【注意】 複数の DSP 演算命令の組み合わせで、ディスティネーションレジスタとして同一のレジスタを指定した場合ウォーニング 701 になります。

```
PADD A0,M0,A0 PMULS X0,Y0,A0 ウォーニング 701
```

(4) コンディション付き DSP 演算命令

DSP 演算命令には DSR レジスタの DC ビットの状態により、実行するかどうかを選択できるものがあります。コンディション DCT は DC ビットが 1 のときだけ命令を実行します。コンディション DCF は DC ビットが 0 のときだけ命令を実行します。

コンディションを付けられる DSP 演算命令には、次のものがあります。

PADD、PAND、PCLR、PCOPY、PDEC、PDMSB、PINC、PLDS、PNEG、POR、PSHA、PSHL、PSTS、PSUB、PXOR

(5) DSP 演算命令一覧

表 4.5 に、DSP 演算命令の一覧を示します。表中の S_x 、 S_y 、 D_z 、 D_u 、 S_e 、 S_f 、 D_g のところに指定できるレジスタは、「表 4.3 DSP レジスタ直接に割り当て可能なレジスタ」を参照してください。

表 4.5 DSP 演算命令一覧

ニーモニック	アドレス形式	ニーモニック	アドレス形式
PABS	S_x, D_z	-	-
PABS	S_y, D_z	-	-
PADD	S_x, S_y, D_z	-	-
PADD	S_x, S_y, D_u	PMULS	S_e, S_f, D_g
PADDC	S_x, S_y, D_z	-	-
PAND	S_x, S_y, D_z	-	-
PCLR	D_z	-	-
PCMP	S_x, S_y	-	-
PCOPY	S_x, D_z	-	-
PCOPY	S_y, D_z	-	-
PDEC	S_x, D_z	-	-
PDEC	S_y, D_z	-	-
PDMSB	S_x, D_z	-	-
PDMSB	S_y, D_z	-	-
PINC	S_x, D_z	-	-
PINC	S_y, D_z	-	-
PLDS	$D_z, MACH$	-	-
PLDS	$D_z, MACL$	-	-
PMULS	S_e, S_f, D_g	-	-
PNEG	S_x, D_z	-	-
PNEG	S_y, D_z	-	-
POR	S_x, S_y, D_z	-	-
PRND	S_x, D_z	-	-
PRND	S_y, D_z	-	-
PSHA	$\#imm, D_z$	-	-
PSHA	S_x, S_y, D_z	-	-
PSHL	$\#imm, D_z$	-	-
PSHL	S_x, S_y, D_z	-	-
PSTS	$MACH, D_z$	-	-
PSTS	$MACL, D_z$	-	-
PSUB	S_x, S_y, D_z	-	-
PSUB	S_x, S_y, D_u	PMULS	S_e, S_f, D_g
PSUBC	S_x, S_y, D_z	-	-
PXOR	S_x, S_y, D_z	-	-

4.2.2 データ転送命令

(1) ニーモニック

データ転送命令にはデュアルメモリ転送とシングルメモリ転送があります。デュアルメモリ転送は、XメモリとDSPレジスタ間、YメモリとDSPレジスタ間のデータ転送を同時に行います。シングルメモリ転送では、任意のメモリとDSPレジスタ間のデータ転送を行います。

表 4.6 にデータ転送命令のニーモニック一覧を示します。

表 4.6 データ転送命令のニーモニック一覧

分類		ニーモニック
デュアルメモリ転送	X メモリ転送	NOPX MOVX
	Y メモリ転送	NOPY MOVY
シングルメモリ転送		MOVS

(2) オペレーションサイズ

NOPX、NOPY 命令は、サイズ指定できません。

MOVX、MOVY 命令は、ワードサイズ (.W) の指定しかできません。サイズを指定しないと、ワードサイズと解釈します。

MOVS 命令は、ワードサイズ (.W) とロングワードサイズ (.L) の両方が指定できます。サイズを指定しないと、ロングワードサイズと解釈します。

(3) アドレス形式

表 4.7 に、データ転送命令で指定可能なアドレス形式を示します。

表 4.7 データ転送命令のアドレス形式

アドレス形式	表記法
DSP レジスタ直接	Dz
レジスタ間接	@Az
ポストインクリメントレジスタ間接	@Az+
インデックス付きレジスタ間接/ポストインクリメント	@Az+Iz
プリデクリメントレジスタ間接	@-Az

アドレス形式のうち、インデックス付きレジスタ間接/ポストインクリメントは、DSP のデータ転送命令に特有なアドレス形式です。この形式は、レジスタ Az の指す内容を参照した後、Az の内容を Iz の値だけインクリメントすることを示します。

(4) アドレス形式に指定可能なレジスタ

表 4.8 に DSP レジスタ直接、レジスタ間接、ポストインクリメントレジスタ間接、インデックス付きレジスタ間接/ポストインクリメント、プリデクリメント間接に指定可能なレジスタを示します。

表の中で、Dx、Dy、Ds、Da、Ax、Ay、As、Ix、Iy、Is については、「表 4.9 データ転送命令一覧」を参照してください。

表 4.8 データ転送命令のアドレス形式に指定可能なレジスタ

		汎用レジスタ								DSP レジスタ									
		R2	R3	R4	R5	R6	R7	R8	R9	A0	A1	M0	M1	X0	X1	Y0	Y1	A0G	A1G
Dz	Dx																		
	Dy																		
	Ds																		
	Da																		
Az	Ax																		
	Ay																		
	As																		
Iz	Ix																		
	Iy																		
	Is																		

【注意】 同一命令内の DSP 演算命令とデータ転送命令の間で同一のレジスタをディスティネーションに指定した場合はウォーニング 703 になります。

PADD A0,M0,Y0 NOPX MOVY.W @R6+,Y0 ウォーニング 703

(5) データ転送命令一覧

表 4.9 にデータ転送命令の一覧を示します。

表中の Dx、Dy、Ds、Da、Ax、Ay、As、Ix、Iy、Is のところに指定できるレジスタは、「表 4.8 データ転送命令のアドレス形式に指定可能なレジスタ」を参照してください。

表 4.9 データ転送命令一覧

種別	ニーモニック	アドレス形式
X データ転送命令	NOPX	—
	MOVX.W	@Ax, Dx
	MOVX.W	@Ax+, Dx
	MOVX.W	@Ax+lx, Dx
	MOVX.W	Da, @Ax
	MOVX.W	Da, @Ax+
	MOVX.W	Da, @Ax+lx
Y データ転送命令	NOPY	—
	MOVY.W	@Ay, Dy
	MOVY.W	@Ay+, Dy
	MOVY.W	@Ay+ly, Dy
	MOVY.W	Da, @Ay
	MOVY.W	Da, @Ay+
	MOVY.W	Da, @Ay+ly
シングルデータ転送命令	MOVS.W	@-As, Ds
	MOVS.W	@As, Ds
	MOVS.W	@As+, Ds
	MOVS.W	@As+ls, Ds
	MOVS.W	Ds, @-As
	MOVS.W	Ds, @As
	MOVS.W	Ds, @As+
	MOVS.W	Ds, @As+ls
	MOVS.L	@-As, Ds
	MOVS.L	@As, Ds
	MOVS.L	@As+, Ds
	MOVS.L	@As+ls, Ds
	MOVS.L	Ds, @-As
	MOVS.L	Ds, @As
	MOVS.L	Ds, @As+
	MOVS.L	Ds, @As+ls

5. アセンブラ制御命令

5.1 アセンブラ制御命令の概要

アセンブラ制御命令は、アセンブラが解釈し、実行する命令です。

表 5.1 に、アセンブラ制御命令の一覧を示します。

表 5.1 アセンブラ制御命令一覧（その 1）

分類	ニーモニック	機能
CPU に関するもの	.CPU	CPU 種別を指定する
セクションまたは ロケーションカウンタ に関するもの	.SECTION	セクションを宣言する
	.ORG	ロケーションカウンタ値を設定する
	.ALIGN	ロケーションカウンタ値を補正する
シンボルに関するもの	.EQU	シンボルに値を設定する（再設定が不可能）
	.ASSIGN	シンボルに値を設定する（再設定が可能）
	.REG	レジスタ別名を定義する
	.FREG	浮動小数点レジスタ名を定義する
データまたはデータ領域を 確保するもの	.DATA	整数データを確保する
	.DATAB	整数データブロックを確保する
	.SDATA	文字列データを確保する
	.SDATAB	文字列データブロックを確保する
	.SDATAC	計数付き文字列データを確保する
	.SDATAZ	ゼロ終端文字列データを確保する
	.FDATA	浮動小数点データを確保する
	.FDATAB	浮動小数点データブロックを確保する
	.XDATA	固定小数点データを確保する
	.RES	データ領域を確保する
	.SRES	文字列データ領域を確保する
	.SRESC	計数付き文字列データ領域を確保する
	.SRESZ	ゼロ終端文字列データ領域を確保する
	.FRES	浮動小数点データ領域を確保する
外部定義または外部参照に に関するもの	.EXPORT	外部定義シンボルを宣言する
	.IMPORT	外部参照シンボルを宣言する
	.GLOBAL	外部参照シンボルまたは外部定義シンボルを宣言する
オブジェクトモジュールに に関するもの	.OUTPUT	オブジェクトモジュールの出力を制御する
	.DEBUG	シンボルデバッグ情報の部分出力を制御する
	.ENDIAN	エンディアン種別を指定する
	.LINE	行番号を変更する

表 5.1 アセンブラ制御命令一覧（その2）

分類	ニーモニック	機能
アセンブルリストに関するもの	.PRINT	アセンブルリストの出力を制御する
	.LIST	ソースプログラム・リストの部分出力を制御する
	.FORM	アセンブルリストの行数と桁数を設定する
	.HEADING	ソースプログラム・リストのヘッダを設定する
	.PAGE	ソースプログラム・リストを改ページする
	.SPACE	ソースプログラム・リストに空行を出力する
その他	.PROGRAM	オブジェクトモジュール名を設定する
	.RADIX	基数のない整数定数を何進数とするかを指定する
	.END	ソースプログラムの終わりを宣言する

5.2 アセンブラ制御命令リファレンス

5.2.1 CPU に関するアセンブラ制御命令

CPU に関するアセンブラ制御命令には、次のものがあります。

.CPU	CPU 種別を指定します。
------	---------------

.CPU CPU 種別を指定する

書式

.CPU CPU 種別

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .CPU を記述します。

(3) オペランド

CPU 種別を記述します。

指定内容	CPU 種別
SH1	SH-1 用にあセンブルする。
SH2	SH-2 用にあセンブルする。
SH2E	SH-2E 用にあセンブルする。
SH3	SH-3 用にあセンブルする。
SH3E	SH-3E 用にあセンブルする。
SH4	SH-4 用にあセンブルする。
SHDSP	SH-DSP 用にあセンブルする。
SH3DSP	SH3-DSP 用にあセンブルする。

指定内容によって、CPU の種別が決まります。

指定を省略した場合は、SHCPU 環境変数で設定した CPU 種別が有効となります。

【参照】 SHCPU 環境変数 操作編 「1.3 SHCPU 環境変数」

解説

(1) .CPU は、アセンブルするソースプログラムの対象とする CPU を指定するアセンブラ制御命令です。

アセンブラは、指定した CPU 用にあセンブルします。

(2) .CPU 種別は次の通りです。

SH1	SH-1 用
SH2	SH-2 用
SH2E	SH-2E 用
SH3	SH-3 用
SH3E	SH-3E 用
SH4	SH-4 用
SHDSP	SH-DSP 用
SH3DSP	SH3-DSP 用

(3) .CPU は、ソースプログラムの最初に記述してください。アセンブラリストに関する制御命令を除いて、ソースプログラムの最初でない場合はエラーとなります。

(4) .CPU の指定は、1 回限り有効です。

(5) CPU 種別の優先順位は、-CPU、.CPU、SHCPU 環境変数の順となります。

コーディング例

```
.CPU          SH2

SECTION A, CODE, ALIGN=4
MOV.L R0, R1
MOV.L R0, R2
```

SH2 用にあsembleします。

【参照】 -CPU 操作編 「2.2.1 CPU に関するコマンドライン・オプション」
-CPU

5.2.2 セクションまたはロケーションカウンタに関するアセンブラ制御命令

セクションまたはロケーションカウンタに関するアセンブラ制御命令には、次のものがあります。

.SECTION	セクションを宣言します。
.ORG	ロケーションカウンタ値を設定します。
.ALIGN	ロケーションカウンタ値を、境界調整数の倍数に補正します。

.SECTION セクションを宣言する

書式

```
.SECTION   セクション名 [,セクション属性 [, { LOCATE = 先頭アドレス
                                           ALIGN = 境界調整数 } ]]
```

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .SECTION を記述します。

(3) オペランド

(a) 第1オペランド : セクション名

セクション名は、シンボルの一種です。

【参照】セクション名のつけ方 言語編 「1.3.2 シンボルの名づけ方」

(b) 第2オペランド : セクション属性

指定内容	セクションの種類
CODE	コードセクション
DATA	データセクション
STACK	スタックセクション
COMMON	コモンセクション
DUMMY	ダミーセクション

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、セクションの用途別の種類が決まります。

指定を省略すると、コードセクションになります。

(c) 第3オペランド：先頭アドレス または 境界調整数

指定内容	セクションの種類
LOCATE=先頭アドレス	絶対アドレスセクション
ALIGN=境界調整数	相対アドレスセクション
指定なし	相対アドレスセクション（境界調整数 = 4）

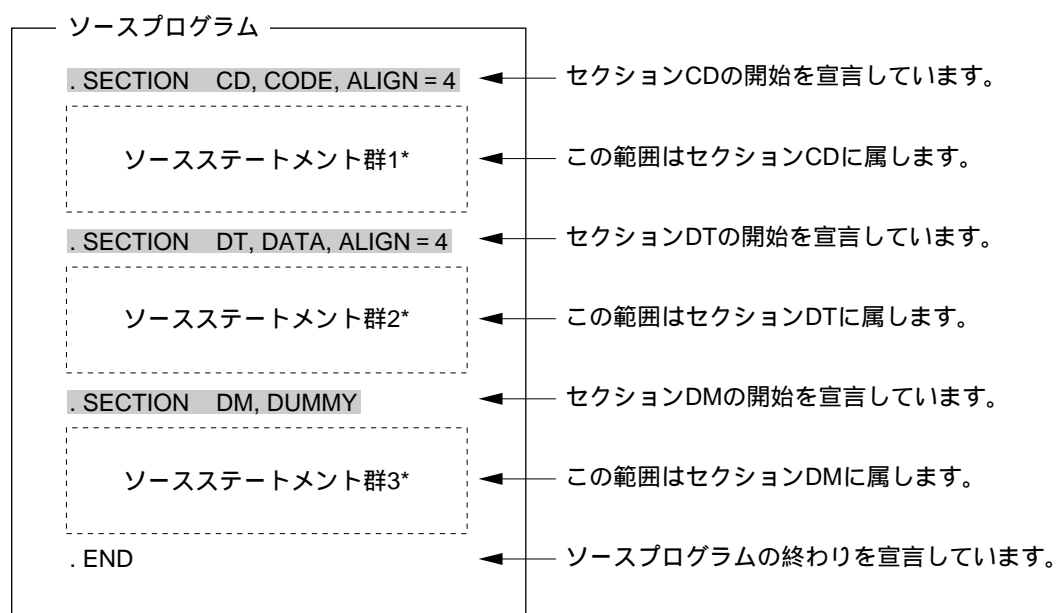
指定内容によって、絶対アドレスセクションになるか、相対アドレスセクションになるかが決まります。

解説

(1) .SECTION は、セクションを宣言するアセンブラ制御命令です。

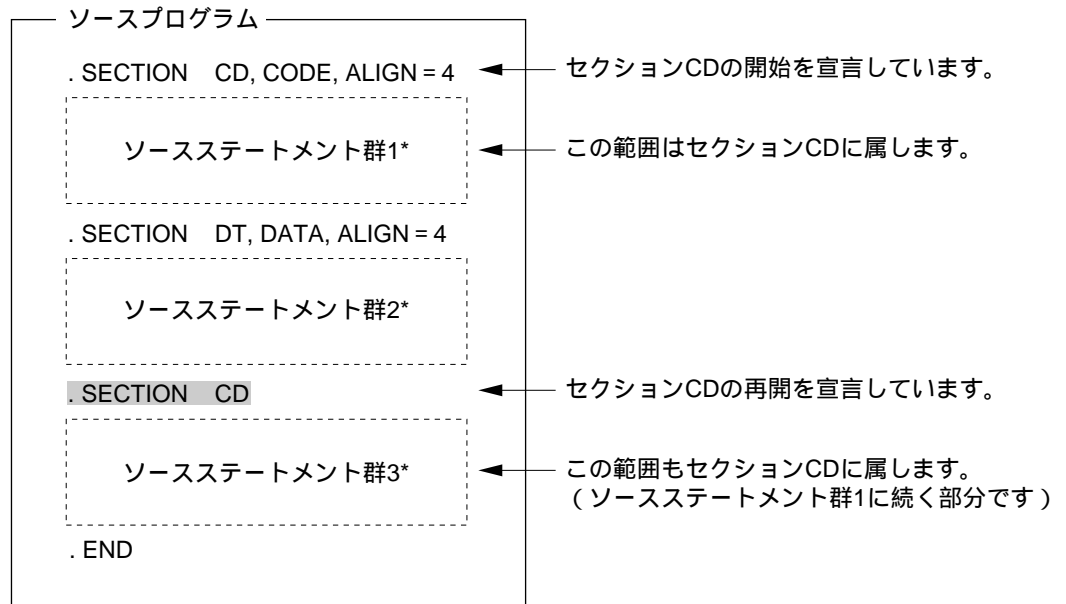
セクションとはプログラムの1区切りであり、リンケージの処理単位です。

セクションの宣言について、簡単な例をあげて説明します。



【注】* この例では、「ソースステートメント群1～3」に
.SECTIONが現れないことを仮定しています。

- (2) すでに宣言してあるセクションを同じファイルの中で再び宣言し、再開できます。
セクションの再開について、簡単な例をあげて説明します。



【注】* この例では、「ソースステートメント群1～3」に
. SECTIONが現れないことを仮定しています。

【注意】 セクションを再開する場合、第2オペランドと第3オペランドの指定を省略してください（そのセクションを開始したときの指定が、そのまま有効です）。

- (3) 絶対アドレスセクションを開始するときには、第3オペランドに「LOCATE = 先頭アドレス」を指定してください。先頭アドレスは、そのセクションが始まる絶対アドレスです。

先頭アドレスは、次のように指定します。

- ・絶対値を指定する。
- かつ、
- ・前方参照シンボルを使わずに指定する。

先頭アドレスとして許される値は、H'00000000 ~ H'FFFFFFFF です。

(10進表現では0 ~ 4,294,967,295)

- (4) 相対アドレスセクションを開始するときには、第3オペランドに「ALIGN = 境界調整数」を指定してください。リンケージエディタは、そのセクションの先頭が境界調整数の倍数にあたる絶対アドレスにくるように調整します。

境界調整数は、次のように指定します。

- ・絶対値を指定する。
かつ、
- ・前方参照シンボルを使わずに指定する。

境界調整数として許される値は、2のべき乗 ($2^0, 2^1, 2^2, \dots, 2^{31}$) です。

コードセクションの場合は、4以上の値 ($2^2, 2^3, 2^4, \dots, 2^{31}$) です。

- (5) 次のいずれかの場合に、アセンブラはデフォルトセクションを用意します。

- ・セクションを宣言しないうちに、実行命令または拡張命令を記述している。
- ・セクションを宣言しないうちに、データを確保するアセンブラ制御命令を記述している。
- ・セクションを宣言しないうちに、.ALIGN アセンブラ制御命令を記述している。
- ・セクションを宣言しないうちに、.ORG アセンブラ制御命令を記述している。
- ・セクションを宣言しないうちに、ロケーションカウンタを参照している。
- ・セクションを宣言しないうちに、ラベルだけの行を記述している。

デフォルトセクションとは、次のようなセクションです。

- ・セクション名 : P
- ・セクションの種類 : コードセクション
相対アドレスセクション (境界調整数 = 4)

コーディング例

```

    . ALIGN      4
    . DATA. L   H' 11111111
    ~~~~~
SECTION CD, CODE, ALIGN = 4

    MOV         R0, R1
    MOV         R0, R2
    ~~~~~
SECTION DT, DATA, LOCATE = H' 00001000
X1:  . DATA. L   H' 22222222
     . DATA. L   H' 33333333
     ~~~~~
    . END

```

; この範囲は、デフォルトセクションPに属します。
 ; デフォルトセクションPはコードセクションであり、
 ; 境界調整数 = 4の相対アドレスセクションです。

; この範囲は、セクションCDに属します。
 ; セクションCDはコードセクションであり、
 ; 境界調整数 = 4の相対アドレスセクションです。

; この範囲は、セクションDTに属します。
 ; セクションDTは、データセクションであり、
 ; 先頭アドレス = H' 00001000の絶対アドレス
 ; セクションです。

【注】この例では、「～」の部分に .SECTION が現れないことを仮定しています。

.ORG ロケーションカウンタ値を設定する

書式

.ORG ロケーションカウンタ値

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .ORG を記述します。

(3) オペランド

ロケーションカウンタに設定したい値を記述します。

解説

(1) .ORG は、ロケーションカウンタ値を設定するアセンブラ制御命令です。

.ORG によって、実行命令やデータを特定のアドレスに配置できます。

(2) ロケーションカウンタ値は、次のように指定します。

- ・絶対値またはセクション内のアドレス値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

ロケーションカウンタ値として許される値は、 H'00000000 ~ H'FFFFFFFF です。

(10進表現では0 ~ 4,294,967,295)

絶対アドレスセクションで指定する場合は、

ロケーションカウンタ値 セクション先頭アドレス (符号なしの値として比較)

となるようにしてください。

(3) アセンブラは、ロケーションカウンタ値を次のように見なします。

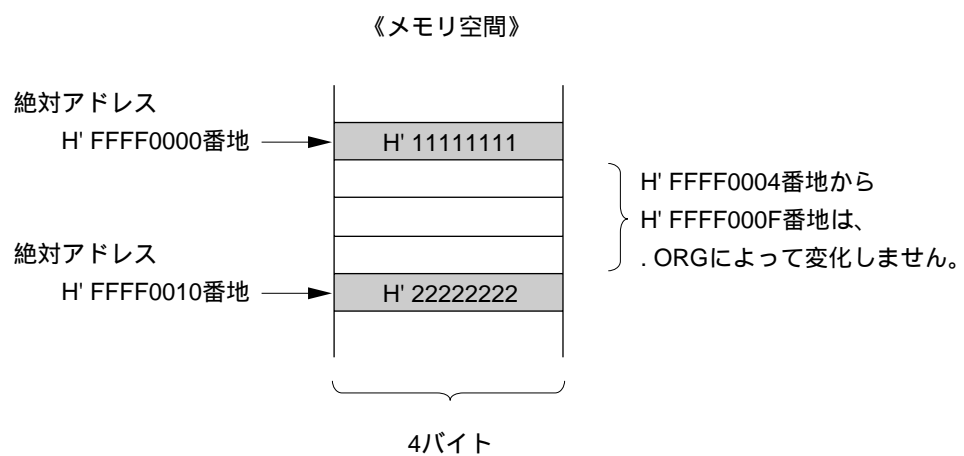
- ・絶対アドレスセクション内では、絶対アドレス値と見なす。

- ・相対アドレスセクション内では、相対アドレス値 (セクション先頭からの相対的な距離) と見なす。

コーディング例

```
.SECTION DT,DATA,LOCATE=H'FFFF0000
.DATA.L H'11111111
.ORG H'FFFF0010 ;ロケーションカウンタ値を設定しています。
.DATA.L H'22222222 ;整数データ H'22222222 を、絶対アドレスの
~ ;H'FFFF0010 番地に確保しています。
```

・コーディング例の図解



.ALIGN ロケーションカウンタ値を補正する

書式

.ALIGN 境界調整数

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .ALIGN を記述します。

(3) オペランド

境界調整数 (ロケーションカウンタ値を調整する基準) として設定したい値を記述します。

解説

(1) .ALIGN は、ロケーションカウンタ値を境界調整数の倍数に補正するアセンブラ制御命令です。 .ALIGN によって、実行命令やデータを特定の境界 (アドレスの区切り) に配置できます。

(2) ロケーションカウンタ値は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

境界調整数として許される値は、2 のべき乗 ($2^0, 2^1, 2^2, \dots, 2^{31}$) です。

(3) 相対セクションで .ALIGN を使用する場合、

.SECTION で指定する境界調整数 .ALIGN で指定する境界調整数

となるようにしてください。

(4) コードセクションに `.ALIGN` を記述すると、アセンブラは `NOP` 命令のオブジェクトコード*をメモリ上に埋めこみ、ロケーションカウンタ値を補正します。半端なバイトサイズの領域には、`H'09` を埋めこみます。

データセクション、ダミーセクションまたはスタックセクションに `.ALIGN` を記述すると、アセンブラは単にロケーションカウンタ値を補正するだけであり、メモリ上にオブジェクトコードを埋めこみません。

【注】* このようなオブジェクトコードは、アセンブルリスト上には表れません。

コーディング例

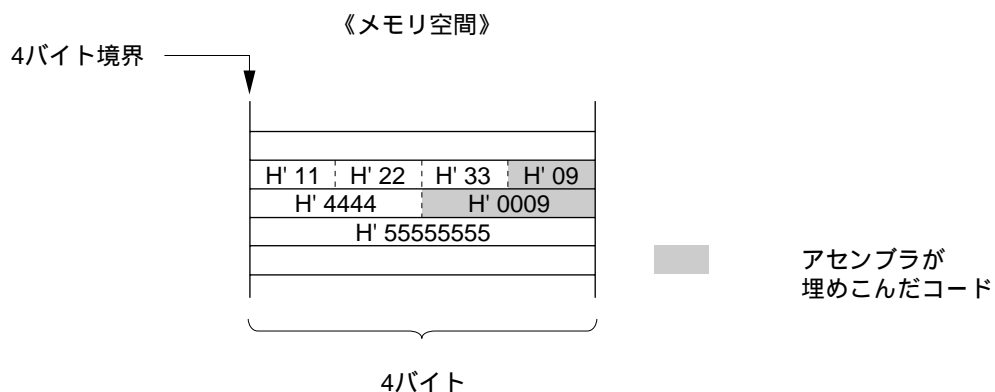
```
.SECTION P,code
~

.DATA.B H'11
.DATA.B H'22
.DATA.B H'33
.ALIGN 2 ;ロケーションカウンタ値を2の倍数に補正しています。
.DATA.W H'4444

.ALIGN 4 ;ロケーションカウンタ値を4の倍数に補正しています。
.DATA.L H'55555555
~
```

・コーディング例の図解

バイトサイズの整数データ `H'11` が、もともと4バイト境界に位置するものと仮定します。アセンブラは、下図のようにオブジェクトコードを埋めこんで境界調整します。



5.2.3 シンボルに関するアセンブラ制御命令

シンボルに関するアセンブラ制御命令には、次のものがあります。

.EQU	シンボルに値を設定します。
.ASSIGN	シンボルに値を設定または再設定します。
.REG	レジスタ別名を定義します。
.FREG	浮動小数点レジスタ名を定義します。

.EQU シンボルに値を設定する（再設定は不可能）

書式

シンボル [:] .EQU シンボル値

ステートメントの要素

(1) ラベル

値を設定したいシンボルを記述します。

(2) オペレーション

ニーモニック.EQU を記述します。

(3) オペランド

シンボルに設定したい値を記述します。

解説

(1) .EQU は、シンボルに値を設定するアセンブラ制御命令です。

.EQU で定義したシンボルは、再定義できません。

(2) シンボル値は、次のように指定します。

- ・絶対値、アドレス値、外部参照シンボル値* を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

シンボル値として許される値は、H'00000000 ~ H'FFFFFFFF です。

(10進表現では -2,147,483,648 ~ 4,294,967,295)

【注】* 外部参照値、外部参照値 + 定数、外部参照値 - 定数が記述できます。

コーディング例

~

```

X1:  .EQU    10           ;X1 の値は 10 になります。
X2:  .EQU    20           ;X2 の値は 20 になります。
      CMP/EQ  #X1,R0      ;CMP/EQ #10,R0 と同じです。
      BT      LABEL1
      CMP/EQ  #X2,R0      ;CMP/EQ #20,R0 と同じです。
      BT      LABEL2
~

```


.ASSIGN シンボルに値を設定する（再設定が可能）

書式

シンボル [:] .ASSIGN シンボル値

ステートメントの要素

（１）ラベル

値を設定したいシンボルを記述します。

（２）オペレーション

ニーモニック .ASSIGN を記述します。

（３）オペランド

シンボルに設定したい値を記述します。

解説

- （１）.ASSIGN は、シンボルに値を設定するアセンブラ制御命令です。
.ASSIGN で定義したシンボルは、.ASSIGN で再定義できます。

- （２）シンボル値は、次のように指定します。
- ・絶対値またはアドレス値を指定する。
- かつ、
- ・前方参照シンボルを使わずに指定する。

シンボル値として許される値は、H'00000000 ~ H'FFFFFFFF です。

（10進表現では -2,147,483,648 ~ 4,294,967,295）

- （３）.ASSIGN による定義は、定義の箇所よりも前方で有効です。

- （４）.ASSIGN で定義したシンボルには、次の使用上の制限があります。
- ・外部参照または外部定義できない。
 - ・デバッガで参照できない。

コーディング例

~

```
X1:  .ASSIGN  1
X2:  .ASSIGN  2
      CMP/EQ  #X1,R0    ;CMP/EQ #1,R0 と同じです。
      BT      LABEL1
      CMP/EQ  #X2,R0    ;CMP/EQ #2,R0 と同じです。
      BT      LABEL2
```

~

```
X1:  .ASSIGN  3
X2:  .ASSIGN  4
      CMP/EQ  #X1,R0    ;CMP/EQ #3,R0 と同じです。
      BT      LABEL3
      CMP/EQ  #X2,R0    ;CMP/EQ #4,R0 と同じです。
      BF      LABEL4
```

~

.REG レジスタ別名を定義する

書式

シンボル [:] .REG レジスタ名

または

シンボル [:] .REG (レジスタ名)

ステートメントの要素

(1) ラベル

レジスタ別名として定義したいシンボルを記述します。

(2) オペレーション

ニーモニック .REG を記述します。

(3) オペランド

別名をつきたいレジスタの名前を記述します。

解説

(1) .REG は、レジスタ別名を定義するアセンブラ制御命令です。

.REG で定義したレジスタ別名は、元のレジスタ名と全く同等に使用できます。

.REG で定義したレジスタ別名は、再定義できません。

(2) 別名をつけることができるのは、汎用レジスタ (R0 ~ R15, SP) だけです。

(3) .REG による定義は、定義の箇所よりも前方で有効です。

(4) .REG で定義したシンボルには、次の使用上の制限があります。

- ・外部参照または外部定義できない。
- ・シミュレータ・デバッガで参照できない。

コーディング例

~

```
MIN:  .REG    R10
MAX:  .REG    R11
      MOV     #0,MIN    ;MOV  #0,R10 同じです。
      MOV     #99,MAX   ;MOV  #99,R11 同じです。
      CMP/HS  MIN,R1
      BF      LABEL
      CMP/HS  R1,MAX
      BF      LABEL
```

~

.FREG 浮動小数点レジスタ名を定義する

書式

シンボル [:] .FREG 浮動小数点レジスタ名

または

シンボル [:] .FREG (浮動小数点レジスタ名)

ステートメントの要素

(1) ラベル

浮動小数点レジスタ名として定義したいシンボルを記述します。

(2) オペレーション

ニーモニック .FREG を記述します。

(3) オペランド

別名をつきたい浮動小数点レジスタの名前を記述します。

解説

(1) .FREG は、浮動小数点レジスタ名を定義するアセンブラ制御命令です。

.FREG で定義した浮動小数点レジスタ名は、元のレジスタ名と全く同等に使用できます。

.FREG で定義したレジスタ別名は、再定義できません。

(2) 別名をつけることができるのは、浮動小数点レジスタ FR_m (m = 0 ~ 15)、DR_n、XD_n (n = 0,2,4,6,8,10,12,14)、FV_i (i = 0,4,8,12) です。

(3) .FREG による定義は、定義の箇所よりも前方で有効です。

(4) .FREG で定義したシンボルには、次の使用上の制限があります。

- ・外部参照または外部定義できない。
- ・シミュレータ・デバッガで参照できない。

(5) .FREG は、CPU 種別が SH-2E、SH-3E、SH-4 のとき使用できます。

コーディング例

~

```
MAX:  .FREG      FR11
      FMOV      @FR1,MAX      ;FMOV @FR1,FR11 と同じです。
```

```
      FCMP/EQ   MAX,FR2      ;FCMP/EQ FR11,FR2 と同じです。
```

```
      BF        LABEL
```

~

5.2.4 データまたはデータ領域を確保するアセンブラ制御命令

データまたはデータ領域を確保するアセンブラ制御命令には、次のものがあります。

.DATA	整数データを確保します。
.DATAB	整数データブロックを確保します。
.SDATA	文字列データを確保します。
.SDATAB	文字列データブロックを確保します。
.SDATAC	計数付き文字列データを確保します。
.SDATAZ	ゼロ終端文字列データを確保します。
.FDATA	浮動小数点データを確保します。
.FDATAB	浮動小数点データブロックを確保します。
.XDATA	固定小数点データを確保します。
.RES	データ領域を確保します。
.SRES	文字列データ領域を確保します。
.SRESC	計数付き文字列データ領域を確保します。
.SRESZ	ゼロ終端文字列データ領域を確保します。
.FRES	浮動小数点データ領域を確保します。

.DATA 整数データを確保する

書式

[シンボル [:]] .DATA [.オペレーションサイズ] 整数データ [, 整数データ ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.DATA を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
B	バイト
W	ワード (2 バイト)
L	ロングワード (4 バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータのサイズが決まります。

指定を省略すると、ロングワードになります。

(3) オペランド

データとして確保したい値を記述します。

解説

(1) .DATA は、整数データをメモリ上に確保するアセンブラ制御命令です。

(2) 整数データには、相対値や前方参照シンボルを含めて、任意の値を指定できます。

(3) オペレーションサイズによって、指定できる整数データの範囲が異なります。

オペレーションサイズ	整数データの範囲*	
B	H'00000000 ~ H'000000FF	(0 ~ 255)
	H'FFFFFFF80 ~ H'FFFFFFF	(-128 ~ -1)
W	H'00000000 ~ H'0000FFFF	(0 ~ 65,535)
	H'FFFF8000 ~ H'FFFFFFF	(-32,768 ~ -1)
L	H'00000000 ~ H'FFFFFFF	(0 ~ 4,294,967,295)
	H'80000000 ~ H'FFFFFFF	(-2,147,483,648 ~ -1)

【注】* () 内は 10 進表現

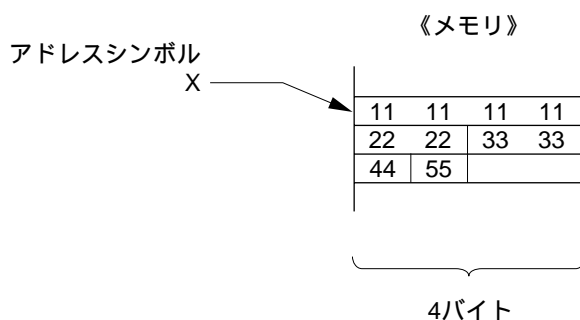
コーディング例

```

~
.ALIGN    4                ; ( ロケーションカウンタ値を補正しています )
X:  .DATA.L  H'11111111    ;
     .DATA.W  H'2222,H'3333 ; 整数データを確保しています。
     .DATA.B  H'44,H'55    ;
~

```

・コーディング例の図解



【注】データは16進数です。

.DATAB 整数データブロックを確保する

書式

[シンボル[:]] .DATAB [.オペレーションサイズ] ブロック数, 整数データ

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.DATAB を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
B	バイト
W	ワード (2 バイト)
L	ロングワード (4 バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータのサイズが決まります。

指定を省略すると、ロングワードになります。

(3) オペランド

(a) 第1オペランド : ブロック数

データの個数を記述します。

(b) 第2オペランド : 整数データ

データとして確保したい値を記述します。

解説

(1) .DATAB は、整数データを指定の数だけ連続して、メモリ上に確保するアセンブラ制御命令です。

(2) ブロック数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

整数データには、相対値や前方参照シンボルを含めて、任意の値を指定できます。

(3) オペレーションサイズによって、指定できるブロック数の範囲および整数データの範囲が異なります。

オペレーションサイズ	ブロック数の範囲 *
B	H'00000001 ~ H'FFFFFFFF (1 ~ 4,294,967,295)
W	H'00000001 ~ H'7FFFFFFF (1 ~ 2,147,483,647)
L	H'00000001 ~ H'3FFFFFFF (1 ~ 1,073,741,823)

オペレーションサイズ	整数データの範囲 *
B	H'00000000 ~ H'000000FF (0 ~ 255)
	H'FFFFFFF80 ~ H'FFFFFFFF (-128 ~ -1)
W	H'00000000 ~ H'0000FFFF (0 ~ 65,535)
	H'FFFF8000 ~ H'FFFFFFFF (-32,768 ~ -1)
L	H'00000000 ~ H'FFFFFFFF (0 ~ 4,294,967,295)
	H'80000000 ~ H'FFFFFFFF (-2,147,483,648 ~ -1)

【注】 * () 内は 10 進表現

コーディング例

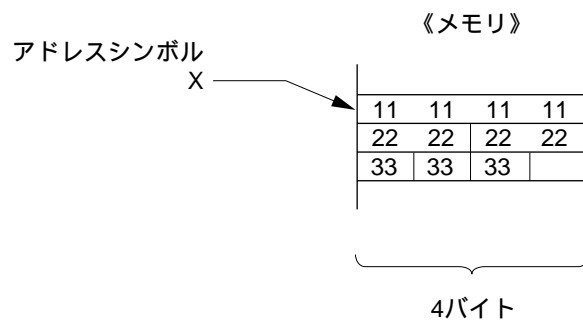
~

```
.ALIGN      4                ; (ロケーションカウンタ値を補正しています)
```

```
X:  .DATAB.L  1,H'11111111 ;  
    .DATAB.W  2,H'2222    ;整数データブロックを確保しています。  
    .DATAB.B  3,H'33      ;
```

~

・コーディング例の図解



【注】データは16進数です。

.SDATA 文字列データを確保する

書式

[シンボル [:]] .SDATA "文字列 " [, "文字列 " ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック `.SDATA` を記述します。

(3) オペランド

データとして確保したい文字列を記述します。

解説

(1) `.SDATA` は、文字列データをメモリ上に確保するアセンブラ制御命令です。

【参照】 文字列 言語編「1.7 文字列」

(2) 文字列に制御文字をつけ加えることができます。

記述形式は次のとおりです。

"文字列 " <制御文字の ASCII コード >

制御文字の ASCII コードは、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

コーディング例

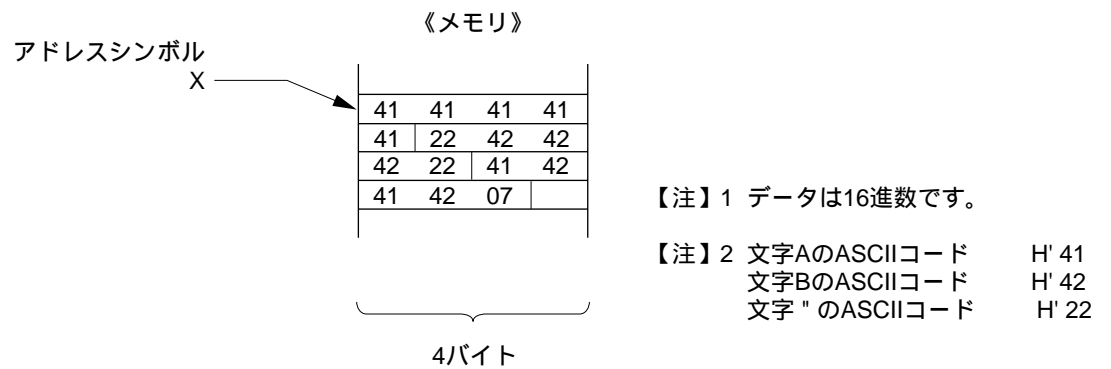
~

```
.ALIGN 4 ; (ロケーションカウンタ値を補正しています)
```

```
X: .SDATA "AAAAA" ; 文字列データを確保しています。
   .SDATA " " "BBB" " " ; 文字列にダブルコーテーションを含む例です。
   .SDATA "ABAB" <H' 07> ; 文字列に制御文字をつけ加えた例です。
```

~

・コーディング例の図解



.SDATAB 文字列データブロックを確保する

書式

[シンボル[:]] .SDATAB ブロック数,"文字列"

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック.SDATAB を記述します。

(3) オペランド

(a) 第1オペランド : ブロック数

文字列データの個数を記述します。

(b) 第2オペランド : 文字列

データとして確保したい文字列を記述します。

解説

- (1) .SDATAB は、文字列データを指定の数だけ連続して、メモリ上に確保するアセンブラ制御命令です。

【参照】 文字列 言語編「1.7 文字列」

- (2) ブロック数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

ブロック数には、1以上の値を指定してください。

ブロック数の上限値は、文字列データの長さによって決まります。

(文字列データの長さ×ブロック数 H'FFFFFFFF (4,294,967,295) バイト)

(3) 文字列に制御文字をつけ加えることができます。

記述形式は次のとおりです。

"文字列 "<制御文字の ASCII コード >

制御文字の ASCII コードは、次のように指定します。

- ・絶対値を指定する。
- かつ、
- ・前方参照シンボルを使わずに指定する。

コーディング例

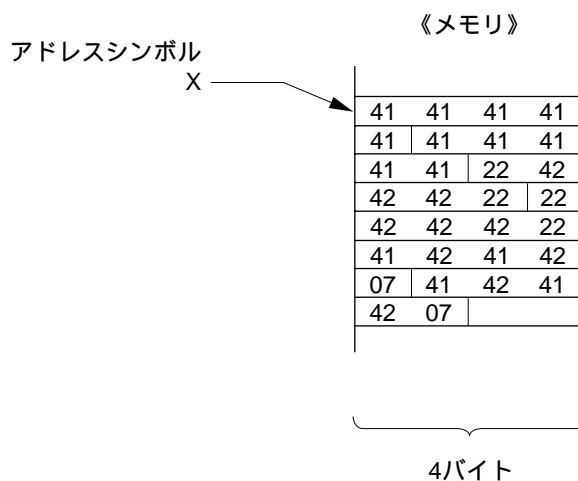
```

~
.ALIGN      4                      ; (ロケーションカウンタ値を補正しています)

X:  .SDATAB  2, "AAAAA"           ; 文字列データブロックを確保しています。
    .SDATAB  2, " " "BBB" " "     ; 文字列にダブルコーテーションを含む例です。
    .SDATAB  2, "ABAB"<H' 07>    ; 文字列に制御文字をつけ加えた例です。
~

```

・コーディング例の図解



【注】1 データは16進数です。

【注】2 文字AのASCIIコード H' 41
 文字BのASCIIコード H' 42
 文字 " のASCIIコード H' 22

.SDATAC 計数付き文字列データを確保する

書式

[シンボル [:]] .SDATAC "文字列 " [, "文字列 " ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック.SDATAC を記述します。

(3) オペランド

データとして確保したい文字列を記述します。

解説

- (1) .SDATAC は、計数付き文字列データをメモリ上に確保するアセンブラ制御命令です。

計数付き文字列とは、文字列の先頭に1バイトの計数をつけ加えたものです。

計数は、文字列データ（計数を含まないデータ）のサイズを、バイト単位で表します。

【参照】 文字列 言語編「1.7 文字列」

- (2) 文字列に制御文字をつけ加えることができます。

記述形式は次のとおりです。

"文字列 " <制御文字の ASCII コード >

制御文字の ASCII コードは、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

コーディング例

~

```
.ALIGN    4                ; (ロケーションカウンタ値を補正しています)
```

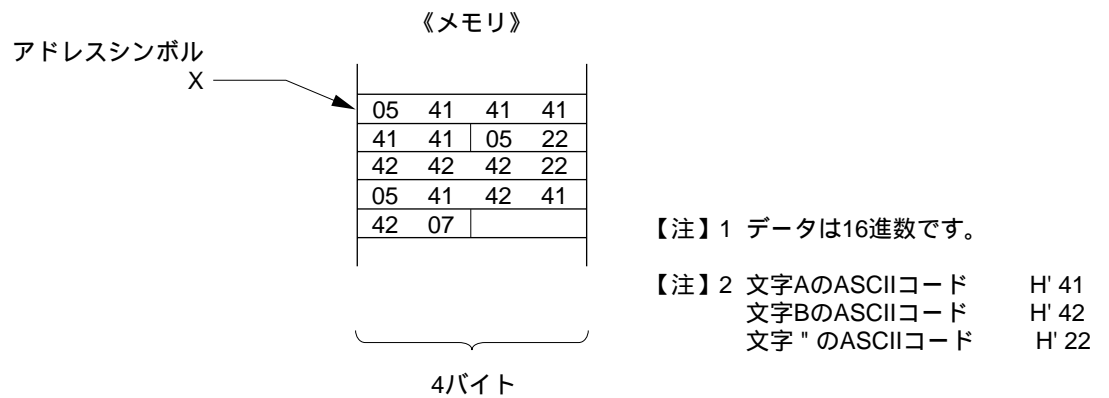
X: .SDATA "AAAAA" ; 計数付き文字列データを確保しています。

```
.SDATAC   " " "BBB" " "     ; 文字列にダブルコーテーションを含む例です。
```

```
.SDATAC   "ABAB" <H' 07> ; 文字列に制御文字をつけ加えた例です。
```

~

・コーディング例の図解



.SDATAZ ゼロ終端文字列データを確保する

書式

[シンボル [:]] .SDATAZ "文字列" [, "文字列" ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック.SDATAZ を記述します。

(3) オペランド

データとして確保したい文字列を記述します。

解説

(1) .SDATAZ は、ゼロ終端文字列データをメモリ上に確保するアセンブラ制御命令です。

ゼロ終端文字列とは、文字列の最後に1バイトのゼロをつけ加えたものです。

【参照】 文字列 言語編「1.7 文字列」

(2) 文字列に制御文字をつけ加えることができます。

記述形式は次のとおりです。

"文字列" <制御文字の ASCII コード>

制御文字の ASCII コードは、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

コーディング例

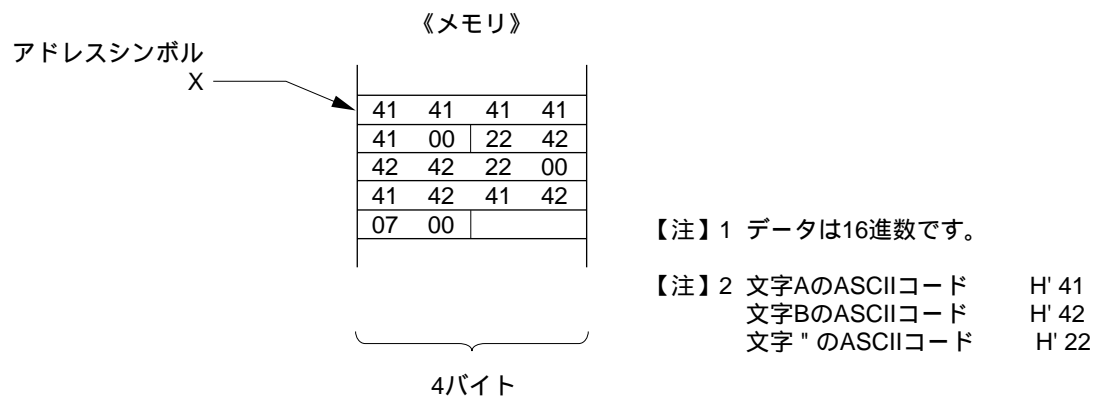
~

```
.ALIGN      4                ; (ロケーションカウンタ値を補正しています)
```

```
X:  .SDATAZ    "AAAAA"        ; ゼロ終端文字列データを確保しています。
    .SDATAZ    " " "BBB" " "   ; 文字列にダブルコーテーションを含む例です。
    .SDATAZ    "ABAB"<H' 07> ; 文字列に制御文字をつけ加えた例です。
```

~

・コーディング例の図解



.FDATA 浮動小数点データを確保する

書式

[シンボル [:]] .FDATA [.オペレーションサイズ] 浮動小数点定数
[, 浮動小数点定数 ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.FDATA を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
S	単精度 (4 バイト)
D	倍精度 (8 バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータのサイズが決まります。

指定を省略すると、単精度になります。

(3) オペランド

データとして確保したい値を記述します。

解説

(1) .FDATA は、浮動小数点定数データをメモリ上に確保するアセンブラ制御命令です。

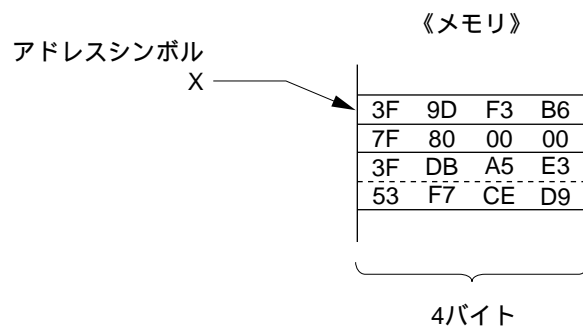
【参照】 浮動小数点定数 言語編「1.4.3 浮動小数点定数」

コーディング例

```
~
.ALIGN      4                ; (ロケーションカウンタ値を補正しています)

X:  .FDATA.S  F'1.234        ; 4 バイトの領域"3F9DF3B6" (F'1.234S)
                                   ; を確保しています。
      .FDATA.S  H'7F800000.S  ; 4 バイトの領域"7F800000" (H'7F800000.S)
                                   ; を確保しています。
      .FDATA.D  F'4.32D-1     ; 8 バイトの領域"3FDBA5E353F7CED9"
                                   ; (F'4.32D-1) を確保しています。
~
```

・コーディング例の図解



.FDATAB 浮動小数点データブロックを確保する

書式

[シンボル[:]] .FDATAB [.オペレーションサイズ] ブロック数,
浮動小数点定数

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.FDATAB を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
S	単精度 (4 バイト)
D	倍精度 (8 バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータのサイズが決まります。

指定を省略すると、単精度になります。

(3) オペランド

(a) 第1オペランド：ブロック数

データの個数を記述します。

(b) 第2オペランド：浮動小数点定数

データとして確保したい浮動小数点定数を記述します。

解説

- (1) .FDATAB は、浮動小数点定数データを指定の数だけ連続して、メモリ上に確保するアセンブラ制御命令です。

(2) ブロック数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボル、外部定義および、相対シンボルを使わずに指定する。

(3) オペレーションサイズによって、指定できるブロック数の範囲が異なります。

オペレーションサイズ	ブロック数の範囲*
S	H'00000001 ~ H'3FFFFFFF (1 ~ 1,073,741,823)
D	H'00000001 ~ H'1FFFFFFF (1 ~ 536,870,911)

【注】* () 内は 10 進表現

【参照】 浮動小数点定数 言語編「1.4.3 浮動小数点定数」

コーディング例

```

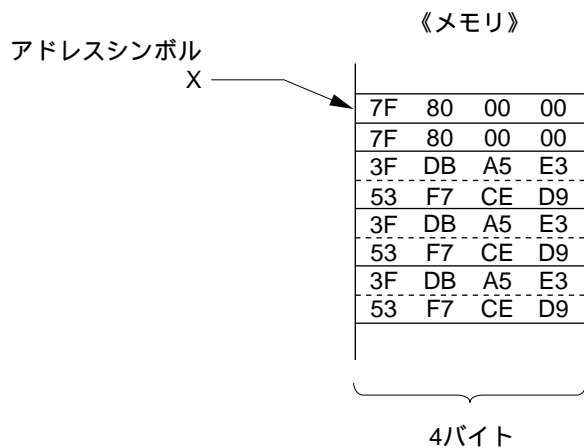
~
.ALIGN      4                ; (ロケーションカウンタ値を補正しています)

X:  .FDATAB.S 2,H'7F800000.S    ; 4 バイトの領域
                                   ; "7F800000" (H'7F800000.S)
                                   ; を 2 個確保します。

      .FDATAB.D 3,F'4.32D-1    ; 8 バイトの領域
                                   ; "3FDBA5E353F7CED9"
                                   ; (F'4.32D-1) を 3 個確保します。
~

```

・コーディング例の図解



.XDATA 固定小数点データを確保する

書式

[シンボル [:]] .XDATA [.オペレーションサイズ] 固定小数点定数
[, 固定小数点定数 ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.XDATA を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
W	ワード (2 バイト)
L	ロングワード (4 バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータのサイズが決まります。

指定を省略すると、ロングワードになります。

(3) オペランド

データとして確保したい値を記述します。

解説

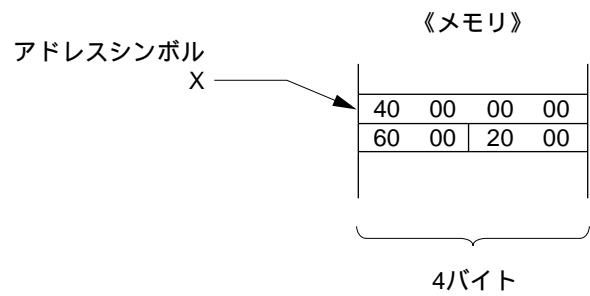
(1) .XDATA は、固定小数点定数データをメモリ上に確保するアセンブラ制御命令です。

【参照】 固定小数点定数 言語編「1.4.4 固定小数点定数」

コーディング例

```
~  
.ALIGN      4                ; (ロケーションカウンタ値を補正しています)  
  
X:  .XDATA.L  0.5          ; 4 バイトの領域 (H'40000000)  
                                ; を確保します。  
    .XDATA.W  0.75,0.25    ; 2 バイトの領域 (H'6000) と (H'2000)  
                                ; を確保します。  
~
```

・コーディング例の図解



.RES データ領域を確保する**書式**

[シンボル[:]] .RES [.オペレーションサイズ] 領域数

ステートメントの要素**(1) ラベル**

必要であれば、目印となるシンボルを記述します。

(2) オペレーション**(a) ニーモニック**

.RES を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
B	バイト
W	ワード(2バイト)
L	ロングワード(4バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータ領域の単位サイズが決まります。

指定を省略すると、ロングワードになります。

(3) オペランド

確保するデータ領域の数(単位サイズ何個分か)を記述します。

解説

(1) .RES は、データ領域をメモリ上に確保するアセンブラ制御命令です。

(2) 領域数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

(3) オペレーションサイズによって、指定できる領域数の範囲が異なります。

オペレーションサイズ	領域数の範囲*
B	H'00000001 ~ H'FFFFFFF (1 ~ 4,294,967,295)
W	H'00000001 ~ H'7FFFFFFF (1 ~ 2,147,483,647)
L	H'00000001 ~ H'3FFFFFFF (1 ~ 1,073,741,823)

【注】* () 内は 10 進表現

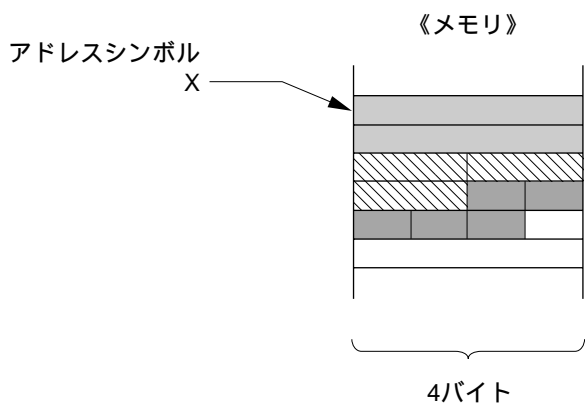
コーディング例

```

~
      .ALIGN    4      ; ( ロケーションカウンタ値を補正しています )
X:    .RES.L     2      ; ロングワードサイズの領域 2 つ分を確保しています。
      .RES.W     3      ; ワードサイズの領域 3 つ分を確保しています。
      .RES.B     5      ; バイトサイズの領域 5 つ分を確保しています。
~

```

・コーディング例の図解



.SRES 文字列データ領域を確保する

書式

[シンボル [:]] .SRES 文字列領域サイズ [, 文字列領域サイズ ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック.SRES を記述します。

(3) オペランド

確保する領域のサイズを、バイト単位で記述します。

解説

(1) .SRES は、文字列用のデータ領域を確保するアセンブラ制御命令です。

(2) 文字列領域サイズは、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

文字列領域サイズとして許される値は、H'00000001 ~ H'FFFFFFFF です。

(10進表現では 1 ~ 4,294,967,295)

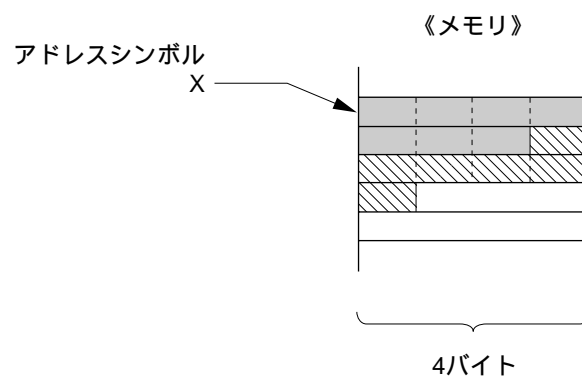
コーディング例

```

~
      .ALIGN 4      ; (ロケーションカウンタ値を補正しています)
X:    .SRES 7       ; 7 バイトの領域を確保しています。
      .SRES 6       ; 6 バイトの領域を確保しています。
~

```

・コーディング例の図解



.SRESC 計数付き文字列データ領域を確保する

書式

[シンボル [:]] .SRESC 文字列領域サイズ [, 文字列領域サイズ ...]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック .SRESC を記述します。

(3) オペランド

文字列データ用の領域（計数の 1 バイトを含まない領域）のサイズを、バイト単位で記述します。

解説

- (1) .SRESC は、計数付き文字列用のデータ領域を、メモリ上に確保するアセンブラ制御命令です。

計数付き文字列とは、文字列の先頭に 1 バイトの計数をつけ加えたものです。

計数は、文字列用のデータ領域（計数を含まない領域）のサイズを、バイト単位で表します。

【参照】 文字列 言語編「1.7 文字列」

- (2) 文字列領域サイズは、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

文字列領域サイズとして許される値は、H'00000000 ~ H'000000FF です。

(10 進表現では 0 ~ 255)

- (3) メモリ上に確保される領域のサイズは、文字列領域サイズ + 計数用の 1 バイトです。

コーディング例

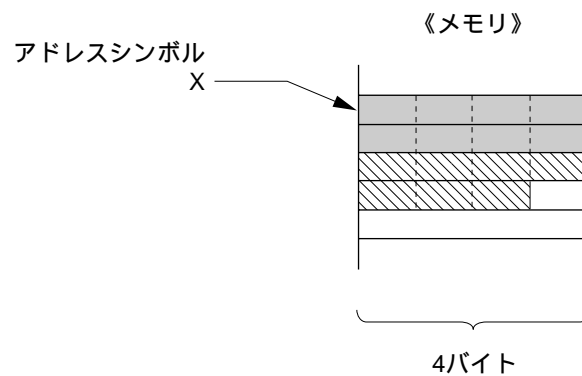
```

~
.ALIGN    4           ; (ロケーションカウンタ値を補正しています)

X:  .SRESC    7       ; 7 バイト + 計数用の 1 バイトを確保しています。
    .SRESC    6       ; 6 バイト + 計数用の 1 バイトを確保しています。
~

```

・コーディング例の図解



.SRESZ ゼロ終端文字列データ領域を確保する**書式**

[シンボル [:]] .SRESZ 文字列領域サイズ [, 文字列領域サイズ ...]

ステートメントの要素**(1) ラベル**

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

ニーモニック .SRESZ を記述します。

(3) オペランド

文字列データ用の領域（終端のゼロを含まない領域）のサイズを、バイト単位で記述します。

解説

- (1) SRESZ は、ゼロ終端文字列用のデータ領域を、メモリ上に確保するアセンブラ制御命令です。ゼロ終端文字列とは、文字列の終端に 1 バイトのゼロをつけ加えたものです。

【参照】 文字列 言語編「1.7 文字列」

- (2) 文字列領域サイズは、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

文字列領域サイズとして許される値は、H'00000000 ~ H'000000FF です。

(10 進表現では 0 ~ 255)

- (3) メモリ上に確保される領域のサイズは、文字列領域サイズ + 終端ゼロ用の 1 バイトです。

コーディング例

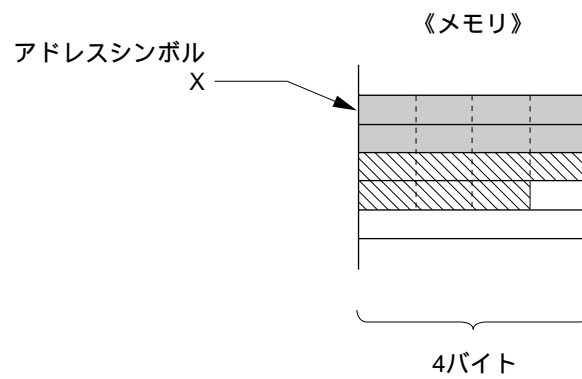
```

~
.ALIGN    4        ; (ロケーションカウンタ値を補正しています)

X:  .SRESZ    7        ; 7バイト+終端ゼロ用の1バイトを確保しています。
    .SRESZ    6        ; 6バイト+終端ゼロ用の1バイトを確保しています。
~

```

・コーディング例の図解



.FRES 浮動小数点データ領域を確保する

書式

[シンボル [:]] .FRES [.オペレーションサイズ] 領域確保数

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

(a) ニーモニック

.FRES を記述します。

(b) オペレーションサイズ

指定内容	データのサイズ
S	単精度 (4 バイト)
D	倍精度 (8 バイト)

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、確保するデータ領域の単位サイズが決まります。

指定を省略すると、単精度になります。

(3) オペランド

確保するデータ領域の数 (単精度サイズ何個分か) を記述します。

解説

(1) .FRES は、浮動小数点データ領域をメモリ上に確保するアセンブラ制御命令です。

(2) 領域数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボル、外部参照および相対シンボルを使わずに指定する。

コーディング例

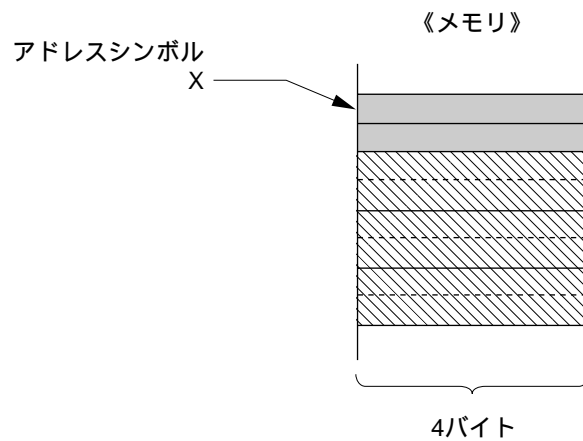
```

~
.ALIGN      4                ; (ロケーションカウンタ値を補正しています)

X:  .FRES.S    2            ; 領域 2 つ分を確保しています。
    .FRES.D    3            ; 領域 3 つ分を確保しています。
~

```

・コーディング例の図解



5.2.5 外部定義または外部参照に関するアセンブラ制御命令

外部定義または外部参照に関するアセンブラ制御命令には、次のものがあります。

.EXPORT	外部定義シンボルを宣言します。 ファイル内で定義しているシンボルを、他のファイルで参照できるようにするための宣言です。
.IMPORT	外部参照シンボルを宣言します。 他のファイルで定義しているシンボルを参照するための宣言です。
.GLOBAL	外部定義シンボルまたは外部参照シンボルを宣言します。 ファイル内で定義しているシンボルを、他のファイルで参照できるようにするため、または、他のファイルで定義しているシンボルを参照するための宣言です。

.EXPORT 外部定義シンボルを宣言する

書式

.EXPORT シンボル [, シンボル ...]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.EXIT を記述します。

(3) オペランド

外部定義シンボルとして宣言したいシンボルを記述します。

解説

(1) .EXPORT は、外部定義シンボルを宣言するアセンブラ制御命令です。

外部定義シンボルの宣言は、ファイル内で定義しているシンボルを、他のファイルで参照するために必要です。

(2) 外部定義シンボルとして宣言できるのは、次のものです。

- ・定数シンボル (.ASSIGN アセンブラ制御命令で定義したものを除く)
- ・絶対アドレスシンボル (ダミーセクションのアドレスシンボルを除く)
- ・相対アドレスシンボル

(3) シンボルを外部参照するには、外部定義シンボルとして宣言するとともに、外部参照シンボルとして宣言しなければなりません。

外部参照シンボルは、シンボルを参照しているファイル側で、.IMPORT アセンブラ制御命令または.GLOBAL アセンブラ制御命令によって宣言します。

コーディング例

ファイルAで定義しているシンボルをファイルBで参照する例です。

・ファイルA

```
.EXPORT X                ;X を、外部定義シンボルとして宣言しています。
~
X:      .EQU    H'10000000 ;X を定義しています。
~
```

・ファイルB

```
.IMPORT X                ;X を、外部参照シンボルとして宣言しています。
~
.ALIGN   4
.DATA.L  X                ;X を参照しています。
~
```

.IMPORT 外部参照シンボルを宣言する

書式

.IMPORT シンボル [, シンボル ...]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .IMPORT を記述します。

(3) オペランド

外部参照シンボルとして宣言したいシンボルを記述します。

解説

(1) .IMPORT は、外部参照シンボルを宣言するアセンブラ制御命令です。

外部参照シンボルの宣言は、他のファイルで定義しているシンボルを参照するために必要です。

(2) ファイル内で定義しているシンボルは、外部参照シンボルとして宣言できません。

(3) シンボルを外部参照するには、外部参照シンボルとして宣言するとともに、外部定義シンボルとして宣言しなければなりません。

外部定義シンボルは、シンボルを定義しているファイル側で、.EXPORT アセンブラ制御命令または .GLOBAL アセンブラ制御命令によって宣言します。

コーディング例

ファイルAで定義しているシンボルをファイルBで参照する例です。

・ファイルA

```
        .EXPORT    X                ;X を、外部定義シンボルとして宣言しています。
        ~
X:      .EQU       H'10000000      ;X を定義しています。
        ~
```

・ファイルB

```
        .IMPORT    X                ;X を、外部参照シンボルとして宣言しています。
        ~
        .ALIGN     4
        .DATA.L    X                ;X を参照しています。
        ~
```

.GLOBAL 外部定義シンボルまたは外部参照シンボルを宣言する

書式

.GLOBAL シンボル [, シンボル ...]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.GLOBAL を記述します。

(3) オペランド

外部定義シンボルまたは外部参照シンボルとして、宣言したいシンボルを記述します。

解説

- (1) .GLOBAL は、外部定義シンボルまたは外部参照シンボルを宣言するアセンブラ制御命令です。

外部定義シンボルの宣言は、ファイル内で定義しているシンボルを他のファイルで参照するために必要です。外部参照シンボルの宣言は、他のファイルで定義しているシンボルをファイル内で参照するために必要です。

- (2) ファイル内で定義しているシンボルを .GLOBAL で宣言すると、そのシンボルは外部定義シンボルになります。

ファイル内で定義していないシンボルを .GLOBAL で宣言すると、そのシンボルは外部参照シンボルになります。

- (3) 外部定義シンボルとして宣言できるのは、次のものです。

- ・定数シンボル (.ASSIGN アセンブラ制御命令で定義したものを除く)
- ・絶対アドレスシンボル (ダミーセクションのアドレスシンボルを除く)
- ・相対アドレスシンボル

- (4) シンボルを外部参照するには、外部定義シンボルとして宣言するとともに、外部参照シンボルとして宣言しなければなりません。

外部定義シンボルは、シンボルを定義しているファイル側で、`.EXPORT` アセンブラ制御命令または `.GLOBAL` アセンブラ制御命令によって宣言します。

外部参照シンボルは、シンボルを参照しているファイル側で、`.IMPORT` アセンブラ制御命令または `.GLOBAL` アセンブラ制御命令によって宣言します。

コーディング例

ファイルAで定義しているシンボルをファイルBで参照する例です。

・ファイルA

```

        .GLOBAL  X           ;X を、外部定義シンボルとして宣言しています。
        ~
X:      .EQU     H'10000000  ;X を定義しています。
        ~

```

・ファイルB

```

        .GLOBAL  X           ;X を、外部参照シンボルとして宣言しています。
        ~
        .ALIGN   4
        .DATA.L  X           ;X を参照しています。
        ~

```

5.2.6 オブジェクトモジュールに関するアセンブラ制御命令

オブジェクトモジュールに関するアセンブラ制御命令には、次のものがあります。

.OUTPUT	オブジェクトモジュールおよびデバッグ情報の出力を制御します。
.DEBUG	シンボルデバッグ情報の部分出力を制御します。
.ENDIAN	エンディアン種別を指定します。
.LINE	行番号を指定します。

.OUTPUT オブジェクトモジュールの出力を制御する

書式

.OUTPUT 出力指定 [, 出力指定]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック**.OUTPUT** を記述します。

(3) オペランド：出力指定

指定内容	出力の制御
OBJ	オブジェクトモジュールを出力する。
NOOBJ	オブジェクトモジュールを出力しない。
DBG	オブジェクトモジュール出力時に、デバッグ情報を出力する。
NODBG	オブジェクトモジュール出力時に、デバッグ情報を出力しない。

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、オブジェクトモジュールとデバッグ情報の出力を制御します。

解説

(1) **.OUTPUT** は、オブジェクトモジュールまたはデバッグ情報の出力を制御するアセンブラ制御命令です。

(2) **.OUTPUT** を2回以上使用して、指定内容が矛盾すると、エラーとなります。

【例】

```

~
.OUTPUT  OBJ
.OUTPUT  NODBG
~

```

OK

```

~
.OUTPUT  OBJ
.OUTPUT  NOOBJ
~

```

エラー

- (3) デバッグ情報の出力に関する指定は、オブジェクトモジュールを出力する場合に限り有効です。
- (4) デバッグ情報を出力する場合、オブジェクトファイルを出力するディレクトリに「dwfinf」という名前のディレクトリを自動生成し、オブジェクトファイル名と同じ主ファイル名で拡張子「dwi」のデバッグ付加情報（ELF/DWARF 付加情報）ファイルを出力します。
- (5) オブジェクトモジュールとデバッグ情報の出力に関して、アセンブラはコマンドライン・オプションによる指定を優先します。

【参照】オブジェクトモジュールの出力

操作編「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」

- OBJECT - NOOBJECT

デバッグ情報の出力

操作編「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」

- DEBUG - NODEBUG

コーディング例

オブジェクトモジュールとデバッグ情報の出力に関して、コマンドライン・オプションによる指定がないことを仮定して、結果を説明しています。

・例1

```
.OUTPUT    OBJ          ;オブジェクトモジュールを出力します。
               ;デバッグ情報は出力しません。

~
```

・例2

```
.OUTPUT    OBJ,DBG      ;オブジェクトモジュールとデバッグ情報を出力します。

~
```

・例3

```
.OUTPUT    OBJ,NODBG    ;オブジェクトモジュールを出力します。
               ;デバッグ情報は出力しません。

~
```

【補足】 デバッグ情報は、デバッガでプログラムをデバッグするときに必要な情報であり、オブジェクトモジュールの一部となります。

デバッグ情報は、ソースステートメントの行に関する情報、シンボルに関する情報などを含みます。

.DEBUG シンボルデバッグ情報の部分出力を制御する

書式

.DEBUG 出力指定

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .DEBUG を記述します。

(3) オペランド：出力指定

指定内容	出力の制御
ON	次のソースステートメントから、シンボルデバッグ情報を出力する。
OFF	次のソースステートメントから、シンボルデバッグ情報を出力しない。

【注】 網掛け部分は指定する前の設定

指定内容によって、シンボルデバッグ情報の出力を制御します。

解説

- (1) .DEBUG は、シンボルデバッグ情報の部分出力を制御するアセンブラ制御命令です。
デバッグに必要なシンボルに限定してシンボルデバッグ情報を出力すると、アセンブル時間を短縮できるなどの利点があります。
- (2) .DEBUG による指定は、オブジェクトモジュールを出力し、かつ、デバッグ情報を出力する場合に限り有効です。

【参照】

オブジェクトモジュールの出力

言語編「5.2.6 オブジェクトモジュールに関するアセンブラ制御命令」

.OUTPUT

操作編「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」

- OBJECT - NOOBJECT

デバッグ情報の出力

言語編「5.2.6 オブジェクトモジュールに関するアセンブラ制御命令」

.OUTPUT

操作編「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」

- DEBUG - NODEBUG

コーディング例

~

```
.DEBUG    OFF    ;アセンブラは、次のソースステートメントから  
              ;シンボルデバッグ情報を出力しません。
```

~

```
.DEBUG    ON     ;アセンブラは、次のソースステートメントから  
              ;シンボルデバッグ情報を出力します。
```

~

【補足】 シンボルデバッグ情報とは、デバッグ情報のうちのシンボルに関するものをいいます。

.ENDIAN エンディアン種別を指定する

書式

.ENDIAN エンディアン種別
エンディアン種別：{ BIG | LITTLE }

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .ENDIAN を記述します。

(3) オペランド：エンディアン種別

指定内容	出力の制御
BIG	BIG エンディアンでアセンブルする。
LITTLE	LITTLE エンディアンでアセンブルする。

【注】 網掛け部分は指定を省略したときの選択

解説

(1) .ENDIAN は、エンディアンの種別を指定するアセンブラ制御命令です。

(2) .ENDIAN は、ソースプログラムの最初に記述してください。

(3) -ENDIAN が指定されている場合は、.ENDIAN は無効になります。

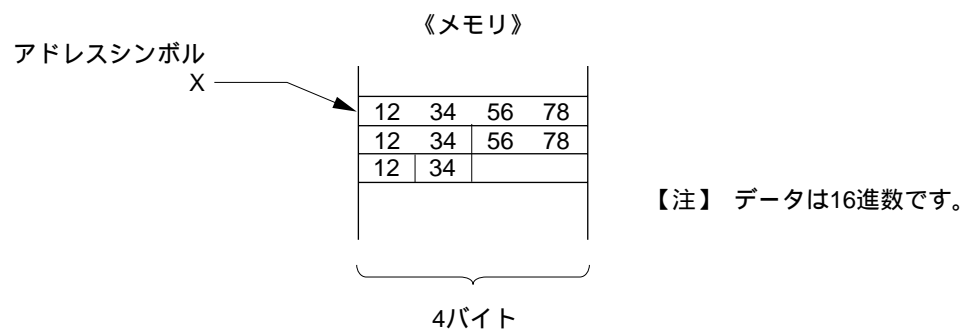
【参照】-ENDIAN 操作編「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」 -ENDIAN

コーディング例 1

BIG エンディアンを指定した場合

```
.CPU      SH1                ; CPU に SH1 を指定します。
.ENDIAN    BIG                ; BIG エンディアンを指定します。
~
X:  .DATA.L  H'12345678      ;
X:  .DATA.W  H'1234,H'5678   ; 整数データを確保しています
X:  .DATA.B  H'12,H'34       ;
~
```

・コーディング例の図解



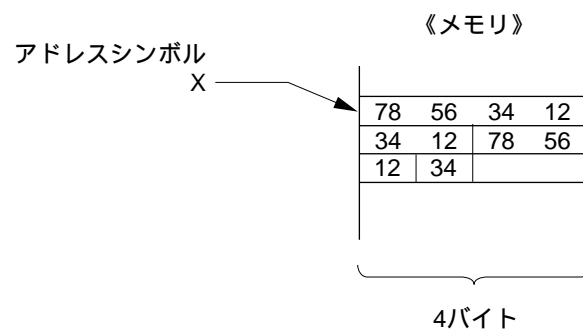
コーディング例2

LITTLE エンディアンを指定した場合

```
.CPU      SH3                ; CPU に SH3 を指定します。
.ENDIAN    LITTLE            ; LITTLE エンディアンを指定します。
~

X:  .DATA.L  H'12345678      ;
X:  .DATA.W  H'1234,H'5678   ; 整数データを確保しています
X:  .DATA.B  H'12,H'34       ;
~
```

・コーディング例の図解



【注】 データは16進数です。

.LINE 行番号を変更する

書式

.LINE ["ファイル名 ",] 行番号

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.LINE を記述します。

(3) オペランド

(a) 第1オペランド : ファイル名

アセンブラのエラーメッセージ、あるいはデバッグ時に参照するファイル名を記述します。

(b) 第2オペランド : 行番号

アセンブラのエラーメッセージ、あるいはデバッグ時に参照する行番号を記述します。

解説

- (1) .LINE は、アセンブラのエラーメッセージ、あるいはデバッグ時に参照する行番号とファイル名を変更するアセンブラ制御命令です。
- (2) プログラム内の最初の.LINE 以降は、次の.LINE まで行番号、ファイル名を更新しません。
- (3) C コンパイラ (V3.0 以降) では、デバッグオプションを指定してアセンブラソースを出力する時に、C ソースファイル行に対応する.LINE を生成します。
- (4) ファイル名を省略すると、ファイル名は変更されず、行番号だけが変更されます。

コーディング例

```
shc -code=asmcode -debug test.c
```

Cソースプログラム (test.c)

```
int      func()
{
    int  i, j;

    j=0;
    for (i=1;i<=10;i++){
        j+=i;
    }
    return(j);
}
```



アセンブリソースプログラム (test.src)

```
      .EXPORT      _func
      .SECTION     P, CODE, ALIGN=4
      .LINE        "/asm/test.c", 1
      _func:
                        ;function: func
                        ;frame size=0

      .LINE        "/asm/test.c", 5
      MOV          #0, R5
      .LINE        "/asm/test.c", 6
      MOV          #10, R6
      MOV          #1 R4

L212:
      .LINE        "/asm/test.c", 7
      ADD          R4, R5
      ADD          #1, R4
      .LINE        "/asm/test.c", 6
      CMP/GT      R6, R4
      BF          L212
      .LINE        "/asm/test.c", 10
      RTS
      .LINE        "/asm/test.c", 9
      MOV          R5, R0
      .END
```

5.2.7 アセンブルリストに関するアセンブラ制御命令

アセンブルリストに関するアセンブラ制御命令には、次のものがあります。

.PRINT	アセンブルリストの出力を制御します。
.LIST	ソースプログラム・リストの部分出力を制御します。
.FORM	アセンブルリストの行数と桁数を設定します。
.HEADING	ソースプログラム・リストのページヘッダを設定します。
.PAGE	ソースプログラム・リストを改ページします。
.SPACE	ソースプログラム・リストに空行を出力します。

【補足】 アセンブルリストはアセンブル結果を出力するリストであり、ソースプログラム・リスト、クロスリファレンス・リスト、セクション情報リストを含みます。

【参照】 アセンブルリストについての詳細
「付録C アセンブルリスト出力例」

.PRINT アセンブルリストの出力を制御する

書式

```
.PRINT 出力指定 [ , 出力指定 ... ]
```

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック **.PRINT** を記述します。

(3) オペランド：出力指定

指定内容	出力方法
LIST	アセンブルリストを出力する。
NOLIST	アセンブルリストを出力しない。
SRC	アセンブルリスト出力時に、ソースプログラム・リストを出力する。
NOSRC	アセンブルリスト出力時に、ソースプログラム・リストを出力しない。
CREF	アセンブルリスト出力時に、クロスリファレンス・リストを出力する。
NOCREF	アセンブルリスト出力時に、クロスリファレンス・リストを出力しない。
SCT	アセンブルリスト出力時に、セクション情報リストを出力する。
NOSCT	アセンブルリスト出力時に、セクション情報リストを出力しない。

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、アセンブルリストの出力を制御します。

解説

(1) .PRINT は、アセンブルリストの出力を制御するアセンブラ制御命令です。

(2) .PRINT を 2 回以上使用して、指定内容が矛盾すると、エラーとなります。

【例】

<pre> ~ .PRINT LIST .PRINT NOSRC ~ </pre>	OK	<pre> ~ .PRINT LIST .PRINT NOLIST ~ </pre>	エラー
---	----	--	-----

(3) ソースプログラム・リスト、クロスリファレンス・リスト、セクション情報リストの出力に関する指定は、アセンブルリストを出力する場合に限り有効です。

(4) アセンブルリストの出力に関して、アセンブラはコマンドライン・オプションによる指定を優先します。

【参照】アセンブルリストの出力

操作編「2.2.3 アセンブルリストに関するコマンドライン・オプション」

```

- LIST    - NOLIST
- SOURCE  - NOSOURCE
- CROSS__REFERENCE  - NOCROSS__REFERENCE
- SECTION - NOSECTION

```

コーディング例

アセンブルリストの出力に関して、コマンドライン・オプションによる指定がないことを仮定して、結果を説明しています。

・例 1

```

.PRINT LIST ; 全種類のアセンブルリストを出力します。
~

```

・例 2

```

.PRINT LIST,NOSRC,NOCREF ; セクション情報リストだけを出力します。
~

```

.LIST ソースプログラム・リストの部分出力を制御する

書式

.LIST 出力指定 [, 出力指定 ...]

出力種別：{ON | OFF | COND | NOCOND | DEF | NODEF | CALL | NOCALL |
EXP | NOEXP | CODE | NOCODE}

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.LIST を記述します。

(3) オペランド

出力指定を記述します。

解説

(1) .LIST は出力指定により次のような働きをするアセンブラ制御命令です。

- (a) ソースステートメントの部分的な出力、出力抑止の制御
- (b) 条件つきアセンブリ機能およびマクロ機能に関するソースステートメントの部分的な出力、出力抑止の制御
- (c) オブジェクトコード表示行の部分的な出力、出力抑止の制御

(2) 各出力指定により制御される内容は以下のとおりです。

項目	出力指定		意味	制御内容
	出力	出力抑止		
(a)	ON	OFF	ソースステートメント	本命令以降のソースステートメント
(b)	COND	NOCOND	条件つき不成立	.AIF, .AIFDEF の不成立部分
	DEF	NODEF	定義	マクロ定義部分 .AREPEAT, .AWHILE 定義部分 .INCLUDE 制御文 .ASSIGNA, .ASSIGNC 制御文
	CALL	NOCALL	コール	マクロコール文 .AIF, .AIFDEF, .AENDI 制御文
	EXP	NOEXP	展開	マクロ展開部分 .AREPEAT, .AWHILE 展開部分
(c)	CODE	NOCODE	オブジェクトコード表示 行	制御命令のオブジェクトコード表示 が、ソースステートメントの行数を 超える部分

【注】 網掛け部分は指定を省略したときの選択

(3) .LIST による指定は、ソースプログラム・リストを出力する場合に限り有効です。

【参照】 ソースプログラム・リストの出力

言語編「5.2.7 アセンブルリストに関するアセンブラ制御命令」 .PRINT

操作編「2.2.3 アセンブルリストに関するコマンドライン・オプション」
- LIST - NOLIST - SOURCE - NOSOURCE

(4) ソースプログラム・リストの部分出力に関して、アセンブラはコマンドライン・オプションによる指定を優先します。

【参照】 ソースプログラム・リストの部分出力

操作編「2.2.3 アセンブルリストに関するコマンドライン・オプション」
- SHOW - NOSHOW

(5) .LIST 自体は、ソースプログラム・リスト上に表示されません。

コーディング例

ソースプログラム・リストの部分出力に関して、コマンドライン・オプションによる指定がないことを仮定して、結果を説明しています。

	<u>.LIST NOCOND,NODEF</u>		ソースプログラムリストの
	.MACRO SHLRN COUNT,Rd		部分出力を制御します。
		:	
SHIFT	.ASSIGNA \COUNT	:	
		:	
	.AIF \&SHIFT GE 16	:	
	SHLR16 \Rd	:	
SHIFT	.ASSIGNA \&SHIFT-16	:	
	.AENDI	:	
		:	
	.AIF \&SHIFT GE 8	:	
	SHLR8 \Rd	:	
SHIFT	.ASSIGNA \&SHIFT-8	:	
	.AENDI	:	
		:	
	.AIF \&SHIFT GE 4	:	汎用多ビットシフトを
	SHLR2 \Rd	:	マクロ定義しています。
	SHLR2 \Rd	:	
SHIFT	.ASSIGNA \&SHIFT-4	:	
	.AENDI	:	
		:	
	.AIF \&SHIFT GE 2	:	
	SHLR2 \Rd	:	
SHIFT	.ASSIGNA \&SHIFT-2	:	
	.AENDI	:	
		:	
	.AIF \&SHIFT GE 1	:	
	SHLR \Rd	:	
	.AENDI	:	
	.ENDM	:	
		:	
	SHLRN 23,R0		マクロコール

コーディング例のソースリスト出力結果

.LIST アセンブラ制御命令によって、マクロ定義部分、.ASSIGNA 制御文、.ASSIGNC 制御文、.AIF または .AIFDEF の不成立部分の出力が抑止されています。

31	31		
32	32	SHLRN	23,R0
33	M		
35	M		
36	M	.AIF 23	GE 16
37 00000000 4029	C	SHLR16	R0
39	M	.AENDI	
40	M		
41	M	.AIF 7	GE 8
45	M		
46	M	.AIF 7	GE 4
47 00000002 4009	C	SHLR2	R0
48 00000004 4009	C	SHLR2	R0
50	M	.AENDI	
51	M		
52	M	.AIF 3	GE 2
53 00000006 4009	C	SHLR2	R0
55	M	.AENDI	
56	M		
57	M	.AIF 1	GE 1
58 00000008 4001	C	SHLR	R0
59	M	.AENDI	

.FORM アセンブルリストの行数と桁数を設定する

書式

.FORM サイズ指定 [, サイズ指定 ...]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.FORM を記述します。

(3) オペランド：サイズ指定

指定内容	リストのサイズ
LIN = 行数	指定の数が、1 ページあたりの行数となる。
COL = 桁数	指定の数が、1 行あたりの桁数となる。

指定内容によって、アセンブルリストの行数、桁数が決まります。

解説

(1) .FORM は、アセンブルリストの 1 ページあたりの行数と 1 行あたりの桁数を設定するアセンブラ制御命令です。

(2) 行数と桁数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

行数として許される値は、20 ~ 255 です。

桁数として許される値は、79 ~ 255 です。

(3) .FORM は、1 つのソースプログラムで何回でも使えます。

(4) アセンブルリストの行数と桁数に関して、アセンブラはコマンドライン・オプションによる指定を優先します。

【参照】アセンブルリストの行数の設定

操作編「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- LINES

アセンブルリストの桁数の設定

操作編「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- COLUMNS

(5) アセンブルリストの行数と桁数に関して、.FORM による指定も、コマンドライン・オプションによる指定もない場合、次のようになります。

- ・ 行数 60
- ・ 桁数 132

コーディング例

アセンブルリストの行数と桁数の設定に関して、コマンドライン・オプションによる指定がないことを仮定して、結果を説明しています。

```

~
.FORM      LIN=60, COL=200 ;このページから、アセンブルリスト1ページを
                                ;60行にします。
                                ;また、この行から、アセンブルリスト1行を
                                ;200桁にします。

~
.FORM      LIN=55, COL=150 ;このページから、アセンブルリスト1ページを
                                ;55行にします。
                                ;また、この行から、アセンブルリスト1行を
                                ;150桁にします。

~

```

.HEADING ソースプログラム・リストのヘッダを設定する

書式

.HEADING "文字列 "

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .HEADING を記述します。

(3) オペランド：文字列

ソースプログラム・リストのヘッダを記述します。

解説

- (1) .HEADING は、ソースプログラム・リストのヘッダを設定するアセンブラ制御命令です。ヘッダとして設定できるのは、60 文字以内の文字列です。

【参照】 文字列 言語編「1.7 文字列」

- (2) .HEADING は、1 つのソースプログラムの中で何回でも使えます。

.HEADING による設定の有効範囲は、次のようになります。

- ・ ページの 1 行目で設定している場合、そのページから有効
- ・ ページの 2 行目以降で設定している場合、次のページから有効

コーディング例

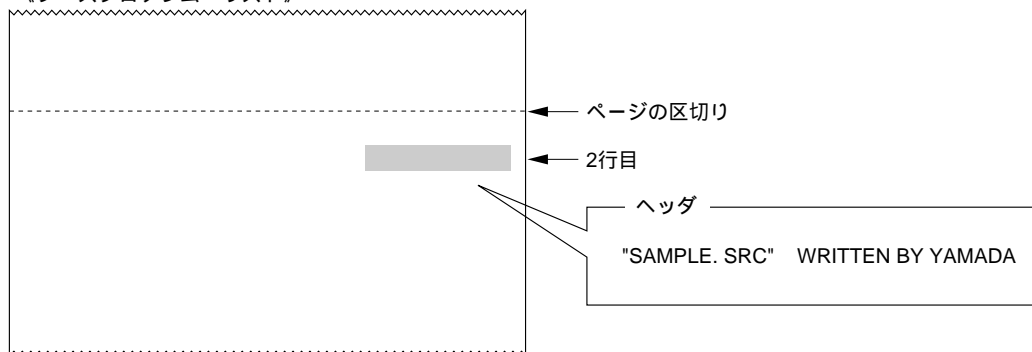
~

```
.HEADING  " "SAMPLE.SRC" "  WRITTEN BY YAMADA"
```

~

・コーディング例の図解

《ソースプログラム・リスト》



.PAGE ソースプログラム・リストを改ページする

書式

.PAGE

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック.PAGE を記述します。

(3) オペランド

記述できません。

解説

- (1) .PAGE は、ソースプログラム・リストを任意の位置で改ページするアセンブラ制御命令です。
- (2) .PAGE がリストの 1 行目にある場合、その改ページ指定は無効になります。
- (3) .PAGE 自体は、ソースプログラム・リスト上に表示されません。

コーディング例

```

~
MOV      R0,R1
RTS
MOV      R0,R2
.PAGE    ;セクションが切り替わるので、改ページを指定しています。
.SECTION DT,DATA,ALIGN=4
.DATA.L  H'11111111
.DATA.L  H'22222222
.DATA.L  H'33333333
~

```

・コーディング例の図解

《ソースプログラム・リスト》

18	00000022	6103	18	MOV	R0, R1
19	00000024	000B	19	RTS	
20	00000026	6203	20	MOV	R0, R2
*** SuperH RISC engine ASSEMBLER Ver. 4.0 *** 01 / 12 / 98 10 : 23 : 30					
PROGRAM NAME =					
22	00000000		22	. SECTION	DT, DATA, ALIGN
23	00000000	11111111	23	. DATA. L	H' 11111111
24	00000004	22222222	24	. DATA. L	H' 22222222
25	00000008	33333333	25	. DATA. L	H' 33333333

改
ペ
ー
ジ

【注】ソースプログラム・リストの見方は、「付録C アセンブルリスト出力例」を参照

.SPACE ソースプログラム・リストに空行を出力する

書式

.SPACE [行数]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .SPACE を記述します。

(3) オペランド：行数

出力したい空行の数を指定します。

省略すると、1 になります。

解説

(1) .SPACE は、空行を指定の行数分、ソースプログラム・リストに出力するアセンブラ制御命令です。.SPACE で出力する空行には、行番号などの表示がありません。

(2) 行数は、次のように指定します。

- ・絶対値を指定する。

かつ、

- ・前方参照シンボルを使わずに指定する。

行数として許される値は、1 ~ 50 です。

(3) .SPACE で空行を出力して改ページが生じる場合、アセンブラは、改ページ以降の空行を出力しません。

(4) .SPACE 自体は、ソースプログラム・リスト上に表示されません。

コーディング例

```

        .SECTION      DT1,DATA,ALIGN=4

        .DATA.L        H'11111111

        .DATA.L        H'22222222

        .DATA.L        H'33333333

        .DATA.L        H'44444444      ;セクションが切り替わる箇所で、
        .SPACE 5                      ; 5 行の空行を挿入しています。
        .SECTION      DT2,DATA,ALIGN=4

```

~

・コーディング例の図解

《ソースプログラム・リスト》

* * * SuperH RISC engine ASSEMBLER Ver. 4.0 * * *				01 / 12 / 98 10 : 23 : 30
PROGRAM NAME =				
1	00000000		1	. SECTION DT1, DATA, ALIGN = 4
2	00000000	11111111	2	. DATA. L H' 11111111
3	00000004	22222222	3	. DATA. L H' 22222222
4	00000008	33333333	4	. DATA. L H' 33333333
5	0000000C	44444444	5	. DATA. L H' 44444444
7	00000000		7	. SECTION DT2, DATA, ALIGN = 4

【注】ソースプログラム・リストの見方は、「付録C アセンブルリスト出力例」を参照

5.2.8 その他のアセンブラ制御命令

その他のアセンブラ制御命令として、次のものがあります。

.PROGRAM	オブジェクトモジュール名を設定します。
.RADIX	基数のない整数定数を何進数とするかを指定します。
.END	ソースプログラムの終わりを宣言します。

.PROGRAM オブジェクトモジュール名を設定する

書式

.PROGRAM オブジェクトモジュール名

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .PROGRAM を記述します。

(3) オペランド：オブジェクトモジュール名

オブジェクトモジュールを識別するための名前を記述します。

解説

(1) .PROGRAM は、オブジェクトモジュール名を設定するアセンブラ制御命令です。

オブジェクトモジュール名とは、「Hシリーズリンケージエディタ」または「Hシリーズライブラリアン」が、オブジェクトモジュールを識別するために必要とする名前です。

(2) オブジェクトモジュール名の付け方は、シンボルの名づけ方と同じです。

アセンブラは、オブジェクトモジュール名の英大文字と英小文字を区別します。

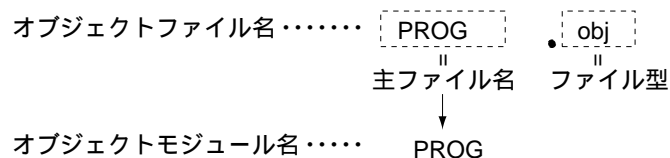
【参照】 シンボルの名づけ方 言語編「1.3.2 シンボルの名づけ方」

(3) .PROGRAM による設定は、最初の1回だけが有効です。(アセンブラは、2回め以降の指定を無視します)

- (4) .PROGRAM による設定がない場合、アセンブラがデフォルト（暗黙）のオブジェクトモジュール名を設定します。

デフォルトのオブジェクトモジュール名は、オブジェクトファイル（オブジェクトモジュールの出力先）の主ファイル名です。

・例



【参照】 操作編「1.2 ファイルの指定形式」

- (5) オブジェクトモジュール名は、プログラムの中で使用しているシンボル名と重複しても構いません。

コーディング例

```
.PROGRAM  PROG1      ;オブジェクトモジュール名として、PROG1 を設定  
                  ;しています。
```

~

.RADIX 基数のない整数定数を何進数とするかを指定する

書式

.RADIX 基数指定

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .RADIX を記述します。

(3) オペランド：基数指定

指定内容	基数のない整数定数
B	2 進数
Q	8 進数
D	10 進数
H	16 進数

【注】 網掛け部分は指定を省略したときの選択

指定内容によって、基数のない整数定数が何進数になるかが決まります。

解説

- (1) .RADIX は、基数のない整数定数を何進数とするかを指定するアセンブラ制御命令です。
- (2) .RADIX による指定を省略した場合、基数のない整数定数は 10 進数です。
- (3) 基数のない整数定数が 16 進数になるよう指定した場合（基数指定 H）、整数定数の一番上位の桁が A～F であるときは、その上に 0 をつけ加えてください。
（アセンブラは、A～F で始まる記述をシンボルと見なします）
- (4) .RADIX による指定は、指定の箇所の前方で有効です。

コーディング例 1

```
~
      .RADIX      D
X:    .EQU        100    ; 100 は、10 進数です。
~
      .RADIX      H
Y:    .EQU        64     ; 64 は、16 進数です。
~
```

コーディング例 2

```
~
      .RADIX      H
Z:    .EQU        0F     ; F と書くとシンボルと見なされてしまうので、
                        ; 先頭に 0 を付けています。
~
```

.END ソースプログラムの終わりと実行開始アドレスを宣言する

書式

.END [実行開始アドレス]

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .END を記述します。

(3) オペランド：実行開始アドレス

シミュレーションを開始したいアドレスを指定します。

解説

(1) .END は、ソースプログラムの終わりを宣言するアセンブラ制御命令です。

.END が出現した時点で、アセンブラはアセンブル処理を終了します。

(2) .END で実行開始アドレスを指定しておくと、シミュレータ・デバッガは指定のアドレスからシミュレーションを開始します。

(3) 実行開始アドレスは、絶対値またはアドレス値で指定します。

(4) 実行開始アドレスには、コードセクションのアドレスを指定してください。

コーディング例

```
.EXPORT      START
.SECTION     CD, CODE, ALIGN=4
START:
    ~
    .END      START      ;ソースプログラムの終了を宣言しています。
```

;シミュレータ・デバッガは、シンボル `START` が示すアドレスから
;シミュレーションを開始します。

6. ファイルインクルード機能

ファイルインクルードとは、アセンブルするソースファイルに他のソースファイルを取り込む機能です（以下、取り込まれる側のソースファイルをインクルードファイルといます）。

ファイルインクルード機能に関する制御文として、.INCLUDE があります。

プログラマが.INCLUDE 制御文を記述した位置に、指定のインクルードファイルが取り込まれます。

コーディング例

ソースプログラム

```
.INCLUDE "FILE. H"

. SECTION CD1, CODE, ALIGN = 4
MOV #ON, R0

~
```

インクルードファイル FILE. H

```
ON: . EQU 1
OFF: . EQU 0
```

ファイルインクルード結果（ソースリスト）

```
.INCLUDE "FILE. H"
ON: . EQU 1
OFF: . EQU 0

. SECTION CD1, CODE, ALIGN = 4
MOV #ON, R0

~
```

.INCLUDE 指定のインクルードファイルを取り込む

書式

.INCLUDE "ファイル名 "

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック **.INCLUDE** を記述します。

(3) オペランド

取り込むファイルを指定します。

解説

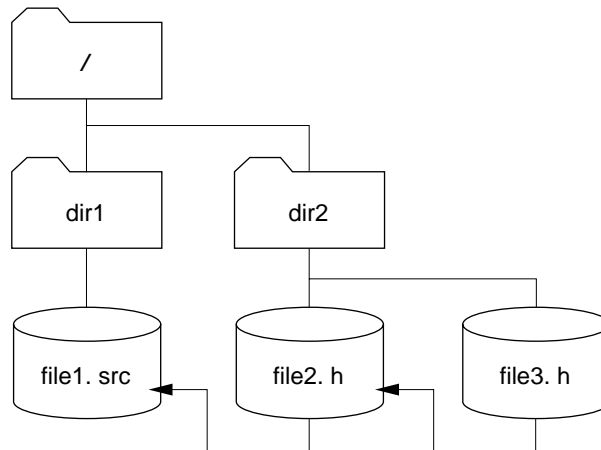
- (1) **.INCLUDE** は、指定したインクルードファイルを取り込む制御文です。
- (2) ファイル名として主ファイル名だけを指定した場合、ファイル型なしのファイル名が有効となります。(アセンブラによるファイル型の仮定なし)
【参照】 操作編「1.2 ファイルの指定形式」
- (3) ファイル名は、ディレクトリを含めた形で指定することができます。
ディレクトリは、絶対パス(ルートディレクトリからの経路)または、相対パス(カレントディレクトリ*からの経路)で指定します。
- (4) インクルードファイル中へ、さらに別のファイルを取り込むこともできます。インクルードは30段階までネスト(多重の状態)することができます。
- (5) **.INCLUDE**で指定したディレクトリ名は、**- INCLUDE**により変更することができます。

【注】 * ソースファイル中の**.INCLUDE** のカレントディレクトリはアセンブラを起動したときのディレクトリとなります。
インクルードファイル中の**.INCLUDE** のカレントディレクトリはそのインクルードファイルが存在するディレクトリとなります。

【参照】 - INCLUDE 操作編「2.2.4 ファイルインクルード機能に関するコマンド
ライン・オプション」

コーディング例

ディレクトリが下図のような構造になっているとき、以下のことを実行するとします。



- ・ ルートディレクトリ (/) からアセンブラを起動
- ・ 入力ソースファイルは /dir1/file1.src
- ・ file1.src に file2.h をインクルード
- ・ file2.h に file3.h をインクルード

起動コマンドは、次のようになります。

```
%asmsh /dir1/file1.src (RET)
```

file1.src には、つぎのインクルード制御文が必要になります。

```
.INCLUDE "dir2/file2.h" ; /がカレントディレクトリです。(相対パス指定)
```

または

```
.INCLUDE "/dir2/file2.h" ; 絶対パス指定
```

また、file2.h には、つぎのインクルード制御文が必要になります。

```
.INCLUDE "file3.h" ; /dir2 がカレントディレクトリです。(相対パス指定)
```

または

```
.INCLUDE "/dir2/file3.h" ; 絶対パス指定
```

【注意】 オペレーティングシステムが Windows[®]95 および Windows[®]NT の場合、スラッシュ (/) を次のように替えてください。

- ・日本語環境 円マーク (¥)
- ・英語環境 バックスラッシュ (\)

7. 条件つきアセンブリ機能

7.1 条件つきアセンブリ機能の概要

条件つきアセンブリ機能は、次のようなアセンブルを簡単に実現します。

- ・ ソースプログラムの文字列を他の文字列に置き換える
- ・ ソースプログラムの一部分をアセンブルするか否か、条件によって切り替える
- ・ ソースプログラムの一部分を、繰り返し展開してアセンブルする

7.1.1 プリプロセッサ変数

アセンブル条件を記述するための変数を、プリプロセッサ変数といいます。プリプロセッサ変数の型には、整数型と文字型があります。

(1) 整数型プリプロセッサ変数

.ASSIGNA 制御文で定義します（再定義が可能）。プリプロセッサ変数の先頭にバックスラッシュ（\）とアンパサンド（&）を付けることにより、内容を参照できます。*

コーディング例

```
FLAG:      .ASSIGNA 1
~
.AIF \&FLAG EQ 1      ; FLAG が 1 のとき、
MOV R0,R1             ; MOV R0,R1 をアセンブルします。
.AENDI
~
```

(2) 文字型プリプロセッサ変数

.ASSIGNC 制御文で定義します（再定義が可能）。プリプロセッサ変数の先頭にバックスラッシュ（\）とアンパサンド（&）を付けることにより、内容を参照できます。*

コーディング例

```
FLAG:      .ASSIGNC "ON"
~
.AIF "\&FLAG" EQ "ON" ; FLAG が"ON"のとき、
MOV R0,R1             ; MOV R0,R1 をアセンブルします。
.AENDI
~
```

【注】* 日本語環境で使用する場合は、\'の代わりに'を使用してください。

7.1.2 置換シンボル

.DEFINE 制御文で定義します。

ソースプログラムの一部分を指定によって、置き換えることができます。

コーディングは、次のようになります。

コーディング例

```
SYM1:      .DEFINE "R1"
~
MOV.L      SYM1,R0 ; MOV.L R1,R0 に置き換えられます。
~
```

7.1.3 条件つきアセンブル

ソースプログラムの一部分をアセンブルするか否か、条件によって切り替えることができます。

条件つきアセンブリの条件には、関係演算子で判別する比較型条件つきアセンブルと置換シンボルで判別する定義型条件つきアセンブルがあります。

(1) 比較型条件つきアセンブル

比較型条件つきアセンブルは、条件の成立か不成立かによりアセンブルする範囲を切り替えます。

コーディングは、次のようになります。

```
~
.AIF  比較型条件
      条件が成立したときアセンブルする部分

.AELIF 比較型条件
      条件が成立したときアセンブルする部分
.AELSE
      全ての条件が成立しないときアセンブルする部分
~
```

この部分は省略可能

コーディング例

```

~
.AIF  "\&FLAG" EQ "ON"
MOV  R0,R10      ; FLAG が
MOV  R1,R11      ;   "ON" のとき
MOV  R2,R12      ;   アセンブルします。
.AELSE
MOV  R10,R0      ; FLAG が
MOV  R11,R1      ;   "ON" でないとき
MOV  R12,R2      ;   アセンブルします。
.AENDI
~

```

(2) 定義型条件つきアセンブル

定義型条件つきアセンブルは、置換シンボルが定義されているか否かによりアセンブルする範囲を切り替えます。

コーディングは、次のようになります。

```

~
.AIFDEF  定義型条件
      条件の置換シンボルが定義されているときアセンブルする部分

.AELSE
      置換シンボルが定義されていないときアセンブルする部分
.AENDI
~

```

この部分は省略可能

コーディング例

~

.AIFDEF FLAG

MOV R0,R10

; .AIFDEF 制御文より後方で

MOV R1,R11

; FLAG が .DEFINE 制御文で定義されているとき

MOV R2,R12

; アセンブルします。

.AELSE

MOV R10,R0

; .AIFDEF 制御文より後方で

MOV R11,R1

; FLAG が .DEFINE 制御文で定義されていないとき

MOV R12,R2

; アセンブルします。

.AENDI

~

7.1.4 繰り返し展開

ソースプログラムの一部分を、指定の回数だけ繰り返し展開してアセンブルすることができます。

コーディングは、次のようになります。

```

~
    .AREPEAT 繰り返し回数
        繰り返しの対象となる部分
    .AENDR
~

```

コーディング例

```

; 64 ビット ÷ 32 ビットの除算を例に挙げます。
; R1:R2 ( 64 ビット ) ÷ R0 ( 32 ビット ) = R2 ( 32 ビット )
; :符号無し
TST      R0,R0      ; ゼロ除算チェック
BT       zero_div
CMP/HS   R0,R1      ; オーバフローチェック
BT       over_div
DIV0U    ; フラグの初期化
.AREPEAT 32
    ROTCL  R2      ; 32 回繰り返してアセンブルします。
    DIV1   R0,R1    ;
.AENDR
    ROTCL  R2      ; R2 = 商

```

7.1.5 条件つき繰り返し展開

ソースプログラムの一部分を、条件が成立している間、繰り返し展開してアセンブルすることができます。

コーディングは、次のようになります。

```
~  
.AWHILE 条件  
    繰り返しの対象となる部分  
.AENDW  
~
```

コーディング例

```
TblSiz:  .ASSIGNA  50                                ; 積和演算を例に挙げます。  
        MOV      A_Tbl1,R1                          ;  R1:データテーブル1の先頭アドレス  
        MOV      A_Tblb,R2                          ;  R2:データテーブル2の先頭アドレス  
        CLRMAC   ; MACレジスタの初期化  
        .AWHILE  \&TblSiz GT 0                      ;  TblSiz が 0 より大きい間、  
            MAC.W  @R0+,@R1+                        ;  積和演算を繰り返してアセンブルします。  
TblSiz:  .ASSIGNA  \&TblSiz-1                        ;  TblSiz から 1 を引きます。  
        .AENDW  
        STS      MACL,R0                            ;  結果を R0 に得ます。
```

7.2 条件つきアセンブリ機能に関する制御文

条件つきアセンブリ機能の制御文には、次のものがあります。

.ASSIGNA	整数型プリプロセッサ変数を定義します。再定義が可能です。
.ASSIGNC	文字型プリプロセッサ変数を定義します。再定義が可能です。
.DEFINE	プリプロセッサ置換文字列を定義します。
.AIF .AELIF .AELSE .AENDI	ソースプログラムの一部分をアセンブルするか否か、条件によって切り替えます。条件成立の場合、.AIF 以降、不成立の場合、.AELIF または .AELSE 以降のソースプログラムをアセンブルします。
.AIFDEF .AELSE .AENDI	ソースプログラムの一部分をアセンブルするか否か、置換シンボルの定義によって切り替えます。置換シンボルが定義されている場合、.AIFDEF 以降、定義されていない場合、.AELSE 以降のソースプログラムをアセンブルします。
.AREPEAT .AENDR	ソースプログラムの一部分 (.AREPEAT と .AENDR の間) を指定の回数だけ繰り返し展開してアセンブルします。
.AWHILE .AENDW	ソースプログラムの一部分 (.AWHILE と .AENDW の間) を条件が成立している間、繰り返し展開してアセンブルします。
.AERROR	プリプロセッサ展開時のエラー処理をします。
.EXITM	.AREPEAT、.AWHILE による繰り返し展開を中断します。
.ALIMIT	プリプロセッサでの .AWHILE の展開の上限値を設定します。

.ASSIGNA 整数型プリプロセッサ変数を定義する（再定義が可能）

書式

プリプロセッサ変数名 [:] .ASSIGNA 値

ステートメントの要素

（１）ラベル

プリプロセッサ変数につける名前を記述します。

（２）オペレーション

ニーモニック .ASSIGNA を記述します。

（３）オペランド

プリプロセッサ変数に与える値を指定します。

解説

（１）.ASSIGNA は、プリプロセッサ変数を定義する制御文です。

プリプロセッサ変数名のつけ方は、シンボルの名づけ方と同じです。

また、プリプロセッサ変数名の最大文字数は 32 文字で、英大文字と英小文字を区別します。

（２）.ASSIGNA で定義したプリプロセッサ変数は、.ASSIGNA によって再定義できます。

（３）プリプロセッサ変数の値は、次の形式で指定します。

- ・ 定数（整数定数、文字定数）
- ・ 既に定義したプリプロセッサ変数
- ・ 上記を項とする式

（４）定義したプリプロセッサ変数は、本制御文より前方のソースステートメントに対して有効です。

(5) プリプロセッサ変数は、以下の箇所で参照できます。

- ・.ASSIGNA 制御文
- ・.ASSIGNC 制御文
- ・.AIF 制御文
- ・.AELIF 制御文
- ・.AREPEAT 制御文
- ・.AWHILE 制御文
- ・マクロ本体 (.MACRO ~ .ENDM 間のソースステートメント)

プリプロセッサ変数を参照する場合、前にバックスラッシュ(\)とアンパサンド(&)を付けて記述してください。*

\&プリプロセッサ変数名[]

アポストロフィ(')は、プリプロセッサ変数名とソースステートメントの区別を明確にしたい場合に記述します。

【注】* 日本語環境で使用する場合、'\の代わりに'¥'を使用してください。

(6) コマンドライン・オプションでプリプロセッサ文字列が定義されている場合、同名のプリプロセッサ変数に対する.ASSIGNAは無効となります。

コーディング例

```

Rn:      .REG      R0
SHIFT:   .ASSIGNA  27
; 汎用多ビットシフト命令を作ります
; SHIFT の値だけ右にシフトします
; Rn に R0 を設定します。
; SHIFT に 27 を設定します。

      .AIF \&SHIFT GE 16 ; 条件: SHIFT 16
      SHLR16 Rn          ; 条件成立なら、Rn を右に 16 ビットシフトします。
SHIFT: .ASSIGNA \&SHIFT-16 ; SHIFT から 16 を減じます。
      .AENDI

      .AIF \&SHIFT GE 8  ; 条件: SHIFT 8
      SHLR8 Rn           ; 条件成立なら、Rn を右に 8 ビットシフトします。
SHIFT: .ASSIGNA \&SHIFT-8 ; SHIFT から 8 を減じます。
      .AENDI

      .AIF \&SHIFT GE 4  ; 条件: SHIFT 4
      SHLR2 Rn           ; 条件成立なら、Rn を右に 4 ビットシフトします。
      SHLR2 Rn           ;
SHIFT: .ASSIGNA \&SHIFT-4 ; SHIFT から 4 を減じます。
      .AENDI

      .AIF \&SHIFT GE 2  ; 条件: SHIFT 2
      SHLR2 Rn           ; 条件成立なら、Rn を右に 2 ビットシフトします。
SHIFT: .ASSIGNA \&SHIFT-2 ; SHIFT から 2 を減じます。
      .AENDI

      .AIF \&SHIFT EQ 1  ; 条件: SHIFT = 1
      SHLR Rn            ; 条件成立なら、Rn を右に 1 ビットシフトします。
      .AENDI

```

展開結果は次のようになります。

```

SHLR16 R0 ; 条件成立なら、Rn を右に 16 ビットシフトします。
SHLR8 R0  ; 条件成立なら、Rn を右に 8 ビットシフトします。
SHLR2 R0  ; 条件成立なら、Rn を右に 2 ビットシフトします。
SHLR R0   ; 条件成立なら、Rn を右に 1 ビットシフトします。

```

.ASSIGNC 文字型プリプロセッサ変数を定義する（再定義が可能）

書式

プリプロセッサ変数名 [:] .ASSIGNC "文字列"

ステートメントの要素

(1) ラベル

プリプロセッサ変数に付ける名前を記述します。

(2) オペレーション

ニーモニック .ASSIGNC を記述します。

(3) オペランド

文字列をダブルコーテーション (") で囲んで記述します。

解説

(1) .ASSIGNC は、文字型プリプロセッサ変数を定義する制御文です。

プリプロセッサ変数名のつけ方は、シンボルの名づけかたと同じです。

また、プリプロセッサ変数名の最大文字数は 32 文字で、英大文字と英小文字を区別します。

(2) .ASSIGNC で定義したプリプロセッサ変数は、.ASSIGNC によって再定義できます。

(3) 文字列は、文字および既に定義したプリプロセッサ変数をダブルコーテーション (") で囲んで指定します。

(4) 定義したプリプロセッサ変数は、本制御文より前方のソースステートメントに対して有効です。

(5) プリプロセッサ変数は、以下の箇所で参照できます。

- ・ .ASSIGNA 制御文
- ・ .ASSIGNC 制御文
- ・ .AIF 制御文
- ・ .AELIF 制御文
- ・ .AREPEAT 制御文
- ・ .AWHILE 制御文
- ・ マクロ本体 (.MACRO ~ .ENDM 間のソースステートメント)

プリプロセッサ変数を参照する場合、前にバックスラッシュ(\)とアンパサンド(&)を付けて記述してください。*

\&プリプロセッサ変数名[]

アポストロフィ(')は、プリプロセッサ変数名とソースステートメントの区別を明確にしたい場合に記述します。

【注】* 日本語環境で使用する場合、\'の代わりに¥を使用してください。

(6) コマンドライン・オプションでプリプロセッサ文字列が定義されている場合、同名のプリプロセッサ変数に対する .ASSIGNC は無効となります。

コーディング例

```
FLAG:  .ASSIGNC  "ON  "           ; FLAG に "ON " を設定します。
~
.AIF  "\&FLAG" EQ "ON  " ; FLAG が "ON" のとき、
MOV  R0,R1                ; MOV R0,R1 をアセンブルします。
.AENDI
~
FLAG:  .ASSIGNC  "\&FLAG "       ; FLAG にスペース ( " ") を付加します。
FLAGA: .ASSIGNC  "OFF"          ; FLAGA に "OFF" を設定します。
FLAG:  .ASSIGNC  "\&FLAG'AND \&FLAGA"
                                           ; FLAG と AND の区別を明確にするため "'" を使います。
                                           ; FLAG は結果的に "ON AND OFF" になります。
~
```


.DEFINE プリプロセッサ置換文字列を定義する

書式

シンボル [:] .DEFINE "置換文字列"

ステートメントの要素

(1) ラベル

対応する置換シンボルを記述します。

(2) オペレーション

ニーモニック .DEFINE を記述します。

(3) オペランド

シンボルに対応する置換文字列をダブルコーテーション (") で囲んで記述します。

解説

(1) .DEFINE は、シンボルの対応する置換文字列に置き換えることを指定する制御文です。

(2) .DEFINE と .ASSIGNC との違いは以下の点です。

- (a) .ASSIGNC で定義したシンボルはプリプロセッサ文でしか使用できませんが、.DEFINE で定義したシンボルは、任意のステートメントで使用できます。
- (b) .ASSIGNA、.ASSIGNC で定義したシンボルは、「\&シンボル」の形式で参照しますが、.DEFINE で定義したシンボルは「シンボル」の形式で参照します。
- (c) .DEFINE で定義したシンボルは、再定義できません。

(3) コマンドライン・オプションで置換シンボルが定義されている場合、同名のシンボルに対する .DEFINE は無効となります。

コーディング例

SYM1: .DEFINE "R1"

~

MOV.L SYM1,R0

~

; MOV.L R1,R0 に置き換えられます。

【注意】

- (1) 先頭が a~f または、A~F で始まる 16 進数は、.DEFINE で同名のシンボルが定義された場合、置換対象になります。置換対象外にするには、先頭に 0 を付加してください。

```
A0: .DEFINE    "0"
      MOV.B    #H'A0,R0      ; MOV.B #H'0,R0 に置き換えられます。
      MOV.B    #H'0A0,R0    ; 置き換えられません。
```

- (2) 基数 (B'、Q'、D'、H') は、.DEFINE で同名のシンボルが定義された場合、置換対象になります。B、Q、D、H、b、q、d、h 一文字のシンボルを定義するときは注意してください。

```
B: .DEFINE    "H"
      MOV.B    #B'10,R0      ; MOV.H #H'10,R0 に置き換えられます。
```

.AIF, .AELIF, .AELSE, .AENDI 比較型条件つきアセンブル

書式

.AIF 項1 関係演算子 項2

AIF の条件成立時にアセンブルするソースステートメント

[.AELIF 項1 関係演算子 項2

AELIF の条件成立時にアセンブルするソースステートメント]

[.AELSE

全ての条件不成立時にアセンブルするソースステートメント]

.AENDI

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

.AIF、.AELIF (省略可)、.AELSE (省略可)、.AENDI を記述します。

(3) オペランド

.AIF : 比較型条件を記述します。記述方法は 解説で詳しく説明しています。

.AELIF : 比較型条件を記述します。記述方法は 解説で詳しく説明しています。

.AELSE : 記述できません。

.AENDI : 記述できません。

解説

(1) .AIF、.AELSE、.AELIF、.AENDI はアセンブルするか否かを条件によって切り替える制御文です。.AELIF と .AELSE は省略できます。

(2) .AELIF は、.AIF と .AELSE の間ならば繰り返し指定できます。

(3) 比較型条件は次のように記述してください。

.AIF 項1 関係演算子 項2

.AELIF 項1 関係演算子 項2

項は、値または文字列を記述します。ただし、値と文字列を比較すると常に条件不成立となります。

値は、定数またはプリプロセッサ変数で指定します。

文字列は、文字、またはプリプロセッサ変数をダブルコーテーション (") で囲んで指定します。ダブルコーテーション (") 自体を文字として指定する場合は、ダブルコーテーションを 2 つ続けて記述 ("") します。

(4) 関係演算子の条件は以下のとおりです。

EQ	項 1 = 項 2
NE	項 1 ≠ 項 2
GT	項 1 > 項 2
LT	項 1 < 項 2
GE	項 1 ≥ 項 2
LE	項 1 ≤ 項 2

【注】 値は 32 ビット符号つき整数として比較します。

文字列の比較は EQ, NE のみ有効です。

コーディング例

```

~
.AIF \&TYPE EQ 1
MOV R0,R3 ; TYPE が
MOV R1,R4 ; 1 のとき
MOV R2,R5 ; アセンブルします。
.AELIF \&TYPE EQ 2
MOV R0,R6 ; TYPE が
MOV R1,R7 ; 2 のとき
MOV R2,R8 ; アセンブルします。
.AELSE
MOV R0,R9 ; TYPE が
MOV R1,R10 ; 1 でも 2 でもないとき
MOV R2,R11 ; アセンブルします。
.AENDI
~

```

.AIFDEF, .AELSE, .AENDI 定義型条件つきアセンブル

書式

.AIFDEF 置換シンボル

置換シンボルが定義されていた時にアセンブルするソースステートメント

[.AELSE

置換シンボルが定義されていない時にアセンブルするソースステートメント]

.AENDI

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

.AIFDEF、.AELSE (省略可)、.AENDI を記述します。

(3) オペランド

.AIFDEF : 定義型条件を記述します。記述方法は 解説で詳しく説明しています。

.AELSE : 記述できません。

.AENDI : 記述できません。

解説

(1) .AIFDEF、.AELSE、.AENDI はアセンブルするか否かを置換シンボルの定義によって切り替える制御文です。.AELSE は省略できます。

(2) 定義型条件は次のように記述してください。

.AIFDEF 置換シンボル

置換シンボルは、.DEFINE 制御文で定義します。

記述した置換シンボルがコマンドライン・オプションで定義されている、または本制御文より後方のソースステートメントで定義されている場合、条件成立となります。

記述した置換シンボルが本制御命令より前方のソースステートメントで定義されている、または定義がない場合、条件不成立となります。

コーディング例

~

.AIFDEF FLAG

MOV R0,R3

; FLAG が .DEFINE 制御文で定義されているとき

MOV R1,R4

; アセンブルします。

.AELSE

MOV R0,R6

; FLAG が .DEFINE 制御文で定義されていないとき

MOV R1,R7

; アセンブルします。

.AENDI

~

.AREPEAT, .AENDR 繰り返し展開

書式

```
.AREPEAT 回数  
繰り返し展開してアセンブルするソースステートメント  
.AENDR
```

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

.AREPEAT、.AENDR を記述します。

(3) オペランド

.AREPEAT : 繰り返し展開する回数を記述します。

.AENDR : 記述できません。

解説

- (1) .AREPEAT、.AENDR は指定された回数だけ繰り返し展開してアセンブルする制御文です。
- (2) .AREPEAT で指定された回数だけ、.AREPEAT ~ .AENDR の間に記述したソースステートメントを繰り返し展開してアセンブルします (ソースステートメントを繰り返しコピーするのと同じで、実行時のループにはなりません)。
- (3) 回数は、定数またはプリプロセッサ変数で指定します。
- (4) 回数に 0 以下の値を指定した場合は、展開しません。

コーディング例

```
                                ; 64 ビット ÷ 32 ビットの除算を例に挙げます。
                                ; R1:R2 ( 64 ビット ) ÷ R0 ( 32 ビット ) = R2 ( 32 ビット ) : 符号無し
TST      R0,R0                ; ゼロ除算チェック
BT       zero_div
CMP/HS   R0,R1                ; オーバフローチェック
BT       over_div
DIV0U                                ; フラグの初期化
.AREPEAT 32
    ROTCL R2                    ; 32 回繰り返してアセンブルします。
    DIV1  R0,R1                ;
.AENDR
ROTCL    R2                    ; R2 = 商
```


.AWHILE, .AENDW 条件つき繰り返し展開

書式

```
.AWHILE 項1 関係演算子 項2  
繰り返し展開してアセンブルするソースステートメント  
.AENDW
```

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

.AWHILE、.AENDW を記述します。

(3) オペランド

.AWHILE：繰り返し展開する条件を記述します。

.AENDW：記述できません。

解説

(1) .AWHILE、.AENDW は条件が成立している間だけ繰り返し展開してアセンブルする制御文です。

(2) .AWHILE で指定した条件が成立している間、.AWHILE～.AENDW の間に記述したソースステートメントを繰り返し展開してアセンブルします（ソースステートメントを繰り返しコピーするのと同じで、実行時のループにはなりません）。

(3) 条件は次のように記述してください。

```
.AWHILE 項1 関係演算子 項2
```

項は、値または文字列を記述します。ただし、値と文字列を比較すると常に条件不成立となります。

値は、定数またはプリプロセッサ変数で指定します。

文字列は、文字、またはプリプロセッサ変数をダブルコーテーション（"）で囲んで指定します。ダブルコーテーション（"）自体を文字として指定する場合は、ダブルコーテーションを2つ続けて記述（""）します。

条件つき繰り返し展開は、最終的に条件を不成立にして展開を終了します。

【注意】 条件が不成立にならない場合は、65,535 回または ALIMIT 制御文で指定した展開回数を繰り返しますので、条件の指定にはよく注意してください。

(4) 関係演算子の条件は以下のとおりです。

EQ	項 1 = 項 2
NE	項 1 ≠ 項 2
GT	項 1 > 項 2
LT	項 1 < 項 2
GE	項 1 ≥ 項 2
LE	項 1 ≤ 項 2

【注】 値は 32 ビット符号つき整数として比較します。
文字列の比較は EQ, NE のみ有効です。

コーディング例

	; 積和演算を例に挙げます。
TblSiz: .ASSIGNA 50	; TblSiz: データテーブルの大きさ
MOV A_Tbl1, R1	; R1: データテーブル 1 の先頭アドレス
MOV A_Tbl2, R2	; R2: データテーブル 2 の先頭アドレス
CLRMAC	; MAC レジスタの初期化
.AWHILE \&TblSiz GT 0	; TblSiz が 0 より大きい間、
MAC.W @R0+, @R1+	; 積和演算を繰り返してアセンブルします。
TblSiz: .ASSIGNA \&TblSiz-1	; TblSiz から 1 を引きます。
.AENDW	
STS MACL, R0	; 結果を R0 に得ます。

.AERROR プリプロセッサ展開時のエラー処理をする

書式

.AERROR

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック **.AERROR** を記述します。

(3) オペランド

記述できません。

解説

(1) **.AERROR** をアセンブルすると、エラー667 を発生し、アセンブラをエラー終了します。

(2) **.AERROR** は、プリプロセッサ変数の値のチェック等に使用することができます。

コーディング例

```
~  
.AIF      \&FLG EQ 1  
    MOV   R1,R10  
    MOV   R2,R11  
.AELSE  
    .AERROR           ; \&FLG が 1 以外の場合エラーとします。  
.AENDI  
~
```

.EXITM 展開の中断終了

書式

.EXITM

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .EXITM を記述します。

(3) オペランド

記述できません。

解説

(1) .EXITM は繰り返し展開 (.AREPEAT ~ .AENDR) および、条件つき繰り返し展開 (.AWHILE ~ .AENDW) の展開を中断させるための制御文です。

(2) 各展開では、本制御文が出現した時点で、展開を中断します。

(3) 本制御文は、マクロ展開の中断終了にも使用します。マクロ命令と繰り返し展開を組み合わせて使用する場合は、本制御文の位置に注意してください。

【参照】 マクロ展開 言語編「8.2 マクロ機能に関する制御文」

コーディング例

~

```
COUNT: .ASSIGNA 0           ; COUNT に 0 を設定しています。
      .AWHILE  1 EQ 1       ; 無限展開 (常に条件成立) を指定しています。
      ADD     R0,R1
      ADD     R2,R3
```

```
COUNT: .ASSIGNA \&COUNT+1 ; COUNT に 1 を加えます。
      .AIF     \&COUNT EQ 2 ; 条件は COUNT = 2 です。
      .EXITM   ; 条件成立で .AWHILE を中断終了します。
      .AENDI
      .AENDW
```

~

COUNT が更新され、.AIF の条件が成立すると、.EXITM がアセンブルされます。.EXITM がアセンブルされた時点で、.AWHILE の展開を中断終了します。

したがって、展開結果は以下のようになります。

```
ADD R0,R1      ..... COUNT が 0 のとき
ADD R2,R3
ADD R0,R1      ..... COUNT が 1 のとき
ADD R2,R3
```

この後、COUNT は 2 となり、展開は中断終了します。

.ALIMIT プリプロセッサでの.AWHILE の展開の上限値を設定する

書式

.ALIMIT 回数

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .ALIMIT を記述します。

(3) オペランド

ステートメントの展開回数を記述します。

解説

- (1) 条件つき繰り返し展開 (.AWHILE ~ .AENDW) で、ステートメントの展開回数が .ALIMIT で指定した上限値を越えると、ウォーニング 854 となり、展開を打ち切ります。

- (2) 展開回数の限界値は、.ALIMIT を指定しないときは 65,535 です。

コーディング例

```
        .ALIMIT      20
        ~
FLG:     .ASSIGNA     0
        .AWHILE      \&FLG EQ 0      ; 20 回展開した後、展開を打ち切り
        NOP          ; ウォーニングとします。
        .AENDW
        ~
```

8. マクロ機能

8.1 マクロ機能の概要

本アセンブリ言語では、プログラム中でよく使用する一連の処理に名前をつけ、1つの命令（マクロ命令）として定義することができます。このような定義をマクロ定義といいます。

マクロ定義の方法は、次のとおりです。

```
~  
.MACRO   マクロ名  
マクロ本体  
.ENDM  
~
```

マクロ名はマクロ命令につける名前、マクロ本体はマクロ命令の内容です。

定義したマクロ命令を呼び出して使用することを、マクロコールといいます。
マクロコールの方法は、次のとおりです。

```
~  
定義済みのマクロ名  
~
```

マクロ定義とマクロコールの例を以下に示します。

コーディング例

~

```
.MACRO SUM ; R0,R1,R2,R3 の合計を求める処理を、  
MOV R0,R10 ; マクロ命令 SUM として定義します。  
ADD R1,R10  
ADD R2,R10  
ADD R3,R10  
.ENDM  
~
```

```
SUM ; マクロ命令 SUM を呼び出します。  
  
; マクロ本体 MOV R0,R10  
; ADD R1,R10  
; ADD R2,R10  
; ADD R3,R10 が展開されます。
```

定義したマクロ命令を、一部変更して展開することも可能です。

手順は次のとおりです。

(1) マクロ定義

- (a) .MACRO 文で仮引数を定義（マクロ名につづいて記述）します。
- (b) マクロ本体の記述に仮引数を使います。（仮引数の先頭にバックスラッシュ（\）を付けます）

(2) マクロコール

マクロパラメータを付けてマクロ命令を呼出します。

マクロ命令展開の際、仮引数は対応するマクロパラメータに置き換えられます。

コーディング例

~

```
.MACRO  SUM ARG1    ; 仮引数 ARG1 を定義します。  
MOV  R0, \ARG1      ; ARG1 を使ってマクロ本体を記述しています。  
ADD  R1, \ARG1  
ADD  R2, \ARG1  
ADD  R3, \ARG1  
.ENDM
```

~

```
SUM R10    ; マクロパラメータ R10 を付けて、マクロ命令 SUM を呼び出します。  
            ; マクロ本体中の仮引数がマクロパラメータで置き換えられ、  
            ;      MOV  R0,R10  
            ;      ADD  R1,R10  
            ;      ADD  R2,R10  
            ;      ADD  R3,R10    が展開されます。
```

8.2 マクロ機能に関する制御文

マクロ機能の制御文には、次のものがあります。

. MACRO	マクロ命令を定義します。
. ENDM	
. EXITM	マクロ命令の展開を中断します。

.MACRO, .ENDM マクロ命令を定義する

書式

.MACRO マクロ名 [仮引数 [= 仮引数のデフォルト] [, 仮引数 ...]]

.ENDM

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

.MACRO、.ENDM を記述します。

(3) オペランド

.MACRO : 定義するマクロ命令の名前と仮引数 (省略可) を記述します。仮引数を記述した場合、仮引数のデフォルト (省略可) を記述します。

.ENDM : 記述できません。

解説

(1) .MACRO、.ENDM はマクロ定義 (一連のソースステートメントに名前を付けてまとめて扱うこと) を行なう制御文です。

(2) マクロ定義

.MACRO ~ .ENDM 間のソースステートメント (マクロ本体) を、マクロ命令として名前を付けることをマクロ命令を定義する (マクロ定義) といいます。

(3) マクロ名

マクロ名はマクロ命令に付ける名前です。

(4) 仮引数

マクロ命令を展開する際にマクロ本体の一部を置換して展開したい場合に指定します。仮引数はマクロ展開を行なう (マクロコール) 時に指定された文字列 (マクロパラメータ) に置換されます。

(a) 仮引数の書き方

仮引数の書き方は、シンボル名の書き方を同じです。

また、仮引数名の最大文字数は 32 文字で、英大文字と英小文字を区別します。

(b) 仮引数の参照

マクロ本体では、置換したい部分に仮引数名を記述します。

マクロ本体での仮引数の参照方法は、次のとおりです。

`\仮引数名[] *`

アポストロフィー (') は、仮引数名とソースステートメントの区別を明確にしたい場合に記述します。

【注】* 日本語環境で使用する場合、' の代わりに ¢ を使用してください。

(5) 仮引数のデフォルト

仮引数には、仮引数のデフォルトを設定できます。仮引数のデフォルトには、マクロコール時にマクロパラメータを省略した場合に置換する文字列を指定します。

仮引数のデフォルトに次の文字を含む場合は、文字列をダブルコーテーション (")、または、アングルブラケット (<>) で囲んでください。

- ・スペース
- ・タブ
- ・カンマ (,)
- ・セミコロン (;)
- ・ダブルコーテーション (")
- ・アングルブラケット (<>)

マクロ展開では、文字列を囲んだダブルコーテーション (") やアングルブラケット (<>) は取り除いて置換します。

(6) マクロ命令の制限

(a) マクロ命令は、次の場所では定義できません。

- ・マクロ本体 (.MACRO ~ .ENDM)
- ・.AREPEAT ~ .AENDR の間
- ・.AWHILE ~ .AENDW の間

(b) マクロ本体には、.END は記述できません。

(c) .ENDM のラベルには、シンボルは記述できません。.ENDM のラベルにシンボルを記述した場合は、.ENDM を無視します。この場合、エラーは表示しません。

コーディング例

~

```
.MACRO  SUM                ; R0,R1,R2,R3 の合計を求める処理を、  
MOV  R0,R10                ; マクロ命令 SUM として定義します。  
ADD  R1,R10  
ADD  R2,R10  
ADD  R3,R10  
.ENDM
```

~

```
SUM                ; マクロ命令 SUM を呼び出します。  
; マクロ本体      MOV  R0,R10  
;                  ADD  R1,R10  
;                  ADD  R2,R10  
;                  ADD  R3,R10   が展開されます。
```

.EXITM マクロ展開の中断終了

書式

.EXITM

ステートメントの要素

(1) ラベル

記述できません。

(2) オペレーション

ニーモニック .EXITM を記述します。

(3) オペランド

記述できません。

解説

(1) .EXITM はマクロ展開を中断させるための制御文です。

本制御文は、マクロ本体(.MACRO ~ .ENDM 間)で、指定できます。

(2) マクロ展開では、本制御文が出現した時点で、展開を中断終了します。

(3) 本制御文は、.AREPEAT、.AWHILE の繰り返し展開の中断終了にも使用します。マクロ命令と繰り返し展開を組み合わせる場合は、本制御文の位置に注意してください。

コーディング例

```

.MACRO      SUM P1
MOV         R0,R10    - - - -
ADD         R1,R10    - - - - (1)
ADD         R2,R10    - - - -
\P1         - - - - (2)
ADD         R3,R10
.ENDM

~

SUM         .EXITM

```

(2)で、.EXITM を展開し、マクロ展開を中断終了します。(1)の部分が展開されます。

8.3 マクロ本体

.MACRO と .ENDM の間に記述した一連のソースステートメントをマクロ本体と呼びます。

マクロ本体は、マクロコール(マクロ命令を呼び出すこと)により、展開してアセンブルされます。

本節ではマクロ本体が持つ機能と記述方法を説明します。

(1) 仮引数の参照

マクロ展開で、マクロパラメータと置換したい部分に仮引数を記述します。

仮引数の参照方法は以下のとおりです。

\仮引数[] *

【注】* 日本語環境で使用する場合、\'\'の代わりに¥¥を使用してください。

アポストロフィ(')は、仮引数名とソースステートメントの区別を明確にしたい場合に記述します。

コーディング例

```
.MACRO    PLUS1 P,P1      ; P,P1 は仮引数です。
ADD       #1,\P1         ; 仮引数 P1 を参照しています。
.SDATA    "\P'1"         ; 仮引数 P を参照しています。
.ENDM

PLUS1     R,R1            ; PLUS1 を展開します。
```

展開の結果は次のようになります。

```
ADD       #1,R1          ; 仮引数 P1 を参照しています。
.SDATA    "R1"           ; 仮引数 P を参照しています。
```

(2) プリプロセッサ変数の参照

マクロ本体では、プリプロセッサ変数を参照できます。

プリプロセッサ変数の参照方法は次のとおりです。

\&プリプロセッサ変数名[] *

【注】* 日本語環境で使用する場合、\'\'の代わりに¥¥を使用してください。

アポストロフィ(')は、プリプロセッサ変数とソースステートメントの区別を明確にしたい場合に記述します。

コーディング例

```
.MACRO    PLUS1
ADD      #1,R\&V1    ; プリプロセッサ変数 V1 を参照しています。
.SDATA   "\&V'1"     ; プリプロセッサ変数 V を参照しています。
.ENDM

V        .ASSIGNC "R"      ; プリプロセッサ変数 V を定義しています。
V1       .ASSIGNA 1        ; プリプロセッサ変数 V1 を定義しています。

PLUS1    ; PLUS1 を展開します。
```

展開結果は次のようになります。

```
ADD      #1,R1        ; プリプロセッサ変数 V1 を参照しています。
.SDATA   "R1"         ; プリプロセッサ変数 V を参照しています。
```

(3) マクロ生成番号

マクロ本体にラベルがある場合などは、複数回マクロコールをするとシンボル名が重複してしまいます。

このような事態を回避するためにはマクロ生成番号を使用してください。

マクロ生成番号は、マクロ展開で固有の 5 桁の 10 進数 (00000 ~ 99999) を展開します。マクロ生成番号は次のように記述してください。

\@ *

【注】* 日本語環境で使用する場合は、\の代わりに¥を使用してください。

シンボル名の一部としてマクロ生成番号を記述しておく、マクロコールのたびに固有のシンボル名となり、重複を避けることができます。

1 つのマクロ本体に、2 つ以上のマクロ生成番号を記述できますが、1 回のマクロコールでは、同じマクロ生成番号が展開されます。

【注意】 マクロ生成番号は数字に展開されるので、シンボル名の先頭には記述しないでください。

【参照】 言語編「1.3.2 シンボルの名づけ方」

コーディング例

```

        .MACRO      RES_STR STR,Rn
        MOV.L      #str\@,\Rn
        BRA        end_str\@
        NOP
str\@    .SDATA      "\STR"
        .ALIGN     2
end_str\@
        .ENDM
RES_STR  "ONE",R0      ; RES_STR を展開するたびに、
RES_STR  "TWO",R1      ; 異なるシンボルを生成します。

```

展開結果は次のようになります。

```

        MOV.L      #str00000,R0
        BRA        end_str00000
        NOP
str00000 .SDATA      "ONE"
        .ALIGN     2
end_str00000
        MOV.L      #str00001,R1
        BRA        end_str00001
        NOP
str00001 .SDATA      "TWO"
        .ALIGN     2
end_str00001

```

(4) マクロ処理除外

マクロ本体内にバックスラッシュ (\) があると、マクロ置換処理の対象になります。したがって、バックスラッシュ (\) を ASCII 文字として記述したい場合は、マクロ置換処理から除外する必要があります。

マクロ処理除外の書き方は、次のとおりです。

\(マクロ処理除外文字列)*

【注】 * 日本語環境で使用する場合は、\'の代わりに'を使用してください。

マクロ展開ではバックスラッシュ (\) とカッコは取り除きます

コーディング例

```
.MACRO BACK_SLASH_SET
\ (MOV      #"\",R0)           ; \はASCII 文字として展開されます。

.ENDM
```

展開結果は次のようになります。

```
MOV      #"\",R0           ; \はASCII 文字として展開されます。
```

(5) マクロ内コメント

マクロ本体のコメントを、マクロ展開では展開したくない(リスティングファイルに同じコメントが何度も現れるのを避けたい) 場合に、マクロ内コメントを記述します。

マクロ内コメントの書き方は次のとおりです。

```
\;コメント *
```

【注】* 日本語環境で使用する場合、\'の代わりに'を使用してください。

コーディング例

```
.MACRO PUSH Rn
MOV.L      \Rn,@-R15  \; \Rn はレジスタです。
.ENDM
```

```
PUSH      R0
```

展開結果は次のようになります(コメントは展開されません)。

```
MOV.L      R0,@-R15
```

(6) 文字列操作関数

マクロ本体には、文字列操作関数を記述できます。

文字列操作関数には次のものがあります。

```
.LEN  関数    : 文字列の長さ
.INSTR 関数    : 文字列の検索
.SUBSTR 関数    : 文字列の切り出し
```

【参照】 .LEN 言語編「8.5 文字列操作関数 .LEN」
.INSTR 言語編「8.5 文字列操作関数 .INSTR」
.SUBSTR 言語編「8.5 文字列操作関数 .SUBSTR」

8.4 マクロコール

マクロ定義により定義されたマクロ命令を展開することをマクロコールといいます。
マクロコールの書き方は次のとおりです。

書式

[シンボル[:]] マクロ名[マクロパラメータ[,マクロパラメータ …]]

ステートメントの要素

(1) ラベル

必要であれば、目印となるシンボルを記述します。

(2) オペレーション

展開したいマクロ名を記述します。呼び出すマクロ命令は、マクロコール以前にマクロ定義してください。

(3) オペランド

マクロパラメータには、マクロ展開で置換する文字列を指定します。この場合、マクロ名に対応するマクロ定義(.MACRO)で、仮引数を宣言しておく必要があります。

解説

(1) マクロパラメータの指定方法

マクロパラメータの指定方法には、位置指定とキーワード指定があります。

(a) 位置指定

マクロ定義(.MACRO)で宣言した仮引数の並び順と、マクロパラメータの並び順を一致させて指定する方法です。

(b) キーワード指定

マクロ定義(.MACRO)で宣言した仮引数の仮引数名に、イコール(=)で区切って指定する方法です。

(2) マクロパラメータの書き方

マクロパラメータに次の文字を含む場合は、文字列をダブルコーテーション(")、または、アングルブラケット(<>)で囲んでください。

- ・スペース
- ・タブ
- ・カンマ(,)
- ・セミコロン(;)
- ・ダブルコーテーション(")
- ・アングルブラケット(<>)

マクロ展開では、文字列を囲んだダブルコーテーションやアングルブラケットは取り除いて置換します。

コーディング例

```

.MACRO  SUM FROM=0,TO=9 ; マクロ命令 SUM、仮引数 FROM,TO を定義します。
      MOV      R\FROM,R10
COUNT .ASSIGNA      \FROM+1
      .AWHILE \&COUNT LE \TO
      MOV      R\&COUNT,R10
COUNT .ASSIGNA      \&COUNT+1
      .AENDW
      .ENDM

```

仮引数を用いてマクロ本体を記述しています。

```

SUM      0,5
SUM      TO=5

```

どちらも 同じ展開結果になります。

マクロ本体中の仮引数がマクロパラメータで置き換えられ、展開結果は次のようになります。

```

MOV      R0,R10
MOV      R1,R10
MOV      R2,R10
MOV      R3,R10
MOV      R4,R10
MOV      R5,R10

```


8.5 文字列操作関数

マクロ本体で利用できる文字列操作関数には、次のものがあります。

. LEN	文字列の長さを返します。
. INSTR	文字列の検索を行ないます。
. SUBSTR	文字列の切り出しを行ないます。

.LEN 文字列の長さを返す文字列操作関数

書式

.LEN [] ("文字列")

解説

(1) .LEN は、文字列の長さを数え、文字数を基数を省略した 10 進数に置換します。

(2) 文字列は、文字をダブルコーテーション (") で囲んで指定します。

ダブルコーテーション (") 自体を文字として指定する場合は、2 つ続けて記述します。

(3) 文字列には、マクロの仮引数、プリプロセッサ変数を指定できます。

.LEN ("\仮引数名")

.LEN ("&プリプロセッサ変数名") *

【注】* 日本語環境で使用する場合、\'\'の代わりに¥¥を使用してください。

(4) 本関数を記述できるのは、マクロ本体 (.MACRO ~ .ENDM) だけです。

コーディング例

~

```
.MACRO  RESERVE_LENGTH P1
```

```
.ALIGN  4
```

```
.SRES   .LEN( "\P1" )
```

```
.ENDM
```

~

```
RESERVE_LENGTH ABCDEF
```

```
RESERVE_LENGTH ABC
```

展開結果は次のようになります。

```
.ALIGN  4
```

```
.SRES   6           "ABCDEF" の字数は 6 です。
```

```
.ALIGN  4
```

```
.SRES   3           "ABC" の字数は 3 です。
```

.INSTR 文字列の検索を行なう文字列操作関数

書式

.INSTR [] ("文字列 1", "文字列 2" [, 検索開始位置])

解説

- (1) .INSTR は、文字列 1 に文字列 2 が含まれているかを検索し、文字列の先頭を 0 とした検索位置を基数を省略した 10 進数に置換します。
文字列 1 に文字列 2 が含まれていない場合は、-1 に置換します。
- (2) 文字列は、文字をダブルコーテーション (") で囲んで指定します。
ダブルコーテーション (") 自体を文字として指定する場合は、2 つ続けて記述します。
- (3) 検索開始位置は、文字列 1 の先頭を 0 とした数値で指定します。省略した場合は 0 を設定します。
- (4) 文字列、検索開始位置には、マクロの仮引数、プリプロセッサ変数を指定できます。
.INSTR("\仮引数名",)
.INSTR("\&プリプロセッサ変数名",) *
【注】* 日本語環境で使用する場合、\'\'の代わりに¥¥を使用してください。
- (5) 本関数を記述できるのは、マクロ本体 (.MACRO ~ .ENDM) だけです。

コーディング例

```
~
.MACRO FIND_STR P1
.DATA.W  .INSTR("ABCDEFG", "\P1", 0)
.ENDM
~
FIND_STR    CDE
FIND_STR    H
```

展開結果は次のようになります。

```
.DATA.W 2      "ABCDEFG" の 2 文字目 (先頭を 0 とします) に "CDE" があります。
.DATA.W -1     "ABCDEFG" の中に "H" はありません。
```

.SUBSTR 文字列の切り出しを行なう文字列操作関数

書式

.SUBSTR [] ("文字列", 切り出しの開始位置, 切り出しの長さ)

解説

- (1) .SUBSTR は、文字列から文字列の先頭を 0 とした切り出しの開始位置から、切り出しの長さ分の文字列を切り出し、ダブルコーテーション (") で囲んだ文字列に置換します。
 - (2) 文字列は、文字をダブルコーテーション (") で囲んで指定します。
ダブルコーテーション (") 自体を文字として指定する場合は、2 つ続けて記述します。
 - (3) 切り出しの開始位置は、0 以上が指定できます。
切り出しの長さは、1 以上が指定できます。
 - (4) 切り出しの開始位置、切り出しの長さが不適当な場合には、空文字("")に置換します。
 - (5) 切り出しの開始位置、切り出しの長さには、マクロの仮引数、プリプロセッサ変数を指定できます。
.SUBSTR("\仮引数名",)
.SUBSTR("&プリプロセッサ変数名",) *
- 【注】* 日本語環境で使用する場合は、\'\'の代わりに''を使用してください。
- (6) 本関数を記述できるのは、マクロ本体 (.MACRO ~ .ENDM) だけです。

コーディング例

~

```
.MACRO RESERVE_STR P1=0,P2
.SDATA .SUBSTR("ABCDEFGH",\P1,\P2)
.ENDM
```

~

```
RESERVE_STR 2,2
RESERVE_STR ,3 ; マクロパラメータ P1 を省略しています。
```

展開結果はそれぞれ次のようになります。

```
.SDATA "CD"
.SDATA "ABC"
```

9. リテラルプール自動生成機能

9.1 リテラルプール自動生成機能の概要

2 バイト長、4 バイト長の定数データ（以下、リテラルといいます）をレジスタへ転送するには、リテラルプール（リテラルの集まり）を確保し、PC 相対アドレス形式で参照しなければなりません。リテラルプールを配置する場合、次のような考慮をする必要があります。

- ・データ転送命令が参照できる範囲にデータが位置しているか？
- ・2 バイト長のデータは 2 バイト境界、4 バイト長のデータは 4 バイト境界に位置しているか？
- ・1 つのデータを、複数のデータ転送命令で共有できないか？
- ・プログラム中のどこにリテラルプールを配置するか？

リテラルプール自動生成機能とは、定数データのレジスタ転送に対応する PC 相対の MOV 命令（または MOVA 命令）と .DATA 制御文を 1 つの命令から自動的に生成する機能です。

例えば、以下のプログラム（a）は本機能を用いれば（b）のように記述できます。

プログラム（a）

```
MOV.L    DATA1,R0
MOV.L    DATA2,R1
~
.ALIGN   4
DATA1    .DATA.L  H'12345678
DATA2    .DATA.L  500000
```

プログラム（b）

```
MOV.L    #H'12345678,R0
MOV.L    #500000,R1
~
```

9.2 リテラルプール自動生成機能に関する拡張命令

拡張命令 (`MOV.W #imm,Rn`、`MOV.L #imm,Rn`、`MOVA #imm,R0`) の記述に対して、アセンブラは必要なリテラルプールを自動生成し、PC 相対のディスプレースメント値を計算します。

表 9.1 に、拡張命令のソースステートメントとその展開結果を示します。拡張命令のソースステートメントは、1 つの実行命令と 1 つのリテラルデータとに展開されます。

表 9.1 拡張命令の種類と展開結果

拡張命令	展開結果
<code>MOV.W #imm,Rn</code>	<code>MOV.W @(disp,PC),Rn</code> と 2 バイト長のリテラルデータ
<code>MOV.L #imm,Rn</code>	<code>MOV.L @(disp,PC),Rn</code> と 4 バイト長のリテラルデータ
<code>MOVA #imm,R0</code>	<code>MOVA @(disp,PC),R0</code> と 4 バイト長のリテラルデータ

9.3 リテラルプール自動生成機能のサイズモード

リテラルプール自動生成には、オペレーションサイズを指定したデータ転送命令（拡張命令）により、リテラルプールを自動生成するサイズ指定モードと、オペレーションサイズの指定がないデータ転送命令をアセンブラが `imm` の値により、適切な命令を選択するサイズ選択モードの2つのサイズモードがあります。

表 9.2 に、データ転送命令とサイズモードの関係を示します。

表 9.2 データ転送命令とサイズモードの関係

データ転送命令	サイズ指定モード	サイズ選択モード
MOV <code>#imm,Rn</code>	実行命令	アセンブラが選択
MOV.B <code>#imm,Rn</code>	実行命令	実行命令
MOV.W <code>#imm,Rn</code>	拡張命令	拡張命令
MOV.L <code>#imm,Rn</code>	拡張命令	拡張命令

（１）サイズ指定モード

サイズ指定モードでは、オペレーションサイズ指定がないデータ転送命令（`MOV #imm,Rn`）は、通常の実行命令として扱われます。

サイズ指定モードは、コマンドライン・オプションで `- AUTO_LITERAL` が指定されていない場合に有効となります。

（２）サイズ選択モード

サイズ選択モードでは、拡張命令のほかにオペレーションサイズを指定していないデータ転送命令（`MOV #imm,Rn`）の記述に対して、アセンブラが `imm` の値の範囲を判定し、必要ならばリテラルプールを自動生成します。`imm` 値は、符号付きの範囲で判定します。

サイズ選択モードは、コマンドライン・オプションで `- AUTO_LITERAL` を指定した場合に有効となります。

表 9.3 に、サイズ選択モードで `imm` の値の範囲により選択される命令を示します。

表 9.3 サイズ選択モードで選択される命令

imm の指定方法	imm の値の範囲*	選択される命令
定数値または 後方参照の絶対値	H'FFFFFF80 ~ H'0000007F (-128 ~ 127)	MOV.B #imm,Rn
	H'FFFF8000 ~ H'FFFFFF7F (-32,768 ~ -129)	MOV.W #imm,Rn
	H'00000080 ~ H'00007FFF (128 ~ 32,767)	展開結果 (MOV.W @(disp,PC),Rn と 2 バイト長のリテラルデータ)
	H'80000000 ~ H'FFFF7FFF (-2,147,483,648 ~ -32,769) H'00008000 ~ H'7FFFFFFF (32,768 ~ 2,147,483,647)	MOV.L #imm,Rn 展開結果 (MOV.L @(disp,PC),Rn と 4 バイト長のリテラルデータ)
相対値または 前方参照の絶対値	imm の値に依存しない	MOV.L #imm,Rn 展開結果 (MOV.L @(disp,PC),Rn と 4 バイト長のリテラルデータ)

【注】 * () 内は 10 進表記

【参照】 - AUTO_LITERAL

操作編 「2.2.8 リテラルプール自動生成機能に関するコマンドライン・オプション」

9.4 リテラルプールの出力

リテラルプールが出力されるポイントは、次のどちらかです。

- ・無条件分岐とそのディレイスロット命令に続く位置
- ・プログラマが .POOL を記述した位置

アセンブラは拡張命令の位置から見て前方で最も近い出力ポイントに、対応するリテラルを出力します。

アセンブラは、同じポイントに出力するリテラルデータをまとめて 1 つのリテラルプールを生成します。

【注意】 ディレイスロット命令にラベルが指定されている場合、リテラルプール出力ポイントにはなりません。

9.4.1 無条件分岐を利用したリテラルプール出力

出力例を以下に示します。

コーディング例

ソースプログラム

```
. SECTION CD1, CODE, LOCATE = H' 0000F000
CD1_START:
  MOV. L    #H' FFFF0000, R0
  MOV. W    #H' FF00, R1
  MOV. L    #CD1_START, R2
  MOV      #H'FF, R3
  RTS
  MOV      R0, R10
. END
```

リテラルプール自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000	2	CD1_START
3	0000F000 D003	3	MOV. L #H' FFFF0000, R0
4	0000F002 9103	4	MOV. W #H' FF00, R1
5	0000F004 D203	5	MOV. L #CD1_START, R2
6	0000F006 E3FF	6	MOV #H'FF, R3
7	0000F008 000B	7	RTS
8	0000F00A 6A03	8	MOV R0, R10
9			***** BEGIN-POOL *****
10	0000F00C FF00		DATA FOR SOURCE-LINE 4
11	0000F00E 0000		ALIGNMENT CODE
12	0000F010 FFFF0000		DATA FOR SOURCE-LINE 3
13	0000F014 0000F000		DATA FOR SOURCE-LINE 5
14			***** END-POOL *****
15		9	. END

9.4.2 .POOL の位置へのリテラルプール出力

無条件分岐を利用したリテラルプール出力がディスプレースメントの範囲内に行えない場合（プログラム中に無条件分岐が少ない場合など）アセンブラはエラー402 を出力します。このような場合、.POOL 制御命令をディスプレースメントの範囲内に記述してください。

ディスプレースメントの範囲は次のとおりです。

- ・オペレーションサイズがワード（W）のとき : 0 ~ 511 バイト
- ・オペレーションサイズがロング（L）のとき : 0 ~ 1023 バイト

.POOL の位置へのリテラルプール出力では、そのリテラルプールを飛び越すための分岐命令も一緒に出力されます。

コーディング例

ソースプログラム

```
. SECTION CD1, CODE, LOCATE = H' 0000F000
CCD1_START
    MOV. L    #H' FFFF0000, R0
    MOV. W    #H' FF00, R1
    MOV. L    #CD1_START, R2
    MOV      #H' FF, R3
    .POOL
    .END
```

リテラルプール自動生成結果（ソースリスト）

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000	2	CD1_START:
3	0000F000 D003	3	MOV. L #H' FFFF0000, R0
4	0000F002 9103	4	MOV. W #H' FF00, R1
5	0000F004 D203	5	MOV. L #CD1_START, R2
6	0000F006 E3FF	6	MOV #H' FF, R3
7	0000F008	7	.POOL
8			***** BEGIN-POOL *****
9	0000F008 A006		BRA TO END-POOL
10	0000F00A 0009		NOP
11	0000F00C FF00		DATA FOR SOURCE-LINE 4
12	0000F00E 0000		ALIGNMENT CODE
13	0000F010 FFFF0000		DATA FOR SOURCE-LINE 3
14	0000F014 0000F000		DATA FOR SOURCE-LINE 5
15			***** END-POOL *****
16		8	.END

9.5 リテラルの共有

アセンブラは、同一のリテラルプールに入る複数の拡張命令の `imm` を共有する処理を行ないます。以下に示すもので同一のものを共有します。

- (1) シンボル
- (2) 定数
- (3) シンボル ± 定数

上記の他、アセンブル時に同一の値を持つと判断できる式は共有する場合があります。

`imm` の値が同じでも、拡張命令のオペレーションサイズが異なる場合は、リテラルデータは共有しません。また、出力先リテラルプールが異なる場合も、データは共有しません。

複数の拡張命令が1つのリテラルデータを共有する例を、以下に示します。

コーディング例

ソースプログラム

```
. SECTION CD1, CODE, LOCATE = H' 0000F000
CD1_START:
MOV. L   #H' FFFF0000, R0
MOV. W   #H' FF00, R1
MOV. L   #H' FFFF0000, R2
MOV      #H' FF, R3
RTS
MOV      R0, R10
.END
```

リテラルプール自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000	2	CD1_START:
3	0000F000 D003	3	MOV. L #H' FFFF0000, R0
4	0000F002 9103	4	MOV. W #H' FF00, R1
5	0000F004 D202	5	MOV. L #H' FFFF0000, R2
6	0000F006 E3FF	6	MOV #H' FF, R3
7	0000F008 000B	7	RTS
8	0000F00A 6A03	8	MOV R0, R10
9			***** BEGIN-POOL *****
10	0000F00C FF00		DATA FOR SOURCE-LINE 4
11	0000F00E 0000		ALIGNMENT CODE
12	0000F010 FFFF0000		DATA FOR SOURCE-LINE 3, 5
13			***** END-POOL *****
14		9	. END

9.6 リテラルプール出力の抑止

プログラム中に無条件分岐が多過ぎると、次のような問題が生じます。

- ・小さなリテラルプールがいくつも出力される。
- ・リテラルを共有できない。

このような場合には、次の方法でリテラルプール出力を抑止してください。

```

~
遅延分岐命令
ディレイスロット命令
.NOPOOL
~

```

コーディング例

ソースプログラム

CASE1:		MOV. L #H' FFFF0000, R0	-----	拡張命令1
		RTS		
		NOP		
		.NOPOOL	-----	このポイントへの リテラルプール出力はありません。
CASE2:		MOV. L #H' FFFF0000, R0	-----	拡張命令2
		RTS		
		NOP	-----	リテラルプールの出力ポイントです。

リテラルプール自動生成結果（ソースリスト）

20	0000F000		20	CASE1:
21	0000F000	D002	21	MOV. L #H' FFFF0000, R0
22	0000F002	000B	22	RTS
23	0000F004	0009	23	NOP
24			24	.NOPOOL
25	0000F006		25	CASE2:
26	0000F006	D001	26	MOV. L #H' FFFF0000, R0
27	0000F008	000B	27	RTS
28	0000F00A	0009	28	NOP
29				***** BEGIN-POOL *****
30	0000F00C	FFFF0000		DATA FOR SOURCE-LINE 21, 26
31				***** END-POOL *****

9.7 リテラルプール自動生成に関する注意事項

(1) 拡張命令を記述してエラーとなる場合

ディレイスロット命令に、拡張命令を記述できません (エラー151)。

境界調整数2未満の相対セクションには、拡張命令を記述できません (エラー152)。

境界調整数4未満の相対セクションには、MOV.L #imm,Rn、MOVA #imm,R0 を記述できません (エラー152)。

(2) .POOL を記述してエラーとなる場合

遅延分岐命令に続いて、.POOL を記述できません (エラー522)。

(3) .NOPOOL を記述してエラーとなる場合

.NOPOOL は、ディレイスロット命令に続いて記述された場合に有効です。それ以外の位置に記述された場合、エラー521 となります。

(4) 拡張命令を展開した結果、他の実行命令のディスプレースメントが範囲外となる場合

アセンブラは、リテラルプールを生成し、ディスプレースメントが範囲外となる命令をエラー402 とします。

対策：.NOPOOL を使用するなどして、リテラルプールの出力ポイントを移動するか、エラーとなる命令の位置またはアドレッシングモードを変更してください。

(5) リテラルプールの出力位置が見つからない場合

拡張命令の位置からみて、次の条件を満足するリテラルプール出力ポイントが見つからない場合

- ・ 同ファイル
- ・ 同セクション
- ・ 前方

アセンブラは、その拡張命令が存在するセクションの最後にリテラルプールとそれを飛び越す BRA 命令 (ディレイスロット命令は NOP) を出力し、ウォーニング 876 を発行します。

(6) 拡張命令のディスプレースメントが範囲外となる場合

リテラルプールを生成したが、拡張命令からのディスプレースメントが範囲外となる場合、その拡張命令はエラー402 となります。

対策：POOL を用いるなどして、範囲内となる場所にリテラルプールが生成されるようにしてください。

(7) サイズ指定モードとサイズ選択モードの相違

Ver.2.0 のリテラルプール機能は、サイズ指定モードのみのサポートでしたが、Ver.3.1 以降ではサイズ選択モードを追加しました。Ver.2.0 で作成したソースプログラムを、Ver.3.1 以降のサイズ選択モードでアセンブルするとオペレーションサイズ指定のないデータ転送命令において、imm 値が H'00000080 ~ H'000000FF (128 ~ 255) の範囲で相違がでます。

サイズ指定モードとサイズ選択モードの出力例を次に示します。

コーディング例

ソースプログラム

```

. SECTION CD1, CODE, LOCATE = H' 0000F000
MOV. L    #H' FF, R0
MOV. W    #H' FF, R1
MOV. B    #H' FF, R2
MOV       #H' FF, R3
RTS
MOV       R0, R10
. END

```

サイズ指定モードのリテラルプール自動生成結果（ソースリスト）

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000 D003	2	MOV. L #H' FF, R0
3	0000F002 9103	3	MOV. W #H' FF, R1
4	0000F004 E2FF	4	MOV. B #H' FF, R2
5	0000F006 E3FF	5	MOV #H' FF, R3
6	0000F008 000B	6	RTS
7	0000F00A 6A03	7	MOV R0, R10
8			***** BEGIN-POOL *****
9	0000F00C 00FF		DATA FOR SOURCE-LINE 3
10	0000F00E 0000		ALIGNMENT CODE
11	0000F010 000000FF		DATA FOR SOURCE-LINE 2
12			***** END-POOL *****
13		8	. END

R3の内容は、H'FFFFFFFFとなります。

サイズ選択モードのリテラルプール自動生成結果（ソースリスト）

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000 D003	2	MOV. L #H' FF, R0
3	0000F002 9103	3	MOV. W #H' FF, R1
4	0000F004 E2FF	4	MOV. B #H' FF, R2
5	0000F006 9301	5	MOV #H' FF, R3
6	0000F008 000B	6	RTS
7	0000F00A 6A03	7	MOV R0, R10
8			***** BEGIN-POOL *****
9	0000F00C 00FF		DATA FOR SOURCE-LINE 3, 5
10	0000F00E 0000		ALIGNMENT CODE
11	0000F010 000000FF		DATA FOR SOURCE-LINE 2
12			***** END-POOL *****
13		8	. END

R3の内容は、H'000000FFとなります。

10. リピートループ命令自動生成機能

10.1 リピートループ命令自動生成機能の概要

SH-DSPにおいて、RSレジスタとREレジスタは、LDRS命令とLDRE命令によって、リピート開始アドレスとリピート終了アドレスを設定します。そのアドレスの設定値は、リピートループ間の命令数により異なります。アドレスを記述するとき、表10.1に示すような考慮をする必要があります。

表 10.1 リピートループ間の命令数とアドレスの設定値

命令数	1 命令	2 命令	3 命令	4 命令以上
RS レジスタ	s_addr0+8	s_addr0+6	s_addr0+4	s_addr
RE レジスタ	s_addr0+4	s_addr0+4	s_addr0+4	e_addr3+4

- ・ s_addr0 : リピート開始アドレスの 1 命令前のアドレス
- ・ s_addr : リピート開始アドレス
- ・ e_addr3 : リピート終了アドレスの 3 命令前のアドレス

リピートループ命令自動生成機能とは、リピートループ間の命令数からアドレス値をRS、REレジスタに転送するPC相対のLDRS、LDRE命令と、リピート回数を設定するSETRC命令を1つの命令から自動的に生成する機能です。

例えば、以下のプログラム (a) は本機能を用いると (b) のように記述できます。

プログラム (a)

```

LDRS s_addr0+6
LDRE s_addr0+4
SETRC #10
s_addr0: NOP
PADD A0,M0,A0 ;リピート開始アドレス
PCMP x1,M0 ;リピート終了アドレス

```

プログラム (b)

```

REPEAT s_addr,e_addr,#10
NOP
s_addr: PADD A0,M0,A0 ;リピート開始アドレス
e_addr: PCMP X1,M0 ;リピート終了アドレス

```

10.2 リピートループ命令自動生成機能に関する拡張命令

拡張命令 (REPEAT s_label,e_label,#imm、REPEAT s_label,e_label,Rn、REPEAT s_label,e_label) の記述に対してアセンブラは必要な命令を自動生成し、PC 相対のディスプレースメント値を計算します。

表 10.2 に、拡張命令のソースステートメントとその展開結果を示します。拡張命令のソースステートメントは、2 つまたは 3 つの実行命令に展開されます。

表 10.2 拡張命令の種類と展開結果

拡張命令	展開結果
REPEAT s_label,e_label,#imm	LDRS@(disp,PC)と LDRE@(disp,PC)と SETRC#imm
REPEAT s_label,e_label,Rn	LDRS@(disp,PC)と LDRE@(disp,PC)と SETRC Rn
REPEAT s_label,e_label	LDRS@(disp,PC)と LDRE@(disp,PC)

10.3 REPEAT の記述方法

書式

[シンボル [:]] REPEAT 開始アドレス,終了アドレス[,リピート回数]

ステートメントの要素

(1) 開始アドレス、終了アドレス

リピートループの開始アドレスと終了アドレスをラベルで記述します。

(2) リピート回数

リピート回数をイミディエイトまたは汎用レジスタで記述します。

解説

- REPEAT は開始アドレスから終了アドレスまでの命令をリピートするための実行命令 (LDRS、LDRE) を自動的に生成します。
- リピート回数を指定した場合、SETRC を生成します。リピート回数を省略した場合、SETRC を生成しません。

10.4 コーディング例

基本例（リピートする命令数が4命令以上のとき）

```

        REPEAT RptStart,RptEnd,#5

        PCLR Y0

        PCLR A0
RptStart: MOVX @R4+,X1  MOVY @R6+,Y1
          PADD A0,Y0,Y0  PMULS X1,Y1,A0
          DCT PCLR A0
          AND  R0,R4
RptEnd:   AND  R0,R6

```

このプログラムは、RptStart から RptEnd までの5命令を5回繰り返し実行します。

展開イメージ

```

        LDRS RptStart
        LDRE RptEnd3+4
        SETRC #5
        PCLR Y0
        PCLR A0
RptStart: MOVX @R4+,X1  MOVY @R6+,Y1
RptEnd3:  PADD A0,Y0,Y0  PMULS X1,Y1,A0 ;ラベルは実際には生成されません
          DCT PCLR A0
          AND  R0,R4
RptEnd:   AND  R0,R6

```

リピートする命令数が1命令のとき

開始アドレスと終了アドレスを同じラベルで指定してください。

```
REPEAT Rpt,Rpt,R0
MOVX @R4+,X1  MOVY @R6,Y1
Rpt:  PADD A0,Y0,Y0  PMULS X1,Y1,A0  MOVX @R4+,X1  MOVY @R6+,Y1
```

展開イメージ

```
LDRS RptStart0+8
LDRE RptStart0+4
SETRC R0
RptStart0: MOVX @R4+,X1  MOVY @R6,Y1  ; ラベルは実際には生成されません
Rpt:  PADD A0,Y0,Y0  PMULS X1,Y1,A0  MOVX @R4+,X1  MOVY @R6+,Y1
```

リピートする命令数が2命令のとき

```
REPEAT RptStart,RptEnd,#10
PCLR Y0
RptStart: MOVX @R4+,X1  MOVY @R6+,Y1
RptEnd:  PADD A0,Y0,Y0  PMULS X1,Y1,A0
```

展開イメージ

```
LDRS RptStart0+6
LDRE RptStart0+4
SETRC #10
RptStart0: PCLR Y0  ; ラベルは実際には生成されません
RptStart: MOVX @R4+,X1  MOVY @R6+,Y1
RptEnd:  PADD A0,Y0,Y0  PMULS X1,Y1,A0
```


リピートする命令数が3命令のとき

```

        REPEAT RptStart,RptEnd,R0
        PCLR Y0
RptStart: MOVX @R4+,X1  MOVY  @R6+,Y1
        PMULS X1,Y1,A0
RptEnd:  PADD A0,Y0,Y0

```

展開イメージ

```

        LDRE RptStart0+4
        LDRS RptStart0+4
        SETRC R0
RptStart0: PCLR Y0      ; ラベルは実際には生成されません
RptStart: MOVX @R4+,X1  MOVY  @R6+,Y1
        PMULS X1,Y1,A0
RptEnd:  PADD A0,Y0,Y0

```

リピート回数を省略したとき

リピート回数を省略すると、アセンブラは SETRC を展開しません。LDRS、LDRE と SETRC を分離したいとき使用します。

```

        REPEAT RptStart,RptEnd
        ; この部分に LDRS, LDRE が展開されます
        MOV #10,R0
OuterLoop:
        SETRC #16
        PCLR Y0
        PCLR A0
RptStart: MOVX @R4+,X1  MOVY  @R6+,Y1
        PADD A0,Y0,Y0  PMULS X1,Y1,A0
        DCT PCLR A0
        AND  R0,R4
RptEnd:  AND  R0,R6
        DT  R0
        BF OuterLoop

```

10.5 REPEAT 拡張命令に関する注意事項

開始アドレス、終了アドレスに関する注意事項

開始アドレスおよび終了アドレスに指定できるラベルは、同じセクション内のラベル、または、同じローカルブロック内のローカルラベルです。

開始アドレスは REPEAT 拡張命令より前方(上位アドレス)になければなりません。

また、終了アドレスは開始アドレスより前方(上位アドレス)になければなりません。

【参照】 ローカルラベル 言語編「1.8 ローカルラベル」

ループ内に記述する命令に関する注意事項

- (1) ループ内にデータまたはデータ領域を確保する制御命令、または、.ORG 制御命令を記述した場合、アセンブラはウォーニングを出力し、1 制御命令を 1 命令としてリピートする命令数をカウントします。.ALIGN 制御命令を記述して、アライメントが生成された場合、アセンブラはウォーニングを出力し、.ALIGN 制御命令を 1 命令としてリピートする命令数をカウントします。

該当する制御命令

.DATA、.DATAB、.SDATA、.SDATAB、.SDATAC、.SDATAZ、.FDATA、
.FDATAB、.XDATA、.RES、.SRES、.SRESC、.SRESZ、.FRES、.ALIGN、.ORG

- (2) アセンブラはループ内ではリテラルプールの自動生成を抑止します。したがって、無条件分岐命令があっても、リテラルプールの出力対象になりません。また、.POOL 制御命令を記述した場合、アセンブラはウォーニングを出力し、.POOL 制御命令を無視します。

ループ直前の命令に関する注意事項

リピートする命令数が 3 命令以下の場合、ループ直前の命令は実行命令または DSP 命令でなければなりません。

したがって、リピートする命令数が 3 命令以下で、開始アドレスの直前が以下の場合、アセンブラはエラーを出力します。

- (1) データまたはデータ領域を確保する制御命令、または、.ORG 制御命令
.DATA、.DATAB、.SDATA、.SDATAB、.SDATAC、.SDATAZ、.FDATA、
.FDATAB、.XDATA、.RES、.SRES、.SRESC、.SRESZ、.FRES、.ORG
- (2) リテラルプール自動生成機能で生成されたリテラルプール

開始アドレスの直前が無条件分岐命令+ディレイスロット命令、または.POOL 制御命令の場合、リテラルプールが自動的に生成される可能性があります。リテラルプールの生成を抑止するためには、ディレイスロット命令の直後に.NOPOOL 制御命令を記述してください。

(3) .ALIGN 制御命令で 1 バイトのアライメントが生成された場合

.ALIGN 制御命令を指定したとき、直前が奇数アドレスである場合、1 バイトのアライメントが生成される場合があります（例えば、ロケーションカウンタが 3 で .ALIGN 4 を指定したときなど）。この場合、実行命令でないデータがループの直前に出力されたことになり、エラーになります。2 バイト以上のアライメントが出力された場合、直前の命令は NOP になり、正常に動作します。

その他の注意事項

- (1) REPEAT 拡張命令～開始アドレスの間には 1 命令以上の実行命令または DSP 命令がなければなりません。1 命令以上の実行命令または DSP 命令が存在しない場合、アセンブラはエラーを出力します。
- (2) REPEAT 拡張命令～終了アドレスの間に別の REPEAT 拡張命令を記述することはできません。REPEAT 拡張命令をネストして記述した場合、アセンブラはエラーを出力し、別の REPEAT を無視します。

操作編

目 次

1.	起動.....	289
2.	コマンドライン・オプション.....	295

1. 起動

1.1 コマンドラインの形式

アセンブラを起動するには、ホストコンピュータがコマンド入力待ち状態のとき、次のようにコマンドラインを入力します。

% asmsh 入力ソースファイル , 入力ソースファイル ... I[コマンドライン・オプション ...]

(1) (2) (3)

- (1) アセンブラの起動コマンドです。
- (2) アセンブルするソースプログラムのファイルです。
複数のソースファイルを、1 度に指定できます。
- (3) アセンブル方法を細かく指示するための、オプション指定です。

【注意】 アセンブラ起動時に複数の入力ソースファイルを指定すると、それらのソースファイルを指定の順に連結したものが、アセンブル処理の単位になります。この場合、.END アセンブラ制御命令は、最後のファイルだけに記述してください。

【補足】 アセンブラは、アSEMBル処理が正常に終了したかどうかの情報を、リターンコードとしてオペレーティングシステムへ返します。リターンコードは、オペレーティングシステムおよび発生したエラーのレベルにより、次のようになります。

オペレーティングシステム	UNIX	Windows®95 および Windows®NT
正常終了	0	0
ウォーニング発生	0	0
エラー発生	1	2
フェイタルエラー（致命的なエラー）発生	1	4

OS へのリターン値は、-ABORT の指定により変更できます。

【参照】 操作編「2.2.6 アセンブラの実行に関するコマンドライン・オプション」
-ABORT

1.2 ファイルの指定形式

アセンブラが扱うファイルは、次の形式で指定します。

主ファイル名 . [ファイル型]

本マニュアルで「ファイル名」という場合、主ファイル名とファイル型を合わせた名前を意味しています。

例

(ファイル名)

file.src 主ファイル名は file、ファイル型は src です。

prog.obj 主ファイル名は prog、ファイル型は obj です。

ファイル型は、内容によってファイルを区別する識別子です。

主ファイル名が同じでもファイル型が異なれば、別のファイルです。

例

file.src

file.obj



それぞれ別のファイルです。

アセンブラが扱うファイルには、次のものがあります。

- ・ソースファイル

ソースプログラムのファイルです。ソースファイルを主ファイル名だけで指定すると、ファイル型は src になります。

- ・オブジェクトファイル

オブジェクトモジュールの出力先となるファイルです。オブジェクトファイルを主ファイル名だけで指定すると、ファイル型は obj になります。オブジェクトファイルの指定を省略すると、主ファイル名は入力ソースファイル (1 つめのファイル) と同じ、ファイル型は obj になります。

- ・リストファイル

アセンブルリストの出力先となるファイルです。リストファイルを主ファイル名だけで指定すると、ファイル型は lis になります。リストファイルの指定を省略すると、主ファイル名は入力ソースファイル (1 つめのファイル) と同じ、ファイル型は lis になります。

1.3 SHCPU 環境変数

アセンブラは、SHCPU 環境変数に設定した CPU 用にアセンブルします。
CPU 種別の設定方法は、以下のとおりです。

(1) UNIX の場合

(a) C Shell の場合

```
setenv SHCPU CPU 種別 *
```

(b) Bourne/Korn Shell の場合

```
SHCPU=CPU 種別 *
```

```
export SHCPU *
```

(2) Windows[®]95 および Windows[®]NT の場合

```
SET SHCPU=CPU 種別 *
```

【注】* 本環境変数は、必ず大文字で指定してください。

指定できる CPU 種別は SH1、SH2、SH2E、SH3、SH3E、SH4、SHDSP、SH3DSP です。

CPU 種別の優先順位は、- CPU、.CPU アセンブラ制御命令、SHCPU 環境変数の順となります。

2. コマンドライン・ オプション

2.1 コマンドライン・オプションの概要

コマンドライン・オプションは、アセンブル方法を細かく指示するための指定です。

表 2.1 に、コマンドライン・オプションの一覧を示します。

表 2.1 コマンドライン・オプション一覧

項番	分 類	コマンドライン・オプション	機 能
2.2.1	CPU に関するもの	-CPU	CPU 種別を指定します。
2.2.2	オブジェクトモジュールに関するもの	-[NO]OBJECT	オブジェクトモジュールの出力を制御します。
		-[NO]DEBUG	デバッグ情報の出力を制御します。
		-ENDIAN	エンディアン種別を指定します。
2.2.3	アセンブルリストに関するもの	-[NO]LIST	アセンブルリストの出力を制御します。
		-[NO]SOURCE	ソースプログラムリスト出力を制御します。
		-[NO]CROSS_REFERENCE	クロスリファレンスリスト出力を制御します。
		-[NO]SECTION	セクション情報リストの出力を制御します。
		-[NO]SHOW	ソースプログラムリストの部分出力を制御します。
		-LINES	アセンブルリストの行数を設定します。
		-COLUMNS	アセンブルリストの桁数を設定します。
2.2.4	ファイルインクルード機能に関するもの	-INCLUDE	インクルードファイルの取り込み先を指定します。
2.2.5	条件つきアセンブリ機能に関するもの	-ASSIGNA	整数型のプリプロセッサ変数を定義します。
		-ASSIGNC	文字型のプリプロセッサ変数を定義します。
		-DEFINE	文字列の置き換えを定義します。
2.2.6	アセンブラの実行に関するもの	-EXPAND	プリプロセッサの展開結果を出力します。
		-ABORT	アセンブラが異常終了するエラーのレベルを変更します。
2.2.7	ソースファイル内の日本語記述に関するもの	-SJIS	ソースファイル内の漢字コードをシフト JIS コードとして扱います。
		-EUC	ソースファイル内の漢字コードを EUC コードとして扱います。
		-OUTCODE	オブジェクトコードへ出力する漢字コードを指定します。
2.2.8	リテラルプール自動生成機能に関するもの	-AUTO_LITERAL	リテラルプール自動生成機能のサイズモードを設定します。
2.2.9	コマンドラインの指定方法に関するもの	-SUBCOMMAND	コマンドラインをファイルから入力します。

2. コマンドライン・オプション

項番	分類	コマンドライン・オプション	機能
2.2.10	浮動小数点データに関するもの	-ROUND	浮動小数点データの丸め方法を指定します。
		-DENORMALIZE	浮動小数点データの非正規化数の扱いを指定します。

【補足】 アセンブルリストはアセンブル結果を出力するリストであり、ソースプログラム・リスト、クロスリファレンス・リスト、セクション情報リストを含みます。

【参照】 アセンブルリストについての詳細 「付録C. アセンブルリスト出力例」

2.2 コマンドライン・オプション リファレンス

2.2.1 CPU に関するコマンドライン・オプション

CPU に関するコマンドライン・オプションには、次のものがあります。

-CPU	CPU 種別を指定します。
------	---------------

-CPU CPU の種別を指定する

書式

-CPU = CPU 種別

解説

(1) **-CPU** は、アセンブルするソースプログラムの対象とする CPU 種別を指定するコマンドライン・オプションです。

(2) CPU 種別の内容は次のとおりです。

SH1	SH-1 用
SH2	SH-2 用
SH2E	SH-2E 用
SH3	SH-3 用
SH3E	SH-3E 用
SH4	SH-4 用
SHDSP	SH-DSP 用
SH3DSP	SH3-DSP 用

アセンブラ制御命令との関係

コマンドライン・オプション	アセンブラ制御命令	SHCPU 環境変数	結 果
-CPU	(指定に関わらず)	(指定に関わらず)	-CPU の CPU 種別
(指定なし)	.CPU CPU 種別	(指定に関わらず)	.CPU の CPU 種別
	(指定なし)	SHCPU=CPU 種別	SHCPU 環境変数の CPU 種別
		(指定なし)	SH1

【参照】 CPU 種別

言語編「5.2.1 CPU に関するアセンブラ制御命令」 **.CPU**

SHCPU 環境変数

操作編「1.3 **SHCPU** 環境変数」

2.2.2 オブジェクトモジュールに関するコマンドライン・オプション

オブジェクトモジュールに関するコマンドライン・オプションには、次のものがあります。

-OBJECT -NOOBJECT	オブジェクトモジュールの出力を制御します。
-DEBUG -NODEBUG	デバッグ情報の出力を制御します。
-ENDIAN	エンディアン種別を指定します。

-OBJECT , -NOBJECT オブジェクトモジュールの出力を制御する**書式****-OBJECT** [= 出力オブジェクトファイル]**-NOBJECT** (下線の部分は省略形)**解説**

(1) -OBJECT は、オブジェクトモジュールを出力する指定です。

-NOBJECT は、オブジェクトモジュールを出力しない指定です。

(2) 出力オブジェクトファイルは、オブジェクトモジュールの出力先です。

(3) 出力オブジェクトファイルの指定を省略すると、次のようになります。

- ・ファイル型の指定を省略した場合

ファイル型は obj になります。

- ・主ファイル名、ファイル型とも指定を省略した場合

主ファイル名は入力ソースファイル (1 つめに指定したもの) と同じ、ファイル型は obj になります。

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果
-OBJECT	(指定に関わらず)	オブジェクトモジュールを出力する。
-NOBJECT	(指定に関わらず)	オブジェクトモジュールを出力しない。
(指定なし)	.OUTPUT OBJ	オブジェクトモジュールを出力する。
	.OUTPUT NOOBJ	オブジェクトモジュールを出力しない。
	(指定なし)	オブジェクトモジュールを出力する。

【注意】 入力ソースファイルと出力オブジェクトファイルに、同じファイルを指定しないでください。同じファイルを指定した場合、入力ソースファイルが上書きされます。

-DEBUG, -NODEBUG デバッグ情報の出力を制御する**書式****-DEBUG****-NODEBUG** (下線の部分は省略形)**解説**

(1) -DEBUG は、デバッグ情報を出力する指定です。

-NODEBUG は、デバッグ情報を出力しない指定です。

(2) -DEBUG、-NODEBUG による指定は、オブジェクトモジュールを出力する場合に限り有効です。

(3) デバッグ情報を出力する場合、オブジェクトファイルを出力するディレクトリに「dwfinf」という名前のディレクトリを自動生成し、オブジェクトファイル名と同じ主ファイル名で拡張子「dwi」のデバッグ付加情報 (ELF/DWARF 付加情報) ファイルを出力します。

【参照】 オブジェクトモジュールの出力言語編 「5.2.6 オブジェクトモジュールに関するアセンブラ制御命令」
.OUTPUT

操作編 「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」 - OBJECT - NOOBJECT

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果 (オブジェクトモジュール出力時)
-DEBUG	(指定に関わらず)	デバッグ情報を出力する。
-NODEBUG	(指定に関わらず)	デバッグ情報を出力しない。
(指定なし)	.OUTPUT DBG	デバッグ情報を出力する。
	.OUTPUT NODBG	デバッグ情報を出力しない。
	(指定なし)	デバッグ情報を出力しない。

【補足】 デバッグ情報は、デバッガでプログラムをデバッグするのに必要です。内容として、ソースプログラムの行に関する情報や、シンボルに関する情報 (シンボルデバッグ情報) などを含みます。

-ENDIAN エンディアン種別を指定する

書式

-ENDIAN =エンディアン種別

エンディアン種別 { BIG | LITTLE }

(下線の部分は省略形)

解説

(1) -ENDIAN は、ターゲットのマイコンのバイトの並び方が、BIG エンディアンか LITTLE エンディアンかを決定するコマンドライン・オプションです。

(2) 省略時は BIG エンディアンになります。

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果
-ENDIAN=BIG	(指定に関わらず)	BIG エンディアンでアセンブルする。
-ENDIAN=LITTLE	(指定に関わらず)	LITTLE エンディアンでアセンブルする。
(指定なし)	.ENDIAN BIG	BIG エンディアンでアセンブルする。
	.ENDIAN LITTLE	LITTLE エンディアンでアセンブルする。
	(指定なし)	BIG エンディアンでアセンブルする。

【参照】 .ENDIAN

言語編 「5.2.6 オブジェクトモジュールに関するアセンブラ制御命令」

.ENDIAN

2.2.3 アセンブルリストに関するコマンドライン・オプション

アセンブルリストに関するコマンドライン・オプションには、次のものがあります。

- LIST - NOLIST	アセンブルリストの出力を制御します。
- SOURCE - NOSOURCE	ソースプログラム・リストの出力を制御します。
- CROSS__REFERENCE - NOCROSS__REFERENCE	クロスリファレンス・リストの出力を制御します。
- SECTION - NOSECTION	セクション情報リストの出力を制御します。
- SHOW - NOSHOW	ソースプログラム・リストの部分出力を制御します。
- LINES	アセンブルリストの、1 ページあたりの行数を設定します。
- COLUMNS	アセンブルリストの、1 行あたりの桁数を設定します。

-LIST, -NOLIST アセンブルリストの出力を制御する

書式

-LIST [= 出力リストファイル]

-NOLIST (下線部分は省略形)

解説

(1) -LIST は、アセンブルリストを出力する指定です。

-NOLIST は、アセンブルリストを出力しない指定です。

(2) 出力リストファイルは、アセンブルリストの出力先です。

(3) 出力リストファイルの指定を省略すると、次のようになります。

- ・ファイル型の指定を省略した場合

ファイル型は lis になります。

- ・主ファイル名、ファイル型とも指定を省略した場合

主ファイル名は入力ソースファイル (1 つめに指定したもの) と同じ、ファイル型は lis になります。

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果
-LIST	(指定に関わらず)	アセンブルリストを出力する。
-NOLIST	(指定に関わらず)	アセンブルリストを出力しない。
(指定なし)	.PRINT LIST	アセンブルリストを出力する。
	.PRINT NOLIST	アセンブルリストを出力しない。
	(指定なし)	アセンブルリストを出力しない。

【注意】 入力ソースファイルと出力リストファイルに、同じファイルを指定しないでください。同じファイルを指定した場合、入力ソースファイルが上書きされます。

-SOURCE, -NOSOURCE ソースプログラム・リストの出力を制御する

書式

-SOURCE-NOSOURCE

(下線部分は省略形)

解説

(1) -SOURCE は、アセンブルリストにソースプログラム・リストを含める指定です。

-NOSOURCE は、アセンブルリストにソースプログラム・リストを含めない指定です。

(2) -SOURCE、-NOSOURCE による指定は、アセンブルリストを出力する場合に限り有効です。

【参照】 アセンブルリストの出力

言語編 「5.2.7 アセンブルリストに関するアセンブラ制御命令」 .PRINT

操作編 「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- LIST - NOLIST

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果 (アセンブルリスト出力時)
-SOURCE	(指定に関わらず)	ソースプログラム・リストを出力する。
-NOSOURCE	(指定に関わらず)	ソースプログラム・リストを出力しない。
(指定なし)	.PRINT SRC	ソースプログラム・リストを出力する。
	.PRINT NOSRC	ソースプログラム・リストを出力しない。
	(指定なし)	ソースプログラム・リストを出力する。

-CROSS__REFERENCE, -NOCROSS__REFERENCE**クロスリファレンス・リストの出力を制御する****書式****-CROSS__REFERENCE****-NOCROSS__REFERENCE** (下線部分は省略形)**解説**

(1) -CROSS__REFERENCE は、アセンブルリストにクロスリファレンス・リストを含める指定です。

-NOCROSS__REFERENCE は、アセンブルリストにクロスリファレンス・リストを含めない指定です。

(2) -CROSS__REFERENCE、-NOCROSS__REFERENCE による指定は、アセンブルリストを出力する場合に限り有効です。

【参照】 アセンブルリストの出力

言語編 「5.2.7 アセンブルリストに関するアセンブラ制御命令」 .PRINT

操作編 「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- LIST - NOLIST

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果 (アセンブルリスト出力時)
-CROSS__REFERENCE	(指定に関わらず)	クロスリファレンス・リストを出力する。
-NOCROSS__REFERENCE	(指定に関わらず)	クロスリファレンス・リストを出力しない。
(指定なし)	.PRINT CREF	クロスリファレンス・リストを出力する。
	.PRINT NOCREF	クロスリファレンス・リストを出力しない。
	(指定なし)	クロスリファレンス・リストを出力する。

-SECTION, -NOSECTION セクション情報リストの出力を制御する**書式****-SECTION****-NOSECTION** (下線部分は省略形)**解説**

(1) -SECTION は、アセンブルリストにセクション情報リストを含める指定です。

-NOSECTION は、アセンブルリストにセクション情報リストを含めない指定です。

(2) -SECTION、-NOSECTION による指定は、アセンブルリストを出力する場合に限り有効です。

【参照】 アセンブルリストの出力

言語編 「5.2.7 アセンブルリストに関するアセンブラ制御命令」 .PRINT

操作編 「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- LIST - NOLIST

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果 (アセンブルリスト出力時)
-SECTION	(指定に関わらず)	セクション情報リストを出力する。
-NOSECTION	(指定に関わらず)	セクション情報リストを出力しない。
(指定なし)	.PRINT SCT	セクション情報リストを出力する。
	.PRINT NOSCT	セクション情報リストを出力しない。
	(指定なし)	セクション情報リストを出力する。

-SHOW, -NOSHOW ソースプログラム・リストの部分出力を制御する

書式

-SHOW [= 出力種別 [, 出力種別 ...]]**-NOSHOW** [= 出力種別 [, 出力種別 ...]]出力種別 : {CONDITIONALS | DEFINITIONS | CALLS | EXPANSIONS | CODE}

(下線部分は省略形)

【注】 オペレーティングシステムが Windows95[®]および Windows[®]NT の場合、出力種別を 2 つ以上指定する時はカッコ () で囲んで指定してください。

解説

(1) ソースプログラムリストのプリプロセッサ機能のソースステートメントの部分出力、出力抑止、オブジェクトコード表示行の部分出力、出力抑止を指定します。

(2) 出力種別で指定した項目を出力、出力抑止します。出力種別を省略した場合は、全ての項目を出力、出力抑止します。

-SHOW 出力

-NOSHOW 出力抑止

(3) 出力種別の内容は次のとおりです。

出力種別	意 味	内 容
CONDITIONALS	条件つき不成立	.AIF, .AIFDEF の不成立部分
DEFINITIONS	定義	マクロ定義部分 .AREPEAT, .AWHILE 定義部分 .INCLUDE 制御文 .ASSIGNA, .ASSIGNC 制御文
CALLS	コール	マクロコール文 .AIF, .AIFDEF, .AENDI 制御文
EXPANSIONS	展開	マクロ展開部分 .AREPEAT, .AWHILE 展開部分
CODE	オブジェクト コード表示行	制御命令のオブジェクトコード表示が、ソースステートメントの行数を超える部分

【参照】 ソースプログラム・リストの出力

言語編 「5.2.7 アセンブルリストに関するアセンブラ制御命令」 .PRINT

操作編 「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- LIST - NOLIST - SOURCE - NOSOURCE

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果
-SHOW=出力種別	(指定に関わらず)	出 力
-NOSHOW=出力種別	(指定に関わらず)	出力抑止
(指定なし)	.LIST 出力種別 (出力)	出 力
	.LIST 出力種別 (出力抑止)	出力抑止
	(指定なし)	出 力

-LINES アセンブルリストの行数を設定する

書式

-LINES = 行数 (下線部分は省略形)

解説

(1) -LINES は、アセンブルリストの1ページあたりの行数を設定するコマンドライン・オプションです。行数として有効な値は、20～255です。

(2) -LINES による指定は、アセンブルリストを出力する場合に限り有効です。

【参照】 アセンブルリストの出力

言語編 「5.2.7 アセンブルリストに関するアセンブラ制御命令」
.PRINT
操作編 「2.2.3 アセンブルリストに関するコマンドライン・オプション」
- LIST - NOLIST

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドライン・オプション	アセンブラ制御命令	結 果
-LINES=行数	(指定に関わらず)	1ページあたり、-LINES で指定した行数になる。
(指定なし)	.FORM LIN=行数	1ページあたり、.FORM で指定した行数になる。
	(指定なし)	1ページあたり、60行になる。

-COLUMNS アセンブルリストの桁数を設定する**書式**-COLUMNS = 桁数

(下線部分は省略形)

解説

(1) -COLUMNS は、アセンブルリストの 1 行あたりの桁数を設定するコマンドライン・オプションです。桁数として有効な値は、79 ~ 255 です。

(2) -COLUMNS による指定は、アセンブルリストを出力する場合に限り有効です。

【参照】 アセンブルリストの出力

言語編 「5.2.7 アセンブルリストに関するアセンブラ制御命令」 .PRINT

操作編 「2.2.3 アセンブルリストに関するコマンドライン・オプション」

- LIST - NOLIST

アセンブラ制御命令との関係

アセンブラは、コマンドライン・オプションによる指定を優先します。

コマンドラインオプション	アセンブラ制御命令	結 果
-COLUMNS=桁数	(指定に関わらず)	1 行あたり、-COLUMNS で指定した桁数になる
(指定なし)	.FORM COL=桁数	1 行あたり、.FORM で指定した桁数になる。
	(指定なし)	1 行あたり、132 桁になる。

2.2.4 ファイルインクルード機能に関するコマンドライン・オプション

ファイルインクルード機能に関するコマンドライン・オプションには、次のものがあります。

-INCLUDE	インクルードファイルの取り込み先を指定します。
----------	-------------------------

-INCLUDE インクルードファイルの取り込み先を指定する

書式

`-INCLUDE = ディレクトリ名 [, ディレクトリ名…]`

(下線部分は省略形)

解説

- (1) `-INCLUDE` は、インクルードするファイルのディレクトリ名を指定するコマンドライン・オプションです。
- (2) ディレクトリ名はホストマシンの標準的な指定方法に従います。
- (3) ディレクトリ名の指定数は、コマンドラインで1行入力可能な限り有効です。
- (4) 検索の優先度は、まずカレントディレクトリ、続いて`-INCLUDE`で指定したディレクトリを指定した順序にしたがって検索します。

アセンブラ制御文との関係

コマンドライン・オプション	アセンブラ制御文	結 果
<code>-INCLUDE</code>	(指定に関わらず)	(1) <code>.INCLUDE</code> のディレクトリ
		(2) <code>-INCLUDE</code> 指定のディレクトリ*
(指定なし)	<code>.INCLUDE</code> ファイル名	<code>.INCLUDE</code> のディレクトリ

【注】 * `.INCLUDE` で指定したディレクトリの前に`-INCLUDE`で指定したディレクトリを付加する。

【注意】

`asmsh aaa.mar -include=/usr/tmp,/tmp` (UNIX の場合)

(`aaa.mar` 内で `.INCLUDE "file.h"` 指定の場合)

`file.h` をカレントディレクトリ、`/usr/tmp`、`/tmp` の順にサーチします。

【参照】 `.INCLUDE`

言語編「6 ファイルインクルード機能」 `.INCLUDE`

2.2.5 条件つきアセンブリ機能に関するコマンドライン・オプション

条件つきアセンブリ機能に関するコマンドライン・オプションには、次のものがあります。

-ASSIGNA	整数型のプリプロセッサ変数を定義します。
-ASSIGNC	文字型のプリプロセッサ変数を定義します。
-DEFINE	文字列の置き換えを定義します。

-ASSIGNA 整数型のプリプロセッサ変数を定義する**書式**

-ASSIGNA= プリプロセッサ変数名=整数定数 [,プリプロセッサ変数名=整数定数...]
(下線部分は省略形)

解説

- (1) -ASSIGNA は、プリプロセッサ変数に整数定数を設定するコマンドライン・オプションです。
- (2) プリプロセッサ変数名の書き方は、シンボル名の書き方と同じです。
- (3) 整数定数は、基数 (B'、Q'、D'、H') と数値を組み合わせで指定します。基数を省略し、数値のみを限定した場合は 10 進数として扱います。
- (4) 整数定数に指定できる値の範囲は、-2,147,483,648 ~ 4,294,967,295 です。ただし、負の値を設定する場合は、10 進以外の基数で指定してください。

アセンブラ制御文との関係

コマンドライン・オプション	アセンブラ制御文	結 果
-ASSIGNA	.ASSIGNA*	-ASSIGNA 指定の値
	(指定なし)	-ASSIGNA 指定の値
(指定なし)	.ASSIGNA	.ASSIGNA 指定の値

【注】 * -ASSIGNA でプリプロセッサ変数に値を設定した場合、当該プリプロセッサ変数への .ASSIGNA による定義がすべて無効になります。

【注意】 ホスト OS が UNIX の場合は、基数表示のアポストロフィ (') の直前に円記号 (¥) を指定します。また、プリプロセッサ変数名の中にドル記号 (\$) がある場合、ドル記号 (\$) の直前に円記号 (¥) を指定します。

```
asmsh aaa.mar -assigna=_¥$=H¥'FF
```

プリプロセッサ変数_\$に値 H'FF を設定します。ソースプログラム内のプリプロセッサ変数_\$のすべての参照箇所¥&_\$を H'FF に設定します。

【参照】 .ASSIGNA

言語編「7.2 条件つきアセンブリ機能に関する制御文」.ASSIGNA

-ASSIGNC 文字型のプリプロセッサ変数を定義する**書式**

-ASSIGNC= プリプロセッサ変数名="文字列" [,プリプロセッサ変数名="文字列"...]
(下線部分は省略形)

解説

- (1) -ASSIGNC は、プリプロセッサ変数に文字列を設定するコマンドライン・オプションです。
- (2) プリプロセッサ変数名の書き方は、シンボル名の書き方と同じです。
- (3) 文字列は文字をダブルコーテーション (") で囲んで指定します。
- (4) 文字列には、255 文字まで指定できます。

アセンブラ制御文との関係

コマンドライン・オプション	アセンブラ制御文	結 果
-ASSIGNC	.ASSIGNC*	-ASSIGNC 指定の文字列
	(指定なし)	-ASSIGNC 指定の文字列
(指定なし)	.ASSIGNC	.ASSIGNC 指定の文字列

【注】 * -ASSIGNC でプリプロセッサ変数に文字列を設定した場合、当該プリプロセッサ変数への .ASSIGNC による定義がすべて無効になります。

【注意】 ホスト OS が UNIX の場合は、文字列の中に次の文字を指定する際、直前に円記号 (¥) を指定します。また、前後に文字列を指定する場合は、前後の文字列をダブルコーテーション (") で囲みます。

- ・イクスクラメーション (!)
- ・ダブルコーテーション (")
- ・ドル (\$)
- ・逆コーテーション (`)

asmsh aaa.mar -assignc=_¥\$="ON"¥!"OFF" (UNIX の場合)

プリプロセッサ変数_\$に文字列 ON!OFF を設定します。ソースプログラム内のプリプロセッサ変数_\$のすべての参照箇所¥&_\$を文字列 ON!OFF に設定します。

【参照】 .ASSIGNC

言語編「7.2 条件つきアセンブリ機能に関する制御文」 .ASSIGNC

-DEFINE 文字列の置き換えを定義する

書式

-DEFINE= 置換シンボル="文字列" [,置換シンボル="文字列"...]
(下線部分は省略形)

解説

- (1) -DEFINE は、プリプロセッサで置換シンボルを対応する文字列に置き換えることを定義するコマンドライン・オプションです。
- (2) -DEFINE と-ASSIGNC の機能の違いは、.DEFINE と.ASSIGNC の機能の違いに対応します。

アセンブラ制御文との関係

コマンドライン・オプション	アセンブラ制御文	結 果
-DEFINE	.DEFINE*	-DEFINE 指定の文字列
	(指定なし)	-DEFINE 指定の文字列
(指定なし)	.DEFINE	.DEFINE 指定の文字列

【注】 * -DEFINE で置換シンボルに文字列を設定した場合、当該置換シンボルへの.DEFINE による定義がすべて無効になります。

【参照】 .DEFINE

言語編「7.2 条件つきアセンブリ機能に関する制御文」.DEFINE

2.2.6 アセンブラの実行に関するコマンドライン・オプション

アセンブラの実行に関するコマンドライン・オプションには、次のものがあります。

-EXPAND	プリプロセッサの展開結果を出力します。
-ABORT	アセンブラが異常終了するエラーのレベルを変更します。

-EXPAND プリプロセッサの展開結果を出力する

書式

-EXPAND [=出力ファイル名] (下線部分は省略形)

解説

- (1) -EXPAND は、マクロ展開、条件つきアセンブル、ファイルのインクルードを行った後のアセンブラソースファイルを出力するコマンドライン・オプションです。
- (2) 本オプションを指定すると、オブジェクトの生成は行いません。
- (3) 出力ファイルの指定を省略すると、次のようになります。
 - ・ファイル型（拡張子）の指定を省略した場合
ファイル型は exp になります。
 - ・主ファイル名、ファイル型（拡張子）とも指定を省略した場合
主ファイル名は入力ソースファイル（1 つめに指定したもの）と同じ、ファイル型は exp になります。
- (4) 入力ファイルと出力ファイルに同じファイルを指定しないでください。

-ABORT アセンブラが異常終了するエラーのレベルを変更する

書式

-ABORT= エラーレベルエラーレベル：{WARNING | ERROR} (下線部分は省略形)

解説

(1) -ABORT は、エラーレベルを変更するコマンドライン・オプションです。

(2) OS へのリターン値は次のとおりです。

発生個数			オプション指定時の OS へのリターン値			
ウォーニング	エラー	致命的エラー	ABORT=WARNING		ABORT=ERROR*	
			Windows [®] 95 および Windows [®] NT	UNIX	Windows [®] 95 および Windows [®] NT	UNIX
0	0	0	0	0	0	0
1 以上	0	0	2	1	0	0
-	1 以上	0	2	1	2	1
-	-	1 以上	4	1	4	1

【注】 * 下線部は、指定を省略した場合の設定です。

(3) OS へのリターン値が 1 以上の場合は、オブジェクトモジュールの出力を抑止します。

(4) -ABORT による指定は、オブジェクトモジュールを出力する場合に限り有効です。

2.2.7 ソースファイル内の日本語記述に関するコマンドライン・オプション

ソースファイル内の日本語記述に関するコマンドライン・オプションには、次のものがあります。

-SJIS	ソースファイル内の漢字コードをシフト JIS コードとして扱います。
-EUC	ソースファイル内の漢字コードを EUC コードとして扱います。
-OUTCODE	オブジェクトファイルへ出力する漢字コードを指定します。

-SJIS ソースファイル内の漢字コードをシフト JIS コードとして解釈する

書式

-SJIS

解説

(1) -SJIS は、文字列、コメント内の日本語記述を可能とするコマンドライン・オプションです。

- ・ SJIS : 文字列、コメント内の日本語は、シフト JIS コードとして解釈します。
- ・ 指定省略 : 文字列、コメント内の日本語は、ホストマシンに依存する日本語コードとして解釈します。

(2) -SJIS の指定は、-EUC と一緒に指定しないでください。

【参照】 シフト JIS コード 言語編「1.4.2 文字定数」

-EUC ソースファイル内の漢字コードを EUC コードとして解釈する

書式

-EUC

解説

(1) -EUC は、文字列、コメント内の日本語記述を可能とするコマンドライン・オプションです。

- ・EUC : 文字列、コメント内の日本語は、EUC コードとして解釈します。
- ・指定省略 : 文字列、コメント内の日本語は、ホストマシンに依存する日本語コードとして解釈します。

(2) -EUC の指定は、-SJIS と一緒に指定しないでください。

【参照】 EUC コード 言語編「1.4.2 文字定数」

-OUTCODE オブジェクトファイルへ出力する漢字コードを設定する

書式

-OUTCODE =漢字コード

漢字コード：{ SJIS | EUC } (下線部分は省略形)

解説

- (1) **-OUTCODE** は、ソースファイル内の日本語記述を指定した漢字コードに変換し、オブジェクトファイルへ出力するコマンドライン・オプションです。
- (2) **-OUTCODE** とソースファイル内の漢字コード (**-SJIS**、**-EUC**) の指定によるオブジェクトファイルへ出力する漢字コードは次のとおりです。

-OUTCODE の漢字コード	ソースファイル内の漢字コード		
	-SJIS	-EUC	指定なし
SJIS	シフト JIS コード	シフト JIS コード	シフト JIS コード
EUC	EUC コード	EUC コード	EUC コード
指定なし	シフト JIS コード	EUC コード	デフォルト漢字コード

デフォルトの漢字コードは、次のとおりです。

SPARC ステーション	EUC コード
HP9000/700 シリーズ	シフト JIS コード
PC-9800 シリーズ	シフト JIS コード
IBM PC およびその互換機	

【参照】 ソースファイル内の漢字コード

操作編 「2.2.7 ソースファイル内の日本語記述に関するコマンドライン・オプション」 **-SJIS**

操作編 「2.2.7 ソースファイル内の日本語記述に関するコマンドライン・オプション」 **-EUC**

2.2.8 リテラルプール自動生成機能に関するコマンドライン・オプション

リテラルプール自動生成機能に関するコマンドライン・オプションには、次のものがあります。

-AUTO_LITERAL	リテラルプール自動生成機能のサイズモードを設定します。
---------------	-----------------------------

-AUTO_LITERAL リテラルプール自動生成機能のサイズモードを設定する

書式

-AUTO_LITERAL

(下線部分は省略形)

解説

(1) -AUTO_LITERAL は、リテラルプール自動生成機能のサイズモードを指定するコマンドライン・オプションです。

- ・本コマンドライン・オプションを指定した場合、リテラルプール自動生成機能はサイズ選択モードとなり、オペレーションサイズ指定のないデータ転送命令 (MOV #imm,Rn) は、アセンブラが imm の値の範囲を判定し、必要ならばリテラルプールを自動生成します。
- ・本コマンドライン・オプションを指定しない場合、リテラルプール自動生成機能はサイズ指定モードとなり、オペレーションサイズ指定がないデータ転送命令は、1 バイトのデータ転送命令とし、アセンブルします。

(2) サイズ選択モードでは、オペレーションサイズ指定のないデータ転送命令は、符号付きの範囲で判定するため、H'00000080 ~ H'000000FF (128 ~ 255) は、ワードサイズとして扱います。

imm の値の範囲	選択されるサイズまたはエラー	
	サイズ選択モード	サイズ指定モード
H'80000000 ~ H'FFFF7FFF (-2,147,483,648 ~ -32,769)	ロングワード	ウォーニング 835
H'FFFF8000 ~ H'FFFFFF7F (-32,768 ~ -129)	ワード	ウォーニング 835
H'FFFFFF80 ~ H'0000007F (-128 ~ 127)	バイト	バイト
H'00000080 ~ H'000000FF (128 ~ 255)	ワード	バイト
H'00000100 ~ H'00007FFF (256 ~ 32,767)	ワード	ウォーニング 835
H'00008000 ~ H'7FFFFFFF (32,768 ~ 2,147,483,647)	ロングワード	ウォーニング 835

【注】 () 内は 10 進表示

【参照】 サイズ選択モード、サイズ指定モード

言語編「9.3 リテラルプール自動生成機能のサイズモード」

2.2.9 コマンドラインの指定方法に関するコマンドライン・オプション

コマンドラインの指定方法に関するコマンドライン・オプションには、次のものがあります。

-SUBCOMMAND	コマンドラインをファイルから入力します。
-------------	----------------------

-SUBCOMMAND コマンドラインをファイルから入力する

書式

-SUBCOMMAND= サブコマンドファイル名

(下線部分は省略形)

解説

- (1) -SUBCOMMAND は、コマンドラインをファイルから入力するコマンドライン・オプションです。
- (2) 通常のコマンドライン指定と同じ順番で、入力ファイル名とコマンドライン・オプションを指定してください。
- (3) 1 行に 1 つの入力ファイル名またはコマンドライン・オプションを指定してください。
- (4) -SUBCOMMAND は、コマンドラインの末尾に指定してください。残りのファイルをファイルから読み込みます。
- (5) -SUBCOMMAND は、サブコマンドファイル内に指定しないでください。

コーディング例

```
asmsh aaa.src -subcommand=aaa.sub
```

サブコマンドファイルの内容をコマンドラインに展開し、アセンブルします。

aaa.sub の内容

```
bbb.src  
-list  
-noobj
```

展開例は、次のとおりです。

```
asmsh aaa.src,bbb.src -list -noobj
```

【注意】

- (1) サブコマンドファイルの 1 行は最大 300 バイトです。
- (2) サブコマンドファイル全体のサイズは最大 32,767 バイトです。

2.2.10 浮動小数点データに関するコマンドライン・オプション

浮動小数点データに関するコマンドライン・オプションには、次のものがあります。

-ROUND	浮動小数点データの丸め方法を指定します。
-DENORMALIZE	浮動小数点データの非正規化数の扱いを指定します。

-ROUND 浮動小数点データの丸め方法を指定する

書式

-ROUND= 丸め種別丸め種別：{ NEAREST | ZERO } (下線部分は省略形)

解説

(1) **-ROUND** は、浮動小数点データ制御命令に記述した定数を、オブジェクトコードに変換する際、丸め方式を指定するコマンドライン・オプションです。

(2) 丸め方式は、以下の2種類があります。

- ・ round to NEAREST even (NEAREST)
- ・ round to ZERO (ZERO)

(3) 本コマンドライン・オプションを指定しなかった場合、CPU 種別により、丸め方法は次のようになります。

CPU 種別	丸め方法
SH-1	round to NEAREST even
SH-2	round to NEAREST even
SH-2E	round to ZERO
SH-3	round to NEAREST even
SH-3E	round to ZERO
SH-4	round to NEAREST even
SH-DSP	round to NEAREST even
SH3-DSP	round to NEAREST even

【注意】 CPU 種別が SH-2E または SH-3E のとき、丸め方法に round to NEAREST even を選択した場合、ソースプログラムで最初に出現した浮動小数点データ制御命令に対してウォーニング 818 とし、指定された round to NEAREST even でオブジェクトコードを出力します。

【参照】 丸め方法 言語編「1.4.3 浮動小数点定数」

-DENORMALIZE 浮動小数点データの非正規化数の扱いを指定する**書式**

-DENORMALIZE= 指定種別 (下線部分は省略形)

指定種別: { ON | OFF }

解説

(1) -DENORMALIZE は浮動小数点データ制御命令に非正規化数を記述したとき、有効な値とするか、無効な値とするかを指定するコマンドライン・オプションです。

(2) 非正規化数を有効な値 (ON) とした場合と、無効な値 (OFF) とした場合はオブジェクトコードが異なります。

- ・有効とした場合：ウォーニング 842 とし、オブジェクトコードを出力します。
- ・無効とした場合：ウォーニング 841 とし、0 のオブジェクトコードを出力します。

(3) 本コマンドライン・オプションを指定しなかった場合、CPU 種別により、非正規化数の扱いは次のようになります。

CPU 種別	丸め方法
SH-1	有効な値 (ON) とする
SH-2	有効な値 (ON) とする
SH-2E	無効な値 (OFF) とする
SH-3	有効な値 (ON) とする
SH-3E	無効な値 (OFF) とする
SH-4	有効な値 (ON) とする
SH-DSP	有効な値 (ON) とする
SH3-DSP	有効な値 (ON) とする

【注意】 CPU 種別が SH-2E または SH-3E のとき、非正規化数の扱いを有効な値とした場合、ソースプログラムで最初に出現した浮動小数点データ制御命令に対してウォーニング 818 とし、指定どおりに非正規化数の扱いを有効な値としてオブジェクトコードを出力します。

【参照】 非正規化数の扱い 言語編「1.4.3 浮動小数点定数」

付 録

目 次

付録 A.	プログラム作成に関する制限と注意事項.....	337
付録 B.	サンプルプログラム.....	338
付録 C.	アセンブルリスト出力例.....	342
付録 D.	エラーメッセージ.....	347
付録 E.	旧バージョンとの相違.....	370
付録 F.	ASCII コード表.....	372

付録 A. プログラム作成に関する制限と注意事項

表 A.1 プログラム作成に関する制限と注意事項

項番	項目	制限
1	文字の種類	ASCII 文字、シフト JIS コード、EUC コード
2	英大文字と 英小文字の区別	シンボル（セクション名を含む）：区別する
		オブジェクトモジュール名：区別する
		予約語：区別しない
		実行命令ニーモニック：区別しない
		DSP 命令ニーモニック：区別しない
		アセンブラ制御命令ニーモニック：区別しない
		オペレーションサイズ：区別しない
		整数定数の基数：区別しない
3	1 行の長さ	255 バイト以内
4	アセンブル行数	65,535 行以内
5	文字定数	4 文字以内
6	シンボル文字数	251 文字以内
7	シンボル数	65,535 個以内
8	外部参照シンボル数	65,535 個以内
9	外部定義シンボル数	65,535 個以内
10	セクションのサイズ	H'FFFFFFF バイト以内
11	セクション数	65,535 個以内
12	マクロ生成番号の数	100,000 個以内
13	リテラルの数	100,000 個以内

付録 B. サンプルプログラム

本アセンブリ言語で記述したソースプログラムの例を示します。

B.1 サンプルプログラム仕様

機能概要

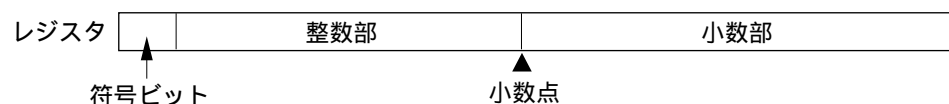
固定小数点の四則演算を行う、マクロ定義およびサブルーチン。

パラメータ 1 OP パラメータ 2 演算結果

OP: +、-、×、÷

【注】 丸め方法は切り捨てです。アンダフロー、オーバフローのチェックは行いません。

データのフォーマット



小数点の位置は、最上位ビット(MSB)からのビット数を、プリプロセッサ変数 POINT に設定します。

入出力

入力 :

レジスタ Parm1 に、パラメータ 1 を設定してください。

レジスタ Parm2 に、パラメータ 2 を設定してください。

ただし、加減算はマクロパラメータにパラメータ 1、パラメータ 2 を設定できません。

出力 :

レジスタ Parm1 に演算結果が設定されます。

使用方法

加算 (+) : マクロコール FIX_ADD [パラメータ 1],[パラメータ 2]

減算 (-) : マクロコール FIX_SUB [パラメータ 1],[パラメータ 2]

乗算 (×) : サブルーチンコール FIX_MUL

除算 (÷) : サブルーチンコール FIX_DIV

使用するレジスタ

次のレジスタを.REG アセンブラ制御命令で定義してください。

Parm1, Parm2, WORK1, WORK2, WORK3, WORK4

B.2 コーディング例

```

.MACRO FIX_ADD Rs=Parm2,Rd=Parm1
ADD    \Rs,\Rd
.ENDM

.MACRO FIX_SUB Rs=Parm2,Rd=Parm1
SUB    \Rs,\Rd
.ENDM

FIX_MUL:
DIV0S    Parm1,Parm2    ;
MOVT     WORK1          ; WORK1 に結果の符号を保持
CMP/PZ   Parm1          ; -+
BT       MUL01          ; | if (Parm1 < 0) Parm1 = -Parm1
NEG      Parm1,Parm1    ; -+
MUL01    CMP/PZ         Parm2    ; -+
BT       MUL02          ; | if (Parm2 < 0) Parm2 = -Parm2
NEG      Parm2,Parm2    ; -+
MUL02    MULU           Parm1,Parm2 ; Parm1(Low) * Parm2(Low)
SWAP.W   Parm1,Parm1    ;
STS      MACL,WORK2     ;
MULU     Parm1,Parm2    ; Parm1(High) * Parm2(Low)
SWAP.W   Parm1,Parm1    ;
SWAP.W   Parm2,Parm2    ;
STS      MACL,WORK3     ;
MULU     Parm1,Parm2    ; Parm1(Low) * Parm2(High)
SWAP.W   Parm1,Parm1    ;
STS      MACL,WORK4     ;
MULU     Parm1,Parm2    ; Parm1(High) * Parm2(High)
CLRT     ;
STS      MACL,Parm1     ;
MOV      WORK3,Parm2    ; -+
SHLR16   WORK3          ; |
SHLL16   Parm2          ; |
ADDC     Parm2,WORK2     ; |
ADDC     WORK3,Parm1     ; |16 ビット乗算結果を合計する
MOV      WORK4,Parm2    ; |
SHLR16   WORK4          ; |
SHLL16   Parm2          ; |
ADDC     WORK4,Parm1     ; -+
.AREPEAT \&POINT        ; -+
SHLL     Parm2          ; | 小数点位置の補正
ROTCL    Parm1          ; |
.AENDR   ; -+
SHLR     WORK1          ; -+
BF       MUL03          ; | 符号を付ける
NEG      Parm1,Parm1    ; -+
MUL03    RTS
NOP

```

```
FIX_DIV:
    MOV          #0,WORK1          ; -+
    DIV0S        WORK1,Parm        ; | 非除数が負ならば1の補数に変換
    SUBC         WORK1,Parm1       ; -+
    .AREPEAT     \&POINT          ; -+
    SHAR         Parm1             ; | 小数点位置の補正
    ROTCR        WORK1            ; |
    .AENDR              ; -+
    DIV0S        Parm2,Parm1       ;
    .AREPEAT     32               ; -+
    ROTCL        WORK1            ; | Parm1:WORK1 / Parm2 -> WORK1
    DIV1         Parm2,Parm1       ; |
    .AENDR              ; -+
    ROTCL        WORK1            ;
    MOV          #0,Parm1          ; -+
    ADDC         Parm1,WORK1       ; | 2の補数に変換
    MOV          WORK1,Parm1       ; -+
    RTS
    NOP
```


付録 C. アセンブルリスト出力例

アセンブルリストはアセンブルの結果を示すリストです。

アセンブルリストには、ソースプログラム・リスト、クロスリファレンス・リスト、セクション情報リストがあります。

以下に示すソースプログラムをアセンブルする場合を例として、アセンブルリストの内容と出力形式について説明します。

なお、本プログラムは、付録 B のサンプルプログラムを利用して以下の計算をするプログラムです。

$$1.5 \times 2.25 + 3 \div 5$$

```
POINT .ASSIGNA 16
Parm1 .REG (R0)
Parm2 .REG (R1)
WORK1 .REG (R2)
WORK2 .REG (R3)
WORK3 .REG (R4)
WORK4 .REG (R5)
      .SECTION SAMPLE, CODE, ALIGN=4
      .INCLUDE "付録 B"
a      .REG (R8)
b      .REG (R9)
c      .REG (R10)
d      .REG (R11)
start
      STS    PR, @-SP
      MOV.L  #H'00018000, a
      MOV.L  #H'00024000, b
      MOV.L  #H'00030000, c
      MOV.L  #H'00050000, d
      MOV    a, Parm1
      MOV    b, Parm2
      BSR    FIX_MUL
      NOP
      MOV    Parm1, a
      MOV    c, Parm1
      MOV    d, Parm2
      BSR    FIX_DIV
      NOP
      FIX_ADD a
      MOV    Parm1, a
      LDS    @SP+, PR
      RTS
      NOP
      .END
```

C.1 ソースプログラム・リスト

ソースプログラム・リストは、ソースステートメントに関する情報（行番号、対応するオブジェクトコードなど）を示します。

図 C.1 は、ソースプログラム・リストの出力例です。

*** SuperH RISC engine ASSEMBLER Ver. 4.0 ***			01/12/98 19:52:40	
PROGRAM NAME =		<u>"SAMPLE" (7)</u>		
1		1	.HEADING ""SAMPLE""	
2		2	POINT	.ASSIGNA 16
3		3	Parm1	.REG (R0)
4		4	Parm2	.REG (R1)
5		5	WORK1	.REG (R2)
6		6	WORK2	.REG (R3)
7		7	WORK3	.REG (R4)
8		8	WORK4	.REG (R5)
~				
20	00000000	9	I1	FIX_MUL :
21	00000000 2107	10	I1	DIV0S Parm1,Parm2 ;
22	00000002 0229	11	I1	MOVT WORK1 ;
23	00000004 4011	12	I1	CMP/PZ Parm1 ; -+
24	00000006 8900	13	I1	BT MUL01 ; if (Parm1
<u>25</u>	<u>00000008 600B</u>	<u>14</u>	<u>I1</u>	<u>NEG Parm1,Parm1 ; -+</u>
(1)	(2)	(3)	(4) (5)	(6)
~				
237		***** BEGIN-POOL *****		
238	00000180 A008	BRA TO END-POOL		
239	00000182 0009	NOP		
240	00000184 00018000	DATA FOR SOURCE-LINE 217 (8)		
241	00000188 00024000	DATA FOR SOURCE-LINE 218		
242	0000018C 00030000	DATA FOR SOURCE-LINE 219		
243	00000190 00050000	DATA FOR SOURCE-LINE 220		
244		***** END-POOL *****		
245		39	.END	
****TOTAL ERRORS 0				
<u>****TOTAL WARNINGS 0</u>				
(9)				

図 C.1 ソースプログラム・リスト出力例

- (1) 行番号（10進）です。
- (2) ロケーションカウンタ値（16進）です。
- (3) オブジェクトコード（16進）です。オペレーションが .RES、.SRES、.SRESC、.SRESZ、.FRES のいずれかであるときは、確保する領域のサイズをバイト単位で示します。
- (4) ソース行番号（10進）です。

(5) 展開区分

ファイルインクルード、条件つきアセンブリ機能、マクロ機能が展開した区分を示します。

In・・・ファイルインクルード（n はインクルードのネストレベルを示します）

C ……条件つきアセンブルの成立、繰り返し展開、条件付き繰り返し展開

M ……マクロ展開

(6) ソースステートメントです。

(7) .HEADING で設定したヘッダです。

(8) リテラルプールです。

(9) エラーとウォーニングの総数です。エラーメッセージは、エラーが存在するソースステートメントの次の行に出力されます。

C.2 クロスリファレンス・リスト

クロスリファレンス・リストは、シンボルに関する情報（属性、値など）を示します。

図 C.2 は、クロスリファレンス・リストの出力例です。

*** SuperH RISC engine ASSEMBLER Ver. 4.0 ***				01/12/98 19:52:4			
*** CROSS REFERENCE LIST							
NAME	SECTION	ATTR	VALUE	SEQUENCE			
FIX_DIV	SAMPLE		00000088	94*	229		
FIX_MUL	SAMPLE		00000000	20*	224		
MAN03		UDEF	00000000	89			
MUL01	SAMPLE		0000000A	24	26*		
MUL02	SAMPLE		00000010	27	29*		
Parm1		REG		3*	21	23	25
				37	37	39	41
				69	71	73	75
				96	97	102	104
				122	124	126	128
				150	152	154	156
				174	176	178	180
				198	200	202	204
Parm2		REG		4*	21	26	28
				44	45	47	49
				70	72	74	76
				144	146	148	150
				168	170	172	174
(1)		(2)	(3)	(4)	(5)		

図 C.2 クロスリファレンス・リスト出力例

- (1) シンボルの名称です。
- (2) シンボルが含まれるセクションの名称です。（セクション名の最初の 8 文字）
- (3) シンボルの属性です。

EXPT	外部定義シンボル
IMPT	外部参照シンボル
SCT	セクション名
REG	.REG アセンブラ制御命令で定義したシンボル
FREG	.FREG アセンブラ制御命令で定義したシンボル
ASGN	.ASSIGN アセンブラ制御命令で定義したシンボル
EQU	.EQU アセンブラ制御命令で定義したシンボル
MDEF	二重定義したシンボル
UDEF	未定義シンボル
表示なし	その他のシンボル

- (4) シンボルの値です。（16 進）
- (5) シンボルを定義、参照しているリスト行番号（10 進）です。
行番号の後ろのアスタリスク（*）は、定義行であることを示します。

C.3 セクション情報リスト

セクション情報リストは、セクションに関する情報（セクションの種類、サイズなど）を示します。

図 C.3 は、セクション情報リストの出力例です。

*** SuperH RISC engine ASSEMBLER Ver. 4.0 ***				01/12/98 19:52:4
*** SECTION DATA LIST				
SECTION	ATTRIBUTE	SIZE	START	
<u>SAMPLE</u>	<u>REL-CODE</u>	<u>000000194</u>	<u> </u>	
(1)	(2)	(3)	(4)	

図 C.3 セクション情報リスト出力例

(1) セクションの名称です。

(2) セクションの種類です。

REL	相対アドレスセクション
ABS	絶対アドレスセクション
CODE	コードセクション
DATA	データセクション
COMMON	コモンセクション
STACK	スタックセクション
DUMMY	ダミーセクション

(3) セクションのサイズです。（16進数、単位はバイト）

(4) 絶対アドレスセクションの先頭アドレスです。

付録 D. エラーメッセージ

D.1 エラーの種類

(1) アセンブラ起動時のエラー

アセンブラの起動コマンドラインに関するエラーです。入力ソースファイルやコマンドライン・オプションの指定に誤りがある場合に発生します。

アセンブラは、ホストシステムの標準エラー出力（通常はディスプレイ装置）に、メッセージを出力します。^{*1}

メッセージの形式は、次のとおりです。^{*2}

“, line 行番号: エラー番号 (E) メッセージ

例

“, line 0:10 (E) NO INPUT FILE SPECIFIED

【注】 *1 オペレーティングシステムが Windows®95 および Windows®NT の場合、アセンブラは、標準出力にメッセージを出力します。

*2 オペレーティングシステムが Windows®95 および Windows®NT の場合
(行番号): エラー番号 (E) メッセージ

例

(0) : 10 (E) NO INPUT FILE SPECIFIED

(2) ソースプログラムのエラー

ソースプログラムの文法エラーです。

アセンブラは、ホストシステムの標準エラー出力またはソースプログラム・リストに、メッセージを出力します。(アセンブル時にソースプログラム・リストを出力する場合には、標準エラー出力にはメッセージが出ません)*¹

メッセージの形式は、次のとおりです。*²

"ソースファイル名", line 行番号 : エラー番号 (E) メッセージ

"ソースファイル名", line 行番号 : エラー番号 (W) メッセージ

例

"PROG.SRC", line 25 : 300 (E) ILLEGAL MNEMONIC

"PROG.SRC", line 33 : 811 (W) ILLEGAL SYMBOL DEFINITION

【注】 *¹ オペレーティングシステムが Windows[®]95 および Windows[®]NT の場合、アセンブラは、標準出力またはソースプログラム・リストに、メッセージを出力します。

*² オペレーティングシステムが Windows[®]95 および Windows[®]NT の場合

ソースファイル名(行番号) : エラー番号 (E) メッセージ

ソースファイル名(行番号) : エラー番号 (W) メッセージ

例

PROG.SRC(25) : 300 (E) ILLEGAL MNEMONIC

PROG.SRC(33) : 811 (W) ILLEGAL SYMBOL DEFINITION

ソースプログラムのエラーの番号は、次のように分類されています。

- 100 ~ 199 番 : ソースプログラム全般の文法エラー
- 200 ~ 299 番 : シンボルに関するエラー
- 300 ~ 349 番 : オペレーションまたはオペランドに関するエラー
- 350 ~ 399 番 : DSP 命令に関するエラー
- 400 ~ 499 番 : 式演算に関するエラー
- 500 ~ 599 番 : アセンブラ制御命令に関するエラー
- 600 ~ 699 番 : ファイルインクルード機能、条件つきアセンブリ機能、マクロ機能に関するエラー
- 700 ~ 799 番 : DSP 命令に関するウォーニング
- 800 ~ 999 番 : ソースプログラム全般のウォーニング

(3) フェイタルエラー (致命的なエラー)

アセンブラの動作環境に関するエラーです。ホストシステムのメモリ容量が不足した場合などに発生します。

アセンブラは、ホストシステムの標準エラー出力に、メッセージを出力します。^{*1}

メッセージの形式は、次のとおりです。^{*2}

"" , line 行番号 : エラー番号 (F) メッセージ

例

"" , line 0 : 903 (F) LISTING FILE OUTPUT ERROR

【注】^{*1} オペレーティングシステムが Windows[®]95 および Windows[®]NT の場合、アセンブラは、標準出力またはソースプログラム・リストに、メッセージを出力します。

^{*2} オペレーティングシステムが Windows[®]95 および Windows[®]NT の場合
(行番号) : エラー番号 (F) メッセージ

例

(0) : 903 (F) LISTING FILE OUTPUT ERROR

フェイタルエラーが発生すると、本アセンブラは処理を中止します。

D.2 エラーメッセージ一覧

表 D.1 起動時のエラーに対するメッセージ

10	メッセージ：NO INPUT FILE SPECIFIED 内容：入力ソースファイルの指定がない。 対策：入力ソースファイルを指定してください。
20	メッセージ：CANNOT OPEN FILE ファイル名 内容：指定のファイルをオープンできない。 対策：ファイル名、ディレクトリ名などを見直してください。
30	メッセージ：INVALID COMMAND PARAMETER 内容：コマンドライン・オプションに誤りがある。 対策：コマンドライン・オプションを見直してください。
40	メッセージ：CANNOT ALLOCATE MEMORY 内容：処理中にメモリが足りなくなった。 対策：ユーザが使用できるメモリ量が、極端に少ない場合に発生します。他に実行中の処理があれば、その処理を終了させてからアセンブラを再起動してください。それでも本エラーが発生する場合は、ホストシステムのメモリ管理の方法を見直してください。
50	メッセージ：COMPLETED FILE NAME TOO LONG ファイル名 内容：ディレクトリを含めたファイル名が長い。 対策：ディレクトリを含めたファイル名を短くしてください。 補足：このときアセンブラが出力するオブジェクトモジュールは、デバッガで扱えない可能性があります。

表 D.2 ソースプログラムのエラーに対するメッセージ

ソースプログラム全般の文法エラー	
100	<p>メッセージ：OPERATION TOO COMPLEX</p> <p>内容：演算が複雑すぎる。</p> <p>対策：演算を簡単にしてください。</p>
101	<p>メッセージ：SYNTAX ERROR IN SOURCE STATEMENT</p> <p>内容：ソースステートメントに構文上の誤りがある。</p> <p>対策：ソースステートメント全体を見直してください。</p>
102	<p>メッセージ：SYNTAX ERROR IN DIRECTIVE</p> <p>内容：アセンブラ制御命令のソースステートメントに構文上の誤りがある。</p> <p>対策：ソースステートメント全体を見直してください。</p>
104	<p>メッセージ：LOCATION COUNTER OVERFLOW</p> <p>内容：ロケーションカウンタ値が最大値を超える。</p> <p>対策：プログラムを縮小してください。</p>
105	<p>メッセージ：ILLEGAL INSTRUCTION IN STACK SECTION</p> <p>内容：スタックセクション内に、実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を記述している。</p> <p>対策：実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を削除してください。</p>
106	<p>メッセージ：TOO MANY ERRORS</p> <p>内容：エラーの数が多いので、表示を打ち切った。</p> <p>対策：ソースステートメント全体を見直してください。</p>
108	<p>メッセージ：ILLEGAL CONTINUATION LINE</p> <p>内容：複数行にわたって記述したソースステートメントに、誤りがある。</p> <p>対策：記述方法を見直してください。</p>
109	<p>メッセージ：LINE NUMBER OVERFLOW</p> <p>内容：1 度にあセンブルする行数が 65,535 行を超える。</p> <p>対策：プログラムを複数のファイルに分割してください。</p>
150	<p>メッセージ：INVALID DELAY SLOT INSTRUCTION</p> <p>内容：ディレイスロット命令（遅延分岐命令の直後にくる実行命令）が不当である。</p> <p>対策：実行命令を記述する順序を変更するなどして、ディレイスロット命令が不当とならないようにしてください。</p>
151	<p>メッセージ：ILLEGAL EXTENDED INSTRUCTION POSITION</p> <p>内容：ディレイスロット命令に拡張命令を記述している。</p> <p>対策：ディレイスロット命令は実行命令を記述してください。</p>

ソースプログラム全般の文法エラー	
152	<p>メッセージ：ILLEGAL BOUNDARY ALIGNMENT VALUE</p> <p>内容：拡張命令を記述しているセクションの境界調整数が不当である。</p> <p>対策：拡張命令を記述するセクションの境界調整数は、2 以上を指定してください。</p>
153	<p>メッセージ：ILLEGAL ADDRESS</p> <p>内容：実行命令、拡張命令を奇数アドレスに記述している。</p> <p>対策：実行命令、拡張命令は偶数アドレスに記述してください。</p>

REPEAT に関するエラー	
160	<p>メッセージ：REPEAT LOOP NESTING</p> <p>内容：REPEAT 拡張命令～終了アドレス間に別の REPEAT 拡張命令が存在する。</p> <p>対策：REPEAT 拡張命令の位置を訂正してください。</p>
161	<p>メッセージ：ILLEGAL START ADDRESS FOR REPEAT LOOP</p> <p>内容：REPEAT 拡張命令～開始アドレス間に実行命令または DSP 命令が存在しない。</p> <p>対策：REPEAT 拡張命令と開始アドレスの間に 1 つ以上の実行命令または DSP 命令を記述してください。</p>
162	<p>メッセージ：ILLEGAL DATA BEFORE REPEAT LOOP</p> <p>内容：REPEAT 拡張命令で指定したループの直前に不当なデータが存在する。</p> <p>対策：ループ直前がアセンブラ制御命令の場合、アセンブラ制御命令を訂正してください。ループ直前がリテラルプールの場合、.NOPOOL 制御命令でリテラルプールの出力を抑止してください。</p> <p>補足：リピートする命令が 3 命令以下の場合、ループの直前は実行命令または DSP 命令でなければなりません。</p>

シンボルに関するエラー	
200	<p>メッセージ：UNDEFINED SYMBOL REFERENCE</p> <p>内容：参照しているシンボルが定義されていない。</p> <p>対策：シンボルを定義してください。</p>
201	<p>メッセージ：ILLEGAL SYMBOL OR SECTION NAME</p> <p>内容：シンボル（セクション名を含む）として予約語を指定している。</p> <p>対策：シンボル（セクション名を含む）を訂正してください。</p>
202	<p>メッセージ：ILLEGAL SYMBOL OR SECTION NAME</p> <p>内容：シンボル（セクション名を含む）に誤りがある。</p> <p>対策：シンボル（セクション名を含む）を訂正してください。</p>
203	<p>メッセージ：ILLEGAL LOCAL LABEL</p> <p>内容：ローカルラベルの指定に誤りがある。</p> <p>対策：ローカルラベルの指定を訂正してください。</p>

オペレーションまたはオペランドに関するエラー	
300	メッセージ：ILLEGAL MNEMONIC 内容：オペレーションに誤りがある。 対策：オペレーションを訂正してください。
301	メッセージ：TOO MANY OPERANDS OR ILLEGAL COMMENT 内容：実行命令のオペランドが多すぎる。または、コメントに誤りがある。 対策：オペランドまたはコメントを訂正してください。
304	メッセージ：LACKING OPERANDS 内容：オペランドが少なすぎる。 対策：オペランドを訂正してください。
307	メッセージ：ILLEGAL ADDRESSING MODE 内容：オペランドに、許されないアドレス形式を指定している。 対策：オペランドを訂正してください。
308	メッセージ：SYNTAX ERROR IN OPERAND 内容：オペランドに文法上の誤りがある。 対策：オペランドを訂正してください。
309	メッセージ：FLOATING POINT REGISTER MISMATCH 内容：単精度オペレーションに対して、倍精度浮動小数点レジスタを指定した。または、 倍精度オペレーションに対して、単精度浮動小数点レジスタを指定した。 対策：オペレーションサイズまたは浮動小数点レジスタを訂正してください。

DSP 命令に関するエラー	
350	<p>メッセージ: SYNTAX ERROR IN SOURCE STATEMENT (ニーモニック)</p> <p>内容: DSP 命令のソースステートメントに構文上の誤りがあります。</p> <p>対策: ソースステートメントを見直してください。</p>
351	<p>メッセージ: ILLEGAL COMBINATION OF MNEMONICS (ニーモニック, ニーモニック)</p> <p>内容: 許されていない DSP 演算命令の組み合わせを指定しています。</p> <p>対策: 命令の組み合わせを訂正してください。</p>
352	<p>メッセージ: ILLEGAL CONDITION (ニーモニック)</p> <p>内容: DSP 演算命令に対するコンディションを不正に指定しています。</p> <p>対策: コンディションの指定を削除するか、DSP 演算命令を変更してください。</p>
353	<p>メッセージ: ILLEGAL POSITION OF INSTRUCTION (ニーモニック)</p> <p>内容: DSP 演算命令の指定位置が誤っています。</p> <p>対策: DSP 演算命令を正しい位置に指定してください。</p>
354	<p>メッセージ: ILLEGAL ADDRESSING MODE (ニーモニック)</p> <p>内容: DSP 演算命令のアドレス形式が誤っています。</p> <p>対策: オペランドを訂正してください。</p>
355	<p>メッセージ: ILLEGAL REGISTER NAME (ニーモニック)</p> <p>内容: DSP 演算命令のレジスタ指定に誤りがあります。</p> <p>対策: レジスタ名を訂正してください。</p>
357	<p>メッセージ: ILLEGAL COMBINATION OF MNEMONICS (ニーモニック)</p> <p>内容: データ転送命令の指定が不正です。</p> <p>対策: データ転送命令を訂正してください。</p>
371	<p>メッセージ: ILLEGAL COMBINATION OF MNEMONICS (ニーモニック, ニーモニック)</p> <p>内容: データ転送命令の組み合わせが誤っています。</p> <p>対策: データ転送命令の組み合わせを訂正してください。</p>
372	<p>メッセージ: ILLEGAL ADDRESSING MODE (ニーモニック)</p> <p>内容: データ転送命令のオペランドとして許されないアドレス形式を指定しています。</p> <p>対策: オペランドを訂正してください。</p>
373	<p>メッセージ: ILLEGAL REGISTER NAME (ニーモニック)</p> <p>内容: データ転送命令のレジスタの指定が誤っています。</p> <p>対策: レジスタ名を訂正してください。</p>

式演算に関するエラー	
400	<p>メッセージ：CHARACTER CONSTANT TOO LONG</p> <p>内容：文字定数が4文字を超えている。</p> <p>対策：文字定数を訂正してください。</p>
402	<p>メッセージ：ILLEGAL VALUE IN OPERAND</p> <p>内容：オペランドとして、範囲外の値である。</p> <p>対策：値を変更してください。</p>
403	<p>メッセージ：ILLEGAL OPERATION FOR RELATIVE VALUE</p> <p>内容：相対値に対して、乗除算または論理演算を行おうとしている。</p> <p>対策：演算内容を訂正してください。</p>
406	<p>メッセージ：ILLEGAL OPERAND</p> <p>内容：浮動小数点データを指定するところに式を指定した。</p> <p>対策：式ではなく、浮動小数点定数を指定してください。</p>
407	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：式の計算中、計算用のメモリが足りなくなった。</p> <p>対策：演算内容を簡単にしてください。</p>
408	<p>メッセージ：DIVISION BY ZERO</p> <p>内容：0 除算を行おうとしている。</p> <p>対策：演算内容を変更してください。</p>
409	<p>メッセージ：REGISTER IN EXPRESSION</p> <p>内容：式の中にレジスタ名が現れた。</p> <p>対策：演算内容を訂正してください。</p>
411	<p>メッセージ：INVALID STARTOF/SIZEOF OPERAND</p> <p>内容：STARTOF 演算または SIZEOF 演算で、誤ったセクション名を指定している。</p> <p>対策：セクション名を訂正してください。</p>
412	<p>メッセージ：ILLEGAL SYMBOL IN EXPRESSION</p> <p>内容：シフト数に相対値を指定している。</p> <p>対策：演算内容を訂正してください。</p>
450	<p>メッセージ：ILLEGAL DISPLACEMENT VALUE</p> <p>内容：ディスプレースメント値が不当である。（負の値を指定した）</p> <p>対策：ディスプレースメント値を訂正してください。</p>
452	<p>メッセージ：ILLEGAL DATA AREA ADDRESS</p> <p>内容：PC 相対データ転送命令のデータ領域のアドレスが不当である。</p> <p>対策：各命令のオペレーションサイズにあった境界のアドレスにアクセスしてください。 （MOV.L, MOVA 命令は4バイト境界、MOV.W は2バイト境界です）</p>

式演算に関するエラー	
453	<p>メッセージ：LITERAL POOL OVERFLOW</p> <p>内容：リテラル未出力の拡張命令が 510 個を超えた。</p> <p>対策：.POOL を用いるなどしてリテラルプールを出力してください。</p>

REPEAT に関するエラー	
460	<p>メッセージ：ILLEGAL SYMBOL</p> <p>内容：REPEAT のオペランドに後方参照のラベル、未定義シンボル、ラベル以外のシンボルを指定した。または、開始アドレスが終了アドレスより大きい（前方にある）。</p> <p>対策：オペランドを訂正してください。</p>
461	<p>メッセージ：SYNTAX ERROR IN OPERAND</p> <p>内容：不当なオペランドを指定した。</p> <p>対策：オペランドを訂正してください。</p>
462	<p>メッセージ：ILLEGAL VALUE IN OPERAND</p> <p>内容：REPEAT 拡張命令とラベルの距離が範囲外である。</p> <p>対策：REPEAT 拡張命令またはラベルの位置を訂正してください。</p>
463	<p>メッセージ：NO INSTRUCTION IN REPEAT LOOP</p> <p>内容：リピートループ内に命令が存在しない。または、終了アドレスに指定した位置に命令が存在しない。</p> <p>対策：開始アドレス～終了アドレス間に命令を記述してください。または、命令の存在するアドレスを終了アドレスに指定してください。</p>

アセンブラ制御命令に関するエラー	
500	<p>メッセージ：SYMBOL NOT FOUND</p> <p>内容：ラベルが必要なアセンブラ制御命令のソースステートメントに、ラベルがない。</p> <p>対策：ラベルを記述してください。</p>
501	<p>メッセージ：ILLEGAL ADDRESS VALUE IN OPERAND</p> <p>内容：セクションの先頭アドレスの指定が誤っている。または、ロケーションカウンタ値の指定が誤っている。</p> <p>対策：先頭アドレスまたはロケーションカウンタ値を訂正してください。</p>
502	<p>メッセージ：ILLEGAL SYMBOL IN OPERAND</p> <p>内容：オペランドに、不当な値（前方参照シンボル、外部参照シンボル、相対アドレスシンボル）を指定している。</p> <p>対策：オペランドを訂正してください。</p>
503	<p>メッセージ：UNDEFINED EXPORT SYMBOL</p> <p>内容：ファイル内で定義していないシンボルを、外部定義シンボルとして宣言している。</p> <p>対策：シンボルを定義してください。または、外部定義シンボルとしての宣言を取りやめてください。</p>

アセンブラ制御命令に関するエラー	
504	<p>メッセージ：INVALID RELATIVE SYMBOL IN OPERAND</p> <p>内容：オペランドに不当な値（前方参照シンボル、外部参照シンボル）を指定している。</p> <p>対策：オペランドを訂正してください。</p>
505	<p>メッセージ：ILLEGAL OPERAND</p> <p>内容：オペランドのつづりに誤りがある。</p> <p>対策：オペランドを訂正してください。</p>
506	<p>メッセージ：ILLEGAL OPERAND</p> <p>内容：オペランドとして許されない要素を指定している。</p> <p>対策：オペランドを訂正してください。</p>
508	<p>メッセージ：ILLEGAL VALUE IN OPERAND</p> <p>内容：オペランドに範囲外の値を指定している。</p> <p>対策：オペランドを訂正してください。</p>
510	<p>メッセージ：ILLEGAL BOUNDARY VALUE</p> <p>内容：境界調整数の指定に誤りがある。</p> <p>対策：境界調整数を訂正してください。</p>
512	<p>メッセージ：ILLEGAL EXECUTION START ADDRESS</p> <p>内容：実行開始アドレスに誤りがある。</p> <p>対策：実行開始アドレスを訂正してください。</p>
513	<p>メッセージ：ILLEGAL REGISTER NAME</p> <p>内容：レジスタ名に誤りがある。</p> <p>対策：レジスタ名を訂正してください。</p>
514	<p>メッセージ：INVALID EXPORT SYMBOL</p> <p>内容：外部定義できないシンボルを、外部定義シンボルとして宣言している。</p> <p>対策：外部定義シンボルとしての宣言を取りやめてください。</p>
516	<p>メッセージ：EXCLUSIVE DIRECTIVES</p> <p>内容：制御命令の指定内容が矛盾している。</p> <p>対策：関連する制御命令を含めて見直してください。</p>
517	<p>メッセージ：INVALID VALUE IN OPERAND</p> <p>内容：オペランドに、不当な値（前方参照シンボル、外部参照シンボル、他セクションの相対アドレスシンボル）を指定している。</p> <p>対策：オペランドを訂正してください。</p>
518	<p>メッセージ：INVALID IMPORT SYMBOL</p> <p>内容：ファイル内で定義しているシンボルを、外部参照シンボルとして宣言している。</p> <p>対策：外部参照シンボルとしての宣言を取りやめてください。</p>

アセンブラ制御命令に関するエラー	
520	メッセージ：ILLEGAL .CPU DIRECTIVE POSITION 内容：.CPU 制御命令がプログラムの先頭でない。または、複数回指定している。 対策：.CPU 制御命令はプログラムの先頭に 1 回だけ指定してください。
521	メッセージ：ILLEGAL .NOPOOL DIRECTIVE POSITION 内容：.NOPOOL の記述位置が不当である。 対策：.NOPOOL はディレイスロット命令に続けて記述してください。
522	メッセージ：ILLEGAL .POOL DIRECTIVE POSITION 内容：.POOL を遅延分岐命令に続けて記述している。 対策：遅延分岐命令の後にはディレイスロット命令を記述してください。
523	メッセージ：ILLEGAL OPERAND 内容：.LINE 制御命令のオペランドに誤りがある。 対策：.LINE 制御命令のオペランドを訂正してください。
525	メッセージ：ILLEGAL .LINE DIRECTIVE POSITION 内容：.LINE 制御命令がマクロ展開または条件付き繰り返し展開中に指定された。 対策：.LINE 制御命令の指定位置を変えてください。

ファイルインクルード機能、条件つきアセンブリ機能、マクロ機能に関するエラー	
600	<p>メッセージ：INVALID CHARACTER</p> <p>内容：ソースプログラムに不当な文字がある。</p> <p>対策：不当な文字を訂正してください。</p>
601	<p>メッセージ：INVALID DELIMITER</p> <p>内容：区切り文字が不当である。</p> <p>対策：区切り文字を訂正してください。</p>
602	<p>メッセージ：INVALID CHARACTER STRING FORMAT</p> <p>内容：文字列に誤りがある。</p> <p>対策：文字列を訂正してください。</p>
603	<p>メッセージ：SYNTAX ERROR IN SOURCE STATEMENT</p> <p>内容：ソースステートメントに構文上の誤りがある。</p> <p>対策：ソースステートメント全体を見直してください。</p>
604	<p>メッセージ：ILLEGAL SYMBOL IN OPERAND</p> <p>内容：制御命令のオペランドが不当である。</p> <p>対策：本制御命令のオペランドにシンボルおよびロケーションカウンタ（\$）は記述できません。</p>
610	<p>メッセージ：MULTIPLE MACRO NAMES</p> <p>内容：.MACRO で定義しようとしているマクロ命令は既に定義されている。</p> <p>対策：マクロ名を訂正してください。</p>
611	<p>メッセージ：MACRO NAME NOT FOUND</p> <p>内容：.MACRO のオペランドにマクロ名がない。</p> <p>対策：マクロ名を記述してください。</p>
612	<p>メッセージ：ILLEGAL MACRO NAME</p> <p>内容：.MACRO のマクロ名に誤りがある。</p> <p>対策：マクロ名を訂正してください。</p>
613	<p>メッセージ：ILLEGAL .MACRO DIRECTVE POSITION</p> <p>内容：マクロ本体（.MACRO ~ .ENDM 間）、.AREPEAT ~ .AENDR 間、.AWHILE ~ .AENDW 間に.MACRO がある。</p> <p>対策：.MACRO を削除してください。</p>
614	<p>メッセージ：MULTIPLE MACRO PARAMETERS</p> <p>内容：マクロ定義（.MACRO）の仮引数の宣言で、仮引数名が重複した。</p> <p>対策：仮引数名を訂正してください。</p>
615	<p>メッセージ：ILLEGAL .END DIRECTIVE POSITION</p> <p>内容：マクロ本体（.MACRO ~ .ENDM 間）に.END がある。</p> <p>対策：.END を削除してください。</p>

ファイルインクルード機能、条件つきアセンブリ機能、マクロ機能に関するエラー	
616	<p>メッセージ：MACRO DIRECTIVES MISMATCH</p> <p>内容：.ENDM が.MACRO に対応していない。または、.EXITM がマクロ本体（.MACRO ~ .ENDM 間）、.AREPEAT ~ .AENDR 間、.AWHILE ~ .AENDW 間以外にある。</p> <p>対策：.ENDM または.EXITM を削除してください。</p>
618	<p>メッセージ：MACRO EXPANSION TOO LONG</p> <p>内容：マクロ展開で 1 行の文字数が 255 文字を超えた。</p> <p>対策：255 文字以下になるように訂正してください。</p>
619	<p>メッセージ：ILLEGAL MACRO PARAMETER</p> <p>内容：マクロコールでマクロパラメータの仮引数名に誤りがある。または、マクロ本体（.MACRO ~ .ENDM 間）の仮引数名に誤りがある。</p> <p>対策：仮引数名を訂正してください。</p> <p>補足：マクロ本体の仮引数名が誤りの場合は、マクロ展開時にエラーになります。</p>
620	<p>メッセージ：UNDEFINED PREPROCESSOR VARIABLE</p> <p>内容：参照しているプリプロセッサ変数が定義されていない。</p> <p>対策：プリプロセッサ変数を定義してください。</p>
621	<p>メッセージ：ILLEGAL .END DIRECTIVE POSITION</p> <p>内容：マクロ展開中に.END がある。</p> <p>対策：.END を削除してください。</p>
622	<p>メッセージ：')' NOT FOUND</p> <p>内容：マクロ処理除外の閉じカッコがない。</p> <p>対策：マクロ処理除外の閉じカッコを記述してください。</p>
623	<p>メッセージ：SYNTAX ERROR IN STRING FUNCTION</p> <p>内容：文字列操作関数に構文上の誤りがある。</p> <p>対策：文字列操作関数を見直してください。</p>
624	<p>メッセージ：MACRO PARAMETERS MISMATCH</p> <p>内容：マクロコールで位置指定のマクロパラメータの数が多い。</p> <p>対策：マクロパラメータの数を訂正してください。</p>
631	<p>メッセージ：END DIRECTIVE MISMATCH</p> <p>内容：対になる制御文で、終了の制御文が一致しない。</p> <p>対策：制御文を見直してください。</p>
640	<p>メッセージ：SYNTAX ERROR IN OPERAND</p> <p>内容：条件つきアセンブリ制御文のオペランドに構文上の誤りがある。</p> <p>対策：ソースステートメント全体を見直してください。</p>

ファイルインクルード機能、条件つきアセンブリ機能、マクロ機能に関するエラー	
641	<p>メッセージ：INVALID RELATIONAL OPERATOR</p> <p>内容：条件つきアセンブリ制御文のオペランドの関係演算子に誤りがある。</p> <p>対策：関係演算子を訂正してください。</p>
642	<p>メッセージ：ILLEGAL .END DIRECTIVE POSITION</p> <p>内容：.AREPEAT ~ .AENDR 間、.AWHILE ~ .AENDW 間に.END がある。</p> <p>対策：.END を削除してください。</p>
643	<p>メッセージ：DIRECTIVE MISMATCH</p> <p>内容：.AREPEAT、.AWHILE に対する.AENDR、.AENDW が対になっていない。</p> <p>対策：制御文を見直してください。</p>
644	<p>メッセージ：ILLEGAL .AENDW OR .AENDR DIRECTIVE POSITION</p> <p>内容：.AIF ~ .AENDI 間に.AENDR、.AENDW がある。</p> <p>対策：.AENDR、.AENDW を削除してください。</p>
645	<p>メッセージ：EXPANSION TOO LONG</p> <p>内容：.AREPEAT、.AWHILE 展開で 1 行の文字数が 255 文字を超えた。</p> <p>対策：255 文字以下になるように訂正してください。</p>
650	<p>メッセージ：INVALID INCLUDE FILE</p> <p>内容：.INCLUDE のファイル名に誤りがある。</p> <p>対策：ファイル名を訂正してください。</p>
651	<p>メッセージ：CANNOT OPEN INCLUDE FILE</p> <p>内容：.INCLUDE のファイル名をオープンできない。</p> <p>対策：ファイル名を訂正してください。</p>
652	<p>メッセージ：INCLUDE NEST TOO DEEP</p> <p>内容：ファイルインクルードのネストが 30 レベルを超えた。</p> <p>対策：ネストを 30 レベル以下にしてください。</p>
653	<p>メッセージ：SYNTAX ERROR IN OPERAND</p> <p>内容：.INCLUDE のオペランドに構文上の誤りがある。</p> <p>対策：オペランドを訂正してください。</p>
660	<p>メッセージ：.ENDM NOT FOUND</p> <p>内容：.MACRO に対する.ENDM がない。</p> <p>対策：.ENDM を記述してください。</p>
662	<p>メッセージ：ILLEGAL .END DIRECTIVE POSITION</p> <p>内容：.AIF ~ .AENDI 間に.END がある。</p> <p>対策：.END を削除してください。</p>

ファイルインクルード機能、条件つきアセンブリ機能、マクロ機能に関するエラー	
663	<p>メッセージ: ILLEGAL .END DIRECTIVE POSITION</p> <p>内容: インクルードファイル中に.END がある。</p> <p>対策: .END を削除してください。</p>
664	<p>メッセージ: ILLEGAL .END DIRECTIVE POSITION</p> <p>内容: .AIF ~ .AENDI 間に.END がある。</p> <p>対策: .END を削除してください。</p>
665	<p>メッセージ: EXPANSION TOO LONG</p> <p>内容: .DEFINE 制御文で 1 行の文字数が 255 文字を超えた。</p> <p>対策: 255 文字以下になるように訂正してください。</p>
667	<p>メッセージ: SUCCESSFUL CONDITION .AERROR</p> <p>内容: .AERROR 制御文を含むステートメントが、.AIF の条件によって処理された。</p> <p>対策: .AERROR を処理しないように条件式を見直してください。</p>
668	<p>メッセージ: ILLEGAL VALUE IN OPERAND</p> <p>内容: .AIFDEF 制御命令のオペランドに誤りがある。</p> <p>対策: 本制御命令のオペランドは、.DEFINE 制御文のシンボルで指定してください。</p>

DSP 命令に関するウォーニング	
700	<p>メッセージ: ILLEGAL VALUE IN OPERAND (ニーモニック)</p> <p>内容: DSP 演算命令のオペランドが値の範囲を超えています。</p> <p>対策: 値を訂正してください。</p>
701	<p>メッセージ: MULTIPLE REGISTER IN DESTINATION (ニーモニック, ニーモニック)</p> <p>内容: DSP 演算命令のデスティネーションオペランドに複数の同一レジスタを指定しています。</p> <p>対策: レジスタの指定を訂正してください。</p>
702	<p>メッセージ: ILLEGAL OPERATION SIZE (ニーモニック)</p> <p>内容: DSP 演算命令またはデータ転送命令のオペレーションサイズが誤っています。</p> <p>対策: オペレーションサイズを訂正または削除してください。</p>
703	<p>メッセージ: MULTIPLE REGISTER IN DESTINATION (ニーモニック, ニーモニック)</p> <p>内容: DSP 演算命令とデータ転送命令で同一のデスティネーションレジスタを指定しています。</p> <p>対策: レジスタの指定を訂正してください。</p>

ソースプログラム全般のウォーニング	
800	<p>メッセージ：SYMBOL NAME TOO LONG</p> <p>内容：シンボルが 251 文字を超えている。</p> <p>対策：シンボルを訂正してください。</p> <p>補足：アセンブラは、252 文字目以降を無視します。</p>
801	<p>メッセージ：MULTIPLE SYMBOLS</p> <p>内容：定義済みのシンボルを、再び定義している。</p> <p>対策：シンボルの再定義を取りやめてください。</p> <p>補足：アセンブラは、2 度目以降の定義を無視します。</p>
807	<p>メッセージ：ILLEGAL OPERATION SIZE</p> <p>内容：オペレーションサイズに誤りがある。</p> <p>対策：オペレーションサイズを訂正してください。</p> <p>補足：アセンブラは、オペレーションサイズの指定を無視します。</p>
808	<p>メッセージ：ILLEGAL CONSTANT SIZE</p> <p>内容：整数定数の記述の一部に、誤りがある。</p> <p>対策：記述を訂正してください。</p> <p>補足：アセンブラが整数定数を誤って（プログラムの意図しない値として）解釈する可能性があります。</p>
810	<p>メッセージ：TOO MANY OPERANDS</p> <p>内容：アセンブラ制御命令のオペランドが多すぎる。または、コメントに誤りがある。</p> <p>対策：オペランドまたはコメントを訂正してください。</p> <p>補足：アセンブラは、余分なオペランドの指定を無視します。</p>
811	<p>メッセージ：ILLEGAL SYMBOL DEFINITION</p> <p>内容：ラベルを記述できないアセンブラ制御命令のソースステートメントに、ラベルをつけている。</p> <p>対策：ラベルを削除してください。</p> <p>補足：アセンブラは、ラベルを無視します。</p>
813	<p>メッセージ：SECTION ATTRIBUTE MISMATCH</p> <p>内容：セクションの再開で、異なる種類のセクションを指定している。または、絶対アドレスセクションの再開で、セクションの先頭アドレスを再び指定している。</p> <p>対策：セクションを再開する場合は、セクションの種類や先頭アドレスを指定しないでください。</p> <p>補足：セクションを開始したときの指定が、そのまま有効です。</p>

ソースプログラム全般のウォーニング	
815	<p>メッセージ：MULTIPLE MODULE NAMES</p> <p>内容：オブジェクトモジュール名を再設定している。</p> <p>対策：オブジェクトモジュールの設定は、1 度だけにしてください。</p> <p>補足：アセンブラは、2 度目以降の設定を無視します。</p>
816	<p>メッセージ：ILLEGAL DATA AREA ADDRESS</p> <p>内容：データまたはデータ領域の配置が不当である。</p> <p>対策：ワード単位のデータやデータ領域は、先頭アドレスが偶数になるように確保してください。ロングワードまたは単精度単位のデータやデータ領域は、先頭アドレスが4 の倍数になるように確保してください。倍精度単位のデータやデータ領域は、先頭アドレスが8 の倍数になるように確保してください。</p> <p>補足：アセンブラは、データまたはデータ領域を指定どおりに配置します。</p>
817	<p>メッセージ：ILLEGAL BOUNDARY VALUE</p> <p>内容：コードセクションの境界調整数が4 未満である。</p> <p>対策：指定は有効です。ただし、実行命令、DSP 命令、拡張命令を奇数アドレスに記述している場合はエラー153 になります。</p> <p>補足：コードセクションの境界調整数に1 を指定する場合は特に注意してください。</p>
818	<p>メッセージ：COMMANDLINE OPTION MISMATCH FOR FLOATING DIRECTIVE</p> <p>内容：CPU 各種が SH-2E または SH-3E のとき、コマンドライン・オプション -ROUND = NEAREST または -DENORMALIZE = ON を指定した。</p> <p>対策：コマンドライン・オプション-ROUND または-DENORMALIZE の指定を変更してください。</p> <p>補足：アセンブラは、コマンドライン・オプション-ROUND または-DENORMALIZE の指定どおりにオブジェクトコードを生成します。</p>
825	<p>メッセージ：ILLEGAL INSTRUCTION IN DUMMY SECTION</p> <p>内容：ダミーセクションに、実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を記述した。</p> <p>対策：実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を削除してください。</p> <p>補足：アセンブラは、実行命令、拡張命令、DSP 命令、データを確保するアセンブラ制御命令の記述を無視します。</p>
826	<p>メッセージ：ILLEGAL PRECISION</p> <p>内容：浮動小数点定数の精度がオペレーションサイズで指定した精度と異なる。</p> <p>対策：オペレーションサイズで指定した精度または浮動小数点定数の精度を訂正してください。</p> <p>補足：アセンブラは、オペレーションサイズで指定した精度を有効として処理します。</p>

ソースプログラム全般のウォーニング	
832	<p>メッセージ：MULTIPLE 'P' DEFINITIONS</p> <p>内容：デフォルトセクション名としての P が、他のシンボルである P と重複した。</p> <p>対策：P が重複しないようにしてください。</p> <p>補足：アセンブラは、P をデフォルトセクション名と見なし、他のシンボル P の定義を無効とします。</p>
835	<p>メッセージ：ILLEGAL VALUE IN OPERAND</p> <p>内容：実行命令のオペランドに範囲外の値を指定した。</p> <p>対策：値を訂正してください。</p> <p>補足：アセンブラは、値を範囲内に補正してオブジェクトコードを生成します。</p>
836	<p>メッセージ：ILLEGAL CONSTANT SIZE</p> <p>内容：整数定数の記述の一部に誤りがある。</p> <p>対策：記述を訂正してください。</p> <p>補足：アセンブラが整数定数を誤って（プログラマの意図しない値として）解釈する可能性があります。</p>
837	<p>メッセージ：SOURCE STATEMENT TOO LONG</p> <p>内容：ソースステートメントの 1 行の長さが、255 バイトを超えている。</p> <p>対策：コメント文を短くするなどして、1 行を 255 バイト以内に納めてください。または、ソースステートメントを複数行に分けて記述してください。</p> <p>補足：アセンブラは、256 バイト目を無視し、257 バイト目以降を次の行と見なします。</p>
838	<p>メッセージ：ILLEGAL CHARACTER CODE</p> <p>内容：コメント、文字列以外にシフト JIS または EUC コードを指定した。または、SJIS、EUC コマンドライン・オプションの指定がない。</p> <p>対策：シフト JIS コードまたは EUC コードはコメント、文字列内に指定してください。または、SJIS、EUC コマンドライン・オプションを指定してください。</p>
839	<p>メッセージ：ILLEGAL FIGURE IN OPERAND</p> <p>内容：固定小数点データで、ワードサイズの場合 6 桁以上、ロングワードサイズの場合 11 桁以上のデータを指定しました。</p> <p>対策：余分な桁を削除してください。</p>
840	<p>メッセージ：OPERAND OVERFLOW</p> <p>内容：浮動小数点データがオーバフローした。</p> <p>対策：値を変更してください。</p> <p>補足：正の値の場合、+ に、負の値の場合、- になります。</p>
841	<p>メッセージ：OPERAND UNDERFLOW</p> <p>内容：浮動小数点データがアンダフローした。</p> <p>対策：値を変更してください。</p> <p>補足：正の値の場合、+0 に、負の値の場合、-0 になります。</p>

ソースプログラム全般のウォーニング	
842	<p>メッセージ：OPERAND DENORMALIZED</p> <p>内容：浮動小数点データに、非正規化数を指定した。</p> <p>対策：浮動小数点データを確認してください。</p> <p>補足：アセンブラは、指定どおりにオブジェクトコードを生成します。（非正規化数を設定します。）</p>
850	<p>メッセージ：ILLEGAL SYMBOL DEFINITION</p> <p>内容：ラベルフィールドにシンボルを指定した。</p> <p>対策：シンボルを削除してください。</p>
851	<p>メッセージ：MACRO SERIAL NUMBER OVERFLOW</p> <p>内容：マクロ生成番号が 99,999 を超えた。</p> <p>対策：マクロコールの回数を減らしてください。</p>
852	<p>メッセージ：UNNECESSARY CHARACTER</p> <p>内容：オペランドの終了後に文字がある。</p> <p>対策：オペランドを訂正してください。</p>
854	<p>メッセージ：.AWHILE ABORTED BY .ALIMIT</p> <p>内容：展開回数が.ALIMIT 制御文で設定した上限値に達したため、展開を中断した。</p> <p>対策：繰り返しを展開する条件を見直してください。</p>
870	<p>メッセージ：ILLEGAL DISPLACEMENT VALUE</p> <p>内容：ディスプレースメント値が不当である。</p> <p>（オペレーションサイズがワードのとき、ディスプレースメントが偶数でない。または、オペレーションサイズがロングワードのとき、ディスプレースメントが4の倍数でない。）</p> <p>対策：アセンブラがディスプレースメントを補正することを考慮してください。</p> <p>補足：アセンブラは、オペレーションサイズに応じてディスプレースメントを補正して、オブジェクトコードを生成します。（オペレーションサイズがワードのとき、ディスプレースメントが偶数になるよう切り捨てる。オペレーションサイズがロングワードのとき、ディスプレースメントが4の倍数になるよう切り捨てる。）</p>
871	<p>メッセージ：PC RELATIVE IN DELAY SLOT</p> <p>内容：オペランドがPC 相対アドレス形式である実行命令が、メモリ上で遅延分岐命令の直後に位置している。</p> <p>対策：遅延分岐によってPC 値が変化することを考慮してください。</p> <p>補足：アセンブラは、指定どおりにオブジェクトコードを生成します。</p>

ソースプログラム全般のウォーニング	
874	<p>メッセージ：CANNOT CHECK DATA AREA BOUNDARY</p> <p>内容：PC 相対データ転送命令において、データ領域の境界をチェックできない。</p> <p>対策：リンクする際には、データ領域の境界に十分注意してください。</p> <p>補足：データ転送命令が相対セクションに属している場合や、データ領域が外部参照シンボルである場合などに、アセンブラは本ウォーニングを出力します。</p>
875	<p>メッセージ：CANNOT CHECK DISPLACEMENT SIZE</p> <p>内容：PC 相対データ転送命令において、ディスプレースメントの大きさをチェックできない。</p> <p>対策：リンクする際には、データ転送命令とデータ領域の距離に十分注意してください。</p> <p>補足：データ転送命令が相対セクションに属している場合や、データ領域が外部参照シンボルである場合などに、アセンブラは本ウォーニングを出力します。</p>
876	<p>メッセージ：ASSEMBLER OUTPUTS BRA INSTRUCTION</p> <p>内容：アセンブラが自動的に BRA 命令を出力した。</p> <p>対策：.POOL などを用いてリテラルプールの出力位置を指定するか、アセンブラが自動的に生成した BRA 命令でプログラムが正常に動作するかどうかを確認してください。</p> <p>補足：リテラルプールの出力ポイントが見つからないため、リテラルプールとそれを飛び越すための BRA 命令を自動出力したことを示します。</p>
880	<p>メッセージ：.END NOT FOUND</p> <p>内容：プログラムに.END がない。</p> <p>対策：.END を記述してください</p>
881	<p>メッセージ：ILLEGAL DIRECTIVE IN REPEAT LOOP</p> <p>内容：リピートループ内に不正なアセンブラ制御命令を指定した。</p> <p>対策：不正なアセンブラ制御命令を削除してください。</p> <p>補足：リピートループ内にデータまたはデータ領域を確保する制御命令、.ALIGN 制御命令および.ORG 制御命令を記述した場合、1 制御命令を 1 命令としてリピートする命令数をカウントします。</p>

表 D.3 フェイタルエラーに対するメッセージ

フェイタルエラーに対するメッセージ	
901	<p>メッセージ：SOURCE FILE INPUT ERROR</p> <p>内容：ソースファイルの入力時に、エラーが発生した。</p> <p>対策：ディスクの空き容量を確認してください。ディスク上の不要なファイルを削除するなどして、必要な空き容量を確保してください。</p>
902	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが不足した。（中間語に関する情報を処理できない）</p> <p>対策：プログラムを分割してください。</p>
903	<p>メッセージ：LISTING FILE OUTPUT ERROR</p> <p>内容：リストファイルの出力時に、エラーが発生した。</p> <p>対策：ディスクの空き容量を確認してください。ディスク上の不要なファイルを削除するなどして、必要な空き容量を確保してください。</p>
904	<p>メッセージ：OBJECT FILE OUTPUT ERROR</p> <p>内容：オブジェクトファイルの出力時に、エラーが発生した。</p> <p>対策：ディスクの空き容量を確認してください。ディスク上の不要なファイルを削除するなどして、必要な空き容量を確保してください。</p>
905	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが不足した。（ソースプログラムの行に関する情報を処理できない）</p> <p>対策：プログラムを分割してください。</p>
906	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが不足した。（シンボルに関する情報を処理できない）</p> <p>対策：プログラムを分割してください。</p>
907	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが不足した。（セクションに関する情報を処理できない）</p> <p>対策：プログラムを分割してください。</p>
908	<p>メッセージ：SECTION OVERFLOW</p> <p>内容：セクションの個数が 65,535 個を超えた。</p> <p>対策：プログラムを分割してください。</p>
909	<p>メッセージ：SYMBOL OVERFLOW</p> <p>内容：シンボルの個数が、65,535 個を超えた。</p> <p>対策：プログラムを分割してください。</p>
910	<p>メッセージ：SOURCE LINE NUMBER OVERFLOW</p> <p>内容：ソースプログラムの行数が、65,535 を超えた。</p> <p>対策：プログラムを分割してください。</p>

フェイタルエラーに対するメッセージ	
911	<p>メッセージ：IMPORT SYMBOL OVERFLOW</p> <p>内容：外部参照シンボルの個数が 65,535 を超えた。</p> <p>対策：外部参照シンボルの個数を減らしてください。</p>
912	<p>メッセージ：EXPORT SYMBOL OVERFLOW</p> <p>内容：外部定義シンボルの個数が 65,535 を超えた。</p> <p>対策：外部定義シンボルの個数を減らしてください。</p>
933	<p>メッセージ：ILLEGAL ENVIRONMENT VARIABLE</p> <p>内容：CPU 種別に誤りがある。</p> <p>対策：CPU 種別を訂正してください。</p>
935	<p>メッセージ：SUBCOMMAND FILE INPUT ERROR</p> <p>内容：サブコマンドファイル入力時にエラーが発生した。</p> <p>対策：ディスクの空き容量を確認してください。ディスク上の不要なファイルを削除するなどして、必要な空き容量を確保してください。</p>
936	<p>メッセージ：SUPPLEMENT FILE OUTPUT ERROR</p> <p>内容：付加情報ファイルの出力時に、エラーが発生した。</p> <p>対策：ディスクの空き容量を確認してください。ディスク上の不要なファイルを削除するなどして、必要な空き容量を確保してください。</p>
950	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが足りない。</p> <p>対策：ソースプログラムを分割してください。</p>
951	<p>メッセージ：LITERAL POOL OVERFLOW</p> <p>内容：リテラルプール用の内部シンボルの個数が 100,000 個を超えた。</p> <p>対策：ソースプログラムを分割してください。</p>
952	<p>メッセージ：LITERAL POOL OVERFLOW</p> <p>内容：リテラルプールの容量があふれた。</p> <p>対策：リテラルプールがあふれる前に、無条件分岐を挿入してください。</p>
953	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが足りない。</p> <p>対策：ソースプログラムを分割してください。</p>
954	<p>メッセージ：LOCAL BLOCK NUMBER OVERFLOW</p> <p>内容：ローカルラベルの有効範囲であるローカルブロックの個数が 100,000 個を超えた。</p> <p>対策：ソースプログラムを分割してください。</p>
956	<p>メッセージ：EXPAND FILE INPUT/OUTPUT ERROR</p> <p>内容：プリプロセッサ展開出力のファイル出力時にエラーが発生した。</p> <p>対策：ディスクの空き容量を確認してください。ディスク上に不必要なファイルを削除するなどして、必要な空き容量を確保してください。</p>
957	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが足りない。</p> <p>対策：ソースプログラムを分割してください。</p>
958	<p>メッセージ：MEMORY OVERFLOW</p> <p>内容：メモリが足りない。</p> <p>対策：ソースプログラムを分割してください。</p>

付録 E. 旧バージョンとの相違

新バージョン Ver.4.0 と旧バージョン Ver.3.1 の相違を示します。

E.1 CPU

新バージョンでは、SH-1、SH-2、SH-3、SH-3E、SH-DSP のソースプログラムのアセンブル機能の他に、SH-2E、SH-4、SH3-DSP のソースプログラムのアセンブル機能が追加されました。

SH-2E、SH-4、SH3-DSP のアセンブル機能の追加により、次の項目が追加および変更されました。

- ・ 予約語
- ・ 実行命令
- ・ .CPU アセンブラ制御命令
- ・ CPU コマンドライン・オプション
- ・ SHCPU 環境変数

【参照】 予約語 言語編「1.2 予約語」
実行命令 言語編「3 実行命令」
.CPU アセンブラ制御命令
 言語編「5.2.1 CPU に関するアセンブラ制御命令」
CPU コマンドラインオプション
 操作編「2.2.1 CPU コマンドライン・オプション」
SHCPU 環境変数 操作編「1.3 SHCPU 環境変数」

E.2 オブジェクトフォーマット

ソフトウェア開発サポートツールでは、旧バージョンまでのオブジェクトフォーマットに加え、ELF/DWARF フォーマットをサポートしました。

本アセンブラは、デバッグ情報を出力する場合に ELF/DWARF 付加情報を出力します。

【参照】 デバッグ情報
 言語編「5.2.6 オブジェクトモジュールに関するアセンブラ制御命令」 .OUTPUT
 操作編「2.2.2 オブジェクトモジュールに関するコマンドライン・オプション」 -DEBUG

E.3 定数

新バージョンでは、浮動小数点定数の単精度のほかに、倍精度をサポートしました。

・浮動小数点定数の倍精度は、SH-4 で使用します。

【参照】 浮動小数点定数 言語編「1.4.3 浮動小数点定数」

E.4 アセンブラ制御命令の変更

新バージョンでは、浮動小数点データに関するアセンブラ制御命令に倍精度のレジスタ、オペレーションサイズを追加しました。これを表 E.1 に示します。

表 E.1 制御命令の変更

制御命令、制御文	変更点	参照
.FREG	浮動小数点レジスタ名の種別を追加	言語編 5.2.3
.FDATA	オペレーションサイズに倍精度を追加	言語編 5.2.4
.FDATAB	オペレーションサイズに倍精度を追加	言語編 5.2.4
.FRES	オペレーションサイズに倍精度を追加	言語編 5.2.4

E.5 コマンドライン・オプションの追加

新バージョンで追加したコマンドライン・オプションを表 E.2 に示します。

表 E.2 コマンドライン・オプションの追加

コマンドライン・オプション	機能	参照
ROUND	浮動小数点データの丸め方法を設定	操作編 2.2.10
DENORMALIZE	浮動小数点の非正規化数の扱いを設定	操作編 2.2.10

付録 F. ASCII コード表

表 F.1 ASCII コード表

下位 4 ビット	上位 4 ビット							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

索引

五十音索引

あ行

アスキーコード表.....	372
アセンブラ.....	5, 9
アセンブラ制御命令.....	15, 115
アセンブリ言語ソースプログラム.....	5, 9
アセンブルリスト.....	5
アセンブルリストに関するアセンブラ制御命令	185
アセンブルリストに関するコマンドライン・オプション	305
アセンブルリストの内容（出力例）	342
アセンブルリストの出力制御.....	186, 306
アセンブルリストの 1 ページあたりの行数設定	192, 312
アセンブルリストの 1 行あたりの桁数設定.....	192, 313
アドレス計算.....	93
アドレス形式.....	68
実行命令のニーモニックおよびオペレーションサイズとの組み合わせ	74
アドレスシンボル.....	20
絶対アドレスシンボル.....	57
相対アドレスシンボル.....	57
アンダフロー境界.....	26
位置指定	257
イミディエイト	69, 72
インクリメント	69
インクルードファイル	209, 210
インクルードファイル取り込み先の指定	315
インクルードのネスト	210
インデックス付き GBR 間接	68
インデックス付きレジスタ間接.....	68
英大文字と英小文字の区別.....	337
SH-DSP 命令.....	99
エミュレータ.....	10
エディタ	9
エラーレベルの変更.....	291

演算（式）	36
演算の順序	38
演算の内部処理	44
演算に関する制限	44
演算子	36
予約語である演算子	19
演算子の優先順位	38
演算子の結合規則	38
エンディアン種別	180, 304
エンディアン種別の指定	180, 304
オーバーフロー境界	26
オブジェクトコード	34
ディレイスロット命令のオブジェクトコード	92
オブジェクトコンバータ	9
オブジェクトファイル	292
オブジェクトファイルの指定	302
オブジェクトモジュール	5, 9
オブジェクトモジュールに関するアセンブラ制御命令	174
オブジェクトモジュールに関するコマンドライン・オプション	301
オブジェクトモジュールの出力制御	175, 302
オブジェクトモジュール名	201
オペランド	16, 17, 67
オペレーション	15, 17, 67
オペレーションサイズ	67
実行命令のニーモニックおよびアドレス形式との組み合わせ	74
オペレーティングシステム	291
オペレーティングシステムへのリターンコード	291

か行

開始アドレス	280
（ソースプログラム・リストの）改ページ	196
拡張命令	75, 266, 280
外部	60
外部参照	61, 64
外部参照に関するアセンブラ制御命令	167
外部参照シンボル	57
外部参照シンボルの宣言	64, 170, 172

外部参照値	57
外部定義	64
外部定義に関するアセンブラ制御命令	167
外部定義シンボル	64
外部定義シンボルの宣言	64, 168, 172
カッコ	38
仮引数	246, 249, 253
仮引数のデフォルト	249
環境変数	293
SHCPU 環境変数	293
漢字	16, 24, 45, 324, 325, 326
キーワード指定	257
基数	23
基数のない整数定数	203
起動	291
境界調整（ロケーションカウンタ値の補正）	34, 127
セクションの境界調整	121
行	
アセンブル行数	337
1 行の最大長	16, 337
行の継続	18
行番号	183
行番号の変更	183
繰り返し展開	219, 233
クロスリファレンス・リスト	345
クロスリファレンス・リストの出力制御	186, 308
計数付き文字列	
計数付き文字列データの確保	147
計数付き文字列データ領域の確保	161
コード	
シフト JIS コード	16, 24, 45, 325, 327
EUC コード	16, 24, 45, 326, 327
コードセクション	51
コードセクションの宣言	121
項	36
後方	60
後方参照	61
固定小数点定数	31

固定小数点データ	32
固定小数点データの確保	155
コマンドライン	291
コマンドライン・オプション	297
コメント	67
複数行にわたるソースステートメントのコメント	18
コモンセクション	52
コモンセクションの宣言	121
コンパイラ	9

さ行

再アセンブル	62
サブコマンドファイル	331
算術演算命令	76, 80, 81, 84, 87
サンプルプログラム	338
式	36
式の要素	36
式の演算順序	38
式の内部処理	44
式に関する注意事項	44
指数部	25
システム制御命令	79, 82, 83, 85, 88, 90
CPU 種別	118, 293, 300
シフト命令	77, 82
シミュレーションの開始アドレス（実行開始アドレス）	205
シミュレータ・デバッガ	9
終了アドレス	280
主ファイル名	292
小数部	25, 28, 29, 32
条件つきアセンブリ機能	215
比較型条件つきアセンブル	216, 229
定義型条件つきアセンブル	217, 231
条件つきアセンブリ機能に関する制御文	221
条件つき繰り返し展開	220, 235
条件つき繰り返し展開の上限値の設定	241

シンボル	15, 20
シンボルの参照	60
シンボルの定義	58
シンボルの名づけ方	22
シンボルの役割	20
シンボルデバッグ情報	178
シンボルデバッグ情報の部分出力	178
実行開始アドレス	205
実行命令	67
実行命令の一覧	74
スタックセクション	53
スタックセクションに関する注意	54
スタックセクションの宣言	121
スタックポインタ (SP)	19
正規化数	26, 27, 29
(文字列につけ加える) 制御文字	143, 146, 147, 149
整数定数	23
基数のない整数定数	203
整数データの確保	138
整数データブロックの確保	140
整数部	25
セクション	38, 51, 121
セクションの開始と再開	122, 123
セクション属性 (用途別の種類)	121
セクションのメモリ上での位置	55
セクション集合	38
セクション集合演算	37
セクション名	20, 58, 121
セクション情報リスト	346
セクション情報リストの出力制御	309
絶対アドレス	55
絶対アドレスシンボル	57
絶対アドレスセクション	55
絶対アドレスセクションの宣言	123
絶対アドレス値	57, 125
絶対値	57
絶対パス	210, 211
ゼロ	26, 29

ゼロ終端文字列

ゼロ終端文字列データの確保	149
ゼロ終端文字列データ領域の確保	163
ゼロ除算	44
前方	60
前方参照	60
ソースステートメント	15
ソースステートメントの構成	15
ソースステートメントの書き方	16
複数行にわたるソースステートメントの書き方	18, 103
ソースファイル	291, 292
入力ソースファイル	291
複数のソースファイルのアセンブル	62, 291
ソースプログラム	5, 9
ソースプログラム・リスト	342
ソースプログラム・リストの改ページ	196
ソースプログラム・リストの出力制御	186, 306
ソースプログラム・リストの部分出力制御	188
ソースプログラム・リストのヘッダ設定	194
ソースプログラム・リストへの空行出力	198
相対アドレス	55
相対アドレスシンボル	57
相対アドレスセクション	55
相対アドレスセクションの宣言	124
相対アドレス値	57, 125
相対値	57
相対パス	210, 211

た行

ダミーセクション	53
ダミーセクションの宣言	121
ダミーセクションに関する注意	54
ダミーセクションを使ったデータ構造記述	53
単精度	25, 26, 27, 28, 29, 67
遅延分岐命令	91
置換シンボル	216, 227
DSP 命令	99, 104

DSP 演算命令	104
定数	23
定数シンボル	57
定数値	57
データ構造	53
データセクション	52
データセクションの宣言	121
データ転送命令	74, 81, 82, 84, 86, 102, 108
シングルデータメモリ転送命令	102, 111
X データメモリ転送命令	102, 111
Y データメモリ転送命令	102, 111
データ領域の確保	157
ディスプレースメント	68
ディスプレースメントの補正	72
ディスプレースメント付き G B R 間接	68
ディスプレースメント付きレジスタ間接	68
ディレイスロット命令	91
デクリメント	69
デバッグ情報	177
デバッグ情報の出力制御	175
シンボルデバッグ情報の部分出力制御	178
デバッグ付加情報 (ELF/DWARF 付加情報)	176, 303

な行

内部シンボル	22
ニーモニック	15, 67, 337
アセンブラ制御命令のニーモニック	115
実行命令のニーモニック	74

は行

バイアス	28
バイアス値	28
倍精度	25, 26, 27, 28, 29, 67
非正規化数	26, 27, 29, 30, 334
非数	26, 29
ファイルインクルード機能	209

ファイルの指定形式.....	292
ファイル型.....	292
ファイル名.....	292
符号ビット.....	28, 32
浮動小数点定数.....	25
浮動小数点データ.....	28
浮動小数点データの確保.....	151
浮動小数点データブロックの確保.....	153
浮動小数点レジスタ名の定義.....	135
浮動小数点データ領域の確保.....	165
プリデクリメント・レジスタ間接.....	68
プリプロセッサ変数.....	215
プリプロセッサ置換文字列の定義.....	227, 320
整数型プリプロセッサ変数の定義.....	222
文字型プリプロセッサ変数の定義.....	225
プリプロセッサ展開	
プリプロセッサ展開結果の出力.....	322
プリプロセッサ展開時のエラー処理.....	237
プログラムカウンタ (PC).....	19
プログラムカウンタの値.....	93
分割プログラミング.....	62
分岐命令.....	78, 80
遅延分岐命令.....	91
並列演算命令.....	100
(ソースプログラム・リストの) ヘッダ.....	194
ポストインクリメント・レジスタ間接.....	68

ま行

マクロ.....	245
マクロ機能.....	245
マクロ機能に関する制御文.....	248
マクロコール.....	257
マクロ処理除外.....	255
マクロ生成番号.....	254
マクロ定義.....	245, 246, 249
マクロ内コメント.....	256
マクロパラメータ.....	246, 247

マクロ本体.....	253
マクロ名.....	249
丸め.....	29, 333
無限大.....	26, 29
文字定数.....	24
文字列.....	45
計数付き文字列データの確保.....	147
計数付き文字列データ領域の確保.....	161
ゼロ終端文字列データの確保.....	149
ゼロ終端文字列データ領域の確保.....	163
文字列データの確保.....	143
文字列データブロックの確保.....	145
文字列データ領域の確保.....	159
文字列操作関数.....	256, 260, 261, 262

や行

優先順位.....	39
演算の優先順位.....	38
演算子の優先順位.....	38
予約語.....	19

ら行

ライブラリアン.....	9, 10
ライブラリファイル.....	9, 10
ラベル.....	15
リストファイル.....	292
リストファイルの指定.....	306
リテラルプール自動生成機能.....	265
リテラルプール自動生成機能に関する拡張命令.....	266
リテラルプール自動生成機能のサイズモード.....	267
(OSへの)リターンコード.....	291
リピート回数.....	280
リピート制御命令.....	89
リピートループ命令自動生成機能.....	279
リピートループ命令自動生成機能に関する拡張命令.....	280
リンケージエディタ.....	9, 10

レジスタ間接.....	68
レジスタ直接.....	68
レジスタ名（予約語）.....	19
レジスタ別名の定義.....	133
ローカルラベル.....	46
ロードモジュール.....	9, 10
Sタイプ形式ロードモジュール.....	9, 10
ロケーション.....	34, 55
ロケーションカウンタ.....	19, 34
ロケーションカウンタ値.....	34
ロケーションカウンタ値の設定.....	125
ロケーションカウンタ値の補正.....	127
論理演算命令.....	77

英数字索引

A0.....	19, 90, 105, 110
A1.....	19, 90, 105, 110
A0G	19, 110
A1G	19, 110
-ABORT.....	323
ADD.....	72, 76
ADDC	76
ADDV	76
.AELIF	229
.AELSE	229, 231
.AENDL.....	229, 231
.AENDR	233
.AENDW.....	235
.AERROR.....	237
.AIF	229
.AIFDEF.....	231
.ALIGN.....	127
.ALIMIT	241
AND.....	72, 77
.AREPEAT.....	233
As.....	110
ASCII.....	372
.ASSIGN.....	131
.ASSIGNA	222
-ASSIGNA	317
.ASSIGNC	225
-ASSIGNC	318
-AUTO_LITERAL	329
.AWHILE.....	235
Ax.....	110
Ay.....	110
Az.....	109, 110
B'.....	23
BF.....	78, 91
BF/S.....	80, 91

BRA.....	78, 91
BRAF.....	80, 91
BSR.....	78, 91
BSRF.....	80, 91
BT.....	78, 91
BT/S.....	80, 91
CLRMAC.....	79
CLRS.....	83
CLRT.....	79
CMP/EQ.....	72, 76
CMP/GE.....	76
CMP/GT.....	76
CMP/HL.....	76
CMP/HS.....	76
CMP/PL.....	76
CMP/PZ.....	76
CMP/STR.....	76
-COLUMNS.....	313
.CPU.....	118
-CPU.....	300
-CROSS_REFERENCE.....	308
D'.....	23
Da.....	110
.DATA.....	138
.DATAB.....	140
DBR.....	19, 88
DCF.....	100, 106
DCT.....	100, 106
.DEBUG.....	178
-DEBUG.....	303
.DEFINE.....	227
-DEFINE.....	320
-DENORMALIZE.....	334
Dg.....	105
disp.....	68, 70, 75
DIV0S.....	76
DIV0U.....	76
DIV1.....	76

DMULS	80
DMULU	80
Dp	105
Ds	110
Du	105
DRn	19, 88
DSR	19, 90
DT	80
DWARF	176, 303
Dx	110
Dy	110
Dz	105, 110
ELF	176, 303
.ENDIAN	180
-ENDIAN	304
.END	205
.ENDM	249
.EQU	130
EUC	326, 327
-EUC	326
.EXITM	239, 252
-EXPAND	322
.EXPORT	168
EXTS	76
EXTU	76
FABS	81, 84, 87
FADD	81, 84, 87
FCMP/EG	81, 84, 87
FCMP/GT	81, 84, 87
FCNVDS	88
FCNVSD	88
.FDATA	151
.FDATAB	153
FDIV	81, 84, 87
FIPR	87
FLDI0	81, 84
FLDI1	81, 84
FLDS	82, 85

FLOAT.....	82, 85, 88
FMAC.....	81, 84
FMOV.....	81, 84, 86
FMUL.....	81, 84, 87
FNEG.....	81, 84, 87
.FORM.....	192
FPUL.....	19, 82, 85
FPSCR.....	19, 82, 85
FRn.....	19, 82, 85
FRCHG.....	88
.FREG.....	135
FSCHG.....	88
FSQRT.....	84
FSTS.....	82, 85
FSUB.....	81, 84, 87
FTRC.....	82, 85, 88
FTRV.....	87
FVn.....	19, 88
GBR.....	19, 68, 75
.GLOBAL.....	172
H'.....	23
.HEADING.....	194
.INCLUDE.....	210
-INCLUDE.....	315
.IMPORT.....	170
#imm.....	75, 266
.INSTR.....	261
Is.....	110
Ix.....	110
Iy.....	110
Iz.....	109, 110
JMP.....	78, 91
JSR.....	78, 91
LDC.....	79, 83, 88, 90, 91
LDRE.....	89, 280
LDRS.....	89, 280
LDS.....	79, 82, 85, 90
LDTLB.....	83

.LEN	260
.LINE	183
-LINES	312
.LIST.....	188
-LIST.....	306
M0.....	19, 105, 110
M1.....	19, 105, 110
MAC.....	76, 80
MACH.....	19, 75
MACL.....	19, 75
.MACRO	249
MOD.....	19, 90
MOV	72, 74, 91, 265
MOVA.....	74, 91, 265
MOVS.....	99, 108, 111
MOVT.....	74
MOVX.....	99, 108, 111
MOVY.....	99, 108, 111
MUL	80
MULS	76
MULU.....	76
NEAREST.....	29, 333
NEG.....	76
NEGC.....	76
-NOCROSS_REFERENCE.....	308
-NODEBUG	303
-NOLIST	306
-NOOBJECT	302
NOP.....	79
.NOPOOL	273, 274
NOPX.....	99, 108, 111
NOPY.....	99, 108, 111
-NOSECTION.....	309
-NOSHOW	310
-NOSOURCE.....	307
NOT.....	77
-OBJECT.....	302
OCBI.....	88

OCBP.....	88
OCBWB.....	88
OR.....	77
.ORG.....	125
-OUTCODE.....	327
.OUTPUT.....	175
PABS	99, 104, 107
PADD.....	99, 104, 107
PADDC	99, 104, 107
.PAGE.....	196
PAND.....	99, 104, 107
PC	19, 68, 70, 75
.POOL.....	269, 271, 274
PCLR.....	99, 104, 107
PCMP	99, 104, 107
PCOPY.....	99, 104, 107
PDEC	99, 104, 107
PDMSB	99, 104, 107
PINC	99, 104, 107
PLDS	99, 104, 107
PMULS	99, 104, 107
PNEG	99, 104, 107
POR.....	99, 104, 107
PR	19, 75
PREF	82
.PRINT.....	186
PRND.....	99, 104, 107
.PROGRAM.....	201
PSHA.....	99, 104, 105, 107
PSHL	99, 104, 105, 107
PSTS.....	99, 104, 107
PSUB	99, 104, 107
PSUBC.....	99, 104, 107
PXOR.....	99, 104, 107
qNAN	26
Q'.....	23
.RADIX	203
RE	19, 90, 279

.REG.....	133
REPEAT.....	280
.RES	157
ROTCL.....	77
ROTCR	77
ROTL.....	77
ROTR.....	77
-ROUND.....	333
RS	19, 90, 279
Rn.....	19, 68, 70, 75
Rn_BANK.....	19
RTE	79, 91
RTS	78, 91
R0.....	19, 68, 75
R15.....	19
.SDATA.....	143
.SDATAB.....	145
.SDATAC	147
.SDATAZ.....	149
Se	105
.SECTION.....	121
-SECTION.....	309
SETRC	72, 89, 280
SETS.....	83
SETT.....	79
Sf.....	105
SGR.....	19, 88
SH1.....	118, 293, 300
SH-1.....	118, 300, 333, 334
SH2.....	118, 293, 300
SH-2.....	118, 300, 333, 334
SH2E.....	118, 293, 300
SH-2E.....	118, 300, 333, 334
SH3.....	118, 293, 300
SH-3.....	118, 300, 333, 334
SH3DSP	118, 293, 300
SH3-DSP	118, 300, 333, 334
SH3E.....	118, 293, 300

SH-3E.....	118, 300, 333, 334
SH4.....	118, 293, 300
SH-4.....	118, 300, 333, 334
SHAD.....	82
SHAL.....	77
SHAR.....	77
SHDSP.....	118, 293, 300
SH-DSP.....	118, 300, 333, 334
SHLD.....	82
SHLL.....	77
SHLL2.....	77
SHLL8.....	77
SHLL16.....	77
SHLR.....	77
SHLR2.....	77
SHLR8.....	77
SHLR16.....	77
-SHOW.....	310
SJIS.....	325, 327
-SJIS.....	325
SLEEP.....	79
sNAN.....	26
-SOURCE.....	307
SP.....	19
SPC.....	19, 83
.SPACE.....	198
SR.....	19, 75
.SRES.....	159
.SRESC.....	161
.SRESZ.....	163
SSR.....	19, 83
STC.....	79, 83, 88, 90
STS.....	79, 82, 85, 90
SUB.....	76
SUBC.....	76
-SUBCOMMAND.....	331
.SUBSTR.....	262
SUBV.....	76

SWAP	74
Sx	105
Sy	105
symbol	71, 75
TAS	77
TRAPA	72, 79, 91
TST	77
VBR	19, 75
XOR	19, 77
XTRCT	74
X0	19, 90, 105, 110
X1	19, 90, 105, 110
XDn	19, 88
.XDATA	155
XMTRX	19, 88
Y0	19, 90, 105, 110
Y1	19, 90, 105, 110
ZERO	29, 333
0 除算	44
2 進数	23, 203
8 進数	23, 203
1 0 進数	23, 203
1 6 進数	23, 203