ユーザーマニュアル

Tatara・Tanren

ハビエル　ラバンデイラ

バージョン 1.2.0

2026

https://tatara.tools

Tatara・Tanren ユーザーマニュアル
バージョン 1.2.0、2026 年
https://tatara.tools

Copyright © 2026 ハビエル　ラバンデイラ

本書は、クリエイティブ・コモンズ 表示 4.0 国際 ライセンス（CC BY 4.0）の下で提供されています：https://creativecommons.org/licenses/by/4.0/deed.ja。本書に含まれるサンプル・プログラムは、Apache License, Version 2.0 の下で提供されています：https://www.apache.org/licenses/LICENSE-2.0。

MSX は、MSX Licensing Corporation の商標です。

MACRO-80（M80）および LINK-80（L80）は、Microsoft Corporation の製品です。
Microsoft は、Microsoft グループの企業の商標です。

アセンブラー AS およびリンカー LD の著作権は、Egor Voznesensky に帰属します。

Nextor の著作権は、Néstor Soriano に帰属します。

CP/M、MSX-DOS、Z80、R800 をはじめ、本書に記載されているその他の製品名は、各社の商標または登録商標です。本書ではこれらの名前を製品を特定するためにのみ使用しており、各社との提携や各社による推奨を意味するものではありません。

目次

このマニュアルについて **xiii**

I はじめの一歩 **1**

1 はじめに **3**
1.1 Tatara と Tanren とは . . . 3
1.2 名前の由来 . . . 4
1.3 できること . . . 4
1.4 必要なもの . . . 6
1.5 次に読むところ . . . 6

2 アセンブラーとリンカーのしくみ **7**
2.1 2 つの段階 . . . 7
2.2 アセンブラーがすること . . . 8
2.3 アセンブラーにわからないこと . . . 9
2.4 リンカーがすること . . . 10
2.5 実例 . . . 10
2.6 コードとデータ：セグメント . . . 14
2.7 2 つの段階に分ける利点 . . . 14
2.8 この章で出てきた用語 . . . 14

3 インストール **17**
3.1 配布されるもの . . . 17
3.2 ファイルの置き場所 . . . 18
3.3 インストールの手順 . . . 19
3.4 PATH と TATARA の設定 . . . 20
3.5 インストールの確認 . . . 21
3.6 アップデート . . . 22

4 最初のプログラム **23**
4.1 始める前に . . . 23
4.2 プログラム . . . 24
4.3 アセンブルする . . . 25
4.4 リンクする . . . 26
4.5 実行する . . . 27

4.6 ビルド・ファイル . . . 27
4.7 2 つのモジュールからなるプログラム . . . 27
4.8 次に読む章 . . . 29

II アセンブラー Tatara 31

5 TATARA の実行 33
5.1 コマンド・ライン . . . 33
5.2 ファイル名 . . . 34
5.3 オプション . . . 36
5.4 TATARA が表示するもの . . . 37
5.5 リスティング . . . 39
5.6 TATARA が実行を始めないとき . . . 40

6 ソースの書式 41
6.1 行 . . . 41
6.2 4 つのフィールド . . . 42
6.3 ラベルを書く場所 . . . 42
6.4 コメント . . . 44
6.5 大文字と小文字 . . . 44
6.6 スペース、タブ、行の長さ、行末 . . . 45

7 名前 47
7.1 名前に使える文字 . . . 47
7.2 長さ . . . 48
7.3 大文字と小文字 . . . 49
7.4 届きにくい名前 . . . 51

8 式 55
8.1 値は 16 ビット . . . 55
8.2 数 . . . 56
8.3 文字 . . . 56
8.4 名前とロケーション・カウンター . . . 57
8.5 演算子 . . . 57
8.6 値の種類 . . . 58

9 命令 61
9.1 命令セット . . . 61
9.2 オペランド . . . 62
9.3 相対ジャンプ . . . 63
9.4 R800 の命令 . . . 64
9.5 未公開命令 . . . 65
9.6 オペランドが正しくないとき . . . 68

10 セグメントとロケーション・カウンター 69
10.1 ロケーション・カウンター . . . 69

10.2 コード、データ、アブソリュート . . . 70
10.3 ORG . . . 71
10.4 領域の確保：DS . . . 72
10.5 名前付きセグメント . . . 72
10.6 トランジェント・データ・セグメント . . . 73
10.7 メッセージ . . . 76

11 データ 77
11.1 DB：バイト . . . 77
11.2 文字列 . . . 78
11.3 DW：ワード . . . 78
11.4 DC：自分で終わりを示す文字列 . . . 79
11.5 別の名前 . . . 80
11.6 メッセージ . . . 80

12 シンボルの定義と共有 81
12.1 名前に値を与える 3 つの方法 . . . 81
12.2 ラベル . . . 82
12.3 EQU . . . 82
12.4 DEFL . . . 83
12.5 モジュールどうしで名前を共有する . . . 83
12.6 END と開始アドレス . . . 85
12.7 メッセージ . . . 86

13 ファイルの取り込み 89
13.1 INCLUDE . . . 89
13.2 TATARA が探す場所 . . . 91
13.3 環境変数 TATARA . . . 92
13.4 ファイル名の中のパス . . . 92
13.5 ファイルを取り込むファイル . . . 93
13.6 同じファイルを 2 回取り込む . . . 93
13.7 インクルード・ファイルの中の END . . . 94
13.8 メッセージ . . . 94

14 インクルード・ファイル 97
14.1 ファイルの中身 . . . 97
14.2 取り込み方 . . . 98
14.3 MSXDOS.INC と system マクロ . . . 99
14.4 BIOS.INC と SUBROM.INC . . . 100
14.5 ERRORS.INC . . . 101
14.6 そのほかのファイル . . . 102

15 条件付きアセンブル 105
15.1 IF、ELSE、ENDIF . . . 105
15.2 2 つのパス . . . 107
15.3 IFDEF と IFNDEF . . . 108

15.4 IF1 と IF2 . . . 109
15.5 IFB、IFNB、IFIDN、IFDIF . . . 109
15.6 入れ子 . . . 110
15.7 メッセージ . . . 111

16 マクロ 113
16.1 マクロの定義と使用 . . . 114
16.2 引数 . . . 115
16.3 &によるテキストの連結 . . . 116
16.4 本体の中のコメント . . . 117
16.5 LOCAL . . . 117
16.6 EXITM . . . 118
16.7 マクロの中のマクロ . . . 119
16.8 大文字と小文字 . . . 120
16.9 メッセージ . . . 120

17 繰り返しブロック 123
17.1 REPT . . . 123
17.2 IRP . . . 124
17.3 IRPC . . . 126
17.4 LOCAL、EXITM、入れ子 . . . 126
17.5 メッセージ . . . 127

18 リスティング 129
18.1 リスティングの 1 行 . . . 130
18.2 ページ . . . 131
18.3 リスティングを止める、再開する . . . 134
18.4 マクロの展開 . . . 135
18.5 アセンブルされない行 . . . 136
18.6 メッセージ . . . 137

19 シンボル表の出力と診断用スイッチ 139
19.1 シンボル表 . . . 139
19.2 シンボル表の読み方 . . . 141
19.3 シンボル表とリスティング . . . 144
19.4 診断用スイッチ . . . 144

20 エラーメッセージ 149
20.1 メッセージの内容 . . . 149
20.2 TATARA は最初のエラーで止まる . . . 151
20.3 経路 . . . 152
20.4 第 2 パスで見つかるエラー . . . 155
20.5 内部エラー . . . 155

III リンカー Tanren 157

21 TANREN の実行 159
21.1 コマンド・ライン . . . 159
21.2 出力ファイル . . . 162
21.3 オプション . . . 162
21.4 TANREN が表示するもの . . . 163
21.5 127 文字のコマンド・ライン . . . 165
21.6 TANREN が始まらないとき . . . 165

22 リンク・ファイル 167
22.1 最初のリンク・ファイル . . . 167
22.2 リンク・ファイルの書き方 . . . 168
22.3 リンク・ファイルとコマンド・ライン . . . 169
22.4 メッセージ . . . 170

23 TANREN がオブジェクト・ファイルを探す場所 173
23.1 3 つの場所 . . . 173
23.2 環境変数 TANREN . . . 175
23.3 名前の中のパス . . . 176
23.4 出力ファイルの書き出し先 . . . 176
23.5 メッセージ . . . 177

24 プログラムの配置 179
24.1 コードが先、データがあと . . . 179
24.2 名前付きセグメント . . . 182
24.3 コードとデータを動かす：/P: と /D: . . . 182
24.4 モジュールをまたぐトランジェント・セグメント . . . 184
24.5 アブソリュート・セグメント . . . 185
24.6 メッセージ . . . 186

25 モジュール間のシンボル 187
25.1 名前の対応のさせかた . . . 187
25.2 外部の名前の値 . . . 189
25.3 どのモジュールも定義していない名前 . . . 190
25.4 2 回定義された名前 . . . 192
25.5 大文字と小文字 . . . 192
25.6 すべてのモジュールがリンクされる . . . 194
25.7 /M の表示のすべて . . . 194
25.8 メッセージ . . . 195

26 出力形式 197
26.1 TANREN が書き出すもの . . . 197
26.2 .COM のプログラム . . . 198
26.3 ほかのアドレス用のイメージ . . . 198
26.4 BLOAD 用のファイル: /B . . . 199

26.5 エントリー・アドレス . . . 201
26.6 ROM イメージ . . . 202
26.7 プログラムの大きさの限度 . . . 204
26.8 書き出すものがないとき . . . 205
26.9 メッセージ . . . 205

27 リンカーのエラーメッセージ 207
27.1 メッセージの形 . . . 207
27.2 TANREN は最初のエラーで止まる . . . 208
27.3 壊れたオブジェクト・ファイル . . . 209
27.4 セグメント . . . 210
27.5 メッセージ . . . 212

IV 実例 213

28 複数のモジュールからなるプログラム 215
28.1 この例 . . . 215
28.2 2 つのモジュール . . . 216
28.3 リンク・ファイル . . . 217
28.4 作って実行する . . . 218
28.5 TANREN がしたこと . . . 220
28.6 試してみること . . . 220

29 トランジェント・セグメントで共有する作業用変数 223
29.1 この例 . . . 223
29.2 ソース . . . 224
29.3 作る . . . 225
29.4 シンボル表を読む . . . 226
29.5 マップを読む . . . 227
29.6 試してみること . . . 229

30 マクロとリスティングのモード 233
30.1 この例 . . . 233
30.2 ソース . . . 234
30.3 作って実行する . . . 235
30.4 3 つのリスティングのモード . . . 237
30.5 試してみること . . . 240

31 長いシンボル名と大文字・小文字を区別する名前 243
31.1 この例 . . . 243
31.2 ソース . . . 244
31.3 作る . . . 245
31.4 2 つのファイルを読む . . . 246
31.5 モジュールをまたぐ長いシンボル名 . . . 247
31.6 マクロの名前と /C . . . 248

32 MSX-BASIC の BLOAD 用のバイナリ 251
32.1 この例 . . . 251
32.2 ソース . . . 252
32.3 作る . . . 253
32.4 BASIC から実行する . . . 254
32.5 ファイルを置ける場所 . . . 254
32.6 試してみること . . . 256

33 16 KB の ROM カートリッジ 259
33.1 この例 . . . 259
33.2 ソース . . . 260
33.3 作る . . . 262
33.4 実行する . . . 263
33.5 試してみること . . . 263

34 複数のディレクトリにまたがるプロジェクト 267
34.1 この例 . . . 267
34.2 ソース . . . 268
34.3 作って実行する . . . 270
34.4 試してみること . . . 271

35 TATARA と TANREN を自身のソースからビルドする 275
35.1 ソース . . . 275
35.2 BUILD.BAT . . . 276
35.3 実行する . . . 279
35.4 ツール自身でツールをビルドする . . . 280

A 疑似命令リファレンス 283
A.1 用途ごとの疑似命令 . . . 283
A.2 アルファベット順の疑似命令 . . . 286
A.3 疑似命令の行のラベル . . . 292
A.4 別名 . . . 293

B 命令一覧 295
B.1 表の読み方 . . . 296
B.2 8 ビットのロード . . . 297
B.3 16 ビットのロード、PUSH と POP . . . 298
B.4 交換、ブロック転送、サーチ . . . 299
B.5 8 ビットの算術演算と論理演算 . . . 300
B.6 汎用と CPU 制御 . . . 301
B.7 16 ビットの算術演算 . . . 302
B.8 ローテートとシフト . . . 303
B.9 ビットのセット、リセット、テスト . . . 305
B.10 ジャンプ . . . 305
B.11 コール、リターン、リスタート . . . 306
B.12 入出力 . . . 307

B.13 未公開命令 . . . 308
B.14 R800 のかけ算 . . . 309
B.15 受け付けない書き方 . . . 309

C 演算子と優先順位 311
C.1 演算子 . . . 311
C.2 範囲の端 . . . 313
C.3 値 . . . 313

D エラーメッセージ 315
D.1 メッセージの形 . . . 315
D.2 TATARA のメッセージ . . . 316
D.3 TANREN のメッセージ . . . 320
D.4 TANREN の警告とお知らせ . . . 322

E 制限値 325
E.1 TATARA . . . 325
E.2 TANREN . . . 326
E.3 コマンド・ライン . . . 327
E.4 決まった制限のないもの . . . 328

F `.tro` オブジェクト・ファイルの形式 329
F.1 オブジェクト・ファイルが伝えること . . . 329
F.2 例 . . . 330
F.3 ファイルの構成 . . . 333
F.4 レコード . . . 334
F.5 TANREN がオブジェクト・ファイルを読む方法 . . . 337

G メモリ 339
G.1 メモリ・マップ . . . 339
G.2 マッパーに置くもの . . . 340
G.3 プログラムを作るのにツールが必要とするメモリの量 . . . 340
G.4 マッパーがいっぱいになったとき . . . 342

H ハードウェアの入手先 343
H.1 ツールに必要なもの . . . 343
H.2 カートリッジ . . . 344
H.3 メーカーと購入先 . . . 345
H.4 入手のしやすさ . . . 346

I 環境変数 347
I.1 環境変数の設定 . . . 347
I.2 PATH . . . 348
I.3 TATARA . . . 348
I.4 TANREN . . . 349
I.5 両方に共通の規則 . . . 350

J M80 とのソースの互換性 353
J.1 そのまま使えるもの . . . 353
J.2 TATARA が受け付けない M80 の疑似命令 . . . 354
J.3 そのほかの書き方 . . . 355
J.4 TATARA が加えるもの . . . 356
J.5 そのほかの違い . . . 357
J.6 ソースの変換 . . . 357

K インクルード・ファイル 359
K.1 MSXDOS.INC . . . 360
K.2 ERRORS.INC . . . 360
K.3 BIOS.INC . . . 361
K.4 WORKAREA.INC . . . 362
K.5 SUBROM.INC . . . 362
K.6 HOOKS.INC . . . 362
K.7 PORTS.INC . . . 362
K.8 ASCII.INC . . . 363
K.9 EXTBIO.INC . . . 363
K.10 参考文献 . . . 363

L 用語集 365

このマニュアルについて

対象とする読者

このマニュアルは、Z80 のアセンブリ言語で自分のプログラムを作りたいと考えている MSX ユーザー、とくにこれから始めようとしている人を対象としています。MSX-DOS の使い方、つまりドライブやディレクトリの切り替え、ファイルの一覧表示、プログラムの実行といった操作はわかっているものとします。しかし、アセンブラーやリンカーを使ったことがある必要はありません。

すでにアセンブリ言語でプログラムを作っている人は、2 章を読み飛ばしてもかまいません。必要になりそうなリファレンス情報は、第 II 部と第 III 部、それに付録にまとめてあります。

このマニュアルで扱わないこと

このマニュアルでは、Tatara と Tanren の使い方を説明します。Z80 の命令セットや、それを使ったプログラムの設計・作成の方法は扱いません。それらについては、Z80 のアセンブリ言語によるプログラミングを解説した書籍や講座を参照してください。

構成

このマニュアルは、4 つの部と付録で構成されています。

第 I 部「はじめの一歩」 2 つのツールが何であるかを述べ、アセンブラーとリンカーがどのように協力して働くのかを説明したあと、ツールをインストールして最初のプログラムを作るまでを案内します。

第 II 部「アセンブラー Tatara」 アセンブラーについて説明します。実行のしかた、ソース・ファイルの書き方、そして式や命令からマクロや条件付きアセンブルまで、言語のすべての機能を取り上げます。

第 III 部「リンカー Tanren」リンカーについて説明します。実行のしかた、オブジェクト・ファイルを探して結合するしくみ、プログラムをメモリのどこに配置するのか、そして書き出せるファイルの種類を取り上げます。

第 IV 部「実例」複数のファイルからなるプログラム、MSX-BASIC 用のバイナリ、ROM カートリッジなど、よくある作業ごとに、完全なプログラムを実際に作ってみます。

付録 調べものをするためのものです。すべての疑似命令、命令、演算子、エラー・メッセージ、制限値を 1 か所にまとめてあります。

アセンブリ言語が初めての人は、まず第 I 部を順に読み、次に第 IV 部の実例を進めてください。実例に出てきたことをくわしく知りたくなったら、そのつど第 II 部や第 III 部を参照するとよいでしょう。経験のあるプログラマーであれば、5 章と 21 章、それに付録を読めば、必要なことはほとんどわかるはずです。

表記について

入力する文字は、**`tatara hello.as hello.tro`** のように太字のタイプライター体で示します。プログラムが表示する文字は、`ERROR: undefined symbol in an expression.` のように通常のタイプライター体で示します。

MSX-DOS2 のプロンプトでの操作例では、その両方を示します。プロンプトは `A:\>`と表記します。カレント・ディレクトリがドライブのルート・ディレクトリでないときは、`A:\SOMEDIR\>`のように、プロンプトにそのディレクトリが含まれます。

MSX-DOS

```
A:\>tanren /v
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools
```

本文中のファイル名は、`HELLO.AS` のように大文字で書きます。MSX-DOS も 2 つのプログラムも、ファイル名をこのように表示するからです。入力するときは、大文字でも小文字でもかまいません。

16 進数は、アセンブラーが読むのと同じ形で、末尾に `h` を付けて書きます。たとえば `0100h` は 16 進数の 100、つまり 10 進数の 256 です。

Tatara と *Tanren* は、2 つのツールの名前です。プログラムそのもの、つまり実行するファイルを指す場合は、TATARA、TANREN のように大文字で書きます。

> 注. 注は、役に立つものの、最初に読むときには飛ばしてもかまわない情報です。

> 警告. 警告は、作業の内容が失われたり、思ったとおりに動かないプログラムができたりするおそれのあることを示します。

対象とするバージョン

このマニュアルは、Tatara と Tanren のバージョン 1.2.0 について説明しています。2 つのツールとこのマニュアルの最新版は、`https://tatara.tools` で公開しています。プログラムと、このマニュアルで使うサンプル・プログラムは、`https://tatara.tools/download` からダウンロードできます。

第 I 部

はじめの一歩

1 はじめに 3

2 アセンブラーとリンカーのしくみ 7

3 インストール 17

4 最初のプログラム 23

第 1 章

はじめに

1.1 Tatara と Tanren とは 3
1.2 名前の由来 4
1.3 できること 4
1.4 必要なもの 6
1.5 次に読むところ 6

1.1 Tatara と Tanren とは

Tatara と Tanren は、Z80 のアセンブリ言語で書いたプログラムを、MSX で動くプログラムにするために必要な 2 つのツールです。

Tatara は**アセンブラー**です。`ld a,5` や `call 5` のような命令を並べたテキスト・ファイル、つまり自分で書いたプログラムを読み込み、Z80 プロセッサーが理解できるマシン語に変換します。

Tanren は**リンカー**です。Tatara が作ったマシン語を 1 つまたは複数のファイルから受け取って結合し、それぞれの部分をメモリのどこに置くかを決めて、実行できる完成したプログラムを書き出します。

なぜツールが 1 つではなく 2 つなのか、それぞれが何をするのかについては、2 章で詳しく説明します。

どちらのツールも、MSX-DOS2 または Nextor のもとで、MSX そのものの上で動きます。プログラムを書き、アセンブルし、リンクし、実行するまでを同じマシンで行うことができ、ほかのコンピューターは必要ありません。MSX のエミュレーター上でも同じように動きます。

Tatara が読むアセンブリ言語は、Microsoft の MACRO-80 と同じものです。MACRO-80 は、CP/M や初期の MSX の時代に、多くのソフトウェアの開発に使われたアセンブラーです。そのため、MACRO-80 向けに書かれたプログラムや書籍

は、Tatara でも利用できます。両者の違いは付録 J にまとめてあります。

1.2　名前の由来

2 つの名前は、日本刀を作る伝統的な方法から取ったものです。

たたらは、粘土で作った炉です。この炉に砂鉄と木炭を入れて燃やし、3 日ほどかけて**玉鋼**（たまはがね）を作ります。玉鋼は強いものの形のない粗い鋼のかたまりで、まだ刀ではありません。

鍛錬は、そのあとに続く鍛造の作業です。刀匠は玉鋼を熱し、打ち、折り返すことを何度もくり返して、完成した刀に仕上げます。この言葉は、ふだんの日本語では訓練や修練という意味でも使われます。

2 つのツールは、プログラムに対してこれと同じ 2 つの仕事をします（図 1.1）。自分で書いたソース・ファイルが原料です。TATARA はそれをオブジェクト・ファイルにします。オブジェクト・ファイルは本物のマシン語ですが、まだ実行できるプログラムではありません。玉鋼がまだ刀ではないのと同じです。TANREN は 1 つ以上のオブジェクト・ファイルを受け取り、それを鍛えて完成したプログラムにします。2 章では、このたとえをもとに説明を進めます。

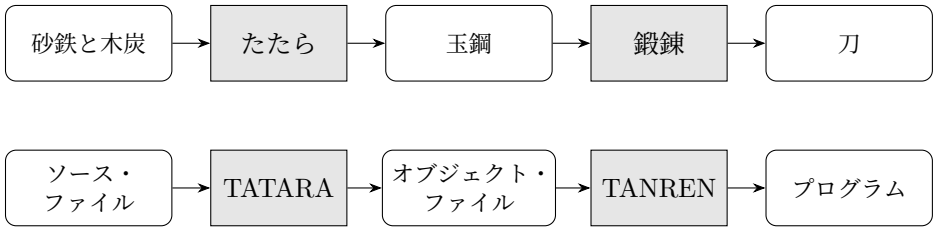

図 1.1: 原料から完成した刀へ、ソース・ファイルから完成したプログラムへ

1.3　できること

最初に読むときのために、主な機能を簡単に挙げておきます。それぞれの項目は、示した章で詳しく説明します。

- **長いシンボル名**　ラベルのようなシンボルの名前には 255 文字まで使えるので、その役目がわかる名前を付けることができます。また、`/C` オプションを使うと名前の大文字と小文字が区別されるようになり、`Count` と `count` は別の名前になります（7 章）。

- **マクロと条件付きアセンブル**　一度書いたコードを何度でも使うことができ、アセンブルするときにプログラムの一部を含めたり除いたりすることもできます（15 章〜17 章）。

- **セグメント**　コードとデータを、セグメントと呼ばれるプログラムの別々の領域に分けて置くことができます。セグメントには名前を付けることもできます。また、プログラムのそれぞれ一部分でしか使わないデータ領域どうしは、同じメモリを共有できます（10 章）。

- **多くのファイルとディレクトリにまたがるプログラム**　`INCLUDE` 疑似命令で取り込むファイルは、3 か所から探されます。さらに、環境変数 `TATARA` と `TANREN` で、ほかに探す場所を指定できます（13 章、23 章）。

- **用意された名前**　Tatara には、MSX-DOS2 のファンクションとエラー・コード、BIOS のエントリー・アドレス、システム・ワーク・エリアなど、合わせて 904 個の名前を定義したインクルード・ファイルが付いています。そのため、プログラムでは `_STROUT` が `09h` であることを調べなくても、名前をそのまま書くことができます（14 章）。

- **大きなプログラム**　どちらのツールも表をメモリ・マッパーに置き、Tanren は作成中のプログラムもメモリ・マッパーに置きます。そのため、作れるプログラムの大きさは、MSX-DOS が空けておくメモリではなく、MSX に搭載されたメモリの量で決まります（26 章、付録 G）。

- **リンカーへの指示をファイルに書ける**　MSX-DOS は、コマンドラインを 127 文字で打ち切ります。多くのファイルからなるプログラムでは、すぐにこの長さに達してしまいます。Tanren は、ファイルの一覧をはじめ、コマンドラインで指定する内容を、テキスト・ファイルから読むことができます（22 章）。

- **自分の書いた行を示すエラー**　エラーが起きると、メッセージには自分で書いたファイルと行が示されます。その行がマクロや取り込んだファイルから来たものであっても同じです（20 章）。

- **さまざまな出力形式**　Tanren は、MSX-DOS 用のプログラム、MSX-BASIC の `BLOAD` 用のバイナリ、任意のアドレス用のコードのイメージ、ROM カートリッジのイメージを書き出すことができます（26 章）。

- **R800 の命令**　Tatara は、MSX turbo R に搭載されている R800 プロセッサーの乗算命令、`MULUB` と `MULUW` もアセンブルできます（9 章）。

- **未公開命令**　Tatara は、`SLL` や、`IXH` と `IXL` を使う命令など、MSX のプログラムが使う Z80 の未公開命令もアセンブルできます（9.5 節）。

1.4　必要なもの

Tatara と Tanren を使うには、次のものが必要です。

- **MSX-DOS2** または **Nextor** が動いている MSX。MSX-DOS1 では、どちらのプログラムもその旨のメッセージを表示して、すぐに終了します。
- **メモリ・マッパー**。MSX-DOS2 はメモリ・マッパーがないと動かないので、MSX-DOS2 が動いている MSX であれば、すでに備わっています。あるプログラムを作るのにツールが必要とするメモリの量は、付録 G で説明します。
- Tatara のアーカイブ `TATARxyz.ZIP` と、それを展開する `UNKAGO.COM`。アーカイブは `https://tatara.tools/download` から、UNKAGO は `https://kago.tools/en/download/` からダウンロードできます。インストールのしかたは 3 章で説明します。
- プログラムを書くための、プレーン・テキストのファイルを保存できるエディター。

MSX-DOS2 または Nextor が動く MSX2 以降の機種であれば、どれでも使えます。これらを動かせる MSX エミュレーターでもかまいません。

1.5　次に読むところ

2 章では、アセンブラーとリンカーが何をするのか、なぜプログラムを 2 つの段階で作るのかを説明します。3 章でツールをインストールし、4 章でそれを使って最初のプログラムを作り、実行します。

第 2 章

アセンブラーとリンカーのしくみ

2.1 2 つの段階 7
2.2 アセンブラーがすること 8
2.3 アセンブラーにわからないこと 9
2.4 リンカーがすること 10
2.5 実例 10
2.6 コードとデータ：セグメント 14
2.7 2 つの段階に分ける利点 14
2.8 この章で出てきた用語 14

この章では、TATARA と TANREN がそれぞれ何をするのか、そしてなぜプログラムを 1 回ではなく 2 つの段階に分けて作るのかを説明します。あわせて、このマニュアルの以降の章で使う用語を紹介し、小さなプログラムを 1 つ取り上げて、自分で書いたソース・ファイルから実行するプログラムができあがるまでを追っていきます。

2.1 2 つの段階

プログラムは、1 つまたは複数の**ソース・ファイル**（source file）から始まります。ソース・ファイルは、エディターを使ってアセンブリ言語で書くテキスト・ファイルです。Tatara のソース・ファイルには、ふつう `.AS` という拡張子を付けます。

TATARA はソース・ファイルを読み、拡張子 `.TRO` の**オブジェクト・ファイル**（object file）を書き出します。オブジェクト・ファイルにはプログラムのマシン語が入っていますが、まだ実行できるプログラムではありません。

TANREN は 1 つ以上のオブジェクト・ファイルを読み、完成した**プログラム・ファイル**（program file）を書き出します。MSX-DOS2 の場合、これは拡張子 `.COM`

のファイルで、MSX-DOS2 のプロンプトでその名前を入力すると実行されます。この 2 つの段階を示したのが図 2.1 です。

図 2.1: ソースからプログラムまでの 2 つの段階

1 章のたとえでいえば、ソース・ファイルは原料、オブジェクト・ファイルは玉鋼、プログラムは完成した刀にあたります。この章の残りでは、それぞれの段階で何が起きているのか、そしてなぜ玉鋼を作っておく価値があるのかを説明します。

2.2　アセンブラーがすること

ソース・ファイルは行の並びです。その多くは、`ld c,_STROUT` のように、人が読める短い言葉で書いた Z80 への命令です。しかし、Z80 は言葉を読むことができません。Z80 が読むのはバイト、つまり 0 から 255 までの数で、それがメモリ上に 1 つずつ並んだものがプログラムの**マシン語**（machine code）です。ソース・ファイルの各行をそのバイトに変換することを**アセンブル**（assembling）といいます。

表 2.1 は、3 つの行と、TATARA がそれをアセンブルした結果のバイトを示したものです。バイトは 16 進数で書いてあります。

行	バイト
`ld c,_STROUT`	`0E 09`
`call BDOS`	`CD 05 00`
`ret`	`C9`

表 2.1: 3 つの行と、それが変換されたバイト

各行の最初のバイトは、それがどの命令なのかを表します。命令にオペランドがあれば、そのあとのバイトがオペランドです。アドレスは 2 バイトを使い、Z80 は下位のバイトを先に置きます。そのため、アドレス `0005h` は、`05`、`00` の順に並びます。

`_STROUT` と `BDOS` は命令ではありません。これらは `09h` と `0005h` という数に付けた名前で、Tatara に付属するインクルード・ファイルの 1 つ、`MSXDOS.INC` で定義されています（14 章）。このファイルを取り込んだプログラムではこれらの名前が使え、アセンブラーはそれぞれの名前を対応する数に置き換えます。このように数の代わりに使う名前を**シンボル**（symbol）といいます。

プログラムには、自分自身の中の場所を指す名前も必要です。たとえば、ループの始まりや、表示するメッセージの位置です。このような名前を**ラベル**（label）といいます。ラベルは、`msg1:` のように行の先頭に書き、あとにコロンを付けます。ラベルは、その行が作る最初のバイトの**アドレス**（address）、つまりそのバイトが置かれるメモリの位置の番号を表します。ラベルもシンボルの一種です。

それぞれのラベルが何を表すのかを知るために、アセンブラーはバイトを数えながら進みます。プログラムの最初の行が 3 バイトを作り、2 行目がさらに 3 バイトを作れば、3 行目は最初の行から 6 バイト先から始まるわけです。

> 注. このマニュアルでは、バイト、16 進数、メモリのアドレスが何であるかはわかっているものとします。これらについては、Z80 のプログラミングを学ぶための書籍や講座で説明されています。

2.3 アセンブラーにわからないこと

バイトを数えれば、アセンブラーには、それぞれのラベルがプログラムの先頭からどれだけ離れているかがわかります。しかし、1 つのソース・ファイルだけではわからないことが 2 つあります。

1. **プログラムがメモリのどこに置かれるか**　たとえば、あるラベルがコードの先頭から 17 バイト目にあることはわかっても、そのコードがどのアドレスから始まるのかはわかりません。それは、あとでプログラムのすべての部分をまとめるときに決まるからです。このように、コードを置く場所によって変わるアドレスを**リロケータブル**（relocatable）といいます。
2. **ほかのファイルにあるものがどこにあるか**　プログラムは複数のソース・ファイルに分けて書くことができ、あるファイルの行から、別のファイルに書かれたルーチンを呼び出すこともできます。TATARA は 1 度に 1 つのファイルをアセンブルするので、別のファイルで定義された名前のアドレスを知ることはできません。

そこで TATARA は、できるかぎりの仕事をしたうえで、終えられなかったことを書き留めておきます。まだアドレスがわからない場所には、マシン語の中に**穴**（hole）を残し、その穴に何が入るのかを記録します。こうしてできるのが、オブジェクト・ファイルです。つまり、穴のあいたマシン語と、それぞれの穴についての記録です。これが玉鋼にあたります。

1 つのソース・ファイルのコードを、オブジェクト・ファイルに入った形で呼ぶときは**モジュール**（module）といいます。モジュールは、自分のシンボルの一部をほかのモジュールに提供したり、ほかのモジュールが提供するシンボルを使ったりすることができます。

- **パブリック・シンボル**（public symbol）は、モジュールが定義し、ほかのモジュールから使えるようにしたシンボルです。PUBLIC 疑似命令で宣言します。
- **外部シンボル**（external symbol）は、モジュールが使うものの、定義はほかのモジュールにあるシンボルです。EXTRN 疑似命令で宣言します。

疑似命令とは、Z80 ではなくアセンブラーに対して指示を与える行のことで、それ自体はマシン語を作りません。

2.4　リンカーがすること

TANREN は 1 つ以上のオブジェクト・ファイルを読み、次の 3 つのことを行います。

1. **モジュールを配置する**　モジュールをメモリ上に順に並べていきます。始まりのアドレスは 0100h で、ここは MSX-DOS2 がプログラムを読み込み、実行を始める場所です。[1]
2. **すべての穴を埋める**　リロケータブルなアドレスの場合は、TATARA が穴に残しておいた値に、そのモジュールを置いたアドレスを足します。外部シンボルの場合は、そのシンボルをパブリックとして宣言しているモジュールを探し、そのシンボルのアドレスを穴に書き込みます。穴を埋めるこの作業を**リロケーション**（relocation、再配置）といいます。
3. **プログラム・ファイルを書き出す**　完成したバイトを、MSX-DOS2 が読み込んで実行できるファイルに書き出します。[2]

この仕事全体を**リンク**（linking）といいます。玉鋼を鍛えて刀にする鍛錬にあたる作業です。

2.5　実例

例として取り上げる TWOMOD は、2 つのモジュールからなるプログラムで、Tatara に付属するサンプル・プログラムの 1 つです。4 章で実際に作り、28 章でもう一度取り上げます。1 つ目のモジュール MAIN.AS は、putstr という名前のルーチンを呼び出して 2 行を表示します。putstr は外部シンボルとして宣言しています。

[1] これはデフォルトの動作です。TANREN は、特に指定しないかぎり MSX-DOS2 用のプログラムを作るからです。開始アドレスは変えることができ、ROM カートリッジや MSX-BASIC の BLOAD 用のバイナリのように、メモリのほかの場所で動くプログラムを作ることもできます（24 章）。

[2] これもデフォルトの動作です。TANREN は、MSX-BASIC の BLOAD 用のバイナリや、ROM カートリッジのイメージを書き出すこともできます（26 章）。

```
; MAIN.AS - one of two modules.
;
; It knows that putstr exists. It does not know where putstr is, and
; it does not need to: EXTRN says the name is somebody else's, and the
; linker fills the address in.

                extrn   putstr          ; defined in PUTSTR.AS

                include msxdos.inc      ; BDOS and _TERM0

                cseg

start:          ld      hl,msg1
                call    putstr
                ld      hl,msg2
                call    putstr
                ld      c,_TERM0
                jp      BDOS

msg1:           db      "Two modules, one program.",13,10,0
msg2:           db      "The linker joined them.",13,10,0

                end     start
```

2 つ目のモジュール PUTSTR.AS は、putstr を定義し、それをパブリックとして宣言しています。

```
; PUTSTR.AS - the other module.
;
; PUBLIC is what makes the name visible outside this file. Without it
; the name would still assemble here and the link of MAIN.AS would
; stop with an undefined symbol - which is worth trying once.

                public  putstr

                include msxdos.inc      ; BDOS, _CONOUT and "system"

                cseg

; putstr - print a string.
;
;   HL is pushed around the BDOS call because MSX-DOS promises nothing
;   about the registers it hands back.
;
; Input:        HL -> the string, ending in a zero byte
; Output:       it is printed
; Modifies:     AF, BC, DE, HL

putstr:         ld      a,(hl)
                or      a
                ret     z
                push    hl
```

```
                ld      e,a
                system  _CONOUT
                pop     hl
                inc     hl
                jr      putstr

                end
```

どちらのモジュールも MSXDOS.INC を取り込んでいます。INCLUDE の行は、そのファイルの行がその場所に書かれているかのように読み込み、それ自体はバイトを作りません。PUTSTR.AS では system _CONOUT を使っています。system は MSXDOS.INC で定義された**マクロ**で、いくつかの行のまとまりに付けた 1 つの名前です。ここでは、ld c,_CONOUT と call BDOS の 2 行を表します。マクロについては 16 章で説明します。

2.5.1　TATARA が MAIN.AS から作るもの

MAIN.AS をアセンブルすると、71 バイトのマシン語ができ、その中には 4 つの穴があります（表 2.2）。オフセットは、モジュールの先頭からのバイト数です。

オフセット	行	バイト	穴
0	ld hl,msg1	21 11 00	リロケータブル：msg1、17 バイト目
3	call putstr	CD 00 00	外部：putstr
6	ld hl,msg2	21 2D 00	リロケータブル：msg2、45 バイト目
9	call putstr	CD 00 00	外部：putstr
12	ld c,_TERM0	0E 00	なし
14	jp BDOS	C3 05 00	なし：BDOS は常に 0005h
17	msg1: db ...	テキスト 28 バイト	なし
45	msg2: db ...	テキスト 26 バイト	なし

表 2.2: TATARA が書き出すモジュール MAIN

リロケータブルな穴には、ラベルのオフセットが入っています。11h は 17 で、msg1 のオフセットです。外部シンボルの穴には 0 が入っています。アセンブラーは、putstr について名前以外は何も知らないからです。オフセット 14 の jp BDOS は穴ではありません。BDOS は 0005h という決まったアドレスで、プログラムをどこに置

いても変わらないからです。

2.5.2 TATARA が PUTSTR.AS から作るもの

PUTSTR.AS をアセンブルすると 14 バイトになり、穴は 1 つもありません。system_CONOUT の中の call BDOS は、決まったアドレスを呼び出します。最後の jr putstr も、アドレスを持っているわけではありません。持っているのは「14 バイト戻る」という距離で、コードをどこに置いても同じだからです。このモジュールには、オフセット 0 に putstr というパブリック・シンボルが 1 つあります。

2.5.3 TANREN がすること

TANREN は、先に指定された MAIN を最初に置き、そのすぐあとに PUTSTR を置きます。そして、表 2.3 のように 4 つの穴を埋めます。

MAIN の配置	0100h（71 バイト、0146h まで）
PUTSTR の配置	0147h（14 バイト、0154h まで）
したがって putstr は	0147h
0101h の穴	0011h + 0100h = 0111h、msg1 のアドレス
0104h の穴	0147h、putstr のアドレス
0107h の穴	002Dh + 0100h = 012Dh、msg2 のアドレス
010Ah の穴	0147h、putstr のアドレス

表 2.3: TANREN による MAIN と PUTSTR のリンク

こうしてできるのが 85 バイトの TWOMOD.COM で、MSX-DOS2 はこれを 0100h に読み込みます。2 つのオブジェクト・ファイルと、それからできるプログラムを示したのが図 2.2 です。

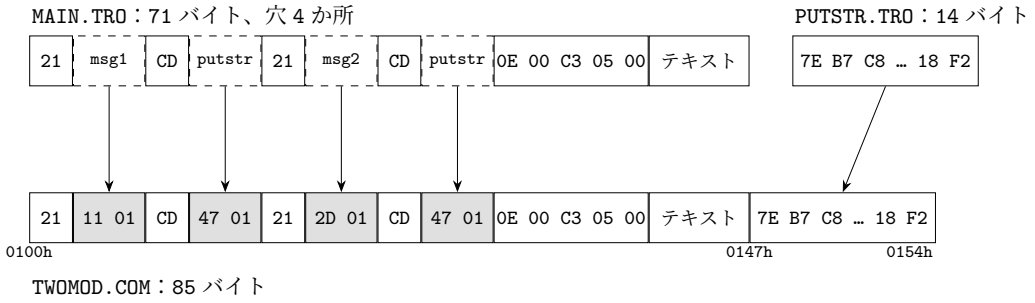

図 2.2: MAIN.TRO と PUTSTR.TRO をリンクして TWOMOD.COM を作る

2.6　コードとデータ：セグメント

プログラムには、命令と、命令が扱うデータという 2 種類のものが入っています。Tatara では、この 2 つを 2 つの**セグメント**（segment）に分けておくことができます。**コード・セグメント**（code segment）は CSEG 疑似命令で、**データ・セグメント**（data segment）は DSEG 疑似命令で始めます。

TANREN はモジュールを配置するとき、すべてのモジュールのコード・セグメントを先に順に並べ、そのあとにデータ・セグメントを並べます。そのため、完成したプログラムでは、それぞれの種類がひとまとまりになります。

TWOMOD は 2 つのメッセージをコード・セグメントに置いています。このくらい小さなプログラムならそれで問題はなく、データ・セグメントは空です。セグメントについては、独自の名前を持つセグメントや、プログラムの別々の部分で共有するデータ領域も含めて、10 章で詳しく説明します。

2.7　2 つの段階に分ける利点

プログラムを 2 つの段階で作ると、1 つの段階で作るよりもコマンドが 1 つ増えます。そのかわりに、次の 4 つの利点が得られます。

再利用 PUTSTR のようなモジュールは、アセンブルし直すことなく、いくつものプログラムにリンクできます。

大きさ 大きなプログラムは複数のファイルに分けたほうが扱いやすく、アセンブルし直すのは変更したファイルだけですみます。

共同作業 ルーチンのライブラリを書いた人は、それをアセンブルしてオブジェクト・ファイルを公開することができます。ほかの人は、そのコードを自分のプログラムにコピーすることなく、自分のプログラムにリンクして使えます。ソース・ファイルがまったくなくてもかまいません。オブジェクト・ファイルがあれば十分です。

出力の選択 プログラムをどこに置くかは最後に決まるので、同じオブジェクト・ファイルから、MSX-DOS2 用のプログラム、MSX-BASIC の BLOAD 用のバイナリ、ROM カートリッジのどれでも作ることができます（26 章）。

2.8　この章で出てきた用語

表 2.4 は、この章で紹介した用語の一覧です。このマニュアルのすべての用語は、用語集（付録 L）にまとめてあります。

用語	意味
ソース・ファイル	アセンブリ言語で書くテキスト・ファイル
マシン語	Z80 が読んで実行するバイト
アセンブル	ソースの行をマシン語に変換すること
シンボル	数の代わりに使う名前
ラベル	プログラム中のアドレスを表すシンボル
アドレス	メモリの位置の番号
リロケータブル	コードを置く場所によって変わるアドレス
穴	マシン語の中で、アドレスがまだわからない場所
オブジェクト・ファイル	TATARA が書き出す、穴のあいたマシン語
モジュール	オブジェクト・ファイルに入った、1 つのソース・ファイルのコード
パブリック・シンボル	モジュールがほかのモジュールに提供するシンボル
外部シンボル	モジュールが使うが、定義はほかのモジュールにあるシンボル
リロケーション	穴を埋めること
リンク	モジュールを配置し、穴を埋め、プログラムを書き出すこと
プログラム・ファイル	TANREN が書き出す、完成したプログラム
セグメント	コードまたはデータという、1 種類の内容のためのプログラムの領域

表 2.4: この章で紹介した用語

第3章

インストール

3.1 配布されるもの 17
3.2 ファイルの置き場所 18
3.3 インストールの手順 19
3.4 PATH と TATARA の設定 20
3.5 インストールの確認 21
3.6 アップデート 22

この章では、ダウンロードしたファイルから始めて、TATARA と TANREN がどのディレクトリからでも実行でき、インクルード・ファイルを見つけられる状態の MSX を用意します。ここで説明するインストールは、このマニュアルの以降の章が前提とするものです。これから先のセッションは、すべてこのディレクトリ構成で示します。

3.1 配布されるもの

Tatara は、1 つの**アーカイブ**（archive）として配布されます。アーカイブとは、複数のファイルを 1 つにまとめたファイルです。名前は `TATARxyz.ZIP` で、`xyz` はバージョンを表します。たとえば `TATAR120.ZIP` はバージョン 1.2.0 です。`https://tatara.tools/download` からダウンロードできます。中には、表 3.1 のように、すでにディレクトリに分けられたインストール一式が入っています。

アーカイブは、MSX では UNKAGO を使って展開します。UNKAGO は、ZIP、LZH、PMA のアーカイブを展開するプログラムです。UNKAGO は必須です。MSX でこのアーカイブを展開するのに使うもので、MSX-DOS2 にも Nextor にも付属していません。`UNKAGO.COM` は `https://kago.tools/en/download/` からダウンロードしてください。TATARA や TANREN と同じく、MSX-DOS2 または Nextor で動きます。

ファイルまたはディレクトリ	内容
BIN	アセンブラーの TATARA.COM と、リンカーの TANREN.COM
INCLUDE	インクルード・ファイル（14 章）
EXAMPLES	このマニュアルのサンプル・プログラム（サンプルごとに 1 つのディレクトリ）
README.TXT	バージョンと、Web サイトのアドレス
LICENSE	Tatara の配布条件である Apache License 2.0

表 3.1: アーカイブの中身

> 注. 以前のリリースは、TATAR117.LZH のような.LZH のアーカイブで配布されていました。これらにはディレクトリがないので、この章で説明する方法ではインストールできません。この章で説明するのは.ZIP のアーカイブです。

このマニュアルでは、2 つのファイルを PC や Mac などの別のコンピューターでダウンロードし、メモリ・カードなどを使って MSX にコピーすることを前提にしています。

3.2　ファイルの置き場所

このマニュアルでは、アーカイブを A:\TATARA に展開し、次のディレクトリができているものとします。

```
A:\TATARA\
    BIN\          TATARA.COMとTANREN.COM
    INCLUDE\      インクルード・ファイル
    EXAMPLES\     サンプル・プログラム（サンプルごとに1つのディレクトリ）
    README.TXT
    LICENSE
```

次の 2 つの条件を満たしていれば、ほかの場所にインストールしてもかまいません。2 つのプログラムがあるディレクトリが PATH に含まれていること、そして環境変数 TATARA がインクルード・ファイルのあるディレクトリを指していることです。どちらも 3.4 節で説明します。以降の章では A:\TATARA で示しますので、別の場所にインストールした場合は、自分のディレクトリに読み替えてください。

> 注. 日本の MSX では、バックスラッシュが円記号で表示されるため、上のディレクトリは `A:¥TATARA¥INCLUDE` のように見えます。これは同じ文字なので、このマニュアルのパスはそのまま使えます。

3.3 インストールの手順

`TATAR120.ZIP` と `UNKAGO.COM` を、MSX の同じディレクトリにコピーします。ファイルをどこに置くかは UNKAGO に指定するので、ディレクトリはどこでもかまいません。次の例では、ドライブ A: のルート・ディレクトリに置いています。

1. **アーカイブを `A:\TATARA` に展開します。** 2 つのファイルがあるディレクトリで、スイッチ `/D:` と展開先のディレクトリを付けて UNKAGO を実行します。`A:\TATARA` がなければ、UNKAGO がそれとその中のディレクトリを作ります。

MSX-DOS

```
A:\>unkago /d:a:\tatara tatar120.zip
UNKAGO LZH/PMA/ZIP Decompressor v1.0.1
Copyright (C) 2026 Javier Lavandeira
https://kago.tools

Extracting BIN\ OK
Extracting BIN\TATARA.COM OK
Extracting BIN\TANREN.COM OK
Extracting INCLUDE\ OK
Extracting INCLUDE\WORKAREA.INC OK
...
Extracting EXAMPLES\TWOMOD\TWOMOD.LNK OK
Extracting README.TXT OK
Extracting LICENSE OK
```

   UNKAGO は、アーカイブの中のディレクトリとファイルごとに 1 行、全部で 50 行を表示します。どの行も `OK` で終わっていれば正常です。上の表示は途中を省略しています。

2. **確認します。`dir a:\tatara\bin`** で `TATARA.COM` と `TANREN.COM` が、**`dir a:\tatara\include`** で表 3.2 の 10 個のファイルが、**`dir a:\tatara\examples`** でサンプルごとの 8 つのディレクトリが表示されれば、正しく展開されています。

3. **片付けます。** `TATAR120.ZIP` はもう必要ないので、削除してかまいません。`UNKAGO.COM` は残しておきます。アップデートの展開にも使います（3.6 節）。

ファイル	内容
ASCII.INC	制御コード
BIOS.INC	メイン ROM のエントリー・ポイント
ERRORS.INC	MSX-DOS と MSX-DOS2 のエラー・コード
EXTBIO.INC	拡張 BIOS と、MSX-MUSIC の FM BIOS
HOOKS.INC	フック
MSXDOS.INC	MSX-DOS のファンクション番号と、system マクロ
PORTS.INC	I/O ポート
SUBROM.INC	サブ ROM のエントリー・ポイント
WORKAREA.INC	システム・ワーク・エリア
README.TXT	各ファイルの簡単な説明

表 3.2: A:\TATARA\INCLUDE の中のファイル

3.4 PATH と TATARA の設定

環境変数（environment variable）とは、文字列を保持する名前で、MSX-DOS2 がメモリに保存し、プログラムから読み出すことができます。このインストールでは、2 つの環境変数を使います。

PATH コマンドを入力すると、MSX-DOS2 はその名前のプログラムを、まずカレント・ディレクトリで探し、次に PATH に書かれたすべてのディレクトリで探します。ここに A:\TATARA\BIN を加えると、どのディレクトリからでも TATARA と TANREN を実行できるようになります。

TATARA ソース・ファイルがほかのファイルを取り込むとき、TATARA はそのファイルを、まずソース・ファイルのあるディレクトリで、次にカレント・ディレクトリで、そして TATARA に書かれたすべてのディレクトリで探します。これを A:\TATARA\INCLUDE に設定すると、どのプログラムでも、コピーを置かずに表 3.2 のファイルを取り込めるようになります。詳しい規則は 13 章で説明します。

いまのセッションで設定するには、次のように入力します。

```
A:\>set path=%path%;a:\tatara\bin
A:\>set tatara="a:\tatara\include"
```

%path%は PATH の現在の値を表します。そのため、1 行目は PATH を置き換えるのではなく、その最後に新しいディレクトリを加えます。**set** だけを入力すると、すべての環境変数とその値が表示されます。PATH には、新しく加えたディレクトリの前に、自分の環境のディレクトリが表示されるはずです。

> 注. MSX-DOS2 のプロンプトでは、上のように TATARA の値を引用符で囲みます。囲まないと、MSX-DOS2 は*** Invalid environment string と表示し、環境変数を設定しません。後で説明する AUTOEXEC.BAT では、引用符は必要ありません。

この方法で設定した環境変数は、MSX をリセットするか電源を切るまで有効です。MSX を起動するたびに設定するには、次の 2 行を AUTOEXEC.BAT に加えます。AUTOEXEC.BAT は起動ドライブのルート・ディレクトリにあるバッチ・ファイルで、MSX-DOS2 が起動するときに実行します。まだない場合は、テキスト・エディターで作ります。すでにほかのコマンドが書かれていてもかまいません。その最後に 2 行を加えます。

```
set path=%path%;a:\tatara\bin
set tatara=a:\tatara\include
```

> 注. TATARA には、PATH と同じように、セミコロンで区切って複数のディレクトリを書けます。プロジェクトが独自のインクルード・ファイルを持つときに便利です（13 章）。

> 注. TANREN にも、オブジェクト・ファイルを探すディレクトリを指定する TANREN という環境変数があります。このインストールでは必要ありません（23 章）。

3.5 インストールの確認

どのディレクトリからでもかまいませんので、それぞれのプログラムを/V オプションを付けて実行します。プログラムの名前とバージョンを表示するだけで、ほかには何もしません。

```
A:\>tatara /v
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools


A:\>tanren /v
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools
```

代わりに次のように表示された場合は、PATH に A:\TATARA\BIN が含まれていません。**set** で確認し、3.4 節のとおりに設定し直してください。

MSX-DOS

```
A:\>tatara /v
*** Unrecognized command
```

環境変数 TATARA は、4 章で、最初のプログラムが MSXDOS.INC を取り込むときに確認します。

3.6　アップデート

新しいバージョンも同じ方法で展開しますが、スイッチ/O を付けます。

MSX-DOS

```
A:\>unkago /o /d:a:\tatara tatarxyz.zip
```

/O を付けないと、UNKAGO はすでにあるファイルを置き換えません。ファイルごとに Skipping BIN\TATARA.COM: it already exists のように表示し、古いバージョンがそのまま残ります。それぞれのプログラムのバージョンは、/V で確認できます。

> 警告. /O はサンプルも置き換えます。サンプルを変更する前に、そのディレクトリをほかの場所にコピーし、コピーのほうで作業してください。そうしないと、次のアップデートで変更が上書きされます。

第 4 章

最初のプログラム

4.1 始める前に 23
4.2 プログラム 24
4.3 アセンブルする 25
4.4 リンクする 26
4.5 実行する 27
4.6 ビルド・ファイル 27
4.7 2 つのモジュールからなるプログラム 27
4.8 次に読む章 29

この章では、サンプルのうち 2 つを作って実行します。1 つのソース・ファイルからなるプログラム HELLO と、2 章で取り上げた 2 つのモジュールからなるプログラム TWOMOD です。すべてのコマンドと、それに対して TATARA と TANREN が表示する内容を示します。

4.1 始める前に

3 章のインストールが済んでいる必要があります。つまり、2 つのプログラムが PATH に含まれ、環境変数 TATARA が設定され、サンプルが A:\TATARA\EXAMPLES にある状態です。この章では、まず A:\TATARA\EXAMPLES\HELLO で、次に A:\TATARA\EXAMPLES\TWOMOD で作業します。最初のディレクトリに移動します。

MSX-DOS

```
A:\>cd \tatara\examples\hello
A:\TATARA\EXAMPLES\HELLO>
```

4.2　プログラム

HELLO.AS は、完全なプログラムとしてはもっとも小さなものです。1 行を表示して、MSX-DOS2 に戻ります。

```
; HELLO.AS - the smallest complete program.
;
; One source, one object, one .COM. It prints a line and gives the
; machine back to MSX-DOS.

                include msxdos.inc      ; BDOS, _STROUT, _TERM0 and the
                                        ;   "system" macro, found through
                                        ;   TATARA - see BUILD.BAT

                cseg                    ; RELOCATABLE code. Nothing here
                                        ;   knows its own address, and the
                                        ;   linker chooses one - 0100h for a
                                        ;   .COM, unless /P: says otherwise

start:          ld      de,msg
                system  _STROUT
                ld      c,_TERM0
                jp      BDOS

msg:            db      "Hello from Tatara.",13,10,"$"

                end     start           ; where it starts, for the linker
```

数行ずつ見ていきます。

- 最初の数行はセミコロンで始まっています。セミコロンは**コメント**の始まりを表し、TATARA はそこから行の終わりまでを無視します。コメントは、プログラムを読む人のためのものです。行の書き方は 6 章で説明します。
- include msxdos.inc は、その位置で MSXDOS.INC を読み込み、その行がそこに書かれているのと同じように扱います。このファイルでは、BDOS、_STROUT、_TERM0 という名前と、system マクロが定義されています（13 章と 14 章）。TATARA は、このファイルを環境変数 TATARA を使って見つけます。
- cseg はコード・セグメントを始めます。これ以降はリロケータブルなコードで、どこに置くかは TANREN が決めます（10 章）。

- `start:` と `msg:` はラベルです。`start` は最初の命令のアドレス、`msg` は文字列のアドレスです。
- `ld de,msg` と `system _STROUT` は、`msg` にある文字列を、終わりを示す`$`の手前まで表示するよう、MSX-DOS2 に頼みます。
- `ld c,_TERM0` と `jp BDOS` でプログラムが終わり、MSX-DOS2 がプロンプトを再び表示します。
- `db` はプログラムにバイトを置きます。まず文字列、次に次の行の先頭に移るための 13 と 10、最後に`$`です（11 章）。
- `end start` はソース・ファイルの終わりを表し、プログラムが `start` から始まることを TANREN に伝えます。

4.3 アセンブルする

`HELLO.AS` をアセンブルして、オブジェクト・ファイル `HELLO.TRO` を作ります。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara hello.as hello.tro
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ended at HELLO.AS(22)
```

TATARA はバナーを表示し、ソース・ファイルを読んでから、要約を表示します。要約は、ソースがどこで終わったかを示します。ここでは、`END` のある `HELLO.AS` の 22 行目です。これで `HELLO.TRO` が書き出されました。2 章で説明したオブジェクト・ファイル、つまり穴のあいたマシン語です。

警告. `TATARA` が設定されていないと、TATARA は `MSXDOS.INC` を見つけられず、次のように表示して止まります。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara hello.as hello.tro
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ERROR: cannot open MSXDOS.INC
    included from HELLO.AS(6)
```

> 2 行目は、そのファイルがどこで取り込まれたか、つまり HELLO.AS の 6 行目であることを示しています。3.4 節のとおりに環境変数を設定し、もう一度アセンブルしてください。

4.4　リンクする

HELLO.TRO をリンクして、プログラム HELLO.COM を作ります。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tanren /o:hello.com hello.tro
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

1 module, 7 records, ends at EOF.
Wrote HELLO.COM, 0100-0121 (34 bytes), entry 0100.
```

/O: オプションは、プログラム・ファイルの名前を指定します。指定しなければ、TANREN は最初のオブジェクト・ファイルの名前をプログラムに付けるので、ここでもやはり HELLO.COM になります。このマニュアルでは、名前がコマンドから分かるように/O: を付けています。

要約の 1 行目は、TANREN が読んだものを示しています。1 つのモジュールで、7 つのレコードからなり、ファイルの終わりまで読んだということです（21 章）。2 行目は、書き出したものを示しています。HELLO.COM はアドレス 0100h から 0121h までの 34 バイトで、0100h から始まります。これで、ディレクトリには 3 つのファイルがそろいました。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>dir hello.*
 Volume in drive A: is MAIN
 Directory of A:\TATARA\EXAMPLES\HELLO

HELLO    AS        603 26-09-28 17:32
HELLO    TRO        93 26-09-29  1:27
HELLO    COM        34 26-09-29  1:27
 730 bytes in 3 files   12664K free
```

ボリュームの名前、日付、空き容量は、それぞれの MSX で異なります。

4.5 実行する

プログラムの名前を入力します。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>hello
Hello from Tatara.
```

MSX-DOS2 は HELLO.COM を、TANREN が配置したアドレス 0100h に読み込み、そこから実行しました。2 つのコマンドで、ソース・ファイルが実行できるプログラムになりました。

4.6 ビルド・ファイル

どのサンプルにも、**ビルド・ファイル**（build file）BUILD.BAT があります。これは、サンプルを作るためのコマンドを実行するバッチ・ファイルです。HELLO のビルド・ファイルは、この章の 2 つのコマンドを /Q オプション付きで実行します。/Q を付けると、バナーと要約が表示されません。エラーは /Q の有無にかかわらず表示されるので、何も表示されなければ成功したということです。内容を見るには **type build.bat** と入力します。

> 警告. ビルド・ファイルは、始めに環境変数 TATARA を設定し、終わりにそれを消します。ビルド・ファイルを実行すると、そのセッションでは TATARA が空になり、次にファイルを取り込むプログラムは cannot open と表示して止まります。3.4 節のとおりに環境変数を設定し直すか、MSX をリセットして AUTOEXEC.BAT で設定されるようにしてください。

このマニュアルでは、ビルド・ファイルを使わずにコマンドを入力します。そのほうが、コマンドを覚えるにはよい方法です。

4.7 2 つのモジュールからなるプログラム

TWOMOD は 2 章のプログラムです。MAIN.AS は 2 行を表示するために putstr を呼び出し、putstr は PUTSTR.AS で定義されています。このディレクトリに移動し、2 つのソース・ファイルを 1 つずつアセンブルします。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>cd ..\twomod
A:\TATARA\EXAMPLES\TWOMOD>tatara main.as main.tro
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ended at MAIN.AS(23)

A:\TATARA\EXAMPLES\TWOMOD>tatara putstr.as putstr.tro
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ended at PUTSTR.AS(32)
```

2 つのオブジェクト・ファイルは、HELLO.TRO のときと同じように TANREN のコマンド・ラインに並べることもできます。このサンプルでは、代わりに**リンク・ファイル**（link file）を使います。リンク・ファイルとは、本来コマンド・ラインに入力する内容を書いておくテキスト・ファイルです。TWOMOD.LNK には、プログラムの名前と 2 つのオブジェクト・ファイルが書かれています。

```
; TWOMOD.LNK - the link, in a file instead of on the command line.
;
; TANREN reads this when it is given @TWOMOD, and the words in it are
; the words it would have read from the command line. A ; starts a
; comment and the rest of the line is ignored.
;
; MSX-DOS gives a command line 127 characters and cuts a longer one
; without saying so. That is the whole reason this exists: the
; assembler is itself linked from a file of nineteen module names.

/o:twomod.com

main.tro putstr.tro
```

MSX-DOS2 のコマンド・ラインには 127 文字までしか入らないので、モジュールの多いプログラムにはリンク・ファイルが必要です（22 章）。TANREN は、@のあとに名前が書かれたリンク・ファイルを読みます。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tanren @twomod
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

2 modules, 14 records, ends at EOF.
Wrote TWOMOD.COM, 0100-0154 (85 bytes), entry 0100.
```

これは表 2.3 の数字と同じです。MAIN が 0100h に、そのすぐあとに PUTSTR が置かれ、全体で 85 バイト、0154h で終わります。プログラムを実行します。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>twomod
Two modules, one program.
The linker joined them.
```

4.8 次に読む章

第 II 部では、アセンブラーについて、実行のしかた（5 章）から順に詳しく説明します。第 IV 部では、ほかのサンプルを 1 つずつ取り上げて解説します。

第 II 部

アセンブラー Tatara

5 TATARA の実行 33

6 ソースの書式 41

7 名前 47

8 式 55

9 命令 61

10 セグメントとロケーション・カウンター 69

11 データ 77

12 シンボルの定義と共有 81

13 ファイルの取り込み 89

14 インクルード・ファイル 97

15 条件付きアセンブル 105

16 マクロ 113

17 繰り返しブロック 123

18 リスティング 129

19 シンボル表の出力と診断用スイッチ 139

20 エラーメッセージ 149

第5章

TATARAの実行

5.1 コマンド・ライン 33
5.2 ファイル名 34
5.3 オプション 36
5.4 TATARA が表示するもの 37
5.5 リスティング 39
5.6 TATARA が実行を始めないとき 40

4 章では、TATARA を **tatara hello.as hello.tro** という 1 つの形で実行しました。この章では、コマンド・ライン全体について説明します。指定するファイル名、ファイル名を省いたときの動作、すべてのオプション、そして TATARA が自分から表示するすべての内容です。セッションは HELLO を使い、A:\TATARA\EXAMPLES\HELLO で実行します。

5.1 コマンド・ライン

TATARA のコマンドは、次の形をしています。

```
tatara [options] source [object [listing]]
```

角かっこの中は省略できます。source はアセンブルするファイル、object は書き出すオブジェクト・ファイル、listing はリスティングを書き出すファイルです。リスティングについては 5.5 節で説明します。

オプションは、/Q のように、スラッシュのあとに 1 文字を続けたものです。文字は大文字でも小文字でもかまいません。オプションは、ファイル名の前、間、後ろのどこに置いてもよく、1 つずつ別の語として書きます。**/ls** ではなく **/l /s** と書いてください。TATARA はスラッシュのあとの最初の 1 文字だけを読み、その語の残り

は無視するので、**/ls** は**/l** と同じになります。

5.2　ファイル名

ファイル名は順番どおりに扱われます。1 つ目がソース・ファイル、2 つ目がオブジェクト・ファイル、3 つ目がリスティング・ファイルです。4 つ目を書くとエラーになります（5.6 節）。

5.2.1　拡張子は付け加えられない

TATARA は、それぞれの名前を入力されたとおりに使い、拡張子を自分で付け加えることはしません。表 5.1 はよく使われる拡張子で、サンプルもこれを使っていますが、入力するときには拡張子も書く必要があります。

ファイル	よく使われる拡張子
ソース・ファイル	`.AS`（`.ASM` と `.Z80` もよく使われます）
オブジェクト・ファイル	`.TRO`
リスティング・ファイル	`.LST` または `.PRN`

表 5.1: よく使われる拡張子

警告. **tatara hello** は、拡張子のない `HELLO` という名前のファイルを探し、見つからなければ止まります。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara hello
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ERROR: cannot open HELLO
```

名前には、MSX-DOS2 のほかのファイル名と同じように、ドライブとディレクトリを含めることができます。**tatara src\main.as obj\main.tro** は、ソースをディレクトリ `SRC` から読み、オブジェクト・ファイルを `OBJ` に書き出します。

5.2.2 オブジェクト・ファイルを省く

2 つ目のファイル名を書かないと、TATARA はソースを同じように読んでアセンブルし、エラーがあれば表示しますが、何も書き出しません。ソースにエラーがあるかどうかをすばやく確かめるのに便利です。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara hello.as
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ended at HELLO.AS(22)
```

5.2.3 オブジェクト・ファイルが書き出されるとき

TATARA はソースを 2 回読み、1 回目の読み込みでソース全体を読み終えてからオブジェクト・ファイルを作ります。1 回目の読み込みでエラーが見つかると、TATARA はその前に止まるので、同じ名前のオブジェクト・ファイルはそのまま残ります。次の例では、BAD.AS には命令ではない行が 1 行だけ書かれています。HELLO.TRO は、前後で同じ時刻の同じファイルのままです。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>dir hello.tro
 Volume in drive A: is MAIN
 Directory of A:\TATARA\EXAMPLES\HELLO

HELLO    TRO        93 26-09-29  2:15
 93 bytes in 1 file   12384K free
A:\TATARA\EXAMPLES\HELLO>tatara bad.as hello.tro
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

BAD.AS(1): ERROR: not a directive, a macro or an instruction.

A:\TATARA\EXAMPLES\HELLO>dir hello.tro
 Volume in drive A: is MAIN
 Directory of A:\TATARA\EXAMPLES\HELLO

HELLO    TRO        93 26-09-29  2:15
 93 bytes in 1 file   12376K free
```

エラー・メッセージには、ファイル名と、かっこに入った行番号、そして何がいけないのかが表示されます。メッセージについては 20 章で説明します。

5.2.4　オブジェクト・ファイルとソースを同じにはできない

2 つ目のファイル名が 1 つ目と同じだと、オブジェクト・ファイルを書き出すとソースが消えてしまうので、TATARA は実行を拒否します。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara hello.as hello.as
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ERROR: the output file is the input file.
```

TATARA は、入力されたとおりの 2 つの名前を比べます。**hello.as** と **a:hello.as** のように、同じファイルを別の書き方で指定した場合には気づかないので、名前の順番には注意してください。

5.3　オプション

表 5.2 は、ふだん使うオプションの一覧です。それぞれ、専用の章か、この章で説明します。

オプション	働き	章
/P	リスティングを画面に表示する	18
/L	3 つ目に書いたファイルにリスティングを書き出す	18
/S	アセンブルの終わりにシンボル表を表示する	19
/C	シンボル名とマクロ名の大文字と小文字を区別する	7
/Q	バナーと要約行を表示しない	5
/V	バナーを表示して終わる	5
/?	使い方の画面を表示して終わる	5

表 5.2: TATARA のオプション

/C が変えるのは 1 つだけです。これを付けなければ、M80 と同じように Start、START、start は同じ名前ですが、付けると 3 つの別々の名前になります。命令、レジスター、疑似命令は、/C の有無にかかわらず、大文字でも小文字でも書けます。

5.3.1 診断用のオプション

使い方の画面には、ほかに 3 つのオプションがあります。これらは TATARA がソースをどう解釈したかを表示するもので、ふだん使うためではなく、TATARA 自身の問題を調べるためのものです。

/F 各行をリスティングする代わりに、TATARA がその行を分けた 4 つのフィールド（6 章）を、それぞれ角かっこに入れて表示します。

/M すべてのマクロ定義を、TATARA が保存したとおりに表示します。

/H 最後に、TATARA がマクロのために確保しているメモリのブロックのうち、まだ使われているものの数を表示します。

5.4 TATARA が表示するもの

5.4.1 バナーと要約行

ソースをアセンブルするときは、いつも最初に**バナー**（banner）が表示されます。プログラムの名前とバージョン、著作権表示、Web アドレスで、そのあとに空行が 1 行続きます。最後には**要約行**（summary line）が表示されます。ソースがどこで終わったか、つまり END のあるファイルと行、END がなければ最後の行です。どちらも 4 章で見ました。END が取り込んだファイルの中にあれば、要約行にはそのファイルの名前が出ます（13 章）。

/Q を付けると、どちらも表示されません。エラーは/Q の有無にかかわらず表示されるので、/Q を付けて何も表示されなければ、成功したということです。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara /q hello.as hello.tro

A:\TATARA\EXAMPLES\HELLO>
```

/V はバナーを表示して終わります。ソースを指定しても、アセンブルはしません。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara /v
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools


A:\TATARA\EXAMPLES\HELLO>
```

5.4.2　使い方の画面

/?は、バナーと**使い方の画面**（usage screen）を表示して終わります。使い方の画面は、コマンド・ラインの要約です。ファイル名を 1 つも書かずに **tatara** とだけ入力しても、同じ画面が表示されます。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara /?
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

Usage:  TATARA [options] <source> [output]
        TATARA /P [options] <source> [output]
        TATARA /L [options] <source> <output> <listing>

Options:  /P Assemble to the screen
          /L Write a listing file, named third on the line
          /S Print the symbol table when the assembly ends
          /C Make symbol and macro names case-sensitive
          /F Print the fields of each line, for diagnosis
          /M Print every macro definition, for diagnosis
          /H Print how many heap blocks are still in use
          /Q Omit the banner above and the summary line
          /V Print the program version
          /? Print this screen

<source>  : The file to assemble, M80 syntax (.as, .asm or .z80)
<output>  : The relocatable object it produces, usually .tro
<listing> : The listing file, usually .lst or .prn
```

この画面は MSX の画面の 24 行をすべて使うので、コマンドの行は上にスクロールして見えなくなります。

5.5 リスティング

リスティング（listing）とは、TATARA がアセンブルしたソースを印刷用に整えたもので、ページに分かれ、各ページに見出しが付きます。/P か/L を付けたとき、または 3 つ目のファイル名を書いたときに作られます。

/P は、リスティングを画面に表示します。次のセッションは、その始めの部分だけを示しています。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara /p hello.as
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

        Tatara v1.2.0   29-Sep-26       PAGE    1


; HELLO.AS - the smallest complete program.
;
; One source, one object, one .COM. It prints a line and gives the
; machine back to MSX-DOS.
```

各ページはフォーム・フィードで始まり、MSX ではこれで画面が消えます。そのため、1 ページより長いリスティングは、読むより早く流れていってしまいます。/P は、短いソースをさっと確認するのに向いています。

/L は、3 つ目に書いたファイルにリスティングを書き出します。このとき、各行の前にアドレスとバイトが付きます。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara /l hello.as hello.tro hello.lst
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ended at HELLO.AS(22)
```

リスティングには、ソースが取り込んだすべてのファイルの行が含まれ、最後にすべてのシンボルの表が付きます。そのため、HELLO.LST でさえ約 20 KB になり、その大部分は MSXDOS.INC です。リスティングの内容と、ソースからそれを制御する方法は、18 章で説明します。

5.6　TATARA が実行を始めないとき

表 5.3 は、TATARA が 1 行もアセンブルしないうちに止まるときのメッセージです。TATARA は最初にコマンド・ラインを調べるので、一部のメッセージはバナーより前に表示されます。

メッセージ	意味
`ERROR: Tatara needs MSX-DOS2 or Nextor.`	MSX が MSX-DOS1 で動いています。
`ERROR: Tatara needs a memory mapper.`	MSX にメモリ・マッパーがありません (1 章)。
`ERROR: unknown option.`	スラッシュのあとに、TATARA のオプションではない文字が続いているか、何も続いていません。
`ERROR: too many filenames.`	ファイル名が 4 つ以上あります。
`ERROR: the output file is the input file.`	オブジェクト・ファイルがソースと同じ名前です。
`ERROR: cannot open` *name*	ソース・ファイルが存在しないか、名前を拡張子なしで入力しました。
`ERROR: cannot create` *name*	ディスクがいっぱいか、書き込み禁止になっているなどの理由で、オブジェクト・ファイルを書き出せません。

表 5.3: アセンブルの前に表示されるメッセージ

そのうちの 2 つは、次のように表示されます。

MSX-DOS

```
A:\TATARA\EXAMPLES\HELLO>tatara /x hello.as
ERROR: unknown option.

A:\TATARA\EXAMPLES\HELLO>tatara a b c d
ERROR: too many filenames.
```

第 6 章

ソースの書式

6.1 行 41
6.2 4 つのフィールド 42
6.3 ラベルを書く場所 42
6.4 コメント 44
6.5 大文字と小文字 44
6.6 スペース、タブ、行の長さ、行末 45

この章では、ソース・ファイルの書き方を説明します。1 行とは何か、TATARA が 1 行を分ける 4 つの部分、そしてラベル、コメント、大文字と小文字、スペースとタブ、行の長さ、行末の規則です。TATARA が、ある行について「not a directive, a macro or an instruction」と表示したのに、その行が正しく見えるときは、この章を読み返してください。

6.1 行

ソース・ファイルはテキスト・ファイルで、TATARA はそれを 1 行ずつ読みます。それぞれの行は、次の 3 つのどれかです。

- 空行。TATARA は読み飛ばします。
- コメント行。読む人のためのコメントだけが書かれた行です。
- **ステートメント**（statement）。Z80 への命令、TATARA への疑似命令、またはマクロの名前（16 章）が書かれた行です。

ソース・ファイルは、プレーン・テキストを保存できるエディターであれば、MSX でも別のコンピューターでも書けます。PC や Mac で書いたファイルをそのままアセンブルできる理由は、6.6 節で説明します。

6.2　4 つのフィールド

TATARA は、どの行も最大 4 つの**フィールド**（field）に分けます。順番はいつも同じです。

```
label      operation  operand       ; comment
```

4 つのフィールドがすべてそろった行を、図 6.1 に示します。

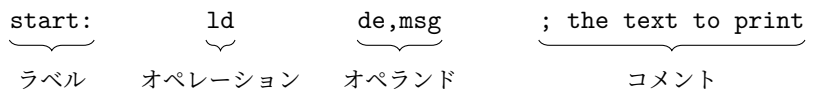

図 6.1: ソースの 1 行と、その 4 つのフィールド

ラベル　その行のアドレスに付ける名前です（2 章）。`start:` がその例です。ラベルを書ける場所は次の節で、名前に使える文字は 7 章で説明します。

オペレーション　その行が何をするかを表します。`ld` のような命令、`db` のような疑似命令、またはマクロの名前です。そのあとの最初のスペースかタブで終わります。このフィールドを**オペレーション**（operation）といいます。

オペランド　オペレーションが扱う対象で、`de,msg` がその例です。オペレーションのあとからコメントまで、コメントがなければ行の終わりまで続き、スペースを含むこともできます。たとえば `db "Hello from Tatara."`では、`"Hello from Tatara."` 全体が**オペランド**（operand）です。オペランドに何を書けるかはオペレーションによって決まります（8 章から 11 章）。

コメント　読む人のための文章で、セミコロンから行の終わりまでです。TATARA はこれを無視します。

どのフィールドも省略できます。よく使われる行の形を表 6.1 に示します。

6.3　ラベルを書く場所

TATARA は、次の 2 つのどちらかの方法でラベルを見分けます。

1. 1 桁目、つまり行の最初の文字から始まる語は、コロンの有無にかかわらずラベルです。
2. 1 桁目から始まらない語は、コロンで終わっている場合だけラベルです。

行	フィールド
`start:`	ラベル
`start:  ld de,msg`	ラベル、オペレーション、オペランド
`    ret`	オペレーション
`    ld c,_TERM0`	オペレーション、オペランド
`    jp BDOS ; back to MSX-DOS2`	オペレーション、オペランド、コメント
`; HELLO.AS - the smallest complete program.`	コメント
（空行）	なし

表 6.1: よく使われる行の形

したがって、字下げした語にコロンがなければ、それはオペレーションです。次のソース INDENT2.AS では、start が字下げされていてコロンがないので、TATARA はこれをオペレーションとみなします。しかし、その名前のオペレーションはありません。

```
  start   ld      a,1
          end
```

MSX-DOS

```
INDENT2.AS(1): ERROR: not a directive, a macro or an instruction.
```

start のあとにコロンを付ければ、同じように字下げした行でも受け付けられます。

警告. 1 桁目から始まるものは、命令であってもすべてラベルになります。次のソース COL1.AS では、命令を字下げせずに入力しています。

```
ld      a,1
        end
```

TATARA は ld をラベル、a,1 をオペレーションとみなし、次のように表示して止まります。

MSX-DOS

```
COL1.AS(1): ERROR: not a directive, a macro or an instruction.
```

このメッセージが正しく見える行を指しているときは、その行がスペースかタブで始まっているかを確かめてください。

putstr:: のようにコロンを 2 つ付けて書いたラベルは、パブリックにもなります。これは PUBLIC 疑似命令と同じ働きです（12 章）。

サンプルは、次の決まりに従って書かれています。まねをするのにもよい書き方です。ラベルは 1 桁目から書いてコロンで終え、同じ行のそれ以外の部分と、ラベルのない行はすべて、タブで字下げします。

6.4　コメント

コメント（comment）はセミコロンで始まり、行の終わりまで続きます。コメントだけの行も書けます。その場合は、HELLO.AS の最初の数行のように、セミコロンを 1 桁目に置くことができます。

文字列の中のセミコロンは、コメントの始まりにはなりません。次の行では、文字列は"a;b"で、コメントは 2 つ目のセミコロンから始まります。

```
        db      "a;b"           ; the semicolon is part of the string
```

同じ行の中で閉じていない引用符は、文字列の始まりになりません。これは、アポストロフィーを含む唯一の Z80 命令で意味を持ちます。

```
        ex      af,af'          ; swap the two AF registers
```

6.5　大文字と小文字

命令、レジスター名、疑似命令は、大文字でも小文字でも、その混じったものでも書けます。LD A,B、ld a,b、Ld A,b は同じ行です。

名前も、TATARA を/C 付きで実行しないかぎり（5 章）、大文字と小文字を区別しません。/C を付けなければ、msg、MSG、Msg は同じ名前です。詳しくは 7 章で説明します。

文字列の中の文字は、大文字と小文字がそのまま保たれます。"Hello"と"HELLO"は別の文字列です。

サンプルのコードは小文字で書かれています。インクルード・ファイルは、MSX の資料と同じように、名前を大文字で書いています。たとえば ld c,_STROUT です。

6.6 スペース、タブ、行の長さ、行末

6.6.1 スペースとタブ

スペースとタブはフィールドを区切ります。2つのフィールドの間には、どちらでも、いくつでも、混ぜて使ってもかまいません。タブを使うとフィールドの縦の位置がそろうので、サンプルではタブを使っています。リスティングでは、タブは次の8の倍数の桁まで進みます（18章）。

6.6.2 行の長さ

1行は255文字まで書けます。それより長い行があると、TATARAは次のメッセージを表示して止まります。

MSX-DOS

```
ERROR: source line too long.
```

6.6.3 行末

テキスト・ファイルでは、それぞれの行の終わりを1つまたは2つの制御文字で表します。MSX-DOSとWindowsはCRとLFの2つを使い、macOSとLinuxはLFだけを使います。TATARAはLFで行を終え、CRは無視するので、どちらのファイルも読めます。そのため、別のコンピューターで書いたソースは、変換せずにMSXにコピーしてアセンブルできます。ファイルの最後の行は、行末の文字がなくても読み込まれます。

ごく古いMacintoshのプログラムが書き出すような、CRだけを使うファイルは、1つの長い行として読まれ、TATARAは上のメッセージを表示して止まります。

6.6.4 Ctrl-Z

MSX-DOSのエディターには、CP/Mからの習慣で、ファイルの最後の行のあとにCtrl-Z（文字コード`1Ah`）を書き込むものがあります。TATARAはCtrl-Zで読み込みをやめ、そのあとにあるものはすべて無視します。

第 7 章

名前

7.1 名前に使える文字 47
7.2 長さ 48
7.3 大文字と小文字 49
7.4 届きにくい名前 51

プログラムには**名前**（name）がたくさん出てきます。ラベル、EQU で定義した値、マクロ、セグメントなどです。2 章ではこれらをシンボルと呼びました。それぞれの種類の名前がどのように値を持つかは、12 章で説明します。この章では、名前の書き方を説明します。使える文字、長さ、大文字と小文字を区別するかどうか、そして避けたほうがよい名前です。

7.1 名前に使える文字

名前は、表 7.1 の文字を好きな順に並べて作ります。ただし、数字で始めることはできません。

文字	
A〜Z、a〜z	英字
0〜9	数字（先頭には使えない）
? @ . _ $	そのほかの 5 つの文字

表 7.1: 名前に使える文字

これは M80 で使える文字と同じです。次のソースの 4 つの名前は、どれもふつうのラベルです。

```
        cseg
?tmp:           nop
@loop:          nop
my.name:        nop
a_b$c:          nop
```

M80 が受け付ける名前のうち、IXH、IXL、IYH、IYL の 4 つは、レジスターなので TATARA では名前に使えません（9.5 節）。

ラベルにこれ以外の文字があると、TATARA は `a name may hold only letters, digits, ? @ . _ and the dollar sign.` で止まります。

多くのプログラムで使われているのはアンダースコアで、長い名前の単語をつなぐのに使います。たとえば print_string です。ほかの文字も使えますが、my.name や a_b$c は名前以外のものと読み違えられることがあるので、サンプルでは使っていません。

数字で始まる語は数です（8 章）。TATARA は 1st: をラベルとして受け付けますが、それを参照することはできません。jp 1st は 1st を書き方の誤った数として読み、次のように表示して止まります。

MSX-DOS

```
DIGIT2.AS(3): ERROR: bad expression.
```

7.2　長さ

シンボル、つまりラベルや EQU、DEFL で定義した値の名前は 255 文字まで書くことができ、すべての文字が意味を持ちます。最後の 1 文字だけが違う 2 つの名前は、別々のシンボルです。次のソースはサンプル NAMES（31 章）の一部で、最後の 5 文字だけが違う 2 つのシンボルを定義しています。

```
first_pass_symbol_table_high_water_mark_in_bytes        defl    1024
first_pass_symbol_table_high_water_mark_in_words        defl    512
```

M80 は名前の最初の 6 文字しか区別しませんでした。CP/M 時代のプログラムの多くが PRTSTR のような略語で書かれているのは、そのためです。TATARA では、略語を使う必要はありません。

シンボルの名前はオブジェクト・ファイルの中でも長さがそのまま保たれます。そのため、あるモジュールのパブリックな名前と、別のモジュールで外部として宣言した同じ名前は、TANREN がリンクするときに全体で照合されます（2 章）。

> 注．ソースの 1 行も 255 文字までなので（6 章）、255 文字の名前はそれだけで 1 行を使い切り、コロンを付ける余地もありません。命令のあとに書く名前は、さらに短くなります。タブ、jp、タブのあとに残るのは 251 文字です。どんなプログラムでも、数十文字の名前があれば十分です。

マクロ、セグメント、グループの名前はもっと短くなります。マクロの名前は 64 文字まで、セグメントやグループの名前（10 章）は 16 文字までです。それより長い名前は、切り詰められるのではなく、次のどちらかのメッセージで拒否されます。

- `a macro name may be at most 64 characters.`
- `a segment or group name may be at most 16 characters.`

7.3 大文字と小文字

M80 と同じように、名前はふつう大文字と小文字を区別しません。Counter、counter、COUNTER は同じ名前です。/C オプション（5 章）を付けると、これらは 3 つの別々の名前になります。

/C が影響するのは、自分で付けた名前だけです。命令、レジスター、疑似命令は、/C の有無にかかわらず、大文字でも小文字でも書けます（6 章）。

サンプル NAMES で、その違いを確かめられます。このサンプルは、大文字と小文字だけが違う 2 つのシンボルを、名前に新しい値を与え直せる DEFL で定義しています。

```
; NAMES.AS - names as long as you like, in either case.
;
; A SYMBOL MAY BE 255 CHARACTERS AND ALL OF THEM COUNT. M80 keeps the
; first six, which is why so much CP/M-era assembly is written in
; abbreviations. The two names below differ only in their last five
; characters and are two different symbols here. 255 is the limit, and
; the test suite has a 240-character name in it; these are 48, which is
; as long as a line can carry and still be read on an MSX screen.

first_pass_symbol_table_high_water_mark_in_bytes        defl    1024
first_pass_symbol_table_high_water_mark_in_words        defl    512

; THE SAME SIX LETTERS IN TWO CASES. Without /C they are ONE symbol,
; and the second line changes its value. With /C they are TWO, with
; values 1 and 2. The symbol table of each run says which - and DEFL
; is used rather than EQU because a redefinition is legal, so the
; source assembles cleanly in both modes.

Counter         defl    1
```

```
counter         defl    2

; DIRECTIVES, MNEMONICS AND REGISTER NAMES ARE CASE-INSENSITIVE IN
; BOTH MODES. /C is about the names you choose, not the ones the Z80
; already has.

                cseg

start:          LD      A,1
                ld      b,a
                DB      Counter
                db      counter
                ret

                end     start
```

/C を付けずにアセンブルすると、Counter と counter は 1 つのシンボルになり、2 行目でそれに新しい値 2 が与えられます。次は、/S が表示するシンボル表（19 章）の最後の部分です。

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>tatara /q /p /s names.as
...
ASEG - absolute

0002h  var        Counter
0200h  var        first_pass_symbol_table_high_water_mark_in_words
0400h  var        first_pass_symbol_table_high_water_mark_in_bytes
```

表には、最初に定義したときに書いたとおりの名前が表示されます。/C を付けると、値が 1 と 2 の 2 つのシンボルになります。

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>tatara /q /p /c /s names.as
...
ASEG - absolute

0001h  var        Counter
0002h  var        counter
0200h  var        first_pass_symbol_table_high_water_mark_in_words
0400h  var        first_pass_symbol_table_high_water_mark_in_bytes
```

/C は、大文字と小文字だけが違う名前がプログラムに必要なときにだけ使ってください。たとえば、大文字と小文字を区別するツールのために書かれたプログラムや、そのようなツールが作ったプログラムです。そうでなければ、区別しないほうが安全です。大文字と小文字を間違えて入力しても、正しいシンボルを指すからです。

警告. 1 つのプログラムのモジュールは、すべて同じ方法でアセンブルしなければなりません。すべて/C 付きか、すべて/C なしです。オブジェクト・ファイルにはどちらだったかが記録され、TANREN は混ざったものをリンクしません。混ざっていると、ある名前があるモジュールでは見つかり、別のモジュールでは見つからないことが起こるからです。次の例では、TWOMOD の MAIN.AS を/C 付きで、PUTSTR.AS を/C なしでアセンブルしてあります。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tanren @twomod
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ERROR: one module was assembled /C and another was not.
```

7.4 届きにくい名前

TATARA には予約語がありません。どんな名前でも定義します。しかし、ある場所では、名前を探す前に語を決まった一覧で調べるので、そこでは一覧の語と同じつづりの名前には届きません。そのような場所は 4 つあります。

オペレーション・フィールド TATARA は、まず疑似命令、次にマクロ、最後に命令を探します。疑似命令と同じ名前のマクロは呼び出すことができず、命令と同じ名前のマクロは、その命令の代わりに使われます（16 章）。

命令のオペランド オペランドの最初の語がレジスターの名前と完全に一致すると、その語はレジスターになります。語は、スペース、タブ、コンマ、かっこ、+、-で終わります。そのため、c と c+0 はどちらもレジスター C で始まりますが、bad はそれ自体で 1 つの語なので、ふつうの名前です。レジスターは、A、B、C、D、E、H、L、BC、DE、HL、SP、AF、AF'、IX、IY、I、R です。

条件 JP、JR、CALL、RET のあとでは、条件 NZ、Z、NC、C、PO、PE、P、M が先に調べられます。

式 NOT、HIGH、LOW は常に演算子です。また、MOD、SHR、SHL、EQ、NE、LT、LE、GT、GE、AND、OR、XOR は、演算子を置ける場所ではどこでも演算子です（8 章）。high という名前のシンボルは定義できますが、db high は bad expression. と表示して止まります。

名前 c は、レジスターであり、条件であり、正しい名前でもあるので、このほとんどを一度に示してくれます。次のソースは c を定義し、4 回使っています。

```
c               equ     5
                cseg
                ld      a,c
                ld      a,0+c
                jp      c,c
                db      c
```

それぞれの行で TATARA が書き出すバイトを、リスティングから取って表 7.2 に示します。

行	バイト	c の意味
`ld a,c`	`79`	レジスター
`ld a,0+c`	`3E 05`	シンボル（5）
`jp c,c`	`DA 05 00`	条件、次にシンボル
`db c`	`05`	シンボル

表 7.2: c という名前のシンボルの 4 つの使い方

2 行目を `ld a,c+0` と書くと、オペランドがレジスターの名前で始まるので、エラーになります。

MSX-DOS

```
REGNAME.AS(4): ERROR: not a form this instruction has.
```

安全な決まりは簡単です。レジスター、条件、演算子、疑似命令、命令と同じつづりの名前を付けないことです。疑似命令は付録 A、命令は付録 B に一覧があります。

7.4.1　疑問符 2 つで始まる名前

疑似命令 `LOCAL`（16 章）は、マクロの中のラベルに名前を作ります。こうすると、マクロを使うたびにラベルが別のものになります。作られる名前は`??0000`、`??0001` のようなもので、ほかの名前と同じようにシンボル表に表示されます。

MSX-DOS

```
CSEG - default code segment, 0009h bytes

0000h              start
0002h              ??0000
0006h              ??0001
```

この形の名前を自分で定義しないでください。LOCAL が作る名前とぶつかるおそれがあります。

7.4.2 ドル記号だけの場合

$だけのときは名前ではなく、ロケーション・カウンター、つまりいまの行のアドレスです（8 章と 10 章）。a_b$c のように名前の中にあるときは、ふつうの文字です。

第 8 章

式

8.1 値は 16 ビット 55
8.2 数 56
8.3 文字 56
8.4 名前とロケーション・カウンター 57
8.5 演算子 57
8.6 値の種類 58

命令や疑似命令が数を必要とする場所には、どこでも**式**（expression）を書くことができます。式とは、数、名前、またはそれらを組み合わせた計算のことで、`10`、`msg`、`msg+2`、`(count-1)*4` などがその例です。TATARA は、その行をアセンブルするときに式の値を計算します。この章では、数と文字の書き方、演算子の働きと計算の順序、そして式が持つ値の種類を説明します。例には、それぞれの式の値を 2 バイトで格納する `DW` を使います（11 章）。

8.1 値は 16 ビット

TATARA が計算する値は、すべて 0 から 65535（`0FFFFh`）までの 16 ビットの数です。どちらかの端を越える計算は、反対側に回り込みます。`65535+1` は 0 になり、`0-1` は `0FFFFh` になります。そのため、負の数は、Z80 自身が扱うのと同じように、16 ビットの補数として格納されます。`-1` は `0FFFFh`、`-6` は `0FFFAh` です。

`ld a,...` や `DB` のオペランドのように、1 バイトしか必要としない場所では、TATARA は値の下位バイトを使います。そこで受け付けられる値については、9 章と 11 章で説明します。

8.2　数

数は、表 8.1 のように、後ろに付けた文字がほかの基数を示さないかぎり、10 進数です。この文字は大文字でも小文字でもかまいません。

基数	接尾辞	例	値
10 進数	なし、または D	100、100d	100
16 進数	H	100h	256
8 進数	O または Q	100o、100q	64
2 進数	B	100b	4

表 8.1: それぞれの基数の数

数は必ず数字で始まります。英字で始まる語は名前です（7 章）。そのため、最初の桁が英字になる 16 進数には、前に 0 を付ける必要があります。0ffh は 255 ですが、ffh は名前なので、定義していなければ TATARA は次のように表示して止まります。

MSX-DOS

```
HEXNAME.AS(2): ERROR: undefined symbol in an expression.
```

警告. B と D は 16 進数の数字であり、接尾辞でもあります。どちらかで終わる数は、2 進数または 10 進数として読まれます。10b は 10Bh ではなく 2 です。16 進数は、0abh や 12bh のように、必ず H で終えてください。

12b の 2 のように、その基数にない数字があると、TATARA は止まります。

MSX-DOS

```
BADDIGIT.AS(2): ERROR: bad expression.
```

8.3　文字

文字定数（character constant）は、アポストロフィーまたは引用符で囲んだ 1 文字か 2 文字です。その値は文字のコードで、'A' と"A"はどちらも 41h です。2 文字の場合は、最初の文字が上位バイトに入るので、"AB"は 4142h になります。

2 種類の引用符は同じ働きをするので、囲みたい文字がそのどちらかのときに便利です。"'"はアポストロフィーです。また、定数の中では、それを囲んでいる記号

を 2 つ続けて書くと、その記号そのものを表します。そのため、''''もアポストロフィーです。

3 文字以上の文字列は DB で使います（11 章）。

8.4 名前とロケーション・カウンター

式の中の名前は、その値を表します。ラベルのアドレス、または EQU や DEFL で与えた数です（12 章）。TATARA はソースを 2 回読むので、名前は定義する行より前でも使えます。次のソースでは、jp later が正しくアセンブルされます。

```
                cseg
                jp      later           ; defined below
later:          ret
```

どこにも定義されていない名前があると、TATARA は止まります。

MSX-DOS

```
UNDEF.AS(2): ERROR: undefined symbol in an expression.
```

$だけのときは、**ロケーション・カウンター**（location counter）を表します。これは、その行のいまのセグメントでのアドレスです（10 章）。よく使われるのは jr $で、自分自身にジャンプして、そのまま待ち続けます。

8.5 演算子

演算子を表 8.2 に示します。英語の単語で書く演算子は、大文字でも小文字でも書けますが、両側にスペースかかっこが必要です。aandb ではなく a and b と書きます。

いくつかの演算子には、補足が必要です。

- 比較の演算子は、真なら 0FFFFh、偽なら 0 を返します。そのため、その結果を AND、OR、NOT で組み合わせることができます。比較は符号なしの数として行われます。-1 は 0FFFFh なので、-1 gt 1 は真です。
- 0 で割ってもエラーにはなりません。M80 と同じように、結果は 0FFFFh です。
- HIGH と LOW は、すぐあとの値にだけ働きます。high 1234h+1 は (high 1234h)+1 で、13h です。

演算子	働き	例と値
+ -	加算、減算	10-4-3 = 3
* /	乗算、除算	17/5 = 3
MOD	除算の余り	17 mod 5 = 2
SHL SHR	左シフト、右シフト	1 shl 4 = 10h
AND OR XOR	ビットごとの論理積、論理和、排他的論理和	0f0h and 3ch = 30h
NOT	すべてのビットを反転	not 0 = 0FFFFh
HIGH LOW	上位バイト、下位バイト	high 1234h = 12h
EQ NE	等しい、等しくない	1 eq 1 = 0FFFFh
LT LE	より小さい、以下	2 le 1 = 0
GT GE	より大きい、以上	2 gt 1 = 0FFFFh
-（値の前）	符号反転	-1 = 0FFFFh

表 8.2: 演算子

8.5.1　優先順位

演算子が 2 つ以上ある式では、それぞれの**優先順位**（precedence）によって、どれを先に計算するかが決まります。表 8.3 に、結び付きの強いものから順にその段階を示します。同じ段階の演算子は左から右へ計算されるので、10-4-3 は (10-4)-3 です。かっこを使うと順序を変えられます。2+3*4 は 14 で、(2+3)*4 は 20 です。

これは M80 と同じ段階です。覚えておくとよいことが 2 つあります。-2*3 は -(2*3) で、not a eq b は not (a eq b) です。迷ったときは、かっこを付けてください。かっこには何の負担もありません。

8.6　値の種類

2 章で説明したように、TATARA はすべてのアドレスを知ることはできず、TANREN が埋めるための穴を残します。そのため、どの式も、値だけでなく種類を持っています。

アブソリュート 10、'A'、EQU で定義した名前のように、TATARA がアセンブルし

段階	演算子
1	HIGH LOW
2	* / MOD SHL SHR
3	値の前の-
4	+ -
5	EQ NE LT LE GT GE
6	NOT
7	AND
8	OR XOR

表 8.3: 演算子の優先順位（結び付きの強い順）

た時点で完成している数です。このような値を**アブソリュート**（absolute）といいます。

リロケータブル コード・セグメントのラベルのような、セグメントの中のアドレスです。TATARA が知っているのはセグメントの先頭からの距離だけで、セグメントを置いたアドレスは TANREN が加えます。

外部 EXTRN で宣言した、ほかのモジュールが定義する名前です。TATARA はその値について何も知らず、TANREN が埋めます。

式が役に立つのは、TANREN がアドレスを 1 つ加えるだけで完成させられる場合だけです。そのため、TATARA が受け付けるのは表 8.4 の組み合わせだけです。それ以外は、両側ともアブソリュートでなければなりません。

式	結果	例
アブソリュート＋リロケータブル、またはその逆	リロケータブル	a1+2
リロケータブル − アブソリュート	リロケータブル	a1-2
リロケータブル − リロケータブル（同じセグメント）	アブソリュート	a2-a1
外部＋アブソリュート、またはその逆	外部	far+2
外部 − アブソリュート	外部	far-2

表 8.4: TATARA が受け付ける組み合わせ

このうちもっとも役に立つのは、同じセグメントの 2 つのラベルの差です。これは長さを表し、セグメントをどこに置いても変わりません。次のソースでは、a2-a1 は 1、a1+2 は a1 の 2 バイト後ろのアドレスですが、a1*2 はコードを移動すると意味がなくなります。

```
                cseg
a1:             nop
a2:             nop
                dw      a2-a1           ; absolute: 1
                dw      a1+2            ; relocatable
                dw      a1*2            ; not allowed
```

MSX-DOS

```
RELOC.AS(6): ERROR: relocation error - a segment-relative value is not allowed
    here.
```

外部の名前は、1 つの式の中で 1 回だけ、数を加えるか引くかして使えます。2-far は受け付けられません。

MSX-DOS

```
EXTRN.AS(5): ERROR: an external symbol may not be used here.
```

符号反転、NOT、HIGH、LOW にも、アブソリュートの値が必要です。また、セグメントはそれぞれが 1 つの種類なので、別々のセグメントにある 2 つのラベルを引き算することもできません（10 章）。

第 9 章

命令

9.1 命令セット 61
9.2 オペランド 62
9.3 相対ジャンプ 63
9.4 R800 の命令 64
9.5 未公開命令 65
9.6 オペランドが正しくないとき 68

この章では、Z80 の命令と、R800 が追加した 2 つの命令を、TATARA 向けにどう書くかを説明します。オペランドの形、インデックス・レジスターとディスプレースメント、相対ジャンプ、そしてオペランドが正しくないときに TATARA が表示するメッセージです。それぞれの命令が何をするかは説明しません。それは、Z80 のプログラミングを学ぶ本や講座の役目です。参考のため、付録 B に命令を種類ごとに一覧にしてあります。

9.1 命令セット

TATARA が受け付けるのは、ザイログが書くとおりの形で書いた Z80 の命令、ザイログが公表しなかったもののどの Z80 でも動く**未公開命令**（undocumented instruction、9.5 節）、そして R800 の `MULUB` と `MULUW`（9.4 節）で、全部で 70 個のニーモニックです。6 章で説明したように、ニーモニックはオペレーション・フィールドに、そのオペランドはコンマで区切ってオペランド・フィールドに書きます。

9.2　オペランド

オペランドは、表 9.1 のどれかの形をしています。どの命令がどの形をとるかは Z80 によって決まっていて、Z80 の本に一覧があります。表のバイトは、それぞれの例に対して TATARA が書き出すものです。

形	例	バイト
レジスター	`ld a,b`	`78`
即値	`ld a,5`	`3E 05`
	`ld hl,1234h`	`21 34 12`
レジスター間接	`ld a,(hl)`	`7E`
	`ex (sp),hl`	`E3`
	`jp (hl)`	`E9`
インデックス	`ld a,(ix+5)`	`DD 7E 05`
	`ld a,(ix-1)`	`DD 7E FF`
	`ld a,(iy)`	`FD 7E 00`
メモリのアドレス	`ld a,(1234h)`	`3A 34 12`
	`ld hl,(1234h)`	`2A 34 12`
ポート	`in a,(98h)`	`DB 98`
	`out (c),a`	`ED 79`
条件	`jr nz,$`	`20 FE`
ビット番号	`set 0,(ix+3)`	`DD CB 03 C6`
決まった値	`rst 38h`	`FF`
	`im 1`	`ED 56`

表 9.1: オペランドの形

9.2.1　値とかっこ

命令が数、アドレス、ポートをとる場所には、どんな式でも書けます（8 章）。かっこのないオペランドは**即値**（immediate value）で、その数そのものが命令の中に入ります。かっこで囲んだオペランドはメモリのアドレスで、`IN` と `OUT` ではポートです。`ld a,5` は A に数 5 を入れ、`ld a,(5)` はアドレス 5 に格納されているバイトを A に入れます。

`ld a,5` のように 1 バイトを占める即値は、式の下位バイトです。TATARA は、値がそこに収まるかどうかを確かめません。`ld a,-1` は意図したとおり `0FFh` を書き

出しますが、`ld a,300` は 300 の下位バイトである `2Ch` を、何も表示せずに書き出します。また、1 バイトには、TANREN があとで仕上げる必要のあるアドレスを入れられないので、1 バイトの値はアブソリュートでなければなりません（8.6 節）。`ld hl,msg` のような 2 バイトの値は、リロケータブルでも外部でもかまいません。

9.2.2 インデックス・レジスター

インデックスのオペランドは `(IX+d)` または `(IY+d)` で、`d` は**ディスプレースメント**（displacement）です。これはインデックス・レジスターに加える −128 から 127 までの符号付きの数で、`+`でも`-`でも書けます。また、`(ix-1)` や `(ix+count*2)` のように、アブソリュートな式であれば何でもかまいません。`(IX)` だけのときは `(IX+0)` の意味です。範囲外のディスプレースメントがあると、TATARA は止まります。

MSX-DOS

```
DISP.AS(2): ERROR: an index displacement must be -128 to 127.
```

9.3 相対ジャンプ

`JR` と `DJNZ` は**相対ジャンプ**（relative jump）です。命令の中には、ジャンプ先のアドレスではなく、そこまでの距離が符号付きの 1 バイトで入ります。ソースには、`JP` と同じようにジャンプ先を書けば、TATARA が距離を計算します。たとえば `jr nz,loop` や `djnz loop` です。

距離は次の命令から測り、−128 から 127 バイトまででなければなりません。自分自身にジャンプする `jr $`は距離 −2 で、`18 FE` にアセンブルされます。それより遠いジャンプ先があると、TATARA は止まります。

MSX-DOS

```
JR.AS(2): ERROR: a JR or DJNZ can only reach -128 to 127.
```

ジャンプ先は、ジャンプと同じセグメントにあるか、アブソリュートなアドレスでなければなりません。TANREN が別の場所に置くコードまでの距離は、プログラムをリンクするまでわからないからです。遠すぎるジャンプ先や、別のモジュールにあるジャンプ先には `JP` を使ってください。

9.4　R800 の命令

MSX turbo R のプロセッサーである R800 は、Z80 の命令に 2 つのかけ算の命令を加えています。TATARA が受け付ける 3 つの形を表 9.2 に示します。

命令	バイト	働き
`mulub a,`*r*	`ED C1+8`*r*	A × *r*、結果は HL
`muluw hl,bc`	`ED C3`	HL × BC、結果は DE:HL
`muluw hl,sp`	`ED F3`	HL × SP、結果は DE:HL

表 9.2: R800 の命令

r は 8 ビットのレジスターで、`mulub a,b` は `ED C1` です。TATARA はどのレジスターでも受け付けますが、R800 の資料が結果を保証しているのは `B`、`C`、`D`、`E` だけです。最初のオペランドは、`MULUB` では `A`、`MULUW` では `HL` でなければなりません。

> 警告. TATARA はどの MSX でもこれらの命令をアセンブルしますが、それを使ったプログラムが正しく動くのは turbo R だけです。Z80 では、かけ算を行いません。

9.4.1　R800 のニーモニック

命令は、それぞれ数バイトとして格納されます。最初の 1 バイト、または `DD` や `ED` のようなプレフィックスがある場合は最初の 2 バイトが、どの命令なのかを表します。この部分を**オペコード**（opcode）といいます。残りの部分があれば、そこには数、アドレス、ディスプレースメントなどのオペランドが入ります。プロセッサーが読むのはバイトだけです。`LD` や `DJNZ` のような**ニーモニック**（mnemonic）は、数を覚えなくてもプログラムが書けるように、人がオペコードに付けた名前です。

ニーモニックは名前にすぎないので、同じオペコードに別の名前の組を付けることもできます。『MSX-Datapack』に収められているアスキーの R800 の資料は、命令全体に新しいニーモニックを付けています。`JP` は `br`（branch）、`DJNZ` は `dbnz`、`EX` は `xch`、`RES` は `clr`、`LDIR` は `movem` です。レジスター名はドットで始まり、角かっこはメモリを表し、アキュムレーターも必ず書きます。ザイログの `SUB B` は `sub .a,.b` になります。2 つのかけ算を除けば、R800 の命令はすべて、名前を変えた Z80 の命令と同じオペコードを持ち、同じ動作をします。

256 バイトのメモリを 0 にする短いルーチンを、両方の書き方で表 9.3 に示します。バイトは同じです。

`B` は 256 回目の減算でようやく 0 になるので、`ld b,0` とすると `DJNZ` は 256 回繰り返します。

	Z80	R800	バイト
clear:	ld hl,buffer	ld .hl,buffer	21 *nn nn*
	ld b,0	ld .b,0	06 00
	xor a	xor .a,.a	AF
loop:	ld (hl),a	ld [.hl],.a	77
	inc hl	inc .hl	23
	djnz loop	dbnz loop	10 FC
	ret	ret	C9

表 9.3: 同じルーチンの Z80 と R800 のニーモニック

この文章を書いている時点では、TATARA が受け付けるのは Z80 のニーモニックだけです。R800 のニーモニックで書かれたプログラムは、TATARA でアセンブルする前に Z80 のニーモニックに書き直す必要があります。そうしてできるプログラムは同じものです。

9.5 未公開命令

ザイログは Z80 の命令を公表しましたが、プロセッサーが実行するオペコードのすべてではありませんでした。残りのうちのいくつかは役に立つ働きをし、どの Z80 でも同じように動き、何十年も MSX のプログラムで使われてきました。TATARA は、そのうちの次の 4 つのグループを受け付けます。

- `SLL`。左へシフトし、ビット 0 を 1 にします。
- **インデックス・ハーフ**の `IXH`、`IXL`、`IYH`、`IYL`。`IX` と `IY` の上位と下位のバイトを、8 ビットのレジスターとして使います。
- `IN F,(C)`。ポートを読んでフラグを設定し、値は残しません。
- `OUT (C),0`。ポートに決まった値を書きます。

R800 の命令と同じように、これらはどの MSX でもアセンブルされ、有効にするオプションはありません。M80 にはどれもありません（付録 J）。表 9.4 にその書き方を示します。付録 B には、104 の書き方すべてをバイトとともに挙げます。

書き方	バイト
sll *r*（B、C、D、E、H、L、A）	CB 30+*r*
sll (hl)	CB 36
sll (ix+*d*)、sll (iy+*d*)	DD CB *d* 36、FD CB *d* 36
ld *x*,*r* と ld *r*,*x*（B、C、D、E、A）	IX なら DD、IY なら FD のあとに、上位の半分なら H、下位の半分なら L にした同じ命令のバイト。ld ixh,b は DD と、ld h,b の 60
ld *x*,*y*（同じレジスターの半分どうし）	同じ。ld ixh,ixl は DD 65
ld *x*,*n*	同じ。ld ixl,5 は DD 2E 05
inc *x*、dec *x*	同じ。inc iyh は FD 24
add a,*x*、adc a,*x*、sub *x*、sbc a,*x*、and *x*、xor *x*、or *x*、cp *x*	同じ。cp ixl は DD BD
in f,(c)	ED 70
out (c),0	ED 71

表 9.4: 未公開命令。*r* は 8 ビットのレジスターで、コードは付録 B のもの。*x* と *y* はインデックス・ハーフ

9.5.1　インデックス・ハーフの仕組み

IXH のためのオペコードはありません。命令の前に DD を置くと、その命令は HL の代わりに IX を使い、それが半分にも及びます。H は IXH に、L は IXL になります。ld a,h は 7C なので、ld a,ixh は DD 7C です。FD なら、同じ命令が IY を使います。

プレフィックスは 1 つのオペランドではなく命令全体にかかり、それでどの書き方があるかが決まります（図 9.1）。

- ld h,ixl は書けません。DD 65 は両方のオペランドを半分にするので、ld ixh,ixl になります。
- ld ixh,iyl は書けません。1 つの命令には 2 つのプレフィックスのどちらか一方しか付けられません。
- ld ixh,(hl) は書けません。プレフィックスがあると (HL) は (IX+*d*) になり、DD 66 *d* は ld h,(ix+*d*) です。

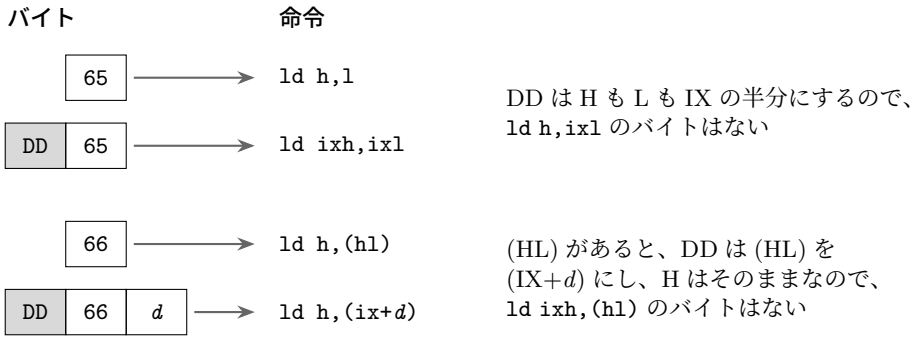

図 9.1: プレフィックスは命令全体にかかる。DD が 2 つの命令にどう働くか

Z80 にその書き方がない場所でも、インデックス・ハーフは拒否されます。ローテートとシフト、BIT、SET、RES、IN と OUT、そして ld (ix+5),ixh のようにインデックス付きのアドレスと並ぶ場合です。どれも、TATARA は not a form this instruction has. で止まります。

9.5.2 プログラムが当てにできないこと

警告. **MSX turbo R での SLL。**R800 は CB 30 から CB 37 を実行しますが、SLA として実行し、ビット 0 は 0 のままです。B が 81h のとき、sll b は Z80 では 03h、R800 では 02h を残します。キャリーはどちらでも同じなので、違いが見えるのはビット 0 だけです。SLL を使うプログラムは、turbo R では同じように動きません。インデックス・ハーフ、IN F,(C)、OUT (C),0 は、R800 でも Z80 と同じように動きます。

警告. **OUT (C),0 はいつも 0 を書くとは限りません。**NMOS の Z80 は 0 を書き、CMOS の Z80 は 0FFh を書きます。openMSX では、どちらのプロセッサーも 0FFh を書きました。プログラムは書かれる値を当てにできません。

IN F,(C) は A を変えず、IN A,(C) と同じようにフラグを設定します。使い道は、ポートのビットをフラグで調べることだけです。

9.5.3 綴りと名前

これらの命令の綴りはそれぞれ 1 つだけです。SL1 や SLIA ではなく SLL、HX、LX、HY、LY ではなく IXH など、IN (C) ではなく IN F,(C)、そして 00h や式ではなく数字の 0 そのものを書いた OUT (C),0 です。ほかの綴りでは、TATARA が知らない語に対するいつものメッセージが出ます。

IXH、IXL、IYH、IYL はレジスターなので、もう名前には使えません。M80 はこれらをラベルとして受け付けますが、TATARA でそのようなラベルがあると、ld a,ixh の意味がメッセージなしで変わってしまいます。そのため TATARA はこれを拒否し、PUBLIC と EXTRN でも拒否します。

MSX-DOS

```
IXHNAM.AS(2): ERROR: IXH, IXL, IYH and IYL are registers, not names.
```

HX、LX、HY、LY はふつうの名前のままなので、これらを座標に使っているプログラムはこれまでどおりアセンブルできます。

ほかの 2 つのグループの未公開オペコードは受け付けません。rlc (ix+2),b のように、結果をメモリとレジスターの両方に入れる書き方と、ED グループの中で NEG、RETN、NOP を繰り返す余分なオペコードです。

9.6 オペランドが正しくないとき

TATARA が使えないオペランドに対して表示するメッセージを、それぞれの原因となる行とともに表 9.5 に示します。

行	メッセージ
ld hl,a	not a form this instruction has.
mulub b,c	not a form this instruction has.
ld a,(ix+200)	an index displacement must be -128 to 127.
jr far	a JR or DJNZ can only reach -128 to 127.
rst 5	RST takes 0, 8, 10h and so on to 38h.
im 3	IM takes 0, 1 or 2.
bit 8,a	a bit number must be 0 to 7.

表 9.5: オペランドについてのメッセージ

not a form this instruction has. は一般的なメッセージで、命令はあるものの、そのオペランドの形はないという意味です。Z80 は 8 ビットのレジスターを 16 ビットのレジスターにコピーできないので、ld hl,a がその例です。その命令にどの形があるかは、Z80 の本か付録 B で確かめてください。

第 10 章

セグメントとロケーション・カウンター

10.1 ロケーション・カウンター 69
10.2 コード、データ、アブソリュート 70
10.3 ORG 71
10.4 領域の確保：DS 72
10.5 名前付きセグメント 72
10.6 トランジェント・データ・セグメント 73
10.7 メッセージ 76

2 章では、セグメントを紹介しました。セグメントとは、コードまたはデータという 1 種類の内容を持つプログラムの領域で、TANREN がそれぞれ別々にメモリに配置するものです。この章では、セグメントについて詳しく説明します。TATARA がそれぞれのセグメントでアドレスを数える方法、セグメントを選び、そのアドレスを設定し、その中に領域を確保する疑似命令、関係のあるコードやデータをまとめておける名前付きセグメント、そして Tatara 独自の機能で、変数どうしでメモリを共有できるトランジェント・データ・セグメントです。

10.1 ロケーション・カウンター

TATARA はセグメントをアセンブルしながら、それぞれの行が作るバイトを数えます（2 章）。この数が**ロケーション・カウンター**（location counter）です。ラベルは、その行でのロケーション・カウンターの値を持ち、式の中の$はその値を読みます（8 章）。

ロケーション・カウンターは、セグメントごとに 1 つあります。ソースがセグメ

ントを切り替えると、TATARA はいまのカウンターをしまっておき、もう一方のセグメントのカウンターを使います。戻ってきたときは、止まったところから続けます。次のソースでは、コードとデータが交互に書かれていますが、それぞれの種類は別々に数えられます。

```
                cseg
start:          ld      hl,count
                inc     (hl)
                dseg
count:          ds      1
                cseg
again:          ret
                dseg
total:          ds      2
```

TATARA のリスティングには、各行のアドレスが表示されます。

MSX-DOS

```
 0000'   21 00 00             start:          ld      hl,count
 0003'   34                                   inc     (hl)
                                              dseg
 0000'                        count:          ds      1
                                              cseg
 0004'   C9                   again:          ret
                                              dseg
 0001'                        total:          ds      2
```

again はコード・セグメントのオフセット 4、つまり inc (hl) のすぐあとにあり、total はデータ・セグメントのオフセット 1、つまり count のすぐあとにあります。アドレスのあとのアポストロフィーは、そのアドレスがリロケータブル、つまり TANREN が仕上げるセグメントの中のオフセットであることを示します（2 章）。リスティングでは、コードとデータに同じ記号を使います。詳しくは 18 章で説明します。

10.2　コード、データ、アブソリュート

3 種類のセグメントと、それぞれを選ぶ疑似命令を表 10.1 に示します。ソースはコード・セグメントから始まるので、自分のデータを持たないプログラムには、疑似命令はまったく必要ありません。

コードとデータを分けておくと、TANREN はデータをコードのすぐあと以外の場所に置くことができます。たとえば、ROM カートリッジは変数を RAM に置かなければなりません（33 章）。TANREN は、ふつうはすべてのコード・セグメントを先に置き、そのあとにデータ・セグメントを置きます。その順序の詳細と、別のアド

疑似命令	セグメント	アドレス
CSEG	コード・セグメント	リロケータブル（TANREN が配置）
DSEG	データ・セグメント	リロケータブル（TANREN が配置）
ASEG	アブソリュート・セグメント	書いたとおりに固定

表 10.1: 3 種類のセグメント

レスを選ぶオプションについては、24 章で説明します。

アブソリュート・セグメント（absolute segment）では、どのアドレスも書いたとおりで、TANREN はそれを動かしません。4000h にある ROM カートリッジのヘッダーのように、決まったアドレスに置く必要があるコードやデータのためのものです。

> 警告. .COM のプログラムは 0100h に読み込まれ、ひと続きのバイトの塊になっています。4000h のように、そこから遠いアブソリュートなアドレスを使うと、ファイルはその間のすべてを含むことになり、約 16 KB の大きさになります。.COM のプログラムでアブソリュート・セグメントを使うのは、その理由がわかっている場合だけにしてください。

10.3 ORG

ORG は、いまのセグメントのロケーション・カウンターを設定します。アブソリュート・セグメントではオペランドはアドレスで、コード・セグメントとデータ・セグメントではセグメントの先頭からのオフセットです。

```
                aseg
                org     4000h
rom:            db      'AB'            ; at address 4000h
                cseg
                org     10h
code10:         nop                     ; 10h bytes into the code
```

ORG のオペランドは、TATARA がその行に最初に来たときにわかっていなければならないので、それより下で定義する名前は使えません。また、アブソリュートか、いまのセグメントの中のアドレスでなければなりません。データ・セグメントの中のアドレスは、コード・セグメントのロケーション・カウンターにはできないので、TATARA は 8 章のリロケーション・エラーを表示して止まります。

10.4　領域の確保：DS

DS（別名 DEFS）は、ロケーション・カウンターを指定したバイト数だけ進め、それによって変数のための領域を確保します。

```
                dseg
buffer:         ds      128             ; 128 bytes
count:          ds      1               ; 1 byte
```

バイト数は、ORG のオペランドと同じように、アブソリュートで、最初の読み込みでわかっていなければなりません。DS がとるのはバイト数だけで、領域を埋める値のような 2 つ目のオペランドを書くと bad expression. になります。DS はバイトを何も書き出さず、ロケーション・カウンターを進めるだけです。そのため、プログラムは変数を読む前に値を入れておく必要があります。

10.5　名前付きセグメント

プログラムには、コード・セグメントもデータ・セグメントも、それぞれ 2 つ以上置くことができます。CSEG や DSEG のあとに名前を書くと、**名前付きセグメント**（named segment）が選ばれます。

```
                cseg    music           ; all the music code
                dseg    strings         ; all the text
                dseg    variables       ; all the variables
```

名前付きセグメントは、同じ種類のふつうのセグメントと同じように働き、独自のロケーション・カウンターを持ちます。同じ名前を選ぶ行は、すべて同じセグメントに追加されます。そのため、1 つの目的のためのコードを、いくつもの場所や、いくつものソース・ファイルに分けて書いても、最後には 1 つにまとまります。TANREN は、リンクするすべてのモジュールから、同じ名前のセグメントをつなぎ合わせます。

次のソースでは、コード・セグメントとセグメント music が、それぞれ 2 回選ばれています。

```
                cseg
main:           nop
                cseg    music
play:           nop
                nop
                cseg
more:           nop
                cseg    music
stop:           ret
```

シンボル表（19 章）には、それぞれのセグメントがその大きさとともに表示され、それぞれのラベルが属するセグメントの下に表示されます。

MSX-DOS

```
CSEG - default code segment, 0002h bytes

0000h             main
0001h             more

music - named code segment, 0003h bytes

0000h             play
0002h             stop
```

TANREN は 2 つのセグメントを互いにどこに置くかわからないので、それぞれのセグメントは 1 つの値の種類になります（8.6 節）。そのため、別々のセグメントにあるラベルどうしの差は、リロケーション・エラーとして受け付けられません。

10.6 トランジェント・データ・セグメント

たいていのプログラムには、プログラムの一部分でしか使わない変数があります。ユーザーが入力した行を読むルーチンは、読んでいる間、その行を入れるバッファーを必要とします。報告を表示するルーチンは、作っている行を入れるバッファーを必要とします。2 つのルーチンが同時に動くことがなければ、2 つのバッファーが同時に使われることもないので、同じメモリを使うことができます。`.COM` のプログラムがコードとデータを合わせて 64 KB 未満で収めなければならない MSX では、この節約が大きな意味を持つことがあります。

ほかのアセンブラーでは、これはプログラマーの仕事です。プログラマーは、共有するアドレスを、たいていは `EQU` を使って手で計算し、変数の大きさが変わるたびに正しく直さなければなりません。Tatara では、**トランジェント・データ・セグメント**（transient data segment）を使って、これを自動的に行います。

10.6.1 しくみ

トランジェント・データ・セグメントは、名前のあとに `TRANSIENT` という語を付けた名前付きデータ・セグメントです。その変数は**グループ**（group）に分けられ、それぞれのグループは名前を付けた `GROUP` の行で始まります。

```
                dseg    scratch,transient
                group   reading         ; used only while reading
inbuf:          ds      64
inlen:          ds      2
                group   writing         ; used only while writing
outbuf:         ds      32
outlen:         ds      2
```

規則はわずかです。

- グループの中では、変数はふつうのデータ・セグメントと同じように順に並び、重なることはありません。`inbuf` と `inlen` は、同時に使うことができます。
- どのグループも、セグメントの先頭から始まります。`outbuf` は `inbuf` と同じアドレスにあります。
- セグメントの大きさは、いちばん大きいグループの大きさです。
- どの変数も、どこかのグループに入っていなければなりません。最初の `GROUP` より前に `DS` の行があると、エラーになります。

この例の 4 つの変数を、ふつうのデータ・セグメントとトランジェントなセグメントに置いたところを、図 10.1 に示します。ふつうのセグメントでは 100 バイトを使いますが、トランジェントなセグメントではグループ `reading` の大きさである 66 バイトで済み、`writing` の 34 バイトは何の負担にもなりません。

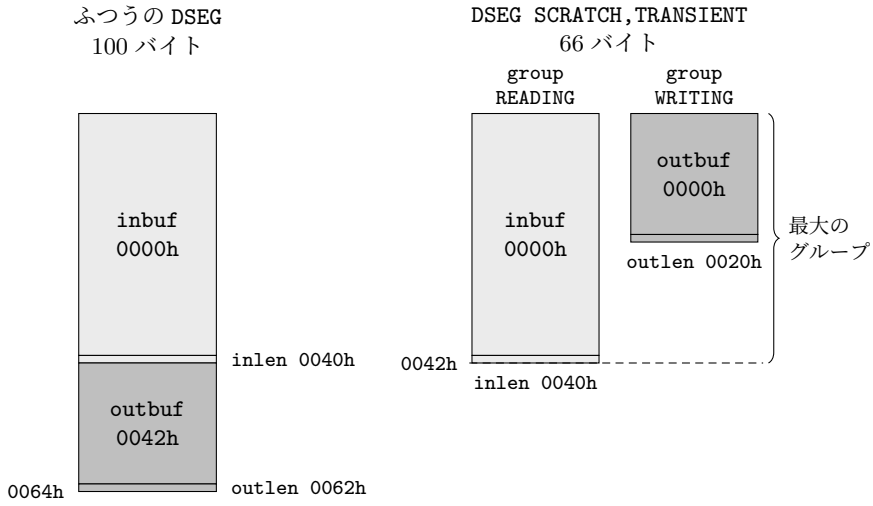

図 10.1: 同じ 4 つの変数を、ふつうのデータ・セグメントとトランジェントなセグメントに置いたところ（縮尺どおり）

TATARA のシンボル表には、セグメントとそれぞれのグループが、その大きさと、グループの中の各変数のオフセットとともに表示されます。

MSX-DOS

```
scratch - named data segment, transient, group reading, 0042h bytes

0000h             inbuf
0040h             inlen

scratch - named data segment, transient, group writing, 0022h bytes

0000h             outbuf
0020h             outlen
```

10.6.2 安全に使うために

グループが表すのは、どの変数が同時に使われるかということだけです。それ以外はプログラマーの責任です。TATARA はプログラムの流れを追って、2 つのグループが同時に使われないことを確かめたりはしません。

> 警告. あるグループの変数は、同じセグメントのほかのすべてのグループで同じオフセットにある変数と、同じアドレスを持ちます。一方に書き込むと、ほかのものが壊れます。`writing` を使うルーチンが、まだ表示していない行が `outbuf` に残っている間に `reading` を使うルーチンを呼び出すと、ユーザーが入力した行でその行が上書きされてしまいます。互いに使っている間も残っていなければならない変数は、同じグループに入れるか、別々のトランジェント・セグメントに入れるか、ふつうのデータ・セグメントに入れてください。

規則を補う点が 2 つあります。

- 互いに重なるのは、1 つのトランジェント・セグメントのグループどうしだけです。別々のセグメントがメモリを共有することはないので、交代で動くルーチンの組ごとに、トランジェント・セグメントを 1 つずつ持つことができます。
- グループの名前はグローバルです。別のモジュールにある同じセグメントの同じ名前のグループは同じグループで、TANREN がそれらをつなぎ合わせます。これを使った複数のモジュールからなるプログラムは、29 章で説明します。

別々のグループにあるラベルは、互いに引き算することができません。どちらも、もう一方の位置について何も示さないからです。TATARA はリロケーション・エラーを表示します。

10.7　メッセージ

セグメントについてのメッセージを、それぞれの原因となる行とともに表 10.2 に示します。

行	メッセージ
`dseg vars,forever`	`bad ASEG, CSEG or DSEG line.`
`dseg` と 17 文字の名前	`a segment or group name may be at most 16 characters.`
`dseg vars` のあとの `dseg vars,transient`	`this segment was declared differently before.`
コード・セグメントの中の `group oops`	`GROUP needs a name, no label, and a transient DSEG.`
最初の `GROUP` より前の `x: ds 1`	`this transient DSEG needs a GROUP first.`
`dw second-first`（2 つのグループまたは 2 つのセグメントのラベル）	`relocation error - a segment-relative value is not allowed here.`
`org v`（`v` は別のセグメント）	`relocation error - a segment-relative value is not allowed here.`

表 10.2: セグメントについてのメッセージ

プログラムに置けるセグメント、グループ、外部の名前の数には限りがあり、それを超えると `too many segments, groups or externals.` と表示されます。その限度は付録 E に一覧があります。

第 11 章

データ

11.1 DB：バイト 77
11.2 文字列 78
11.3 DW：ワード 78
11.4 DC：自分で終わりを示す文字列 79
11.5 別の名前 80
11.6 メッセージ 80

プログラムには、命令だけでなくデータも必要です。表示するメッセージ、数の表、アドレスなどです。この章では、プログラムにデータを置く疑似命令を説明します。バイトと文字列のための DB、16 ビットのワードのための DW、そして自分で終わりを示す文字列のための DC です。領域を確保するだけで中身を入れない DS は、10 章で説明しました。

11.1 DB：バイト

DB は、コンマで区切った式の並びをとり、それぞれを 1 バイトとして格納します。

```
table:          db      1,2,3           ; 01 02 03
                db      'A'+1           ; 42
                db      -1              ; FF
```

それぞれの式には、8 章で説明したものなら何でも書けます。命令の場合と同じように（9 章）、TATARA は値の下位バイトを格納し、値が収まるかどうかは確かめません。`db 300` は `2Ch` を格納します。

1 バイトには、TANREN があとで仕上げる必要のあるアドレスを入れられないので、値はアブソリュートでなければなりません。DB にラベルや外部の名前を書くと、TATARA はリロケーション・エラーを表示して止まります。

MSX-DOS

```
DBREL.AS(3): ERROR: relocation error - a segment-relative value is not allowed
    here.
```

11.2　文字列

DB の中の**文字列**（string）は、引用符で囲んだテキストで、1 文字につき 1 バイトとして格納されます。文字列と数は、1 つの並びの中に混ぜて書けます。

```
msg:            db      'Hello',13,10,'$'
```

これは 48 65 6C 6C 6F 0D 0A 24 というバイトになります。5 つの文字、次の行の先頭に移るための 13 と 10、そして MSX-DOS2 のファンクション _STROUT が表示を止める $ です（4 章）。

文字列は、アポストロフィーでも引用符でも囲めます。文字列の中では、囲んでいる記号を 2 つ続けて書くとその記号そのものを表し、もう一方の記号を使うこともできます。次の 2 行は、同じ 4 バイトを格納します。

```
                db      "It's"
                db      'It''s'
```

空の文字列 '' は、何も格納しません。

> 注. 1 文字か 2 文字の文字列は、文字定数（8 章）と同じ形をしています。TATARA は、M80 と同じように、閉じる引用符のあとに何があるかでこの 2 つを見分けます。コンマか行の終わりが続けば文字列で、そうでなければ式の中の文字定数です。そのため、db 'AB' は 41 42 の 2 バイトですが、db 'AB' and 0ffh は 1 バイトで、値 4142h から上位バイトを取り除いた 42h になります。

11.3　DW：ワード

DW は、それぞれの式を**ワード**（word）として格納します。ワードとは 2 バイトのことで、Z80 がすべての 16 ビットの値を格納するのと同じように、下位バイトが先に来ます（2 章）。dw 1234h は 34 12 です。

2 バイトには、TANREN が仕上げるアドレスを入れられるので、ここではラベル、ラベルをもとにした式、外部の名前を使えます（8.6 節）。そのため、ジャンプ先のルーチンの一覧のような、アドレスの表を作るには DW を使います。

```
jumps:          dw      start, again, finish
```

DW の中の 1 文字か 2 文字の文字列は文字定数です。dw 'AB' は、値 4142h を下位バイトから格納した 42 41 になります。それより長い文字列があると、TATARA は bad DB, DW or DC operand. と表示して止まります。

11.4 DC：自分で終わりを示す文字列

DC は文字列を 1 つとり、DB と同じように格納しますが、最後の文字のビット 7 をセットします。dc 'Stop' は 53 74 6F F0 で、最後のバイトは p の 70h に 80h を加えたものです。

ふつうのテキストの文字はすべてビット 7 が 0 なので、プログラムはそのビットで文字列の最後の文字を見分けることができ、$のような終わりの印を必要としません。次のルーチンは、DC で作った文字列を表示します。文字列のアドレスは HL に入れておきます。E の文字を表示する MSX-DOS2 のファンクション_CONOUT を、MSXDOS.INC の system マクロ（14 章）を使って呼び出しています。

```
print:          ld      a,(hl)          ; the next character
                inc     hl
                push    af              ; keep it, and its bit 7
                and     7fh             ; the character without bit 7
                ld      e,a
                push    hl              ; MSX-DOS2 may change HL
                system  _CONOUT
                pop     hl
                pop     af
                bit     7,a             ; was it the last one?
                jr      z,print         ; no: go round again
                ret
```

AND、BIT、スタックを初めて見る人は、Z80 のプログラミングを学ぶ本で確かめてください。ここで大事なのは、ビット 7 がセットされたバイトでループが自然に終わることです。

DC には、文字列をちょうど 1 つ、それも空でないものを書かなければなりません。空の文字列には印を付ける最後の文字がないからです。dc '' と dc 5 は、どちらも bad DB, DW or DC operand. と表示して止まります。

11.5　別の名前

いくつかの疑似命令には別の名前があります。ほかのアセンブラー向けに書かれたソースをそのままアセンブルできるように、TATARA はそれらも受け付けます。表 11.1 に一覧を示します。

別の名前	同じもの
DEFB、DEFM	DB
DEFW	DW
DEFS	DS

表 11.1: データの疑似命令の別の名前

11.6　メッセージ

データについてのメッセージを、それぞれの原因となる行とともに表 11.2 に示します。

行	メッセージ
db 1,,2	bad DB, DW or DC operand.
dw 'ABC'	bad DB, DW or DC operand.
dc ''、dc 5	bad DB, DW or DC operand.
db label、 db far（外部）	relocation error - a segment-relative value is not allowed here.

表 11.2: データについてのメッセージ

第 12 章

シンボルの定義と共有

12.1 名前に値を与える 3 つの方法 81
12.2 ラベル 82
12.3 EQU 82
12.4 DEFL 83
12.5 モジュールどうしで名前を共有する 83
12.6 END と開始アドレス 85
12.7 メッセージ 86

シンボルとは、数の代わりに使う名前です（2 章）。この章では、名前に値を与える 3 つの方法、1 つの名前は 1 つの値しか持たないという規則、モジュールどうしで名前を共有するための疑似命令、そしてソースを終わらせ、プログラムがどこから始まるかを示す `END` を説明します。

12.1 名前に値を与える 3 つの方法

表 12.1 に、その 3 つを示します。

書き方	値	変更
`name:`	その行でのロケーション・カウンター	できない
`name equ` *value*	与えた値	できない
`name defl` *value*	与えた値	できる

表 12.1: 名前に値を与える 3 つの方法

数に名前を付けると、プログラムが読みやすく、変更しやすくなります。`ld c,9`

では何をするのかわかりませんが、`ld c,_STROUT` ならわかります。また、その数が変わっても、直すのは名前を定義している行だけです。`MSXDOS.INC` の名前が `EQU` で定義されているのは、まさにこのためです（14 章）。

12.2　ラベル

ラベルは、それを書いた行でのロケーション・カウンターの値、つまりその行が作る最初のバイトのアドレスを持ちます（10 章）。ラベルは 1 回だけ定義します。同じ名前のラベルがもう 1 つあると、TATARA は止まります。

MSX-DOS

```
LABTWICE.AS(3): ERROR: this name already has a value.
```

12.3　EQU

`EQU` は、名前に固定の値、つまりそのあとに書いた式の値を与えます。

```
five            equ     5
screen          equ     4000h
here            equ     $               ; an address in the code
```

名前はコロンを付けずに書きます。コロンを付けると、TATARA はそれをラベルとみなし、`EQU` には名前がないことになってしまいます。`five: equ 5` と書くと、TATARA は `EQU, DEFL and MACRO take a name, not a label.` と表示して止まります。

値にはどんな式でも書けます（8 章）。`here` のように、アドレスでもかまいません。その場合、値は同じ場所にあるラベルと同じようにリロケータブルになります。ただし、外部の名前は書けません。外部の名前は、プログラムをリンクするまで値を持たないからです。

`EQU` で定義した名前は、それを定義する行より前でも使えます。次のソースでは、`later` を `EQU` の前の行で使っていますが、TATARA は正しく `0Ah` としてアセンブルします。

```
                ld      de,later
later           equ     five*2
```

`EQU` が与える値は変わりません。同じ名前を同じ値で定義し直すのはかまいませんが、別の値で定義すると、TATARA は `this name already has a value.` と表示して止まります。

12.4 DEFL

DEFL も名前に値を与えますが、その名前にはあとで、ソースに必要なだけ何度でも、新しい値を与えることができます。名前を使うそれぞれの行は、ソースのその位置での値を使います。

```
count           defl    0
                db      count           ; 00
count           defl    count+1
                db      count           ; 01
count           defl    count+1
                db      count           ; 02
```

DEFL で定義した名前は**変数シンボル**（variable symbol）と呼ばれることがあり、シンボル表には var と表示されます。変数シンボルは、ラベルや EQU と同じ名前を持てません。`n equ 1` のあとに `n defl 2` と書くと、TATARA は `this name already has a value.` と表示して止まります。EQU と同じように、名前はコロンを付けずに書きます。

DEFL がいちばん役に立つのは、条件付きアセンブルやマクロの中で、ソースを読み進めるにつれて変わるカウンターやスイッチとして使うときです（15 章から 17 章）。

> 注．ほかのコンピューターで M80 を知っている人は、DEFL の別名として SET という疑似命令を期待するかもしれません。しかし、Z80 モードの M80 にも TATARA にも、この疑似命令はありません。set はビットをセットする Z80 の命令だからです。`x set 5` はその命令として読まれ、TATARA は `not a form this instruction has.` と表示して止まります。定義し直せるシンボルは DEFL だけです（マクロも定義し直せますが、それは別の話で、16 章で説明します）。
> 同じように、TATARA には.Z80 という疑似命令もありません。M80 では Z80 のニーモニックを受け付けるためにこれが必要でしたが、TATARA は Z80 のニーモニックしか受け付けず、切り替えるモードもありません。M80 向けに書かれたソースが.Z80 で始まっていると、その行で `not a directive, a macro or an instruction.` と表示して止まるので、その行は削除してかまいません。

12.5 モジュールどうしで名前を共有する

2 章で説明したように、プログラムはいくつかのモジュールで作ることができます。モジュールは、PUBLIC でほかのモジュールに名前を提供し、EXTRN でほかのモジュールの名前を使います。どちらもコンマで区切った名前の並びをとり、それぞれに別名があります。PUBLIC の別名は ENTRY、EXTRN の別名は EXT です。また、`both::` のようにコロンを 2 つ付けて書いたラベルもパブリックになります（6 章）。

次のモジュールは、それぞれ別の書き方で 4 つの名前を提供しています。

```
                public  greet,value
                entry   other
                cseg
greet:          ret
other:          ret
both::          ret
value           equ     5
```

PUBLIC は、この例のように定義より前に書いても、あとに書いてもかまいません。パブリックにできるのは、ラベル、EQU の値、そして DEFL の名前です。シンボル表には、パブリックな名前に印が付きます。

MSX-DOS

```
ASEG - absolute

0005h  public     value

CSEG - default code segment, 0003h bytes

0000h  public     greet
0001h  public     other
0002h  public     both
```

次のモジュールは、それらの名前を使います。

```
                extrn   greet,value
                ext     other,both
                cseg
start:          call    greet
                call    other
                call    both
                ld      hl,value
                ret
                end     start
```

TATARA が外部の名前について知っているのは、ほかのモジュールが定義しているということだけなので、シンボル表には 4 つの名前が EXTERNAL - resolved by the linker の下に表示されます。外部の値は TANREN が仕上げるので、ld hl,value のように 2 バイトに入れる場所にしか使えません。ld a,value のように 1 バイトに入れようとすると受け付けられません（8.6 節）。2 つのモジュールをリンクすると、完全なプログラムができます。

MSX-DOS

```
A:\>tanren /o:use.com use.tro pub.tro
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

2 modules, 20 records, ends at EOF.
Wrote USE.COM, 0100-010F (16 bytes), entry 0100.
```

TATARA 自身が見つける誤りが 2 つあります。

- `EXTRN` で宣言した名前を同じモジュールで定義すると、TATARA は `this name already has a value.` と表示して止まります。外部でもあり定義されてもいる、ということはありえないからです。
- `PUBLIC` で宣言した名前がどこでも定義されていないと、TATARA は `a PUBLIC name was never defined.` と表示して止まります。この確認は TATARA がオブジェクト・ファイルを書き出すときに行うので、コマンド・ラインにオブジェクト・ファイルの名前がなければ、そのソースは受け付けられます。

どのモジュールも定義していない名前のように、複数のモジュールにまたがる誤りは TANREN が見つけます（27 章）。

12.6 END と開始アドレス

`END` はソースを終わらせます。TATARA はそのあとを何も読みません。`END` を含むファイルを取り込んだファイルの残りも読みません（13 章）。`END` のないソースも受け付けられ、その場合はファイルの終わりがソースの終わりになります。

`END` にオペランドがあれば、それはプログラムの**開始アドレス**（start address）、つまりプログラムが実行を始める場所を表します。TATARA はそれをオブジェクト・ファイルに書き込み、TANREN は要約行に `entry` として表示します（4 章）。

> 警告. MSX-DOS2 は、`END` に何が書かれていても、`.COM` のプログラムを必ずファイルの最初のバイトである `0100h` から実行します。そのため、`.COM` のプログラムでは、`END` で示すラベルがコードの一番最初になければなりません。次のソースのようにデータが先にあると、プログラムはデータを命令として実行し、暴走してしまいます。TANREN はそれでもファイルを書き出しますが、開始アドレスがプログラムの実行が始まる場所ではないことを警告します。

```
                cseg
msg:            db      'Started at start.',13,10,'$'
start:          ld      de,msg
                ...
                end     start
```

MSX-DOS

```
WARNING: ENTRY.COM is entered at 0100, not at 0114.
1 modules, 7 records, ends at EOF.
Wrote ENTRY.COM, 0100-0120 (33 bytes), entry 0114.
```

.COM のプログラムで TANREN がこの警告を出したら、END で示すコードを先頭に移すか、データをデータ・セグメントに移してください（10 章）。

12.7　メッセージ

シンボルの定義と共有についてのメッセージを、それぞれの原因となる行とともに表 12.2 に示します。

行	メッセージ
`five: equu 5`	`EQU, DEFL and MACRO take a name, not a label.`
`five equ 5` のあとの `five equ 6`、同じラベルを 2 回、 `n equ 1` のあとの `n defl 2`、 `EXTRN` でもあり定義もされた名前	`this name already has a value.`
`x equ far`（`far` は外部）	`an external symbol may not be used here.`
名前のない `public`	`bad PUBLIC or EXTRN list.`
`public nowhere`（定義されていない）	`a PUBLIC name was never defined.`
`x set 5`	`not a form this instruction has.`

表 12.2: シンボルについてのメッセージ

もう 1 つ、正しいソースでは表示されることのないメッセージがあります。

MSX-DOS

```
phase error - this label had a different value on pass 1.
```

TATARA はソースを 2 回読み、ラベルはどちらのときも同じ値にならなければなりません。これがどんなときに起こるかは、20 章で説明します。

第 13 章

ファイルの取り込み

13.1 INCLUDE 89
13.2 TATARA が探す場所 91
13.3 環境変数 TATARA. 92
13.4 ファイル名の中のパス 92
13.5 ファイルを取り込むファイル 93
13.6 同じファイルを 2 回取り込む 93
13.7 インクルード・ファイルの中の END. 94
13.8 メッセージ 94

ソースは、INCLUDE を使って、ほかのファイルの行を取り込むことができます。この章では、INCLUDE が何をするか、TATARA がそのファイルをどこで探すか、環境変数 TATARA がどう関わるか、そしてファイルがさらにファイルを取り込むとどうなるかを説明します。

13.1 INCLUDE

include のあとにファイル名を書くと、TATARA はその位置でそのファイルを読みます。INCLUDE の行の代わりに、そのファイルの行がそこに書かれていたかのように扱われます。ファイルが終わると、TATARA は INCLUDE の次の行から続けます。このようにして読むファイルを**インクルード・ファイル**（include file）と呼びます。

```
; DEFS.INC - two names for MAIN.AS
five            equ     5
ten             equ     10
```

```
                include defs.inc        ; the names
                cseg
                ld      a,five
                ld      b,ten
                ret
                end
```

リスティングでは、INCLUDE の行があった位置に DEFS.INC の行が表示されます。ほかのファイルから来た行であることを示す印は付きません。INCLUDE の行そのものは表示されません。

MSX-DOS

```
A:\>tatara /l /p main.as
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

        Tatara v1.2.0   29-Sep-26       PAGE    1


                                ; DEFS.INC - two names for MAIN.AS
  0005                          five            equ     5
  000A                          ten             equ     10
                                                cseg
  0000'   3E 05                                 ld      a,five
  0002'   06 0A                                 ld      b,ten
  0004'   C9                                    ret
                                                end
ended at MAIN.AS(6)
```

ファイル名は、拡張子を付け、引用符で囲まずに、そのまま書きます。そのあとにコメントを書いてもかまいません。TATARA は名前に何も付け加えません。include defs は DEFS という名前のファイルを探し、include "defs.inc"は引用符で始まる名前のファイルを探します。どちらも cannot open と表示して止まります。

インクルード・ファイルには、ソースに書けるものなら何でも書けます。名前、コード、データ、マクロ、そしてさらに INCLUDE の行です。よくある使い方は、いくつものプログラムで共有する名前をまとめておくことです。たとえば、MSXDOS.INC には MSX-DOS2 のファンクションの名前が入っています（14 章）。

ファイルを取り込むことは、モジュールをリンクすることとは違います（2 章）。インクルード・ファイルは、それを取り込んだソースの一部になり、TATARA がそのソースから作る 1 つのオブジェクト・ファイルの一部になります。その名前は取り込んだソースの名前になるので、PUBLIC も EXTRN も必要ありません。

13.2 TATARA が探す場所

DEFS.INC のように、ドライブもディレクトリも付いていないファイル名のファイルは、何か所にもある可能性があります。TATARA は、次の 3 つの場所をこの順に探し、最初に見つかったものを読みます。

1. INCLUDE の行があるファイルのディレクトリ
2. **カレント・ディレクトリ**（current directory）、つまりコマンドを入力したときにいたディレクトリ
3. 環境変数 TATARA に書いたそれぞれのディレクトリ（左から順に）

図 13.1 に、1 つのコマンドについて 3 つの場所を示します。3 つの場所のどれにも DEFS.INC がありますが、TATARA が読むのは、MAIN.AS と同じ SRC にあるものです。

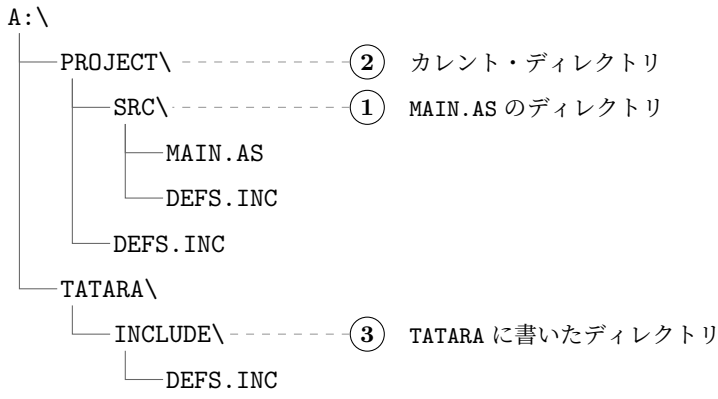

図 13.1: TATARA がインクルード・ファイルを探す 3 つの場所（この順に探す）

取り込む側のファイルのディレクトリが最初に来るのは、プロジェクト自身のインクルード・ファイルが、カレント・ディレクトリや TATARA のディレクトリにある同じ名前のファイルに必ず勝つようにするためです。図 13.1 では、カレント・ディレクトリに A:\PROJECT\DEFS.INC がありますが、TATARA はそこまで行きません。SRC にあるものが先に見つかるからです。

それでも、カレント・ディレクトリは 2 番目に探されます。もし SRC に DEFS.INC がなければ、TATARA はコマンドを入力したディレクトリ、A:\PROJECT にあるものを読みます。

> 注. コマンド・ラインで指定したファイルは探されません。TATARA は、入力されたとおりに、ほかのディレクトリを書かなければカレント・ディレクトリから開きます（5 章）。3 つの場所は INCLUDE にだけ使われます。

13.3　環境変数 TATARA

3 章では、Tatara に付いているインクルード・ファイルのディレクトリを TATARA に設定しました。TATARA には、セミコロンで区切って複数のディレクトリを書くことができ、TATARA はそれらを左から順に探します。AUTOEXEC.BAT などのバッチ・ファイルでは、次のように書きます。

```
set tatara=a:\tatara\include;a:\project\inc
```

プロンプトで入力するときは、3 章と同じように、値を引用符で囲みます。

それぞれのディレクトリの最後に\を付ける必要はありませんが、付けてもかまいません。

この環境変数は、3 つの場所の最後です。上の行の場合、両方のディレクトリにあるファイルは、最初に書いた A:\TATARA\INCLUDE から読まれます。カレント・ディレクトリや、取り込む側のファイルのディレクトリにもあるファイルはそこから読まれ、環境変数はまったく使われません。

13.4　ファイル名の中のパス

INCLUDE のあとのファイル名には、ディレクトリを含めることもできます。

ドライブで始まる名前や、\で始まる名前は、**絶対パス**（absolute path）です。

```
                include a:\tatara\include\msxdos.inc
                include \tatara\include\msxdos.inc
```

絶対パスは、ファイルがどこにあるかを正確に示します。TATARA は書かれたとおりに開き、ほかの場所は探しません。

それ以外の名前は**相対パス**（relative path）で、3 つの場所で順に探されます。include sub\mid.inc は、取り込む側のファイルのディレクトリ、カレント・ディレクトリ、TATARA のそれぞれのディレクトリの下の順に、SUB\MID.INC を探します。

日本の MSX では、3 章と同じように、\は¥と表示されます。同じ文字です。

13.5 ファイルを取り込むファイル

インクルード・ファイルは、さらにほかのファイルを取り込むことができます。TATARA が同時に開けるファイルは 4 つまで、つまりソースと 3 段階のインクルード・ファイルです。4 段階目で TATARA は止まり、そこまでの経路を、1 段階につき 1 行ずつ、内側から順に表示します。

MSX-DOS

```
D2.INC(2): ERROR: too many source files open at once.
    included from D1.INC(2)
    included from D0.INC(2)
    included from DEEPBAD.AS(2)
```

ここでは、`DEEPBAD.AS` が `D0.INC` を取り込み、それが `D1.INC` を取り込み、それが `D2.INC` を取り込み、`D2.INC` の 2 行目が 5 つ目のファイルを求めました。自分自身を取り込むファイルも、同じ制限に当たります。

マクロには別の制限があり、16 章で説明します。

13.6 同じファイルを 2 回取り込む

`EQU` とマクロだけを書いたファイルは、何回でも取り込めます。2 回目にはそれぞれの名前にもう一度同じ値が与えられますが、これは認められています（12 章）。マクロも定義し直すことができます（16 章）。

ラベルを定義しているファイルは、2 回取り込めません。2 回目にそのラベルをもう一度定義することになるからです。

MSX-DOS

```
CODE.INC(2): ERROR: this name already has a value.
    included from TWICELAB.AS(2)
```

ラベルやコードを書いたファイルは、1 か所でだけ取り込んでください。

> 警告. ファイルを 2 回取り込んでも問題がないようにする方法として、**インクルード・ガード**（include guard）が使われることがあります。ファイルの中身を `IFNDEF` で囲み、名前がすでに定義されていれば中身を飛ばすようにして、その内側の最初の行でその名前を定義するものです。TATARA では、これは使えません。

```
                ifndef  gcode_inc
gcode_inc       equ     1
                cseg
shared:         ret
                endif
```

TATARA はソースを 2 回読み（5 章）、1 回目に定義された名前は 2 回目にも定義されたままです。そのため、2 回目には gcode_inc がすでに定義されていて、ファイルを 1 回しか取り込んでいなくても、IFNDEF がその中身を飛ばしてしまいます。TATARA は、ラベル shared が 2 回目に定義されなかったことに気づいて止まります。

MSX-DOS

```
GCODE.INC(4): ERROR: this label was defined on pass 1 and not on pass 2 -
    a conditional skipped it.
```

TATARA のメッセージでは、それぞれの読み込みを**パス**（pass）と呼びます。条件付きアセンブルと 2 つのパスについては、15 章で説明します。

13.7　インクルード・ファイルの中の END

インクルード・ファイルの中の END は、そのファイルだけでなく、ソース全体を終わらせます（12 章）。INCLUDE のあとの行は読まれず、要約行はソースがどこで終わったかを示します。

MSX-DOS

```
ended at ENDIT.INC(2)
```

インクルード・ファイルには END を書かないでください。

13.8　メッセージ

ファイルの取り込みについてのメッセージを、それぞれの原因となる行とともに表 13.1 に示します。

TATARA がインクルード・ファイルを開けないときは、メッセージの前にファイル名と行番号が付きません。その下の行が、そのファイルがどこで求められたかを示します。

MSX-DOS

行	メッセージ
名前のない include	`INCLUDE without a filename.`
include nothere.inc （3 つの場所の どこにもない）	`cannot open NOTHERE.INC`
同時に開く 5 つ目のファイル、 自分自身を 取り込むファイル	`too many source files open at once.`

表 13.1: ファイルの取り込みについてのメッセージ

```
ERROR: cannot open NOTHERE.INC
    included from MIDMISS.INC(2)
    included from NESTMISS.AS(2)
```

インクルード・ファイルの中で起きたそれ以外のエラーは、そのファイルの名前と行番号で始まり、そのあとに同じ経路が続きます。

MSX-DOS

```
BAD.INC(2): ERROR: not a form this instruction has.
    included from MIDBAD.INC(2)
    included from ERRINC.AS(3)
```

この経路の読み方は、マクロについて加わる行も含めて、20 章で説明します。

> 注. M80 は、INCLUDE の別名として MACLIB と$INCLUDE も受け付けます。TATARA は受け付けません。M80 向けに書かれたソースでは、これらを INCLUDE に書き換えてください。

第 14 章

インクルード・ファイル

14.1 ファイルの中身 97
14.2 取り込み方 98
14.3 MSXDOS.INC と system マクロ 99
14.4 BIOS.INC と SUBROM.INC 100
14.5 ERRORS.INC 101
14.6 そのほかのファイル 102

Tatara には、MSX のための名前を集めた 9 つのインクルード・ファイルが付いています。MSX-DOS2 のファンクションとエラー・コード、BIOS とサブ ROM のエントリー・ポイント、フック、システム・ワーク・エリア、I/O ポート、拡張 BIOS、制御コードです。3 章でこれらを `A:\TATARA\INCLUDE` に置き、13 章で `INCLUDE` がそれらをどう見つけるかを説明しました。この章では、それぞれのファイルに何が入っているか、取り込む前に知っておくべきことを説明します。付録 K では、それぞれのファイルをまとめ、その中身を説明している本を挙げます。

14.1 ファイルの中身

表 14.1 に 9 つのファイルを示します。全部で 903 の名前と、`MSXDOS.INC` にある 1 つのマクロ `system` が入っています。

名前は、MSX の資料で使われているもの、つまり MSX Datapack の綴りのとおりです。そのため、ソースの中の名前は、本やほかのプログラムのリストの名前と同じになります。例外は、MSX に I/O ポートの標準の名前がないための `PORTS.INC` と、`SUBROM.INC` と `EXTBIO.INC` の接頭辞です（14.4 節、14.6 節）。

マクロを除けば、名前はすべて `EQU` です。`EQU` は名前に値を与えるだけで、プログラムには何も置きません。そのため、ファイルを取り込んでも増えるのは名前だ

ファイル	内容	名前の数
ASCII.INC	制御コード	37
BIOS.INC	メイン ROM のエントリー・ポイント	92
ERRORS.INC	MSX-DOS と MSX-DOS2 のエラー・コード	76
EXTBIO.INC	拡張 BIOS と、MSX-MUSIC の FM BIOS	85
HOOKS.INC	フック	117
MSXDOS.INC	MSX-DOS のファンクション番号と、system マクロ	93
PORTS.INC	I/O ポート	56
SUBROM.INC	サブ ROM のエントリー・ポイント	36
WORKAREA.INC	システム・ワーク・エリア	311

表 14.1: インクルード・ファイル

けで、バイトは増えません。9 つのファイルをすべて取り込み、ret を 1 つだけ書いたソースは、やはり 1 バイトです。

MSX-DOS

```
A:\>tanren /o:nocost.com nocost.tro
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

1 modules, 5 records, ends at EOF.
Wrote NOCOST.COM, 0100-0100 (1 bytes).
```

それぞれのファイルの最初には、中身の使い方を説明するコメントがあり、それぞれの名前には、それが何かを示す 1 行のコメントがあります。BIOS のルーチンや MSX-DOS2 のファンクションがレジスターに何を必要とし、何を返すかは、MSX の本に書かれています。ファイルにもこのマニュアルにも、それは繰り返していません。

14.2　取り込み方

4 章と同じように、ファイルを名前で取り込みます。

```
                include msxdos.inc
```

取り込んだファイルの名前は、ソースの名前になります。自分のラベルに、それらと同じ名前は付けられません。BEEP が BIOS のエントリーである BIOS.INC を取り込んだソースに beep: と書くと、TATARA は止まります。

MSX-DOS

```
CLASH.AS(4): ERROR: this name already has a value.
```

ソースで使うファイルだけを取り込めば、必要のない名前とぶつかることはありません。

名前だけのファイルは、1 つのソースの中でも、プログラムのすべてのモジュールでも、何回でも取り込めます。それぞれの名前にもう一度同じ値が与えられるだけだからです（13 章）。

名前は大文字で書かれています。TATARA は、/C オプションを付けない限り大文字と小文字を区別しないので（7 章）、ld a,(linl40) は ld a,(LINL40) と同じです。/C を付けるときは、ファイルに書かれているとおりに名前を書いてください。

MSX-DOS

```
CASEC.AS(4): ERROR: undefined symbol in an expression.
```

14.3 MSXDOS.INC と system マクロ

MSXDOS.INC には、MSX-DOS のファンクション番号が、MSX-DOS2 の資料と同じ下線付きの名前で入っています。_STROUT や_TERM0 などです。また、プログラムがファンクションを呼ぶアドレス 0005h の名前 BDOS も入っています。

プログラムは、ファンクション番号を C に、ファンクションが必要とするほかの値をほかのレジスターに入れて、BDOS を呼びます。system マクロは、このうち最初と最後を 1 行で行います。

```
start:          ld      de,msg
                system  _STROUT
                system  _TERM0
msg:            db      'Printed with system.',13,10,'$'
```

リスティングを見ると、TATARA がそれぞれの system の行から何を作るかがわかります。マクロの 2 つの命令で、それぞれに+の印が付いています。

MSX-DOS

```
0000'   11 0D 00              start:          ld      de,msg
                                              system  _STROUT
0003'   0E 09           +                     ld      c,_STROUT
0005'   CD 05 00        +                     call    BDOS
                                              system  _TERM0
0008'   0E 00           +                     ld      c,_TERM0
000A'   CD 05 00        +                     call    BDOS
```

system は、C と、ファンクション自身が変えるものを変えます。呼ぶのは必ず 0005h で、これは MSX-DOS の .COM プログラムには正しいアドレスです。Disk BASIC の下で動くプログラムや、カートリッジから動くプログラムは、0F37Dh でファンクションを呼ぶので、自分で呼び出しを書かなければなりません。

31h より小さいファンクションは MSX-DOS1 にもあります。31h 以上のファンクションには MSX-DOS2 が必要です。

14.4　BIOS.INC と SUBROM.INC

BIOS.INC には、メイン ROM のエントリー・ポイントがアドレス順に入っています。

> 警告. MSX-DOS の下では、0000h から 3FFFh までにメイン ROM はありません。このページは RAM です。そのため、.COM プログラムから BIOS のエントリーをふつうに call しても、BIOS には届きません。例外は、MSX-DOS が同じアドレスに用意している 5 つのエントリーです。RDSLT、WRSLT、CALSLT、ENASLT、CALLF です。それ以外のエントリーには、CALSLT を使った**スロット間コール**（inter-slot call）が必要です。

CALSLT は、呼ぶルーチンのスロットを IY の上位バイトに、そのアドレスを IX に受け取ります。メイン ROM のスロットは、ワーク・エリアにある EXPTBL の最初のバイトなので、ld iy,(EXPTBL-1) で IY の上位バイトに入ります。次のプログラムは、ブザーを鳴らしてから、CHPUT で 1 文字ずつ 1 行を表示します。

```
                include msxdos.inc
                include bios.inc
                include workarea.inc
                cseg
start:          ld      iy,(EXPTBL-1)   ; the MAIN ROM slot, into IYh
                ld      ix,BEEP
                call    CALSLT
                ld      hl,msg
next:           ld      a,(hl)
                or      a
                jr      z,done
                push    hl
                ld      iy,(EXPTBL-1)
                ld      ix,CHPUT
```

```
                call    CALSLT
                pop     hl
                inc     hl
                jr      next
done:           ei
                system  _TERM0
msg:            db      'Printed by the BIOS.',13,10,0
                end     start
```

MSX-DOS

```
A:\>bioscall
Printed by the BIOS.
```

最後の ei は割り込みを有効に戻します。スロット間コールのあとは、割り込みが禁止されたままになるからです。

SUBROM.INC には、MSX2 で加わったサブ ROM のエントリー・ポイントが入っています。サブ ROM は、メイン ROM の名前のいくつかを、別のアドレスの別のルーチンに使っています。たとえば GRPPRT は、メイン ROM では 008Dh、サブ ROM では 0089h です。そのため、SUBROM.INC の名前はすべて SUBROM_GRPPRT のように SUBROM_で始まり、2 つのファイルを一緒に取り込めるようになっています。

サブ ROM のルーチンは、アドレスを IX に入れて、BIOS のエントリー EXTROM を通して呼びます。

```
                ld      ix,SUBROM_REDCLK
                call    EXTROM
```

これは、Disk BASIC やカートリッジからの呼び方です。MSX-DOS からはもっと面倒です。EXTROM 自身が BIOS のエントリーなので CALSLT を通して呼ぶことになりますが、CALSLT は呼ぶアドレスに IX を使うため、サブ ROM のアドレスを IX に入れて EXTROM に渡すことができないからです。その回避方法は、BIOS.INC の EXTROM のあとのコメントに書かれています。

14.5 ERRORS.INC

ERRORS.INC には、MSX-DOS と MSX-DOS2 のエラー・コードが入っています。名前は、MSX-DOS2 の資料と同じようにドットで始まります。.NOFIL は 0D7h、「File not found」です。名前にはドットを使えます（7 章）。コードは 0FFh から下に向かって数えるので、このファイルだけは、ほかと違って降順に並んでいます。

40h 以上の MSX-DOS2 のファンクションは、エラー・コードを A に返し、成功したときは 0 を返します。呼び出しのすぐあとの jr nz でエラーを捕まえられます。そのあと、プログラムはファンクション_TERM でコードを MSX-DOS2 に渡すことが

できます。_TERM はプログラムを終わらせ、そのコードのメッセージを表示します。次のプログラムは、存在しないファイルを開こうとします。

```
                include msxdos.inc
                cseg
start:          ld      de,name
                xor     a               ; open mode 0
                system  _OPEN
                jr      nz,failed       ; A is the error code
                system  _TERM0
failed:         ld      b,a
                system  _TERM
name:           db      'NOTHERE.TXT',0
                end     start
```

MSX-DOS

```
A:\>openerr

*** File not found
```

特定のエラーを調べるには、cp .NOFIL のように、A をその名前と比べます。

14.6　そのほかのファイル

14.6.1　HOOKS.INC

フック（hook）は、システムが決まったときに呼ぶ、RAM の 5 バイトです。たとえば H.TIMI は、タイマー割り込みのたびに、1 秒に 50 回か 60 回呼ばれます。使われていないフックには ret が入っています。フックを使うプログラムは、まずその 5 バイトを保存し、代わりに自分のルーチンへのジャンプを書き込み、終わる前に 5 バイトを元に戻します。終わったプログラムを指したままのフックは、次に呼ばれたときに MSX を暴走させます。

2 つのフックには、MSX-MIDI が名前を付け直したため、それぞれ 2 つの名前があります。H.ONKO と H.MDIN はどちらも 0FF75h、H.FRQI と H.MDTM はどちらも 0FF93h です。これは同じ 5 バイトで、4 つのフックではありません。

14.6.2 WORKAREA.INC

ワーク・エリア（work area）は、システムが変数を置いている、0F323h から上の RAM です。`WORKAREA.INC` では、それぞれのコメントの最初の数字が変数のバイト数です。1 バイトの変数は `ld a,(LINL40)` で、2 バイトの変数は `ld hl,(...)` で読みます。サイズを取り違えて読むと、ほぼ正しい、しかし間違った答えが返ります。

14.6.3 PORTS.INC

MSX には I/O ポートの標準の名前がないので、ポートの名前は Tatara 独自のものです。ポートの名前はすべて `IO_`で始まり（ポート 099h は `IO_VDPCMD` など）、先頭の 0 を含めて 3 桁の 16 進数で書かれています。

14.6.4 EXTBIO.INC

拡張 *BIOS*（extended BIOS）は、MSX-DOS2 自身のマッパー・ルーチンや MSX-MIDI のように、カートリッジが提供するルーチンをプログラムが見つけるためのしくみです。名前は、`EXTBIO_DOS2_DEVICE` のように `EXTBIO_`とデバイスの名前で始まります。資料にある `INIT` や `EOF` のような名前のままでは、たいていのプログラムの名前とぶつかってしまうからです。拡張 BIOS の呼び方は、ファイルの最初のコメントで説明されています。

14.6.5 ASCII.INC

`CHR_NUL` から `CHR_DEL` までの制御コードと、MSX がそれぞれをどう扱うかです。ほかのファイルと違って値は 10 進数です。文字コードは、`db 13,10` のように 10 進数で書くことが多いからです。`CHR_CR` は 13、`CHR_LF` は 10 です。

第 15 章

条件付きアセンブル

15.1 IF、ELSE、ENDIF 105
15.2 2 つのパス 107
15.3 IFDEF と IFNDEF 108
15.4 IF1 と IF2 109
15.5 IFB、IFNB、IFIDN、IFDIF 109
15.6 入れ子 110
15.7 メッセージ 111

条件付きアセンブル（conditional assembly）を使うと、TATARA がソースを読みながら判断する条件によって、ソースの一部の行をアセンブルし、ほかの行を飛ばすことができます。これにより、1 つのソースから複数のプログラムを作れます。たとえば、テスト用のチェックを入れた版と入れない版、MSX1 用と MSX2 用の版を、1 行を変えるだけで切り替えられます。

この章では、条件付きアセンブルの疑似命令と、すべての条件の前提となる、TATARA がソースを 2 回読むしくみ（2 つのパス）を説明します。

15.1 IF、ELSE、ENDIF

`IF` のあとには式を書きます。その値が 0 でなければ、あとに続く行がアセンブルされ、0 ならば飛ばされます。`ENDIF` は `IF` が制御する行の終わりを示します。省略できる `ELSE` は、条件が偽のときに代わりにアセンブルする行の始まりを示します。`IFE` は逆の判定で、値が 0 のときに行がアセンブルされます。

```
debug           equ     1
                cseg
                if      debug
                db      'D'
                else
                db      'R'
                endif
                ife     debug
                db      'E'
                endif
```

ここでは db 'D' がアセンブルされ、db 'R' と db 'E' は飛ばされます。最初の行を debug equ 0 にすると、これが入れ替わります。リスティングにはすべての行が表示されますが、飛ばされた行にはアドレスもバイトもありません。

MSX-DOS

```
0001                            debug           equ     1
                                                cseg
                                                if      debug
0000'   44                                      db      'D'
                                                else
                                                db      'R'
                                                endif
                                                ife     debug
                                                db      'E'
                                                endif
```

飛ばされた行をリスティングから完全に除く方法は、18 章で説明します。

0 以外の値はすべて真とみなされます。8 章の比較演算子は、真のときに 0FFFFh、偽のときに 0 を返すので、if size gt 256 のようにそのまま使えます。

これらの疑似命令のいくつかには別の名前があり、M80 向けに書かれたソースをそのままアセンブルできるように、TATARA はそれらも受け付けます（表 15.1）。

別の名前	同じもの
IFT、COND	IF
IFF	IFE

表 15.1: 条件付きアセンブルの疑似命令の別の名前

M80 は ENDIF の別名として ENDC も受け付けますが、TATARA は受け付けません。M80 向けに書かれたソースでは、ENDIF に書き換えてください。

15.2 2 つのパス

ラベルは、それを定義する行より前でも使えます。次の最初の行がその例です。

```
                jp      later
later:          ret
```

TATARA が `jp later` に来たとき、まだ `later` を見ていないので、そのアドレスはわかりません。TATARA はこれを、ソースを 2 回読むことで解決します。1 回ごとの読み込みを**パス**（pass）と呼びます。

- **第 *1* パス**（first pass）では、TATARA はそれぞれの行が何バイトになるかを調べ、それによってすべてのラベルのアドレスとすべての名前の値を求めます。何も書き出しません。
- **第 *2* パス**（second pass）では、すべての値がわかっているので、TATARA はプログラムのバイト、リスティング、オブジェクト・ファイルを作ります。

図 15.1 の上半分は、この 2 行に対する 2 つのパスを示しています。M80 を含め、ほとんどのアセンブラーはこのように動きます。

ソース			第 1 パス	第 2 パス
	jp	later	0000: 3 バイト、later は不明	0000: C3 03 00
later:	ret		later = 0003	0003: C9

ソース			第 1 パス	第 2 パス
	ifdef	later	まだ定義されていない：飛ばす	定義されている：アセンブルする
	db	1	（飛ばす）	0000: 01
	endif			
after:	nop		after = 0000	**after = 0001：フェーズ・エラー**
later	equ	5	later = 5	later = 5

図 15.1: それぞれのパスが行うこと。上は、定義する行より前で使われたラベル。下は、パスごとに違う答えになる条件

第 1 パスで見つけた名前と値は、第 2 パスでもそのまま使われます。そのため第 2 パスでは、第 1 パスで定義されたすべての名前が、ソースのもっとあとで定義されるものも含めて、すでに定義されています。

これは条件付きアセンブルに関わってきます。TATARA は、それぞれの条件を両方のパスで評価するからです。条件が第 2 パスで違う答えになると、第 2 パスでは

第 1 パスと違う行がアセンブルされ、そのあとのラベルがずれます。図 15.1 の下半分がその例です。IFDEF は later が定義されているかどうかを調べます（15.3 節）。第 1 パスではまだ定義されていないので、db 1 は飛ばされ、after は 0000h になります。第 2 パスでは定義されているので、db 1 がアセンブルされ、after は 0001h になってしまいます。TATARA は after の行で止まります。

MSX-DOS

```
DEFLATE.AS(6): ERROR: phase error - this label had a different value on pass 1.
```

第 1 パスで定義したラベルを条件のせいで第 2 パスが飛ばした場合は、13.6 節のメッセージになります。

したがって、条件は両方のパスで同じ答えにならなければなりません。確実にそうするいちばん簡単な方法は、条件より上で定義された名前だけを使うことです。IF では、TATARA がそれを要求します。あとで定義される名前を使うと、第 1 パスで止まります。

```
; FWD.AS - IF of a name defined later.
                cseg
                if      later
                db      1
                endif
later           equ     1
```

MSX-DOS

```
FWD.AS(3): ERROR: undefined symbol in an expression.
```

15.3　IFDEF と IFNDEF

IFDEF のあとには名前を書き、その名前が定義されていれば行がアセンブルされます。IFNDEF は、定義されていなければアセンブルします。EXTRN で宣言した名前は、定義されているものとみなされます。

よくある使い方は、ソースが自分で値を決められ、決めなければ既定値になるようにするものです。

```
                ifndef  bufsize
bufsize         equ     128
                endif
```

別の大きさにしたいソースは、これらの行より前で bufsize を定義します。必ず**前**でなければなりません。15.2 節で見たように、もっとあとで定義される名前は、

第 1 パスでは未定義、第 2 パスでは定義済みになり、2 つのパスで答えが変わってしまうからです。

同じ理由で、IFNDEF を使ってインクルード・ファイルが 2 回読まれるのを防ぐことはできません（13.6 節）。

15.4 IF1 と IF2

IF1 は第 1 パスでだけ、IF2 は第 2 パスでだけ、その行をアセンブルします。

これらは M80 から来たもので、M80 のソースをアセンブルできるように残してあります。リスティングがそのままプリンターに出力されていた時代に、2 回ではなく 1 回だけ行いたいことのために使われていました。TATARA ではほとんど役に立ちません。リスティングは第 2 パスを表示するので、IF2 の行は表示されますが、IF1 の行は表示されません。また、IF1 や IF2 の中でラベルを定義したりバイトを作ったりすると、2 つのパスが違うものになり、15.2 節の決まりに反します。

```
; PASSLAB.AS - a label inside IF1.
                cseg
                if1
here:           nop
                endif
```

MSX-DOS

```
PASSLAB.AS(4): ERROR: this label was defined on pass 1 and not on pass 2 - a
    conditional skipped it.
```

15.5 IFB、IFNB、IFIDN、IFDIF

この 4 つは、値ではなくテキストを調べます。

- IFB はテキストが空のときに、IFNB は空でないときに、行をアセンブルします。
- IFIDN は 2 つのテキストが同じときに、IFDIF は違うときに、行をアセンブルします。2 つのテキストはコンマで区切ります。

それぞれのテキストは山かっこ（<>）で囲み、かっこはテキストに含まれません。空白もコンマも含まないテキストでは、かっこを省略できます。

```
                ifb     <>              ; true: blank
                ifb     <x>             ; false
                ifnb    <x>             ; true
                ifdif   <abc>,<abd>     ; true
                ifidn   <abc>,<ABC>     ; false: the case differs
                ifidn   abc,abc         ; true
```

比較は、M80 と同じように大文字と小文字も含めて完全に一致するかどうかで、/C の有無は関係ありません。

このようにテキストをそのまま書いたのでは、答えはいつも同じなので、これらの疑似命令はそれだけでは役に立ちません。マクロの中で、使うたびに変わる引数をテキストとして使います（16 章）。

15.6　入れ子

条件付きアセンブルは、別の条件付きアセンブルの中に入れることができます。飛ばされている行の中の IF は評価さえされず、条件がどうであれ、その行はどれもアセンブルされません。

```
                if      0
                if      1
                db      1               ; skipped: the outer IF is false
                else
                db      2               ; skipped too
                endif
                else
                if      1
                db      3               ; assembled
                else
                db      4
                endif
                endif
```

それぞれの ELSE と ENDIF は、まだ閉じていないいちばん内側の IF に対応します。TATARA は、同時に 16 個までの条件付きアセンブルを開いておけます。

IF、ELSE、ENDIF の行にも、ほかの行と同じようにラベルを書けます。M80 と同じように、TATARA がその行に来たときにアセンブル中であれば、ラベルはロケーション・カウンターのアドレスで定義され、飛ばしている最中であれば定義されません。そのため、飛ばされた行のあとの ELSE に書いたラベルは定義されず、飛ばされている行の中の IF に書いたラベルも定義されません。ラベルは、独立した行に書くのがいちばん安全です。

15.7　メッセージ

条件付きアセンブルについてのメッセージを、それぞれの原因となる行とともに表 15.2 に示します。

行	メッセージ
開いている IF がないのに ELSE か ENDIF、 1 つの IF に 2 つ目の ELSE	`ELSE or ENDIF without a matching IF.`
閉じられていない IF	`conditional not closed by ENDIF.`
16 個の IF の中の 17 個目の IF	`conditionals nested too deeply.`
式のない if	`bad expression.`
`if later` （later はもっとあとで定義）	`undefined symbol in an expression.`
endc	`not a directive, a macro or an instruction.`

表 15.2: 条件付きアセンブルについてのメッセージ

閉じられていない条件付きアセンブルに TATARA が気づくのはソースの終わりでですが、見つけやすいように、メッセージは IF の行を示します。複数が開いたままのときは、いちばん外側のものを示します。

第 16 章

マクロ

16.1 マクロの定義と使用 114
16.2 引数 115
16.3 &によるテキストの連結 116
16.4 本体の中のコメント 117
16.5 LOCAL 117
16.6 EXITM 118
16.7 マクロの中のマクロ 119
16.8 大文字と小文字 120
16.9 メッセージ 120

マクロ（macro）は、行のまとまりに付けた名前です。まとまりを定義すると、その名前を命令と同じように使えるようになり、TATARA はそれぞれの使用をそのまとまりの行に置き換えます。1 回の使用からできたものを**展開**（expansion）と呼びます。

14 章の `system` マクロもその 1 つです。`system _STROUT` は、`ld c,_STROUT` と `call BDOS` に置き換えられます。マクロはサブルーチンではありません。サブルーチンはプログラムの中に 1 つだけあり、`call` で呼び出します。マクロの行は、使うたびにプログラムの中にもう一度置かれます。マクロで省けるのは入力の手間で、バイト数ではありません。

この章では、マクロの定義のしかた、マクロへの値の渡し方、マクロの中で使う疑似命令を説明します。名前を付けずに行を繰り返す `REPT`、`IRP`、`IRPC` は 17 章で説明します。これらも、引数、`LOCAL`、`EXITM` については同じ規則に従います。

16.1　マクロの定義と使用

マクロは MACRO で定義します。MACRO の前にはマクロの名前を、あとにはコンマで区切った**パラメーター**（parameter）の名前を書きます。それに続く行がマクロの**本体**（body）で、ENDM で終わります。

```
fill            macro   addr,value
                ld      hl,addr
                ld      (hl),value
                endm
```

EQU と同じように、名前はコロンを付けずに書きます（12 章）。定義はバイトを作りません。TATARA は本体を保存するだけです。マクロは、最初に使う前に定義しておかなければなりません。

マクロを使うには、その名前を命令のように書き、そのあとに**引数**（argument）、つまり今回の使用でパラメーターが表す値を書きます。TATARA は本体の中のそれぞれのパラメーターを、対応する引数に置き換えます。1 番目のパラメーターには 1 番目の引数、2 番目には 2 番目の引数が対応します（図 16.1）。

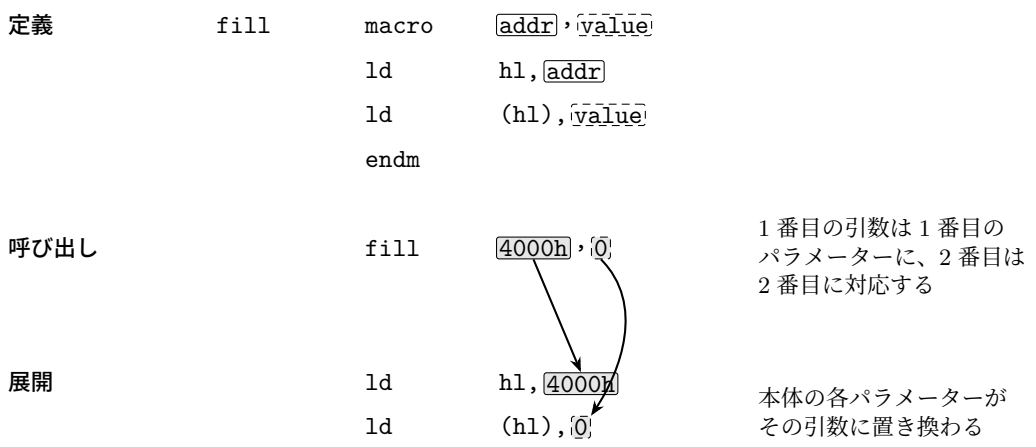

図 16.1: マクロの定義、その使用、そして展開

リスティングには、それぞれの使用のあとにその展開が表示され、作られた行には+の印が付きます。

```
                                        fill    4000h,0
0000'   21 00 40        +               ld      hl,4000h
0003'   36 00           +               ld      (hl),0
                                        fill    4001h,0ffh
0005'   21 01 40        +               ld      hl,4001h
0008'   36 FF           +               ld      (hl),0ffh
```

展開をリスティングに少なく表示する方法や、まったく表示しない方法は、18 章で説明します。

16.2 引数

使用のときの引数はコンマで区切ります。引数の前の空白は引数に含まれません。

- 省略した引数は空になります。パラメーターは何もないものに置き換えられるので、本体はそれに対応していなければなりません。本体に `db 0,b` があると、`b` が空のときは `db 0,` となり、TATARA はこれを受け付けません。空かどうかは `IFB` と `IFNB` で調べられます（16.6 節）。
- 最後のパラメーターより多い引数は、M80 と同じように無視されます。
- コンマを含む引数は山かっこで囲みます。かっこは取り除かれます。`<7,8>`は、`7,8` という 1 つの引数です。
- 文字の前に`!`を付けると、その文字は何であっても引数の一部になります。本来なら引数を終わらせる`>`やコンマ、本来ならコメントを始めるセミコロンなどです。山かっこの中のセミコロンも引数の一部です。

次のマクロは、1 番目の引数を `DB` に入れ、2 番目の引数があればもう 1 つの `DB` に入れます。

```
show            macro   a,b
                db      a
                ifnb    <b>
                db      b
                endif
                endm
```

次の使用では、`show 3` は `b` を空にし、`show 4,5,6` の 6 は無視され、`<7,8>`は 1 つの引数になり、`'!>'` は`>`を含む文字列を渡します。

```
                                        show    3
0002'   03              +               db      3
                                        show    4,5,6
0003'   04              +               db      4
0004'   05              +               db      5
                                        show    <7,8>,9
0005'   07 08           +               db      7,8
0007'   09              +               db      9
                                        show    '!>',10
0008'   3E              +               db      '>'
0009'   0A              +               db      10
```

引数は値ではなくテキストです。`five*2` は、`five*2` という 6 文字として渡されます。引数の前に`%`を付けると、代わりにその値が 10 進数で書かれて渡されます。本体が `db '&x'` のとき（`&`は次の節で説明します）、次の 1 つ目の使用はテキストを、2 つ目は値を渡します。

MSX-DOS

```
                                        show    five*2
0000'   66 69 76 65     +               db      'five*2'
0004'   2A 32           +
                                        show    %five*2
0006'   31 30           +               db      '10'
```

その値は、`IF` の場合と同じように、TATARA がすでに知っているものでなければなりません（15 章）。

16.3　&によるテキストの連結

パラメーターが置き換えられるのは、その名前が独立した語になっているところだけです。ほかのテキストとつなげるには、その間に`&`を書きます。引数が `1` なら、`lab&n` は `lab1` になります。`&`自体は消えます。

文字列の中のパラメーターは、`&`でつながない限り、まったく置き換えられません。文字列`'n'` の `n` はただの文字ですが、`'&n'` の `n` はパラメーターです。

```
label           macro   n
lab&n:          db      'n'
                db      '&n'
                endm
```

MSX-DOS

```
                                        label   1
 0000'   6E             +     lab1:     db      'n'
 0001'   31             +               db      '1'
```

16.4 本体の中のコメント

;で始まるコメントは本体に残り、すべての展開に現れます。;;で始まるコメントはマクロを定義するときに捨てられるので、定義の中でしか見えません。マクロのしくみについてのメモのように、リスティングで繰り返されても意味のないものには;;を使います。

```
note            macro
                nop             ; kept
                nop             ;; dropped
                endm
```

MSX-DOS

```
                                        note
 0000'   00             +               nop             ; kept
 0001'   00             +               nop
```

16.5 LOCAL

ラベルを定義するマクロは、1回しか使えません。2回目の使用でそのラベルをもう一度定義することになるからです（12章）。

```
wait            macro   n
                ld      b,n
loop:           djnz    loop
                endm
```

MSX-DOS

```
LOCALNO.AS(4): ERROR: this name already has a value.
    in wait, called from LOCALNO.AS(8)
```

これは、1つ以上の名前を続けたLOCALで解決します。TATARAは、それぞれの展開の中でそれらの名前を、??0000、??0001のように、毎回違う新しい名前に置き換えます。

```
wait            macro   n
                local   loop
                ld      b,n
loop:           djnz    loop
                endm
```

MSX-DOS

```
                                                wait    10
 0000'   06 0A           +                      ld      b,10
 0002'   10 FE           +      ??0000:         djnz    ??0000
                                                wait    20
 0004'   06 14           +                      ld      b,20
 0006'   10 FE           +      ??0001:         djnz    ??0001
```

作られた名前はふつうのラベルで、シンボル表にも表示されます。自分のラベルには??で始まる名前を使わないでください。LOCAL の行は、本体の最初、ほかのどの行よりも前に書かなければなりません。そうでないと、TATARA は `LOCAL must come before the macro body.` と表示して止まります。

16.6　EXITM

EXITM は、展開をすぐに終わらせます。本体のそれ以降の行は使われません。条件付きアセンブルの中で使い、場合によってマクロを早めに終わらせます。次のマクロは、引数が与えられないと何もしません。

```
put             macro   x
                ifb     <x>
                exitm
                endif
                db      x
                endm
```

MSX-DOS

```
                                                put     1
 0000'   01              +                      db      1
                                                put
                                                put     3
 0001'   03              +                      db      3
```

15 章のテキストの判定が役に立つのは、ここです。IFB と IFNB は引数が与えられたかどうかを調べ、IFIDN と IFDIF は引数をレジスター名のような決まったテキストと比べます。EXITM は、マクロの中で開いたままの条件付きアセンブルも一緒に閉じます。

16.7 マクロの中のマクロ

マクロの本体では、別のマクロを使うことも、定義することもできます。

```
byte            macro   x
                db      x
                endm
pair            macro   x,y
                byte    x
                byte    y
                endm
maker           macro   name,val
name            macro
                db      val
                endm
                endm
```

ここでは、pair 1,2 は byte を通して db 1 と db 2 を作ります。maker three,3 はバイトを作りませんが、three というマクロを定義し、その three が db 3 を作ります。本体の中の MACRO の行は、本体が展開されるまではただのテキストです。それぞれの ENDM は、いちばん内側の MACRO に対応します。

マクロは自分自身を使うこともできます。その場合は、止めるための条件が必要です。次のマクロは 0 まで数え下ります。

```
down            macro   n
                db      n
                if      n gt 0
                down    %n-1
                endif
                endm
```

MSX-DOS

```
                                                down    3
 0000'   03              +                      db      3
 0001'   02              +                      db      2
 0002'   01              +                      db      1
 0003'   00              +                      db      0
```

展開の中の展開は、インクルード・ファイルと合わせて 16 段まで重ねられます。止める条件がないと、自分自身を使うマクロはこの制限に達し、TATARA は sources nested too deeply. と表示して止まります。

マクロは定義し直すことができ、その行からは新しい定義が古い定義に置き換わります。system マクロはこれを利用しています。MSXDOS.INC を取り込むそれぞれのモジュールが、マクロを定義し直すからです（14 章）。

16.8　大文字と小文字

マクロの名前は、ほかの名前と同じ規則に従います。/C がなければ PUT でマクロ put を使えますが、/C があると使えず、TATARA は `not a directive, a macro or an instruction.` と表示して止まります（7 章）。

パラメーターの名前は、M80 と同じように、どちらのモードでも大文字と小文字を区別せずに照合されます。パラメーター val は、本体に val、VAL、Val のどれで書いても置き換えられます。

16.9　メッセージ

マクロについてのメッセージを、それぞれの原因となる行とともに表 16.1 に示します。

行	メッセージ
前に名前のない macro	`MACRO without a name.`
65 文字のマクロの名前	`a macro name may be at most 64 characters.`
`put: macro`	`EQU, DEFL and MACRO take a name, not a label.`
パラメーターが 17 個のマクロ	`too many macro parameters.`
`bad macro a,,b`	`bad macro parameter list.`
ENDM のない MACRO	`macro definition not closed by ENDM.`
MACRO のない ENDM	`ENDM without a macro definition.`
本体の行のあとの LOCAL	`LOCAL must come before the macro body.`
マクロの外の EXITM	`EXITM outside a macro or repeat block.`
終わりなく自分自身を使うマクロ	`sources nested too deeply.`

表 16.1: マクロについてのメッセージ

展開の中の行がエラーを起こすと、メッセージはマクロを定義したファイルの中の本体の行を示し、その下の行が、どのマクロのどこでの使用かを示します。

MSX-DOS

```
ERRIN.AS(4): ERROR: not a form this instruction has.
    in bad, called from ERRIN.AS(8)
```

この経路の詳しい読み方は、20 章で説明します。

第 17 章

繰り返しブロック

17.1 REPT 123
17.2 IRP 124
17.3 IRPC 126
17.4 LOCAL、EXITM、入れ子 126
17.5 メッセージ 127

繰り返しブロック（repeat block）は、TATARA がその場で繰り返す行のまとまりです。その場で定義して使う、名前のないマクロのように働きます。`REPT` は指定した回数だけ、`IRP` はリストの項目ごとに 1 回ずつ、`IRPC` はテキストの文字ごとに 1 回ずつ、その行を繰り返します。

3 つとも、マクロと同じように `ENDM` で終わり、本体、`LOCAL`、`EXITM` については 16 章の規則に従います。

17.1 REPT

`REPT` のあとには回数を書きます。`ENDM` までの行が、その回数だけアセンブルされます。

```
                cseg
here:           rept    3
                nop
                endm
                dw      here
```

MSX-DOS

```
                                here:           rept    3
                                                nop
                                                endm
  0000'   00                +                   nop
  0001'   00                +                   nop
  0002'   00                +                   nop
  0003'   00 00                                 dw      here
```

リスティングには、書いたとおりのブロックに続いて、それぞれの回が、マクロの展開と同じように+の印付きで表示されます。REPT の行のラベルは、最初の回の始まりを示します。here は 0000h です。

回数にはどんな式でも書けますが、TATARA はブロックの行を読む前にそれを計算するので、すでにわかっている数でなければなりません。あとで定義される名前を使うと、TATARA は `undefined symbol in an expression.` と表示して止まり、コード・セグメントのラベルのようなリロケータブルな値を使うと、8 章のリロケーション・エラーで止まります。回数が 0 なら、何もアセンブルされません。

DEFL（12 章）がいちばん役に立つのは、繰り返しブロックの中です。DEFL で定義した名前には回ごとに新しい値を与えられるので、回数を数えられます。

```
n               defl    0
                rept    4
                db      n
n               defl    n+1
                endm
```

MSX-DOS

```
  0000'   00                +                   db      n
  0001'   01                +                   db      n
  0002'   02                +                   db      n
  0003'   03                +                   db      n
```

リスティングにはそれぞれの行が書いたとおり `db n` と表示され、バイトはその回での n の値を示しています。

17.2　IRP

IRP のあとには、パラメーター名、コンマ、山かっこで囲んだ項目のリストを書きます。ブロックは項目ごとに 1 回ずつ、パラメーターをその項目に置き換えてアセンブルされます。

```
                irp     x,<1,2,4,8>
                db      x
                endm
```

MSX-DOS

```
0000'   01              +                       db      1
0001'   02              +                       db      2
0002'   04              +                       db      4
0003'   08              +                       db      8
```

項目は、マクロの引数と同じ規則に従います（16.2 節）。コンマを含む項目は、それ自身の山かっこで囲みます。また、%を付けると、テキストの代わりに値が渡されます。

```
five            equ     5
                irp     s,<'ab',<'c,d'>>
                db      s
                endm
                irp     v,<five*2,%five*2>
                db      '&v'
                endm
```

MSX-DOS

```
0005'   61 62           +                       db      'ab'
0007'   63 2C 64        +                       db      'c,d'
000A'   66 69 76 65     +                       db      'five*2'
000E'   2A 32           +
0010'   31 30           +                       db      '10'
```

リストを囲む山かっこは省略できます。`irp x,1,2,3` は `irp x,<1,2,3>`と同じです。空のリスト `irp x,<>`では、パラメーターが空のまま 1 回実行されます。

IRP がいちばん役に立つのは、マクロの中で、それ自体がリストである引数に対して使うときです。

```
bytes           macro   list
                irp     b,<list>
                db      b
                endm
                endm
```

MSX-DOS

```
                                                bytes   <10,20,30>
0000'   0A              +                       db      10
0001'   14              +                       db      20
0002'   1E              +                       db      30
```

17.3　IRPC

IRPC のあとには、パラメーター名、コンマ、テキストを書きます。ブロックはテキストの文字ごとに 1 回ずつ、パラメーターをその文字に置き換えてアセンブルされます。マクロと同じように、文字列の中のパラメーターには&が必要です（16.3 節）。

```
                irpc    c,TATARA
                db      '&c'
                endm
```

MSX-DOS

```
0000'   54              +                       db      'T'
0001'   41              +                       db      'A'
0002'   54              +                       db      'T'
0003'   41              +                       db      'A'
0004'   52              +                       db      'R'
0005'   41              +                       db      'A'
```

空白やコンマも含めて、すべての文字が数えられます。irpc c,<A B,C>は、A、空白、B、コンマ、C の 5 回になります。山かっこはテキストに空白を入れるときにだけ必要で、テキストには含まれません。空のテキスト irpc c,<>では、1 回も実行されません。IRPC のテキストでは、!と%に特別な意味はありません。

17.4　LOCAL、EXITM、入れ子

LOCAL は、繰り返しブロックでもマクロと同じように働き、回ごとに別の名前を与えます。

```
                rept    2
                local   skip
                jr      skip
                nop
skip:
                endm
```

MSX-DOS

```
0000'   18 01           +                       jr      ??0000
0002'   00              +                       nop
0003'   18 01           +                       jr      ??0001
0005'   00              +                       nop
```

LOCAL がなければ、2 回目で skip をもう一度定義することになります。ラベルだけの行はバイトを作らないので、展開には表示されません。

EXITM は、まだ実行していない回も含めて、ブロック全体を終わらせます。次のブロックは 10 回実行されるはずですが、3 回で止まります。

```
n               defl    0
                rept    10
                if      n eq 3
                exitm
                endif
                db      n
n               defl    n+1
                endm
                db      0ffh
```

MSX-DOS

```
0000'   00              +                       db      n
0001'   01              +                       db      n
0002'   02              +                       db      n
0003'   FF                                      db      0ffh
```

繰り返しブロックは、ほかの繰り返しブロックやマクロの中に入れることができ、それぞれの ENDM はいちばん内側のものを終わらせます。マクロの展開やインクルード・ファイルと合わせて、16 段まで重ねられます（16.7 節）。

17.5　メッセージ

繰り返しブロックについてのメッセージを、それぞれの原因となる行とともに表 17.1 に示します。繰り返しブロックの中でエラーが起きると、メッセージのあとの経路にはブロックが a repeat block と表示されます。

行	メッセージ
`rept count` （`count` はあとで定義）	`undefined symbol in an expression.`
`rept start` （`start` はコード・ セグメントのラベル）	`relocation error - a segment-relative value is not allowed here.`
`irp x` （コンマがない）	`bad macro parameter list.`
`ENDM` のない 繰り返しブロック	`macro definition not closed by ENDM.`

表 17.1: 繰り返しブロックについてのメッセージ

第 18 章

リスティング

18.1 リスティングの 1 行. 130
18.2 ページ 131
18.3 リスティングを止める、再開する 134
18.4 マクロの展開 135
18.5 アセンブルされない行 136
18.6 メッセージ 137

リスティングには、TATARA がソースから何を作ったかが表示されます。それぞれの行が書いたとおりに、アセンブルされたアドレスと、作られたバイトとともに表示されます。プログラムがソースのとおりに動かないときや、ある行が思ったより多くの場所を取るときには、リスティングを見ます。

リスティングの出し方は、5 章で説明しました。リスティングは、コマンド・ラインの 3 つ目に書いたファイルに書き出され、それがなければ、/P を付けたときに画面に表示されます。/L を付けると、それぞれの行の前にアドレスとバイトが付きます。付けなければ、リスティングにはソースだけが表示されます。この章では、/L を付けたリスティングについて説明します。ファイルではなく画面で見るには、両方のオプションを付けます。

```
tatara /l /p modes.as
```

この章の残りでは、リスティングの 1 行に何が表示されるか、リスティングがどのようにページに分かれるか、そしてリスティングに何を表示するかをソースから決める疑似命令について説明します。

18.1　リスティングの 1 行

リスティングの 1 行の欄を図 18.1 に示します。最初にアドレス、次にバイト、そのあとにソース行がそのまま表示されます。

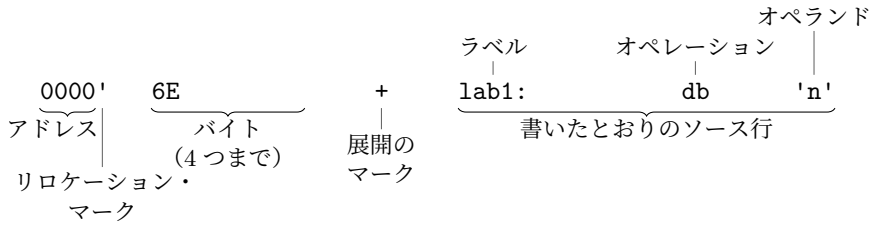

図 18.1: リスティングの 1 行の欄

次のソースには、それぞれの種類の行が 1 つずつあります。

```
                extrn   far
five            equ     5
                aseg
                org     4000h
abs:            nop
                cseg
code:           ld      hl,data
                call    far
                db      1,2,3,4,5,6,7,8,9
only:
                dseg
data:           ds      2
                end
```

MSX-DOS

```
                                        extrn   far
 0005                       five        equ     5
                                        aseg
                                        org     4000h
 4000   00                  abs:        nop
                                        cseg
 0000'  21 00 00            code:       ld      hl,data
 0003'  CD 00 00                        call    far
 0006'  01 02 03 04                     db      1,2,3,4,5,6,7,8,9
 000A'  05 06 07 08
 000E'  09
 000F'                      only:
                                        dseg
 0000'                      data:       ds      2
                                        end
```

- **アドレス**は、その行がアセンブルされたアドレスで、16 進数で表示されます。`only:` や `data: ds 2` のように、バイトを作らずにラベルを定義する行には、そのラベルのアドレスが表示されます。`EQU` の行では、アドレスの欄に名前の値が表示されます。
- **リロケーション・マーク**のアポストロフィーは、リロケータブルなアドレス、つまり TANREN が仕上げるコード・セグメントまたはデータ・セグメントの中のオフセットのあとに付きます（10 章）。4000h のようなアブソリュート・セグメントのアドレスには付きません。コードとデータで同じマークです。
- **バイト**は、その行が作ったバイトで、メモリに並ぶ順に、1 行に 4 つまで表示されます。それより多く作る行は、アドレスとバイトだけの行に続きます。9 バイトの `DB` は 3 行になります。
- そのあとに、**ソース行**が書いたとおりに表示されます。

`EXTRN`、`ASEG`、`ORG` のように、何も作らず、何も定義しない行では、アドレスとバイトの欄は空になります。

オペランドがリロケータブルまたは外部のときは、TANREN が仕上げる前の、オブジェクト・ファイルにあるとおりのバイトが表示されます。`data` ではセグメントの中のオフセットの 0000h、外部の名前 `far` では 0 です。これらのバイトにはマークが付きません。M80 のリスティングでは、このようなオペランドは 16 ビットの値 1 つにマークを付けて、`0000"`や `0000*`のように表示されますが、TATARA はメモリに並ぶ順にバイトを 1 つずつ表示し、マークは付けません。

マクロの展開または繰り返しブロックから作られた行には、ソース行の前の欄に`+` が付きます（16 章）。

18.2 ページ

リスティングはページに分かれます。各ページは**ページ見出し**（page heading）で始まります。ページ見出しは、タイトル、TATARA の名前とバージョン、日付、ページ番号の行と、サブタイトルの行、空の行からなります。プリンターが新しい紙に印刷を始めるように、2 ページ目からは各ページがフォーム・フィードで始まります。1 ページには、標準でソースの 56 行が入ります。

ページは、次の 3 つの疑似命令で制御します。

- `TITLE text` はタイトルを設定します。タイトルはページ見出しの左に表示されます。
- `SUBTTL text` はサブタイトルを設定します。サブタイトルはページ見出しの下の行に表示されます。

- PAGE は、その行のあとで新しいページを始めます。PAGE n は 1 ページの長さを n 行にし、同じように新しいページを始めます。この行数はページ全体の行数で、そのうち 4 行はページ見出しに使われるので、page 12 ではソースに 8 行が残ります。n は 10 から 255 までででなければなりません。1 桁目に書いた M80 の*EJECT と$EJECT は、数のない PAGE の別名です。

TATARA は、タイトルとサブタイトルの最初の 28 文字を残し、それよりあとは捨てます。

次のソースは、タイトルとサブタイトルを設定し、ページを短くして、後半を新しいページから始めます。

```
                title   Page test
                subttl  First part
                page    12
                cseg
                db      1
                db      2
                ...
                db      10
                subttl  Second part
                page
                db      11
                end
```

MSX-DOS

```
Page test       Tatara v1.2.0   30-Sep-26       PAGE    1


                                                title   Page test
                                                subttl  First part
                                                page    12
Page test     Tatara v1.2.0   30-Sep-26       PAGE    1-1
First part

                                                cseg
  0000'   01                                    db      1
  0001'   02                                    db      2
  0002'   03                                    db      3
  0003'   04                                    db      4
  0004'   05                                    db      5
  0005'   06                                    db      6
  0006'   07                                    db      7
Page test     Tatara v1.2.0   30-Sep-26       PAGE    1-2
First part

  0007'   08                                    db      8
  0008'   09                                    db      9
  0009'   0A                                    db      10
```

```
                                                subttl  Second part
                                                page
Page test     Tatara v1.2.0   30-Sep-26       PAGE    1-3
Second part

  000A'   0B                                    db      11
                                                end
```

2ページ目からの見出しが2桁左にずれているのは、それぞれがフォーム・フィードで始まり、フォーム・フィードはページ上で場所を取らないためです。このリスティングから、次のことがわかります。

- TITLE の行は見出しのあとにありますが、最初のページにはすでにタイトルがあります。TATARA はタイトルを第1パスから引き継ぐので（15章）、最初のページにはソースの最後のタイトルが表示されます。それからは、各ページには、そのページが始まったときのタイトルが表示されます。ソースの最初に TITLE を1つだけ書けば、すべてのページにそれが表示されます。
- サブタイトルは第1パスから引き継がれず、SUBTTL の行の次のページから表示されます。ソースの新しい部分を、それ自身のサブタイトルを付けて新しいページから始めるには、上の例のように SUBTTL のあとに PAGE を書きます。
- ページ番号は1、1-1、1-2、1-3と進みます。PAGE で、または前のページがいっぱいになって始まったページは、同じページの一部として番号が付きます。主番号はそのままで、ダッシュのあとの番号が増えます。

主番号は、ソースの中のフォーム・フィードで増えます。フォーム・フィード（文字コード 0Ch）だけの行は新しいページを始め、そのページの番号は2、3と進み、そのあとのページは2-1、2-2となります。これはM80と同じページ番号の付け方で、フォーム・フィードで区切ったソースを、部分ごとに1つの主番号で印刷できます。

18.2.1 最後のページ

ファイルに書き出したリスティングは、番号が S のページで終わります。このページには、ソースが定義したマクロの名前と、シンボルとその値が表示されます。

```
        Tatara v1.2.0   30-Sep-26       PAGE    S


Macros:

Symbols:
000F'   only            0000*   far             0000'   data
4000    abs             0000'   code            0005    five
```

これは 18.1 節のソースのリスティングで、このソースはマクロを定義していません。リロケータブルな値には、リスティングの行と同じアポストロフィーが付きます。外部の名前はこのモジュールでは値を持たないので、0000*と表示されます。シンボル表については、19 章で説明します。/P で画面に表示したリスティングには、このページはありません。

18.3　リスティングを止める、再開する

.XLIST はリスティングを止め、.LIST は再開します。その間の行は普通にアセンブルされますが、リスティングには表示されません。.XLIST の行自体も表示されませんが、.LIST の行は表示されます。

```
                cseg
                db      1
                .xlist
                db      2
                db      3
                .list
                db      4
                end
```

MSX-DOS

```
                                                cseg
0000'   01                                      db      1
                                                .list
0003'   04                                      db      4
                                                end
```

アドレスは 0000h から 0003h に進んでいます。表示されない 2 行も、それぞれ 1 バイトずつ作っているためです。

いちばんよく使うのは、インクルード・ファイルの前後です。そうしないと、インクルード・ファイルの行はすべてリスティングに表示されます（5 章）。

```
                .xlist
                include msxdos.inc
                .list
```

18.4　マクロの展開

マクロの展開や繰り返しブロックをリスティングにどれだけ表示するかは、次の 3 つの疑似命令で決まります。それぞれ、その行から、3 つのうちのほかの疑似命令が来るまで有効です。

- .XALL は標準の設定で、展開の行のうち、バイトを作る行を表示します。
- .LALL は、アセンブルされない条件付きアセンブルの行も含めて、展開のすべての行を表示します。
- .SALL は、展開の行を 1 つも表示しません。マクロを使った行が、展開の代わりであることを示す+を付けて表示されます。

次のマクロには、アセンブルされることのない行があります。ソースは、それぞれの設定でこのマクロを 1 回ずつ使います。

```
two             macro
                db      1
                if      0
                db      2
                endif
                ld      a,3
                endm
                cseg
                two
                .lall
                two
                .sall
                two
                .xall
                two
                end
```

```
                                        two
0000'   01              +               db      1
0001'   3E 03           +               ld      a,3
                                        .lall
                                        two
0003'   01              +               db      1
                        +               if      0
                        +               db      2
                        +               endif
0004'   3E 03           +               ld      a,3
                                        .sall
                        +               two
                                        .xall
                                        two
0009'   01              +               db      1
000A'   3E 03           +               ld      a,3
```

.LALL では、アセンブルされなかった行にはアドレスがありません。.SALL では、展開があったことを示すのは、マクロを使った行の+とアドレスだけです。アドレスは 0004h から 0009h に進んでいて、展開の 3 バイトは 0006h から 0008h にあります。

18.5　アセンブルされない行

条件付きアセンブル（15 章）が飛ばした行をリスティングに表示するかどうかは、さらに 3 つの疑似命令で決まります。

- .LFCOND は標準の設定で、それらの行をアドレスなしで表示します。
- .SFCOND は、それらの行を表示しません。IF、ELSE、ENDIF の行は表示されます。
- .TFCOND は、この 2 つを切り替えます。.LFCOND のあとでは.SFCOND として、.SFCOND のあとでは.LFCOND として働きます。

次のソースには、偽になる条件付きアセンブルが 5 つあり、それぞれの疑似命令の前に 1 つずつ、最後の疑似命令のあとに 1 つあります。

```
                cseg
                if      0
                db      1
                endif
                .sfcond
                if      0
                db      2
                endif
```

```
                .lfcond
                if      0
                db      3
                endif
                .tfcond
                if      0
                db      4
                endif
                .tfcond
                if      0
                db      5
                endif
                end
```

MSX-DOS

```
                                        cseg
                                        if      0
                                        db      1
                                        endif
                                        .sfcond
                                        if      0
                                        endif
                                        .lfcond
                                        if      0
                                        db      3
                                        endif
                                        .tfcond
                                        if      0
                                        endif
                                        .tfcond
                                        if      0
                                        db      5
                                        endif
                                        end
```

2 つ目と 4 つ目の条件付きアセンブルの行は表示されていません。M80 も、このソースを同じように表示します。

18.6 メッセージ

PAGE の長さが 10 から 255 の範囲にないと、TATARA は次のメッセージを表示して止まります。

MSX-DOS

```
PAGEBAD.AS(1): ERROR: a PAGE length must be 10 to 255.
```

ほかのリスティングの疑似命令は、オペランドを取らないか、どんなテキストでも取るので、独自のメッセージはありません。

第 19 章

シンボル表の出力と診断用スイッチ

19.1 シンボル表 139
19.2 シンボル表の読み方 141
19.3 シンボル表とリスティング 144
19.4 診断用スイッチ 144

/S を付けると、TATARA はアセンブルが終わったときに**シンボル表**（symbol table）を表示します。ソースが定義したすべての名前が、その値とともに、セグメントごとにまとめて表示されます。ラベルがどこに置かれたか、EQU にどんな値が与えられたか、モジュールがどの名前をパブリックにし、どの名前をほかのモジュールに求めているかを確かめるには、これがいちばん手早い方法です。

この章では、TATARA がソースをどう読んだかを示す 3 つのオプション、/F、/M、/H についても説明します。これらはプログラムを書くためには必要ありませんが、TATARA の問題を報告するときには、その出力を一緒に送ってください。

19.1 シンボル表

/S は、最後の行をアセンブルしたあと、要約行の前にシンボル表を表示します。そのため、要約行は画面の最後に残ります。リスティング・ファイルを書き出すときでも、シンボル表は必ず画面に表示されます。

次のソースは、それぞれの種類の名前を 1 つずつ定義しています。

```
                public  start,count,limit,level
                extrn   print,exit
limit           equ     100
level           defl    1
level           defl    2
tmp             defl    3
                cseg
start:          ld      hl,msg
                call    print
                call    MIXED
                jp      exit
wait            macro
                local   loop
loop:           djnz    loop
                endm
                wait
                wait
Mixed:          ret
after           equ     start+1
                dseg
count:          ds      1
msg:            db      'Hi$'
                cseg    music
tune:           db      0
                end
```

MSX-DOS

```
A:\>tatara /s syms.as syms.tro
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

ASEG - absolute

0002h  public var level
0003h  var        tmp
0064h  public     limit

CSEG - default code segment, 0011h bytes

0000h  public     start
0001h             after
000Ch             ??0000
000Eh             ??0001
0010h             Mixed

DSEG - default data segment, 0004h bytes

0000h  public     count
0001h             msg
```

```
music - named code segment, 0001h bytes

0000h             tune

EXTERNAL - resolved by the linker

                  exit
                  print

ended at SYMS.AS(26)
```

/Q も付けると、バナーと要約行が省かれ、シンボル表だけが表示されます。

19.2 シンボル表の読み方

19.2.1 セクション

シンボル表には、モジュールのセグメント（10 章）ごとに 1 つずつセクションがあり、そのあとに外部の名前のセクション、PUBLIC で宣言して定義していない名前のセクションが続きます。各セクションは見出しの行で始まります。見出しの形を表 19.1 に示します。

見出しの大きさは、そのセグメントがこのモジュールで占めるバイト数です。アブソリュート・セグメントのバイトはメモリのどこにでも置けるので、アブソリュート・セグメントには大きさがありません。

3 つの標準のセグメントには、シンボルがなくても必ずセクションがあります。何も定義しないソースでも、次のように表示されます。

MSX-DOS

```
ASEG - absolute

(no symbols)

CSEG - default code segment, 0001h bytes

(no symbols)

DSEG - default data segment, 0000h bytes

(no symbols)
```

名前付きセグメント、外部の名前、定義されていない名前のセクションは、入れるものがあるときにだけ表示されます。

見出し	セクション
`ASEG - absolute`	アブソリュート・セグメントと、すべてのアブソリュートな値
`CSEG - default code segment, 0011h bytes`	標準のコード・セグメント
`DSEG - default data segment, 0004h bytes`	標準のデータ・セグメント
`music - named code segment, 0001h bytes`	名前付きコード・セグメント
`scratch - named data segment, transient, group reading, 0042h bytes`	名前付きデータ・セグメント。この例はトランジェントで、グループがある（10.6 節）
`EXTERNAL - resolved by the linker`	`EXTRN` で宣言した名前
`UNDEFINED - named by PUBLIC, never defined`	`PUBLIC` で宣言して定義していない名前

表 19.1: シンボル表のセクションの見出し

19.2.2　シンボルの行

セグメントのセクションの各行には、シンボルの値、種類、名前が表示されます。行は値の順に並びます。

- **値**は、セグメントの中のオフセットです。`Mixed` は、TANREN がコード・セグメントをどこに置いても、その始まりから 10h バイトのところにあります。アブソリュート・セグメントでは、値そのものです。`after equ start+1` のように値がリロケータブルな `EQU` は、その値のセグメントに属します。`DEFL` の名前には、最後に与えられた値が表示されます。`level` は 2 です。
- **種類**は、表 19.2 の言葉のどれかです。
- **名前**は、定義したところで書いたとおりに表示されます。このソースは `Mixed:` を定義する前に `MIXED` を使っていますが、シンボル表には `Mixed` と表示されます。`/C` を付けると、これらは別の名前になります（7 章）。

`??0000` と `??0001` は、`wait` を 2 回使ったときに `LOCAL` が作った名前です（16 章）。これらはふつうのラベルで、使うたびに 1 つずつ作られます。

種類	名前
(空白)	ラベルまたは EQU の名前
public	パブリックにしたラベルまたは EQU の名前
var	DEFL の名前
public var	パブリックにした DEFL の名前

表 19.2: シンボル表の種類

19.2.3 値を持たない名前

外部の名前は、それを使うモジュールの中では値を持ちません。値は TANREN がほかのモジュールから見つけます (25 章)。外部の名前は、値も種類もなしに、特に決まっていない順で表示されます。

PUBLIC で宣言して定義していない名前があると、TATARA はオブジェクト・ファイルを書き出すときに止まります (12 章)。オブジェクト・ファイルがなければ、ソースは受け付けられ、/S はその名前を専用のセクションに表示します。次のソースは、2 つの名前をパブリックと宣言し、そのうち 1 つだけを定義しています。

```
                public  here,nowhere
here:           ret
                end
```

MSX-DOS

```
A:\>tatara /s undef.as
...
CSEG - default code segment, 0001h bytes

0000h  public     here

DSEG - default data segment, 0000h bytes

(no symbols)

UNDEFINED - named by PUBLIC, never defined

                  nowhere
```

19.3　シンボル表とリスティング

リスティング・ファイルは、独自のシンボルのページで終わります（18.2.1 節）。19.1 節のソースでは、次のようになります。

MSX-DOS

```
Macros:
wait

Symbols:
000E'   ??0001          000C'   ??0000          0000*   exit
0003    tmp             0000'   tune            0001'   after
0010'   Mixed           0000'   count           0001'   msg
0000'   start           0002    level           0000*   print
0064    limit
```

このページには同じシンボルが、M80 と同じ形で表示されます。1 行に 3 つずつ、特に決まっていない順で並び、リロケータブルな値には'、外部の名前には*が付きます。値がどのセグメントに属するかや、名前がパブリックかどうかは表示されません。`tune` と `start` はどちらも `0000'` ですが、一方は `music` に、もう一方は標準のコード・セグメントにあります。`/S` のシンボル表にはその両方が表示されるので、それが問題になるときは、こちらを見てください。

`/L` と `/S` を一緒に付けると、両方が出力されます。ページはリスティング・ファイルに、シンボル表は画面に出力されます。

19.4　診断用スイッチ

`/F`、`/M`、`/H` は、TATARA がソースから何を作ったかではなく、ソースをどう読んだかを示します。TATARA がある行を思いがけないように扱ったときは、その行に関係するオプションを付けてもう一度アセンブルし、その出力を報告と一緒に送ってください。MSX-DOS では、>で出力をファイルに保存できます。

```
tatara /f prog.as prog.tro > fields.txt
```

19.4.1　/F：各行のフィールド

`/F` を付けると、各行をリスティングに表示する代わりに、TATARA がその行をどのようにフィールドに分けたか（6 章）が表示されます。次のソースには、コメント、条件付きアセンブルで飛ばされる行、マクロ、インクルード・ファイル、セミコロンを含む文字列があります。

```
; FIELDS.AS - what /F shows.
two             macro   x
                db      x,x
                endm
                cseg
start:          ld      hl,msg          ; a comment
                if      0
                nop
                endif
                two     5
                include finc.inc
msg:            db      'a;b'
                end
```

FINC.INC には、コメントの行と ret があります。

MSX-DOS

```
A:\>tatara /f fields.as fields.tro
...
[][][][; FIELDS.AS - what /F shows.][00][00:0001]
[two][macro][x][][01][00:0002]
[][cseg][][][1A][00:0005]
[start][ld][hl,msg][; a comment][00][00:0006]
[][if][0][][08][00:0007]
[][nop][][][00][00:0008]
[][endif][][][13][00:0009]
[][two][5][][00][00:000A]
[][db][5,5][][20][00:0003]
[][include][finc.inc][][14][00:000B]
[][][][; FINC.INC - included by FIELDS.AS.][00][01:0001]
[][ret][][][00][01:0002]
[msg][db]['a;b'][][20][00:000C]
[][end][][][15][00:000D]
ended at FIELDS.AS(13)
```

出力の各行は、角かっこで囲んだ 6 つの部分からなります。

1. ラベル（コロンは除く）
2. オペレーション
3. オペランド
4. コメント
5. TATARA が疑似命令ごとに付けている番号（16 進数）。疑似命令でない行では 00

6. その行がどこから来たか。ファイルの番号（コマンド・ラインで指定したソースが 00、インクルード・ファイルは開いた順に 01 から）と、そのファイルの中での行番号（16 進数）

出力には、各行が第 1 パスで読まれたとおりに表示されます。この nop のように、条件付きアセンブルで飛ばされた行も表示されます。マクロの展開から作られた行には、それが来たマクロ本体の行が表示されます。db 5,5 は、db x,x を書いた 3 行目です。マクロ本体の行は、マクロを使ったときにだけ表示され、定義したときには表示されません。'a;b' の中のセミコロンはオペランドの一部で、コメントの始まりではありません。

/F の出力は、/P を付けても付けなくても画面に表示されます。

19.4.2　/M：マクロの定義

/M を付けると、マクロの定義を読んだときに、TATARA が記憶したとおりの定義が表示されます。

```
; MDUMP.AS - what /M shows.
put             macro   x,y
                local   skip
                ld      a,x             ; kept
                jr      skip            ;; dropped
lab&y:          db      '&x'
skip:
                endm
                cseg
                put     1,2
                rept    2
                nop
                endm
                end
```

MSX-DOS

```
A:\>tatara /m mdump.as mdump.tro
...
MACRO put: 2 parameters, 1 locals, 4 lines, MDUMP.AS(2)
  4 |           ld      a,<0>           ; kept
  5 |           jr      <L0>
  6 | lab<1>:           db      '<0>'
  7 | <L0>:
ended at MDUMP.AS(14)
```

最初の行には、マクロの名前、パラメーターの数、LOCAL の名前の数、本体の行数、定義が始まる場所が表示されます。そのあとに、本体の各行が、ソースでの行番号とともに表示されます。

- パラメーターは、MACRO の行での順に<0>、<1>のように、LOCAL の名前は<L0>、<L1>のように表示されます。lab&y は lab<1>になっています。&の働きはここで済んでいます（16.3 節）。
- LOCAL の行と ENDM の行は記憶されません。
- ;; で始まるコメントは捨てられています（16.4 節）。

ソースの最後の REPT のブロックは表示されません。/M が表示するのはマクロだけで、繰り返しブロック（17 章）は表示しません。

19.4.3　/H：使用中のメモリ

TATARA は、シンボルやマクロを、メモリ・マッパーの中のメモリのブロックに置き、必要になったときにブロックを取ります。/H を付けると、アセンブルが終わったときにまだ使われているブロックの数が、要約行のあとに表示されます。

MSX-DOS

```
A:\>tatara /h heap1.as heap1.tro
...
ended at HEAP1.AS(8)
HEAP: 7 blocks.
```

この数が 0 になることはありません。シンボル 1 つにブロック 1 つ、マクロの定義には名前と本体のブロックが使われ、TATARA 自身の記録もいくつかのブロックを使うからです。この数は、ほかのソースの数と比べるときにだけ役に立ちます。LOCAL を持つマクロは、使うたびに新しい??の名前を定義するので、使うたびにブロックが 1 つ増えます。同じ wait を 1 回ではなく 3 回使うと、HEAP: 9 blocks. となります。

第 20 章

エラーメッセージ

20.1 メッセージの内容 149
20.2 TATARA は最初のエラーで止まる 151
20.3 経路 152
20.4 第 2 パスで見つかるエラー 155
20.5 内部エラー 155

TATARA は、ソースに誤りを見つけると、**エラーメッセージ**（error message）を 1 つ表示して止まります。この章では、メッセージの読み方、止まったときに TATARA がディスクに何を残すか、そして誤りがインクルード・ファイル、マクロ、繰り返しブロックの中にあるときにメッセージのあとに続く**経路**（trail）の行のたどり方を説明します。すべてのメッセージとその原因は、付録 D に示します。

20.1 メッセージの内容

メッセージは、誤りのあるファイルと行を示し、何が誤っているかを述べます。

```
; LERR.AS - an error half way down a listing.
                cseg
                ld      a,1
                ld      b,2
                ld      (bc),b
                ld      c,3
                end
```

MSX-DOS

```
A:\>tatara /l lerr.as lerr.tro lerr.lst
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

LERR.AS(5): ERROR: not a form this instruction has.
```

- **ファイル**は、その行があるファイルです。コマンド・ラインで指定したソース、インクルード・ファイル、またはマクロを定義したファイルのどれかです。TATARA はファイル名を大文字で表示します。
- **行**は、そのファイルの先頭を 1 として数えます。LERR.AS の先頭のコメントが 1 行目なので、`ld (bc),b` は 5 行目です。
- ERROR: のあとの**テキスト**は、何が誤っているかを述べます。ここでは `ld (bc),b` は正しく書かれていますが、Z80 にはそのような命令がありません。

閉じられていないマクロの定義や条件付きアセンブルは、ソースの最後で初めて見つかります。そのメッセージには、ファイルの最後の行ではなく、それを始めた行、つまり MACRO や IF の行が示されます。

行を示さないメッセージもあります。コマンド・ラインの誤りは、ファイルを読む前に見つかるので、メッセージはバナーの前に表示されます（5.6 節）。

MSX-DOS

```
A:\>tatara /z good.as
ERROR: unknown option.
```

開けなかったファイルは、ERROR: cannot open NOTHERE.INC のように、メッセージの中でその名前が示されます（13 章）。

/Q はバナーと要約行を省きますが、エラーメッセージは省きません。

MSX-DOS

```
A:\>tatara /q lerr.as
LERR.AS(5): ERROR: not a form this instruction has.
```

20.2 TATARA は最初のエラーで止まる

TATARA が一度に表示するメッセージは 1 つだけです。誤りを見つけるとそこで止まり、ソースの残りは見ません。その行を直してもう一度アセンブルすると、次の誤りがあれば、TATARA はそこへ進みます。

TATARA が書きかけのオブジェクト・ファイルを残すことはありません。何が残るかは、誤りをいつ見つけたかによって決まります。

- **第 1 パス**で見つけた場合（ほとんどの誤りはここで見つかります）、TATARA はまだオブジェクト・ファイルを作っていません。以前の実行で作った同じ名前のオブジェクト・ファイルは、そのまま残ります（5 章）。
- **第 2 パス**で見つけた場合、TATARA はすでに新しいオブジェクト・ファイルを作っています。TATARA はそれを削除するので、以前の実行で作ったものも含めて、その名前のオブジェクト・ファイルは残りません。次の例で、`GOOD.AS` は正しいソースで、`PUB.AS` には `PUBLIC` で宣言して定義していない名前があります。この誤りは、オブジェクト・ファイルを仕上げるときに見つかります。

MSX-DOS

```
A:\>tatara good.as pub.tro
...
ended at GOOD.AS(5)
A:\>dir pub.tro
...
PUB      TRO        46 26-09-30  9:04
 46 bytes in 1 file   3800K free
A:\>tatara pub.as pub.tro
...
PUB.AS(7): ERROR: a PUBLIC name was never defined.
A:\>dir pub.tro
...
 0 bytes in 0 files     3808K free
```

第 1 パスの場合は古いオブジェクト・ファイルが残っていることがあり、TANREN は何も言わずにそれをリンクします。エラーのあとは、TATARA がエラーなしでソースをアセンブルするまで、リンクしないでください。

リスティングは第 2 パスで書き出されるので（18 章）、第 1 パスで見つかった誤りでは、リスティングの行が 1 行も書き出される前に TATARA が止まります。20.1 節のコマンド・ラインで指定した `LERR.LST` は作られず、代わりに `/L` と `/P` を付けても、`LERR.AS` ではメッセージだけが表示されます。

20.3　経路

誤りのある行が、コマンド・ラインで指定したソースから直接来たものでないときは、メッセージのあとに経路が続きます。経路は、TATARA がその行にたどり着くまでの各段階を 1 行ずつ、誤りのところから外側へ向かって示します。経路の各行は 4 文字字下げされ、次のどれかの形をしています。

- `included from FILE(line)`：上の行のファイルは、その行の `INCLUDE` でインクルードされました（13 章）。
- `in NAME, called from FILE(line)`：上の行はマクロ `NAME` の一部で、その行がマクロを使いました（16 章）。
- `in a repeat block, called from FILE(line)`：上の行は、その行の `REPT`、`IRP`、`IRPC` のブロックの一部です（17 章）。
- `in a redefined macro, called from FILE(line)`：マクロと同じですが、使われたあとでマクロが新しく定義し直されたので、名前はもうそのマクロのものではありません（16.7 節）。

20.3.1　経路の読み方

次の例では、繰り返しブロックの中に誤りがあります。そのブロックはマクロ `inner` の中にあり、`inner` はマクロ `outer` が使い、`outer` はインクルード・ファイルの中で使われています。`MACS.INC` は 2 つのマクロを定義しています。

```
; MACS.INC - two macros, one using the other, with a repeat block.
outer           macro
                nop
                inner
                endm
inner           macro
                rept    1
                ld      (bc),b
                endm
                endm
```

`USE.INC` は 2 行目で `outer` を使い、`TRAIL.AS` は両方のファイルをインクルードします。

```
; TRAIL.AS - an error in a repeat block, in a macro, in a macro, in an
; include file.
                include macs.inc
                cseg
                include use.inc
                end
```

MSX-DOS

```
MACS.INC(8): ERROR: not a form this instruction has.
    in a repeat block, called from MACS.INC(7)
    in inner, called from MACS.INC(4)
    in outer, called from USE.INC(2)
    included from TRAIL.AS(5)
```

図 20.1 では、メッセージの各行に番号を付け、その行が示すソースの行に同じ番号を付けています。

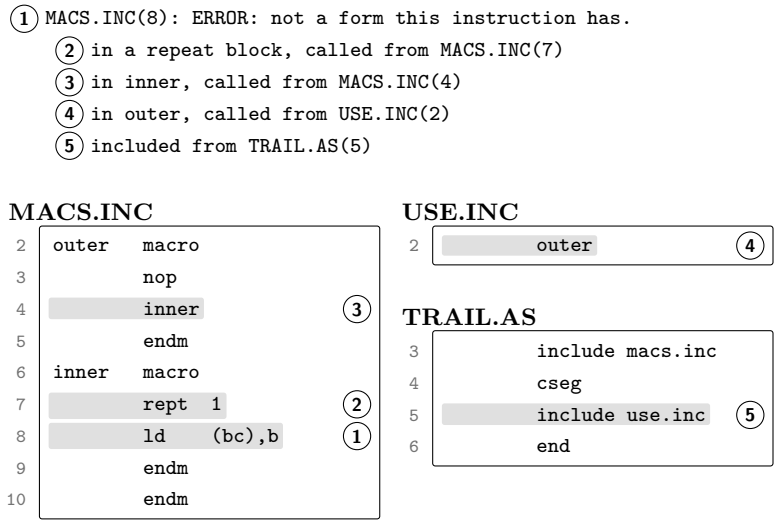

図 20.1: メッセージと経路の各行と、それが示すソースの行

経路は上から順に読みます。

1. 誤りは MACS.INC の 8 行目、ld (bc),b にあります。
2. その行は、7 行目から始まる繰り返しブロックの一部です。
3. その繰り返しブロックは inner の一部で、inner は outer の中の 4 行目で使われています。
4. outer は USE.INC の 2 行目で使われています。

5. USE.INC は、コマンド・ラインで指定したソース TRAIL.AS の 5 行目でインクルードされています。

最初の行は誤りのある場所を示し、最後の行は必ずコマンド・ラインで指定したソースの中にあります。その間の行は、一方からもう一方への道筋を示します。

経路に表示されないものが 2 つあります。USE.INC の 2 行目について独自の included from の行がないのは、called from USE.INC(2) ですでに示されているからです。また、3 行目の include macs.inc はまったく表示されません。経路は、TATARA が誤りのある行にどうたどり着いたかを示すもので、MACS.INC はそれよりずっと前に最後まで読み終わっています。そこから来たのは、マクロの定義だけです。

20.3.2　定義し直されたマクロ

実行中に自分自身を定義し直すマクロ（16.7 節）は、経路の中で自分の名前を持たなくなります。

```
; REDEF.AS - a macro that defines itself again, then fails.
once            macro
once            macro
                endm
                ld      (bc),b
                endm
                cseg
                once
                end
```

MSX-DOS

```
REDEF.AS(5): ERROR: not a form this instruction has.
    in a redefined macro, called from REDEF.AS(8)
```

名前 once は、今では 3 行目で定義した空のマクロのものなので、経路は実行中のマクロにその名前を使えません。

20.4 第 2 パスで見つかるエラー

TATARA はソースを 2 回読みます（15.2 節）。いくつかの誤りは、2 回目に読んで 1 回目と比べたときに初めてわかります。

- `phase error - this label had a different value on pass 1.` ラベルより前の何かが、2 つのパスで異なる大きさにアセンブルされました。多くの場合、条件付きアセンブルの答えが違ったためです（15.2 節）。
- `this label was defined on pass 1 and not on pass 2 - a conditional skipped it.` 上と同じですが、ラベルがまったく飛ばされました。いちばん多い原因は、2 回インクルードしたファイルです（13.6 節）。
- `a PUBLIC name was never defined.` 第 2 パスの最後、オブジェクト・ファイルを仕上げるときに見つかります（12 章）。

これらのエラーのあとで、TATARA は書き出していたオブジェクト・ファイルを削除します（20.2 節）。

20.5 内部エラー

`internal - this line emitted more bytes than fit.` のように `internal -` で始まるメッセージは、**内部エラー**（internal error）です。TATARA は、ソースではなく自分自身の不具合を見つけました。次のものを付けて報告してください。

- ソースと、それがインクルードするすべてのファイル
- コマンド・ライン
- TATARA が表示したすべての出力
- 同じコマンドに `/F` を付けたときの出力。その行がマクロから来たものなら `/M` を付けたときの出力（19.4 節）

`out of mapper memory.` は内部エラーではありません。ソースが、TATARA の表のためにメモリ・マッパーにある以上のメモリを必要としています。

第 III 部

リンカー Tanren

21 TANREN の実行 159
22 リンク・ファイル 167
23 TANREN がオブジェクト・ファイルを探す場所 173
24 プログラムの配置 179
25 モジュール間のシンボル 187
26 出力形式 197
27 リンカーのエラーメッセージ 207

第 21 章

TANRENの実行

21.1 コマンド・ライン 159
21.2 出力ファイル 162
21.3 オプション 162
21.4 TANREN が表示するもの 163
21.5 127 文字のコマンド・ライン. 165
21.6 TANREN が始まらないとき. 165

TANREN は、TATARA が作ったオブジェクト・ファイルを 1 つのプログラムにまとめます。この章では、TANREN の実行のしかた、つまりコマンド・ライン、出力ファイルの名前の付き方、オプション、TANREN が表示するもの、そしてリンクを始める前のメッセージについて説明します。そのほかのことは、あとの章で詳しく説明します。リンク・ファイル（22 章）、TANREN がオブジェクト・ファイルを探す場所（23 章）、プログラムのメモリへの配置（24 章）、モジュールの間での名前の対応（25 章）、出力ファイルの種類（26 章）、エラーメッセージ（27 章）です。

TATARA と同じように、TANREN には MSX-DOS2 または Nextor と、メモリ・マッパーが必要です。

21.1 コマンド・ライン

コマンド・ラインは、**tanren** のあとにオブジェクト・ファイルの名前とオプションを書いたものです。

```
tanren [options] object [object...]
```

拡張子なしで指定したオブジェクト・ファイルには.troが付くので、**tanren main sub** は MAIN.TRO と SUB.TRO をリンクします。名前でもオプションでも、文字は大文字でも小文字でもかまいません。

次の 2 つのモジュールで、小さなプログラムができます。MAIN.AS は、SUB.AS にある greet を呼び出します。どちらも、MSX-DOS のファンクションの名前と system マクロのために MSXDOS.INC をインクルードし、SUB.AS はさらに、表示する行を終わらせるキャリッジ・リターンとライン・フィード、CHR_CR と CHR_LF のために ASCII.INC をインクルードします（14 章）。

```
; MAIN.AS - calls greet, in another module, and ends the program.
                include msxdos.inc
                extrn   greet
                cseg
start:          call    greet
                system  _TERM0
                end     start
```

```
; SUB.AS - greet prints a line.
                include msxdos.inc
                include ascii.inc
                public  greet
                cseg
greet:          ld      de,msg
                system  _STROUT
                ret
                dseg
msg:            db      'Hello from TANREN.',CHR_CR,CHR_LF,'$'
                end
```

それぞれをアセンブルしてオブジェクト・ファイルにし、TANREN が 2 つをリンクして MAIN.COM にします。

MSX-DOS

```
A:\>tatara /q main.as main.tro
A:\>tatara /q sub.as sub.tro
A:\>tanren main sub
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

2 modules, 16 records, ends at EOF.
Wrote MAIN.COM, 0100-0125 (38 bytes), entry 0100.
A:\>main
Hello from TANREN.
```

21.1.1 オブジェクト・ファイルの順序

TANREN は、モジュールを指定した順にメモリに置きます。最初のモジュールのコードをメモリ・アドレス 0100h に、次のモジュールのコードをそのすぐあとに、というように続けます（データの置き場所は 24 章で説明します）。MSX-DOS は .COM のプログラムを必ずメモリ・アドレス 0100h から始めるので、最初に実行するコードのあるモジュールを、最初に指定しなければなりません。

同じ 2 つのモジュールを逆の順に指定すると、greet のコードから始まるプログラムになります。

MSX-DOS

```
A:\>tanren sub main
...
WARNING: SUB.COM is entered at 0100, not at 0109.
2 modules, 16 records, ends at EOF.
Wrote SUB.COM, 0100-0125 (38 bytes), entry 0109.
```

要約行には、MAIN.AS の END の行にあるアドレス start が 0109h になったと表示されますが、.COM のプログラムはそこからは始まりません。MSX-DOS は 0100h にあるものを実行します。TANREN は要約の前に、/Q を付けていてもこのことを警告し、ファイルはそのまま書き出します。この警告は、/B なしで書き出すファイルのエントリー・アドレスが、ファイルの最初のアドレスと違うときに必ず表示されます。/B を付けると、エントリー・アドレスは BLOAD のヘッダーに入り、警告は出ません（26 章）。

21.1.2 名前の間の空白

名前とオプションは、空白またはタブで区切ります。それ以外の文字は、コンマも含めて名前の一部になります。

MSX-DOS

```
A:\>tanren main,sub
...
ERROR: cannot open MAIN,SUB.TRO
```

オプションは、名前の前、あと、名前の間のどこに書いてもかまいません。

21.2　出力ファイル

TANREN は出力ファイルに、最初のオブジェクト・ファイルの名前に拡張子.com を付けた名前を付け、そのオブジェクト・ファイルと同じディレクトリに書き出します。BLOAD 用のファイルを書き出す/B を付けると、拡張子は.bin になります。

MSX-DOS

```
A:\>tanren /b main sub
...
Wrote MAIN.BIN, 0100-0125 (38 bytes), BLOAD header, entry 0100.
```

/O: は出力ファイルの名前を指定します。名前はコロンのすぐあとに書きます。

MSX-DOS

```
A:\>tanren main sub /o:prog.com
...
Wrote PROG.COM, 0100-0125 (38 bytes), entry 0100.
```

名前は書いたとおりに使われます。TANREN は拡張子を付けないので、**/o:prog** では拡張子のない PROG というファイルが書き出され、MSX-DOS はこれをコマンドとして実行しません。

コマンド・ラインの誤りで入力のファイルを上書きしないように、出力ファイルはオブジェクト・ファイルのどれとも同じであってはなりません。

MSX-DOS

```
A:\>tanren /o:sub.tro main sub
...
ERROR: the output file SUB.TRO is also an input file.
```

TANREN は名前を書いたとおりに比べるので、2 つの異なるパスで指定した同じファイルには気が付きません。

21.3　オプション

TANREN のオプションを表 21.1 に示します。アドレスは 1 桁から 4 桁の 16 進数で、**/p:4000** のように h を付けずに書きます。/D:、/O:、/P: にはコロンが必要で、**/p** だけでは TANREN にないオプションになります。

/P: はコードの始まる場所を変え、要約行にそれが表示されます。

MSX-DOS

オプション	働き	章
/B	出力ファイルに BLOAD のヘッダーを書き、標準の拡張子を .bin にする	26
/D:addr	データ・セグメントを addr から始める	24
/M	セグメントとグループの表、そしてシンボルを表示する	25
/O:file	出力ファイルの名前を指定する	この章
/P:addr	コード・セグメントを addr から始める	24
/Q	バナーと要約行を省く	この章
/R	オブジェクト・ファイルのすべてのレコードを表示する（診断用）	付録 F
/V	バナーを表示して終わる	この章
/?	使い方の画面を表示して終わる	この章

表 21.1: TANREN のオプション

```
A:\>tanren /p:4000 /o:high.bin main sub
...
Wrote HIGH.BIN, 4000-4025 (38 bytes), entry 4000.
```

21.4 TANREN が表示するもの

TANREN は、TATARA と同じようにバナーを表示して始まり、2 行の要約で終わります。

MSX-DOS

```
2 modules, 16 records, ends at EOF.
Wrote MAIN.COM, 0100-0125 (38 bytes), entry 0100.
```

- **modules** は、TANREN が読んだオブジェクト・ファイルの数です。**records** は、その中にある情報のまとまりの数で、各セグメントのバイト、名前、TANREN がアドレスを書き込む場所などです（付録 F）。

- **ends at EOF** は、各オブジェクト・ファイルが、最後のレコードが示すところで終わっていたことを示します。そのあとに何かあるときは、代わりに `AND BYTES AFTER THEM.` と表示され、その前の行にそのファイルの名前が表示されます。TATARA はそのようなファイルを作らないので、そのモジュールをもう一度アセンブルしてください（27.3 節）。
- **Wrote** は、出力ファイルの名前、最初と最後のバイトのアドレス、そのバイト数を示します。`/B` を付けると、ディスク上のファイルは `BLOAD` のヘッダーの分だけ 7 バイト長くなり、この行にもそう表示されます。
- **entry** は、`END` の行で指定したアドレスです。`/B` のときにヘッダーに書き込まれます。`/B` なしでは、ファイルの最初のアドレスでなければならず、違うと TANREN は警告します。

書き出すバイトを持つモジュールが 1 つもないときは、2 行目は `Nothing to write: no module has any content.` になります。

`/Q` はバナーと要約を省きます。エラーは`/Q` があってもなくても表示されるので、`/Q` を付けて何も表示されなかったリンクは成功しています。`/V` はバナーを表示して終わり、`/?`、またはオブジェクト・ファイルを 1 つも指定しない **tanren** は、バナーと使い方の画面を表示します。

MSX-DOS

```
A:\>tanren
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

Usage:  TANREN [options] <object|@list> [more...]

Options:  /B Write a BLOAD header on the output file
          /D:<addr> Start the data segments at <addr>
          /M Print the segment and group tables
          /O:<file> Write the image to <file>
          /P:<addr> Start the code segments at <addr>
          /Q Omit the banner above and the summary line
          /R Print every record in the object files
          /V Print the program version
          /? Print this screen

<object>  : A .tro file; .tro is assumed when you leave it off.
@<list>   : A file of the same words; ; is a comment, .lnk
            is assumed.
<addr>    : One to four hex digits, as in /P:4000.
<file>    : The output. Default: the first object with .com
            (.bin with /B). It may not be an input file.
```

21.5 127 文字のコマンド・ライン

MSX-DOS がコマンドに与えるのは、プログラムの名前と空白を含めて 127 文字です。プロンプトでは、COMMAND2 はそれ以上を受け付けず、127 文字を超えてキーを押してもビープ音が鳴るだけです。バッチ・ファイルでは、それより長い行はまったく実行されず、COMMAND2 は`*** Command too long` と表示します。

モジュールの多いプログラムでは、それでは足りないことがあります。**リンク・ファイル**（link file）には、同じ言葉を必要なだけファイルに書いておけるので、コマンド・ラインには **`tanren @prog`** のようにそのファイルだけを書きます（22 章）。

ほかのプログラムがもっと長い行で TANREN を実行すると、MSX-DOS はそれを何も言わずに 127 文字で切ります。TANREN のコマンド・ラインがちょうど 127 文字のときは、何かが切られたかどうかがわからないので、TANREN は`/Q` があっても警告を表示します。

MSX-DOS

```
WARNING: the command line is 127 characters, which is all
MSX-DOS gives - anything past it was dropped in silence.
Use @<file> if a module is missing.
```

21.6 TANREN が始まらないとき

TANREN が何かをリンクする前に表示するメッセージを、表 21.2 に示します。最初の 3 つはバナーの前に表示されます。

どのモジュールも定義していない名前のように、リンクの途中で見つかる誤りについては、27 章で説明します。

メッセージ	意味
`ERROR: Tanren needs MSX-DOS2 or Nextor.`	MSX が MSX-DOS1 で動いています。
`ERROR: Tanren needs a memory mapper.`	MSX にメモリ・マッパーがありません。
`ERROR: unknown option.`	TANREN にないオプションか、コロンのない `/D`、`/O`、`/P` です。
`ERROR: cannot open NAME.TRO`	その名前のオブジェクト・ファイルが見つかりません（TANREN が探す場所は 23 章で説明します）。
`ERROR: the output file NAME is also an input file.`	出力ファイルがオブジェクト・ファイルを上書きしてしまいます。

表 21.2: リンクを始める前のメッセージ

第 22 章

リンク・ファイル

22.1 最初のリンク・ファイル 167
22.2 リンク・ファイルの書き方 168
22.3 リンク・ファイルとコマンド・ライン 169
22.4 メッセージ 170

リンク・ファイル（link file）は、TANREN のコマンド・ラインの言葉、つまりオブジェクト・ファイルの名前とオプションを書いたテキスト・ファイルです。**tanren @prog** はファイル PROG.LNK を読み、その言葉を**@prog** の位置に入力したものとして扱います。リンク・ファイルには 127 文字の制限がなく（21.5 節）、プログラムのモジュールの一覧をコメント付きで 1 か所にまとめておけるので、どのリンクでも同じ一覧が使われます。

22.1 最初のリンク・ファイル

次のリンク・ファイルは、21 章のプログラムをリンクします。

```
; PROG.LNK - the two modules of the example program.

main            ; the start of the program: it must come first
sub             ; greet

/o:prog.com     ; the output file
```

このファイルは、2 つのオブジェクト・ファイル MAIN.TRO と SUB.TRO、そして出力ファイル PROG.COM を指定しています。コマンド・ラインには、リンク・ファイルだけを書きます。

```
A:\>tanren @prog
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

2 modules, 16 records, ends at EOF.
Wrote PROG.COM, 0100-0125 (38 bytes), entry 0100.
A:\>prog
Hello from TANREN.
```

22.2　リンク・ファイルの書き方

リンク・ファイルはテキスト・ファイルで、どのエディターでも書けます。中身はコマンド・ラインと同じ言葉で、次の規則に従います。

- 言葉は、空白、タブ、行の終わりで区切ります。1 行に言葉を 1 つ書いても、いくつ書いても、1 つも書かなくてもかまいません。`main sub /o:one.com` を 1 行に書いた `ONELINE.LNK` は、出力ファイルの名前のほかは `PROG.LNK` と同じ働きをします。
- セミコロンはコメントの始まりで、コメントは行の終わりまで続きます。コメントは 1 行を占めても、行の言葉のあとに続けてもかまいません。
- 拡張子なしで指定したオブジェクト・ファイルには、コマンド・ラインと同じように `.tro` が付きます。
- オプションもほかの言葉と同じで、ファイルのどこに書いてもかまいません。
- ファイルは最後の文字で、または Ctrl+Z があればそこで終わります。最後の行は行の終わりで終わっていなくてもかまいません。

`@`のあとの名前に拡張子がなければ `.lnk` が付くので、**`tanren @prog`** と **`tanren @prog.lnk`** は同じファイルを読みます。TANREN は、リンク・ファイルを書いたとおりの名前で開きます。名前にパスが含まれていなければ、カレント・ディレクトリにあるものが開かれます。

22.2.1　コマンド・ラインに収まらない大きなプログラム

TATARA 自身は 19 のモジュールからなり、次のリンク・ファイルでリンクされます（最初の数行のコメントは、ここでは省いています）。

```
tatara.tro                              ; the driver
cmdline.tro srcline.tro fields.tro      ; the front
dirtab.tro macros.tro expand.tro        ; directives and macros
cond.tro expr.tro                       ; conditionals and expressions
symtab.tro optab.tro insn.tro           ; the tables and the back end
emit.tro objout.tro errs.tro

msxdos.tro strutil.tro alloc.tro hash.tro       ; shared
```

19 のオブジェクト・ファイルの名前だけで、間に空白を 1 つずつ入れて 198 文字になり、コマンド・ラインの 127 文字を大きく超えます。リンク・ファイルを使えば、TATARA.COM を作るコマンドは短くなります。

```
tanren /o:build\tatara.com @tatara.lnk
```

また、コメントでそれぞれのモジュールのまとまりが何であるかを示せますが、これはコマンド・ラインではできません。

22.3 リンク・ファイルとコマンド・ライン

リンク・ファイルの言葉は@の言葉の位置に入るので、コマンド・ラインでは、それ自身の名前やオプションと混ぜて使えます。TANREN は、入力された言葉もファイルから読んだ言葉も、すべて同じように扱います。

- **モジュールの順序**は、いつもと同じように言葉の順序です（21.1 節）。SUBONLY.LNK には sub だけが書いてあります。**tanren main @subonly** では MAIN.TRO が先に、**tanren @subonly main** では SUB.TRO が先になります。

  MSX-DOS

  ```
  A:\>tanren main @subonly
  ...
  Wrote MAIN.COM, 0100-0125 (38 bytes), entry 0100.
  A:\>tanren @subonly main /o:rev.com
  ...
  WARNING: REV.COM is entered at 0100, not at 0109.
  2 modules, 16 records, ends at EOF.
  Wrote REV.COM, 0100-0125 (38 bytes), entry 0109.
  ```

  REV.COM は greet のコードで始まるので、TANREN はそのことを警告します（21.1 節）。

- **オプション**は、@の言葉の前にもあとにも書けます。**tanren /p:4000 @prog /o:high.bin** は、コードを 4000h から始め、HIGH.BIN を書き出します。

- **同じオプションを 2 回指定した**ときは、最後のものが有効になります。PROG.LNK には/o:prog.com があるので、**@prog** のあとの/O: はそれを置き換え、前の/O: はそれに置き換えられます。

MSX-DOS

```
A:\>tanren @prog /o:other.com
...
Wrote OTHER.COM, 0100-0125 (38 bytes), entry 0100.
A:\>tanren /o:first.com @prog
...
Wrote PROG.COM, 0100-0125 (38 bytes), entry 0100.
```

- 1 つのコマンド・ラインに指定できる**リンク・ファイルは 1 つ**だけです。また、リンク・ファイルの中で別のリンク・ファイルを指定することはできません。

TANREN は、リンク・ファイルが指定するオブジェクト・ファイルを、まずそのリンク・ファイルと同じディレクトリで探します。TANREN が探す場所は 23 章で説明します。

22.4　メッセージ

リンク・ファイルについてのメッセージを表 22.1 に示します。TANREN はリンクを始める前にリンク・ファイルを読むので、これらのメッセージはバナーの前に表示されます。メッセージの中では、リンク・ファイルは「file list」と呼ばれています。

メッセージ	意味
ERROR: cannot open the file list NAME.LNK	その名前のリンク・ファイルがないか、@のあとに名前がありません。
ERROR: only one file list may be given.	コマンド・ラインに 2 つ以上のリンク・ファイルがあります。
ERROR: a file list may not name another one.	リンク・ファイルの中の言葉が@で始まっています。
ERROR: too many words on the command line or in the file list.	言葉が全部で 4096 文字を超えているか、128 文字より長い言葉があります。

表 22.1: リンク・ファイルについてのメッセージ

たとえばコマンド・ラインとリンク・ファイルで同じモジュールを 2 回指定すると、リンクの途中で、そのパブリックな名前が 2 回定義されていることが見つかります（27 章）。

第 23 章

TANRENが
オブジェクト・ファイルを探す場所

23.1 3 つの場所 173
23.2 環境変数 TANREN 175
23.3 名前の中のパス 176
23.4 出力ファイルの書き出し先 176
23.5 メッセージ 177

SUB.TRO のように、ドライブもディレクトリも付けずに指定したオブジェクト・ファイルは、複数の場所にある可能性があります。この章では、TANREN がそれをどこで探すか、環境変数 TANREN がどう関わるか、そしてオブジェクト・ファイルがカレント・ディレクトリ以外で見つかったときに出力ファイルがどこに書き出されるかを説明します。

TANREN は、TATARA がインクルード・ファイルを探すのと同じようにオブジェクト・ファイルを探します（13 章）。ただし、INCLUDE の行があるファイルの代わりに、リンク・ファイルが使われます。

23.1 3 つの場所

TANREN は、オブジェクト・ファイルを次の 3 つの場所でこの順に探し、最初に見つかったものを使います。

1. そのオブジェクト・ファイルをリンク・ファイルで指定した場合は、そのリンク・ファイルのディレクトリ（22 章）
2. カレント・ディレクトリ

3. 環境変数 TANREN に書いたそれぞれのディレクトリ（左から順に）

1 つのコマンドについての 3 つの場所を、図 23.1 に示します。BUILD にある PROG.LNK は main と sub を指定しています。TANREN は SUB.TRO をリンク・ファイルと同じ BUILD から読み、カレント・ディレクトリや TATARA\LIB にあるものまでは探しません。BUILD には MAIN.TRO がないので、カレント・ディレクトリにあるものを使います。

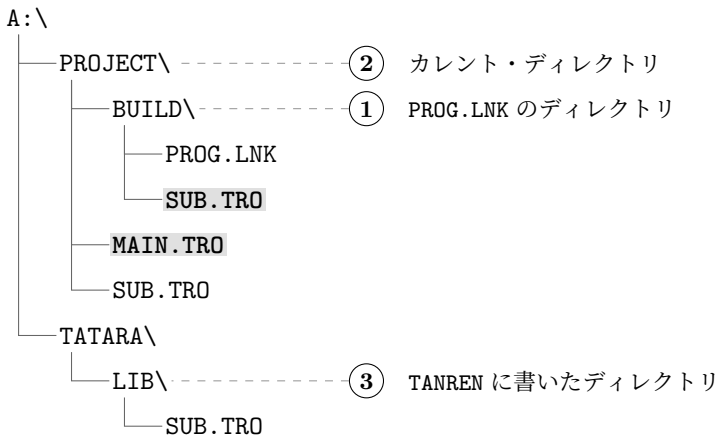

図 23.1: TANREN がオブジェクト・ファイルを探す 3 つの場所とその順序。TANREN が使うファイルには網掛けをしています。

リンク・ファイルのディレクトリが最初なのは、プログラム自身のオブジェクト・ファイルをリンク・ファイルと同じ場所に置いておけば、同じ名前のほかのファイルより必ず優先されるようにするためです。リンク・ファイルではなくコマンド・ラインで指定したオブジェクト・ファイルは、2 番目の場所から探されます。

この章の例では SUB.TRO を何通りか使っていて、それぞれ自分が置かれていたディレクトリを表示するソースから作られています。そのため、プログラムを実行すると、TANREN がどれを使ったかがわかります。リンク・ファイル PROG.LNK を LNK に置き、その横に SUB.TRO がある場合は次のようになります。

MSX-DOS

```
A:\>tanren @lnk\prog /o:lnk.com
...
2 modules, 16 records, ends at EOF.
Wrote LNK.COM, 0100-0123 (36 bytes), entry 0100.
A:\>lnk
SUB.TRO from LNK
```

23.2 環境変数 TANREN

TANREN には、最初の 2 つの場所にないオブジェクト・ファイルを探すディレクトリ、たとえば複数のプログラムが共有するモジュールのディレクトリを書きます。TATARA と同じように働き（13.3 節）、複数のディレクトリはセミコロンで区切って左から順に探されます。それぞれの最後に\を付ける必要はありませんが、付けてもかまいません。AUTOEXEC.BAT などのバッチ・ファイルでは、次のように書きます。

```
set tanren=a:\tatara\lib;a:\project\mod
```

プロンプトで入力するときは、3 章と同じように、値を引用符で囲みます。

TANREN がなければ、最初の 2 つの場所にないオブジェクト・ファイルは見つかりません。

MSX-DOS

```
A:\>tanren main sub
...
ERROR: cannot open SUB.TRO
```

TANREN にそのファイルのあるディレクトリを設定すると、同じコマンドでリンクできます。

MSX-DOS

```
A:\>tanren main sub
...
2 modules, 16 records, ends at EOF.
Wrote MAIN.COM, 0100-0123 (36 bytes), entry 0100.
A:\>main
SUB.TRO from LIB
```

TANREN に書いたディレクトリが存在しないときは、それを飛ばして次のディレクトリが探されます。

TANREN は 3 つの場所の最後です。カレント・ディレクトリにある SUB.TRO は、TANREN のディレクトリにあるものより先に使われます。

MSX-DOS

```
A:\>tanren main sub /o:cur.com
...
Wrote CUR.COM, 0100-0135 (54 bytes), entry 0100.
A:\>cur
SUB.TRO from the current directory
```

23.3　名前の中のパス

B:SUB のようにドライブで始まる名前や、\PROJECT\SUB のように\で始まる名前は、**絶対パス**（absolute path）です。TANREN はそれを書かれたとおりに開き、ほかの場所は探しません。TANREN にそのファイルのあるディレクトリが書いてあっても同じです。

MSX-DOS

```
A:\>tanren main b:sub /o:drv.com
...
ERROR: cannot open B:SUB.TRO
```

それ以外の名前は**相対パス**（relative path）で、3 つの場所で順に探されます。**tanren main lib\sub** は、TANREN がなくても、カレント・ディレクトリの下にある LIB\SUB.TRO を見つけます。

23.4　出力ファイルの書き出し先

/O: を付けないと、出力ファイルには最初のオブジェクト・ファイルの名前に.com を付けた名前が付き（21 章）、そのオブジェクト・ファイルが見つかったディレクトリに書き出されます。最初のオブジェクト・ファイルが TANREN のディレクトリやリンク・ファイルのディレクトリから読まれたときは、出力ファイルもそこに書き出されます。

MSX-DOS

```
A:\>tanren sub main
...
WARNING: LIB\SUB.COM is entered at 0100, not at 0109.
2 modules, 16 records, ends at EOF.
Wrote LIB\SUB.COM, 0100-0123 (36 bytes), entry 0109.
```

ここでは SUB.TRO が TANREN を通して LIB で見つかったので、SUB.COM もカレント・ディレクトリではなく LIB に書き出されました。要約行にはそのパスが表示されます。（警告は、21.1 節と同じくモジュールの順番についてのものです。SUB.TRO を先に指定しているからです。）出力ファイルの書き出し先を確実にするには、/O: で名前を指定してください。

23.5 メッセージ

オブジェクト・ファイルを探すことについてのメッセージを、表 23.1 に示します。

メッセージ	意味
`ERROR: cannot open NAME.TRO`	オブジェクト・ファイルが 3 つの場所のどこにもないか、絶対パスが示す場所にありません。名前は書いたとおりに、大文字で表示されます。
`ERROR: the TANREN variable is too long.`	`TANREN` が 127 文字より長くなっています。

表 23.1: オブジェクト・ファイルを探すことについてのメッセージ

第 24 章

プログラムの配置

24.1 コードが先、データがあと 179
24.2 名前付きセグメント 182
24.3 コードとデータを動かす：/P: と /D: 182
24.4 モジュールをまたぐトランジェント・セグメント 184
24.5 アブソリュート・セグメント 185
24.6 メッセージ 186

TATARA は、すべてのラベルに、セグメントの中で 0 から数えたアドレスを与えます（10 章）。TANREN は、各モジュールの各セグメントをメモリのどこに置くかを決め、それによってすべてのラベルの最終的なアドレスを決めます。この章では、TANREN が従う規則、それを変えるオプション、そして出力ファイルに何が書き出されるかを説明します。

24.1 コードが先、データがあと

TANREN は、すべてのコードを先に置き、すべてのデータをそのあとに置きます。

1. コード・セグメントは、MSX-DOS が `.COM` のプログラムを読み込むメモリ・アドレス 0100h から始まります。
2. データ・セグメントは、コードの最後のバイトのすぐあとから始まります。

複数のモジュールが使うセグメントは 1 つにまとめられ、その中に各モジュールの部分が、コマンド・ラインで指定した順に並びます。

次の 2 つのモジュールには、それぞれコードとデータがあります。`A.AS` には、名前付きコード・セグメント `extra` もあります。

```
; A.AS - code, data, and a named code segment.
                public  astart,aextra,adata
                cseg
astart:         ld      hl,adata
                ret
                cseg    extra
aextra:         nop
                dseg
adata:          ds      4
                end     astart
```

```
; B.AS - code and data.
                public  bcode,bdata
                cseg
bcode:          ld      hl,bdata
                ret
                dseg
bdata:          db      1,2
                end
```

/M を付けると、TANREN はほかの情報とともに、プログラムのセグメントと、それぞれに与えたアドレスを表示します。

MSX-DOS

```
A:\>tanren /m a b
...
Modules: 2
Groups:
Segments:
  flags 00 group FF base 0100 size 0008 end 0107  C
  flags 00 group FF base 0108 size 0001 end 0108 extra
  flags 01 group FF base 0109 size 0006 end 010E  D
Symbols:
  D- 0100 astart
  D- 0108 aextra
  D- 0104 bcode
  D- 0109 adata
  D- 010D bdata
2 modules, 21 records, ends at EOF.
Wrote A.COM, 0100-010E (15 bytes), entry 0100.
```

Segments: の下の各行が 1 つのセグメントで、TANREN が置いた順に並んでいます。最後の言葉はセグメントの名前で、C は標準のコード・セグメント、D は標準のデータ・セグメント、それ以外は名前付きセグメントです。base は最初のアドレス、size はバイト数、end は最後のアドレスです。flags はコードなら 00、データなら 01 です。グループとシンボル、そのほかの /M の表示については、25 章で説明します。

同じプログラムのメモリを図 24.1 に示します。コード・セグメントには A の 4 バイト、続いて B の 4 バイトが 0100h から 0107h まで入り、そのあとの 0108h に extra があります。データ・セグメントは 0109h から始まり、A の 4 バイト、続いて B の 2 バイトが入ります。

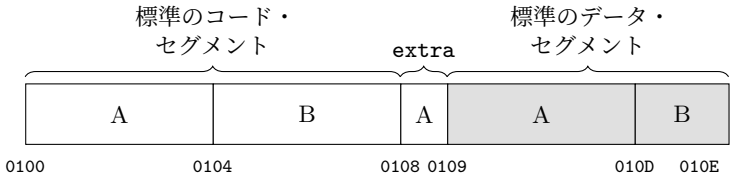

図 24.1: tanren a b のメモリ。それぞれの箱が、セグメントの中の 1 つのモジュールの部分です。網掛けの箱はデータです。

モジュールを逆の順に指定した **tanren b a** では、各セグメントの中で B の部分が先になります。bcode は 0100h に、astart は 0104h に移り、エントリー・アドレスも一緒に移ります。

24.1.1 ファイルに書き出されるもの

出力ファイルには、内容を持つ最初のバイトから最後のバイトまでのメモリが書き出されます。DS はメモリを確保しますが内容を与えないので（11 章）、プログラムの最後にある DS は、ファイルにまったく書き出されません。**tanren b a** では A の ds 4 が最後になるので、ファイルは B の 2 バイトのあとの 010Ah で終わります。

MSX-DOS

```
A:\>tanren /m b a /o:ba.com
...
WARNING: BA.COM is entered at 0100, not at 0104.
2 modules, 21 records, ends at EOF.
Wrote BA.COM, 0100-010A (11 bytes), entry 0104.
```

最初のバイトについても同じです。DS のためにファイルから除かれたメモリは、0 にもされません。プログラムが始まったとき、そこにはそれまでにあった値が残っています。値 0 から始まる必要がある変数は、プログラム自身で 0 にするか、DB や DW で定義してください。

24.2　名前付きセグメント

プログラムには、それぞれの種類のセグメントを複数持たせることができます（10.5 節）。TANREN は、種類ごとに、まず標準のセグメントを置き、次に名前付きセグメントを、コマンド・ラインの順に並べたモジュールの中で最初に現れた順に置きます。次のモジュールは、名前付きセグメントを標準のセグメントより先に選んでいます。

```
; ORDER.AS - named segments written before the default ones.
                public  zfirst,cmain,zdata,dmain
                cseg    zcode
zfirst:         nop
                cseg
cmain:          ret
                dseg    zvars
zdata:          db      1
                dseg
dmain:          db      2
                end     cmain
```

MSX-DOS

```
A:\>tanren /m order
...
Segments:
  flags 00 group FF base 0100 size 0001 end 0100  C
  flags 00 group FF base 0101 size 0001 end 0101 zcode
  flags 01 group FF base 0102 size 0001 end 0102  D
  flags 01 group FF base 0103 size 0001 end 0103 zvars
```

それでも、標準のコード・セグメントは 0100h から始まります。MSX-DOS は .COM のプログラムを必ず 0100h から始めるので、これは重要です。最初に実行するコードは、最初のモジュールの標準のコード・セグメントに入れてください。

24.3　コードとデータを動かす：/P: と /D:

/P: はコードを別のアドレスから始め、データはこれまでと同じようにそのあとに続きます。

MSX-DOS

```
A:\>tanren /m /p:8000 a b /o:p8000.bin
...
Segments:
  flags 00 group FF base 8000 size 0008 end 8007  C
  flags 00 group FF base 8008 size 0001 end 8008 extra
  flags 01 group FF base 8009 size 0006 end 800E  D
...
Wrote P8000.BIN, 8000-800E (15 bytes), entry 8000.
```

MSX-DOS は.COM のプログラムを 0100h に読み込むので、ほかのアドレス用にリンクしたプログラムは.COM のプログラムではありません。たとえば MSX-BASIC の BLOAD のような、別の方法で読み込みます（26 章）。

/D: は、データをコードのすぐあとではなく、指定したアドレスから始めます。データはコードより上にも下にも置けます。**/p:c000 /d:8000** では、コードが C000h に、データがその下の 8000h に置かれます。

MSX-DOS

```
A:\>tanren /m /p:c000 /d:8000 a b /o:high.bin
...
Segments:
  flags 00 group FF base C000 size 0008 end C007  C
  flags 00 group FF base C008 size 0001 end C008 extra
  flags 01 group FF base 8000 size 0006 end 8005  D
...
Wrote HIGH.BIN, 8004-C008 (16389 bytes), entry C000.
```

ファイルが 8000h ではなく 8004h から始まっているのは、データの最初の 4 バイトが内容を持たない A の ds 4 だからです（24.1.1 節）。

データをコードの中から始めることはできません。0100h から 0108h までがコードのとき、**/d:0102** を指定すると、TANREN は/D: would put the data on top of the code. と表示して止まります。

警告.　コードとデータの間にあるものはすべて、0 としてファイルに書き出されます。**/d:c000** を指定すると、コードとデータが合わせて 15 バイトの.COM のプログラムが、48,902 バイトのファイルになります。

MSX-DOS

```
A:\>tanren /m /d:c000 a b /o:dc000.com
...
Wrote DC000.COM, 0100-C005 (48902 bytes), entry 0100.
```

DS だけでできたデータ・セグメントは内容を持たないので、どこに置いてもファイルは大きくなりません。

24.4　モジュールをまたぐトランジェント・セグメント

トランジェント・データ・セグメントのグループは同じメモリを共有し（10.6 節）、TANREN はこれをモジュールをまたいで適用します。トランジェント・セグメントのグループは、どのモジュールにあっても、すべて同じアドレスから始まり、セグメントの大きさはいちばん大きいグループの大きさになります。次の 2 つのモジュールは、それぞれセグメント scratch にグループを 1 つずつ加えます。

```
; T1.AS - the group reading of the transient segment scratch.
                public  inbuf
                dseg    scratch,transient
                group   reading
inbuf:          ds      64
                end
```

```
; T2.AS - the group writing of the same segment.
                public  outbuf
                dseg    scratch,transient
                group   writing
outbuf:         ds      32
                end
```

MSX-DOS

```
A:\>tanren /m a b t1 t2 /o:tr.com
...
Groups:
  reading
  writing
Segments:
  flags 00 group FF base 0100 size 0008 end 0107  C
  flags 00 group FF base 0108 size 0001 end 0108 extra
  flags 01 group FF base 0109 size 0006 end 010E  D
  flags 03 group 00 base 010F size 0040 end 014E scratch
  flags 03 group 01 base 010F size 0020 end 012E scratch
...
Wrote TR.COM, 0100-010E (15 bytes), entry 0100.
```

トランジェント・セグメントには、グループごとに flags が 03 の行が 1 つずつあり、Groups: の中でのグループの番号が 00 から数えて表示されます。どちらのグループも 010Fh から始まるので（図 24.2）、inbuf と outbuf は同じアドレスになり、scratch は reading の大きさである 64 バイトを占めます。scratch には DS しかないので、ファイルは 15 バイトのままです。

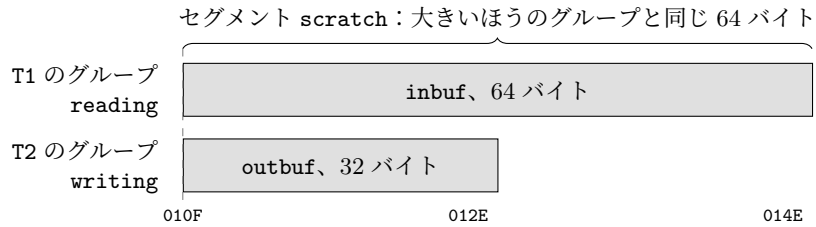

図 24.2: 2 つのモジュールから加えたトランジェント・セグメント scratch の 2 つのグループが、同じアドレスにある様子

24.5　アブソリュート・セグメント

アブソリュート・セグメント（10 章）は動かされません。そのバイトは、ソースに書いたアドレスに置かれます。次のモジュールは、4000h に 2 バイトを置きます。

```
; ABS.AS - two bytes at 4000h.
                public  romhdr
                aseg
                org     4000h
romhdr:         db      'AB'
                end
```

これを A、B と一緒にリンクすると、プログラムは 0100h から 4001h までになり、データの最後のバイトから 4000h までの間はすべて 0 として書き出されます（図 24.3）。

MSX-DOS

```
A:\>tanren /m a b abs /o:gap.com
...
Wrote GAP.COM, 0100-4001 (16130 bytes), entry 0100.
```

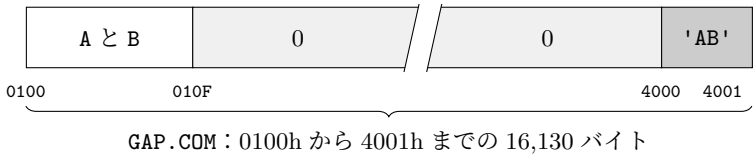

図 24.3: コードよりずっと上にあるアブソリュート・セグメント。間の部分も 0 としてファイルに入ります。

アブソリュート・セグメントは、ROM イメージのように、決まったアドレスにバイトを置かなければならないプログラムのためのものです（26 章）。.COM のプログラムでは、コードよりずっと上にあるバイトはファイルを大きくします。

24.6　メッセージ

配置についてのメッセージを、表 24.1 に示します。

メッセージ	意味
`ERROR: /D: would put the data on top of the code.`	/D: で指定したアドレスがコードの中にあります。
`ERROR: the linked image would run past FFFFh.`	プログラムが、始まるアドレスから FFFFh までに収まりません。たとえば、300 バイトのコードに**/p:ff00** を指定した場合です。

表 24.1: 配置についてのメッセージ

第 25 章

モジュール間のシンボル

25.1 名前の対応のさせかた 187
25.2 外部の名前の値 189
25.3 どのモジュールも定義していない名前 190
25.4 2 回定義された名前 192
25.5 大文字と小文字 192
25.6 すべてのモジュールがリンクされる 194
25.7 /M の表示のすべて 194
25.8 メッセージ 195

モジュールは、PUBLIC でほかのモジュールに名前を提供し、EXTRN でほかのモジュールの名前を使います（12.5 節）。TATARA は外部の名前をすべて未完成のまま残し、TANREN がそれを、ほかのモジュールが与えた値で仕上げます。この章では、TANREN が名前をどのように対応させるか、どのモジュールも定義していない名前や 2 つのモジュールが定義している名前をどう扱うか、そして /M が何を表示するかを説明します。

25.1 名前の対応のさせかた

21 章のプログラムでは、MAIN.AS が greet を EXTRN で宣言して呼び出し、SUB.AS がそれを PUBLIC で宣言して定義しています。

```
; MAIN.AS - calls greet, in another module, and ends the program.
                include msxdos.inc
                extrn   greet
                cseg
start:          call    greet
                system  _TERM0
                end     start
```

```
; SUB.AS - greet prints a line.
                include msxdos.inc
                include ascii.inc
                public  greet
                cseg
greet:          ld      de,msg
                system  _STROUT
                ret
                dseg
msg:            db      'Hello from TANREN.',CHR_CR,CHR_LF,'$'
                end
```

TANREN は、リンク全体で 1 つの名前の表を持ち、モジュールを 2 回読みます。

1. **1 回目のパス**では、すべてのモジュールのパブリックな名前と外部の名前を表に読み込みます。パブリックな名前は、値とともに**定義**されます。`greet` は `SUB` のコードの最初のバイトで、`SUB` のコードは `MAIN` の 8 バイトのあとに来るので、`greet` は 0108h です。外部の名前は**参照**されます。つまり、どこかのモジュールがその値を必要としています。パスの終わりには、参照されている名前がすべて定義されていなければなりません。
2. **2 回目のパス**では、各モジュールのバイトを書き出し、外部の名前が使われている場所すべてに、表の値を入れます。

図 25.1 は、2 つのパスでの `greet` の流れを示しています。TATARA は `call greet` を `CD 00 00` として書き出し、TANREN がそれを 0108h を呼ぶ `CD 08 01` に変えます。

どのバイトを書き出す前にも、すべての名前が表に入っているので、名前についてはモジュールの順番は関係ありません。モジュールは、自分より前に指定したモジュールの名前も、あとに指定したモジュールの名前も使えます。順番が関係するのは、各モジュールがメモリのどこに置かれるかだけです（24 章）。

`/M` は、ほかの情報とともにこの表を表示します。このプログラムでは名前は 1 つです。

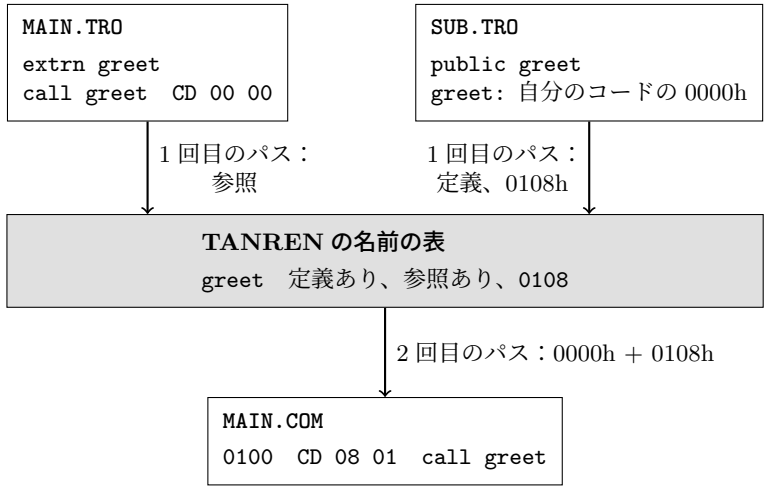

図 25.1: MAIN では外部、SUB ではパブリックな greet が値を得るまで。

```
A:\>tanren /m main sub
...
Symbols:
  DR 0108 greet
2 modules, 16 records, ends at EOF.
Wrote MAIN.COM, 0100-0125 (38 bytes), entry 0100.
```

D は greet が定義されていること、R は参照されていることを示し、0108 がその値です。/M のそのほかの表示は 25.7 節で説明します。

25.2 外部の名前の値

外部の名前には、数を加えることも引くこともできます（8.6 節）。TATARA はその数を値が入る場所に書いておき、TANREN がそれに名前の値を加えます。次の 2 つのモジュールは、コードとデータの中で名前を使っています。

```
; USE.AS - uses two names from VALS.AS, with and without an addend.
                extrn   table,count
                cseg
start:          ld      hl,table+2
                ld      bc,count
                ld      de,count+1
                ret
                dseg
ptrs:           dw      table,table+4
                end     start
```

```
; VALS.AS - a table in the data segment, and an absolute value.
                public  table,count
count           equ     10
                dseg
table:          db      1,2,3,4,5,6
                end
```

MSX-DOS

```
A:\>tanren /m use vals
...
Symbols:
  DR 000A count
  DR 010E table
2 modules, 17 records, ends at EOF.
Wrote USE.COM, 0100-0113 (20 bytes), entry 0100.
```

table は USE のデータのあとの 010Eh にあるので、USE.COM の中では ld hl,table+2 は 21 10 01 に、dw table,table+4 は 0E 01 12 01 になります。

count は EQU で定義されているので、セグメントの中のアドレスではなく**アブソリュート**な値です（8.6 節）。TANREN はこれを動かしません。モジュールがどこに置かれても 000Ah で、ld de,count+1 は 11 0B 00 になります。

25.3　どのモジュールも定義していない名前

参照されているのに定義されていない名前があると、TANREN は 1 回目のパスの終わりで止まり、出力ファイルは書き出されません。MAIN だけをリンクすると、greet を定義するモジュールがありません。

MSX-DOS

```
A:\>tanren main
...
Never defined:
  greet
ERROR: the symbols above were never defined.
```

TANREN は、最初の 1 つだけでなく、足りない名前をすべて表示するので、1 回の実行ですべてがわかります。次のモジュールは 3 つの名前を使い、そのうち beta だけが BETA.AS で定義されています。

```
; MISS.AS - three externals; nobody defines alpha or gamma.
                extrn   alpha,beta,gamma
                cseg
                ld      hl,alpha
                ld      de,beta
                ld      bc,gamma
                ret
                end
```

```
; BETA.AS - defines beta.
                public  beta
                cseg
beta:           ret
                end
```

MSX-DOS

```
A:\>tanren miss beta
...
Never defined:
  gamma
  alpha
ERROR: the symbols above were never defined.
```

名前は、特に決まっていない順で表示されます。

警告. EXTRN で宣言した名前は、モジュールがそれを一度も使わなくても参照されていることになり、TANREN はやはりその定義を求めます。

```
; UNUSED.AS - declares an external and never uses it.
                extrn   nothere
                cseg
start:          ret
                end     start
```

MSX-DOS

```
A:\>tanren unused
...
Never defined:
  nothere
ERROR: the symbols above were never defined.
```

モジュールで外部の名前を使わなくなったら、EXTRN の行からもその名前を消してください。

25.4　2 回定義された名前

1 つの名前を定義できるのは 1 つのモジュールだけです。2 つ目の定義があると、TANREN はモジュールを読んでいる途中で止まり、エラーの前の行に、その名前と、2 つ目の定義が見つかったモジュールを表示します。次の例では、DUP2.AS が 2 つ目の greet を定義していて、OTHER.TRO にアセンブルされています。

```
; DUP2.AS - a second greet.
                public  greet
                cseg
greet:          ret
                end
```

MSX-DOS

```
A:\>tatara /q dup2.as other.tro
A:\>tanren main sub other /o:dup.com
...
  greet is defined again in module dup2
ERROR: that public symbol is already defined.
```

モジュールには、オブジェクト・ファイルではなく、アセンブルしたソースの名前が付くので、メッセージには other ではなく dup2 と表示されます。

1 つのオブジェクト・ファイルを 2 回指定したときも、たとえばコマンド・ラインとリンク・ファイルの両方に書いたときも（22 章）、同じことが起きます。

MSX-DOS

```
A:\>tanren main sub sub /o:dup.com
...
  greet is defined again in module sub
ERROR: that public symbol is already defined.
```

TANREN は、2 回定義された最初の名前で止まり、そのあとで足りない名前を探すことはしません。1 つのリンクに両方の誤りがあるときは、まず 2 回定義された名前を直してから、もう一度リンクしてください。

25.5　大文字と小文字

/C を付けないと、TATARA は名前の大文字と小文字を区別せず（7.3 節）、TANREN も区別しません。次のモジュールは、大文字で GREET を呼びます。

```
; UPPER.AS - calls GREET, in capitals.
                include msxdos.inc
                extrn   GREET
                cseg
start:          call    GREET
                system  _TERM0
                end     start
```

このモジュールは、名前が greet である SUB とリンクできます。/M は、TANREN が最初に見た形で名前を表示します。

MSX-DOS

```
A:\>tanren /m upper sub
...
Symbols:
  DR 0108 GREET
2 modules, 16 records, ends at EOF.
Wrote UPPER.COM, 0100-0125 (38 bytes), entry 0100.
```

/C を付けてアセンブルすると、2 つのモジュールの名前は別の名前になり、一方は定義されているだけ、もう一方は参照されているだけなので、リンクは失敗します。

MSX-DOS

```
A:\>tatara /q /c upper.as upperc.tro
A:\>tatara /q /c sub.as subc.tro
A:\>tanren /m upperc subc
...
Symbols:
  D- 0108 greet
  -R ---- GREET
Never defined:
  GREET
ERROR: the symbols above were never defined.
```

/C を付けてアセンブルしたモジュールと付けずにアセンブルしたモジュールは、一緒にリンクできません（7.3 節）。

MSX-DOS

```
A:\>tanren upperc sub /o:mix.com
...
ERROR: one module was assembled /C and another was not.
```

25.6　すべてのモジュールがリンクされる

TANREN は、渡されたモジュールをそのままリンクします。ほかのどのモジュールからも参照されないモジュールもリンクされ、そのバイトは出力ファイルに入ります。EXTRA.AS の spare は、どこからも参照されていません。

```
; EXTRA.AS - a module nobody refers to.
                public  spare
                cseg
spare:          ld      a,1
                ret
                end
```

MSX-DOS

```
A:\>tanren /m main sub extra /o:ex.com
...
Symbols:
  D- 0111 spare
  DR 0108 greet
3 modules, 22 records, ends at EOF.
Wrote EX.COM, 0100-0128 (41 bytes), entry 0100.
```

spare は定義されていて参照されておらず、その 3 バイトの分だけ、EX.COM は MAIN.COM より 3 バイト長くなります。TANREN は、プログラムに必要なモジュールをライブラリから探すことはしません。コマンド・ラインやリンク・ファイルで指定したモジュールは、使われるかどうかにかかわらず、すべてリンクされます。

25.7　/M の表示のすべて

/M は、バナーのあと、要約行の前に、4 つの部分を表示します。**tanren /m main sub** では次のようになります。

MSX-DOS

```
A:\>tanren /m main sub
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

Modules: 2
Groups:
Segments:
  flags 00 group FF base 0100 size 0011 end 0110  C
  flags 01 group FF base 0111 size 0015 end 0125  D
Symbols:
```

```
  DR 0108 greet
2 modules, 16 records, ends at EOF.
Wrote MAIN.COM, 0100-0125 (38 bytes), entry 0100.
```

- **Modules** は、TANREN が読んだオブジェクト・ファイルの数です。
- **Groups** は、トランジェント・セグメントのグループを 1 行に 1 つずつ表示します。グループには、表示された順に 00 から番号が付きます（24.4 節）。このプログラムにはグループはありません。
- **Segments** は、すべてのセグメントについて、置かれた場所と大きさを表示します（24.1 節）。
- **Symbols** は、すべてのパブリックな名前と外部の名前を、1 行に 1 つずつ、特に決まっていない順で表示します。最初の欄は 2 文字で、どこかのモジュールが名前を定義していれば D、どこかのモジュールが外部として宣言していれば R で、当てはまらない文字の代わりに -が入ります。次が値で、どのモジュールも定義していない名前では----になります。最後が名前です。

/M は、足りない名前があるときも、その一覧の前に表を表示します。足りない名前は-R ----と表示されます。

MSX-DOS

```
A:\>tanren /m main /o:m1.com
...
Modules: 1
Groups:
Segments:
  flags 00 group FF base 0100 size 0008 end 0107  C
  flags 01 group FF base 0108 size 0000 end -----  D
Symbols:
  -R ---- greet
Never defined:
  greet
ERROR: the symbols above were never defined.
```

大きさが 0000 のセグメントには最後のアドレスがないので、end は-----と表示されます。

25.8 メッセージ

名前に関するメッセージを表 25.1 に示します。TANREN のすべてのメッセージは 27 章にあります。

メッセージ	意味
`ERROR: the symbols above were never defined.`	`Never defined:` の下に表示された名前は、どこかのモジュールが外部として宣言していますが、どのモジュールも定義していません。
`ERROR: that public symbol is already defined.`	その前の行の `NAME is defined again in module MOD` が示すパブリックな名前を、2 つ目のモジュールも定義しています。
`ERROR: one module was assembled /C and another was not.`	すべてのモジュールを `/C` 付きで、またはすべてを `/C` なしでアセンブルする必要があります。

表 25.1: 名前に関するメッセージ。

第 26 章

出力形式

26.1 TANREN が書き出すもの 197
26.2 .COM のプログラム 198
26.3 ほかのアドレス用のイメージ 198
26.4 BLOAD 用のファイル: /B 199
26.5 エントリー・アドレス 201
26.6 ROM イメージ 202
26.7 プログラムの大きさの限度 204
26.8 書き出すものがないとき 205
26.9 メッセージ 205

TANREN は、1 回のリンクで 1 つの出力ファイルを書き出します。この章では、そのファイルに何が入るか、ファイルの種類、そしてそれぞれの種類に合わせてプログラムをリンクする方法を説明します。MSX-DOS 用の `.COM` のプログラム、ほかのアドレス用のイメージ、MSX-BASIC の `BLOAD` 用のファイル、ROM イメージの 4 つです。最後に、プログラムの大きさの限度について説明します。

26.1 TANREN が書き出すもの

出力ファイルには、プログラムのメモリのうち、内容のある最初のバイトから最後のバイトまでが入り、それ以外には何も入りません（24.1.1 節）。`/B` が付けるヘッダーを除いて、ファイルには、どのアドレスに置くものかは書かれていません。ファイルを読み込む側が、それを知っている必要があります。

表 26.1 に、4 種類のファイルと、それぞれの作り方を示します。TANREN が標準で作るのは `.COM` だけです。ほかの種類では、出力ファイルの名前を `/O:` で指定す

るか、/B で.bin が付くようにします。

種類	リンクのしかた	読み込むもの	アドレス
.COM	そのまま。**tanren main sub**	名前を入力したときの MSX-DOS	0100h
そのままのイメージ	/P:。名前は/O: で指定します	ほかのプログラム	/P: のアドレス
.BIN	/B と/P:	MSX-BASIC の BLOAD	ヘッダーにあるアドレス
ROM イメージ	ソースに ASEG と、4000h または 8000h の ORG。名前は/O: で指定します	カートリッジから読む MSX	4000h または 8000h。MSX がヘッダーを探す場所です

表 26.1: 出力ファイルの種類。

26.2　.COM のプログラム

オプションを何も付けないと、TANREN は.COM のプログラムを作ります。MSX-DOS はこれを 0100h に読み込んでそこから実行するので、0100h 用にリンクされている必要があり、TANREN はそこからコードを置きます（24 章）。

MSX-DOS

```
A:\>tanren main sub
...
2 modules, 16 records, ends at EOF.
Wrote MAIN.COM, 0100-0125 (38 bytes), entry 0100.
```

.COM のプログラムには/P: を使わないでください。ファイルはやはり 0100h に読み込まれ、中のアドレスがすべて合わなくなります。

26.3　ほかのアドレス用のイメージ

/P: を付けると、TANREN はプログラムをほかのアドレス用にリンクします。ファイルの種類は同じで、中身はバイトだけですが、そのアドレスでしか動かず、MSX-DOS 以外の何かがそこに読み込む必要があります。たとえば、ファイルをメモリに読み込んで呼び出す、ほかのプログラムです。

次のモジュールは、MSX の BIOS の 1 文字を表示するルーチン CHPUT（BIOS.INC）で 1 行を表示するので、MSX-DOS なしで動きます。

```
; HELLOB.AS - prints a line with the BIOS, for BLOAD from BASIC.
                include bios.inc
                include ascii.inc
                cseg
start:          ld      hl,msg
loop:           ld      a,(hl)
                or      a
                ret     z
                push    hl
                call    CHPUT
                pop     hl
                inc     hl
                jr      loop
                dseg
msg:            db      'Hello from BLOAD.',CHR_CR,CHR_LF,0
                end     start
```

C000h 用にリンクすると、34 バイトになります。

MSX-DOS

```
A:\>tanren /p:c000 hellob /o:hellor.bin
...
1 modules, 8 records, ends at EOF.
Wrote HELLOR.BIN, C000-C021 (34 bytes), entry C000.
```

ファイルは 21 0E C0、つまり ld hl,msg で始まります。メッセージは、14 バイトのコードのあとの C00Eh にあります。

26.4 BLOAD 用のファイル: /B

MSX-BASIC は、マシン語のファイルを BLOAD で読み込みます。そのファイルは、どこに置くかを示す 7 バイトの *BLOAD* **ヘッダー**で始まっている必要があり、/B を付けると TANREN がそれを書きます。/B を付けると、/O: で名前を指定しないとき、出力ファイルには.bin が付きます。

MSX-DOS

```
A:\>tanren /b /p:c000 hellob
...
1 modules, 8 records, ends at EOF.
Wrote HELLOB.BIN, C000-C021 (34 bytes), BLOAD header, entry C000.
```

要約行には、いつもと同じく、プログラムのアドレスと大きさが表示されます。ディスク上のファイルはヘッダーの分だけ 7 バイト多い 41 バイトです。HELLOB.BIN のヘッダーを図 26.1 に示します。

- マシン語のファイルであることを示す FE
- プログラムの最初のアドレス、C000h
- プログラムの**最後の**バイトのアドレス、C021h（その次のアドレスではありません）
- プログラムを始めるアドレス、つまりエントリー・アドレス、C000h（26.5 節）

アドレスはどれも 2 バイトで、Z80 と同じく下位バイトが先です。ヘッダーのあとには、HELLOR.BIN と同じ 34 バイトが続きます。

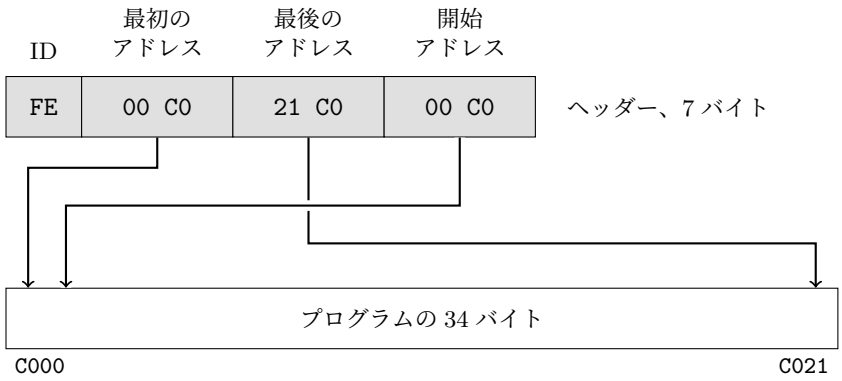

図 26.1: HELLOB.BIN の BLOAD ヘッダーと、そこに入っている 3 つのアドレス。

MSX-BASIC で,R を付けて BLOAD すると、ファイルを読み込んで開始アドレスを呼び出し、プログラムは RET で BASIC に戻ります。

```
Ok
bload"hellob.bin",r
Hello from BLOAD.
Ok
```

MSX-BASIC が動いている間は、8000h より下のアドレスには BASIC の ROM があり、RAM は 8000h から始まります。そのため、.BIN のファイルは、その RAM の中の、BASIC が使っているものを上書きしないアドレス用にリンクする必要があります。0100h 用にリンクした.BIN は、まったく読み込めません。完全な例は 32 章にあります。

26.5 エントリー・アドレス

エントリー・アドレスは、END の行に書いたアドレスです（12.6 節）。複数のモジュールがそれを書いているときは、コマンド・ラインの順で最初のものが使われ、ほかは無視されます。E1.AS と E2.AS は、どちらも書いています。

```
; E1.AS - its END names e1start.
                cseg
e1start:        nop
                ret
                end     e1start
```

```
; E2.AS - its END names e2start, which is not its first byte.
                cseg
                nop
e2start:        ret
                end     e2start
```

MSX-DOS

```
A:\>tanren /b /p:c000 e1 e2 /o:e12.bin
...
Wrote E12.BIN, C000-C003 (4 bytes), BLOAD header, entry C000.
A:\>tanren /b /p:c000 e2 e1 /o:e21.bin
...
Wrote E21.BIN, C000-C003 (4 bytes), BLOAD header, entry C001.
```

E1 を先にするとエントリーは e1start の C000h、E2 を先にすると e2start の C001h になります。

どのモジュールも書いていなければ、要約行に entry は表示されません。

```
; NOEND.AS - no address on its END line.
                cseg
                ld      a,1
                ret
                end
```

MSX-DOS

```
A:\>tanren noend
...
Wrote NOEND.COM, 0100-0102 (3 bytes).
```

BLOAD ヘッダーには必ずエントリー・アドレスがあるので、/B を付けると、TANREN はプログラムの最初のアドレスを使い、要約行にもそれが表示されます。

MSX-DOS

```
A:\>tanren /b /p:c000 noend
...
Wrote NOEND.BIN, C000-C002 (3 bytes), BLOAD header, entry C000.
```

エントリー・アドレスが使われるのは、BLOAD ヘッダーの中だけです。MSX-DOS は、.COM のプログラムをいつも 0100h から実行し（21.1 節）、そのままのイメージもふつうは最初のアドレスから実行されます。そのため、/B なしでは、エントリー・アドレスがそれ以外のとき、TANREN は警告を出し、ファイルはそのまま書き出します。

MSX-DOS

```
A:\>tanren e2 e1 /o:e21raw.bin
...
WARNING: E21RAW.BIN is entered at 0100, not at 0101.
2 modules, 12 records, ends at EOF.
Wrote E21RAW.BIN, 0100-0103 (4 bytes), entry 0101.
```

この警告は/Q を付けても表示されます。上の E21.BIN のように、/B を付けると警告は出ません。ヘッダーが BLOAD に開始アドレスを伝えるからです。

26.6　ROM イメージ

16 KB の ROM カートリッジは 4000h から 7FFFh にあり、MSX が起動するときに探すヘッダーで始まります。EXAMPLES\ROM の ROM.AS は、その完全な例です（最初の数行のコメントは、ここでは短くしています）。

```
; ROM.AS - a 16 KB cartridge.

                include bios.inc        ; INITXT, CHPUT, and the rest

                aseg
                org     4000h

                db      "AB"            ; a cartridge, says the BIOS
                dw      init            ; INIT: called at once
                dw      0               ; STATEMENT: no CALL statements
                dw      0               ; DEVICE: no device
                dw      0               ; TEXT: no BASIC program in here
                dw      0,0,0           ; reserved, and zero

init:           call    INITXT          ; a text screen, from nothing
                ld      hl,msg
ini.lp:         ld      a,(hl)
                or      a
                jr      z,ini.end
                push    hl
                call    CHPUT
                pop     hl
                inc     hl
                jr      ini.lp
ini.end:        jr      ini.end         ; a cartridge has nowhere to go

msg:            db      "Assembled by Tatara. Linked by TANREN.",13,10,0

                org     7fffh           ; the last byte of the 16 KB...
                db      0               ;   ...so that the file is all of it

                end
```

コードはアブソリュート・セグメントにあり、ソースに書いたアドレスに置かれるので（24.5 節）、`/P:` は要りません。

MSX-DOS

```
A:\>tanren rom /o:rom.rom
...
1 modules, 6 records, ends at EOF.
Wrote ROM.ROM, 4000-7FFF (16384 bytes).
```

カートリッジのイメージは 16,384 バイトちょうどでなければならず、最後の 2 行がそれを実現しています。TANREN は内容のある最初のバイトから最後のバイトまでを書き出すので、7FFFh に 1 バイト置くと、その前の空いた部分が 0 としてファイルに入ります。この 2 行がないと、同じプログラムが 77 バイトのファイルになります。

MSX-DOS

```
A:\>tanren romshort /o:romshort.rom
...
Wrote ROMSHORT.ROM, 4000-404C (77 bytes).
```

最後に DS を置いても、そうはなりません。内容のある最後のバイトのあとに DS が確保したメモリは、ファイルに書き出されないからです（24.1.1 節）。カートリッジについては 33 章でくわしく説明します。

26.7　プログラムの大きさの限度

TANREN は、MSX-DOS がプログラムに与えるメモリではなく、メモリ・マッパーの中でプログラムを組み立て、そこからファイルに書き出します。そのため、プログラムは、TANREN 自身が動いているアドレスも含めて、0100h から FFFFh までのアドレス空間全体を使えます。次のモジュールは、0100h に 1 バイト、それより 60 KB 以上上にもう 1 バイトを置いています。

```
; BIG.AS - one byte at the start and one near the top of memory.
                cseg
                db      1
                ds      0EF00h
                db      2
                end
```

MSX-DOS

```
A:\>tanren big
...
1 modules, 6 records, ends at EOF.
Wrote BIG.COM, 0100-F001 (61186 bytes).
```

大きなプログラムには、そのイメージのための、そして TANREN の表のための空いたメモリ・マッパーが十分に必要です。表の大きさは、モジュール、セグメント、名前の数など、プログラムによって変わります。メモリ・マッパーが足りなくなると、TANREN は `ERROR: out of mapper memory.` と表示して止まります。

`.COM` のプログラムは、実行するときに、MSX-DOS が与えるメモリに収まらなければなりません。そのメモリは FFFFh よりかなり下で終わります。この限度は MSX と MSX-DOS によって決まり、リンカーによるものではないので、TANREN はこれを確認しません。

26.8 書き出すものがないとき

モジュールに DS しかないプログラムには、内容のあるバイトがないので、TANREN はファイルを書き出しません。

```
; EMPTY.AS - only DS: nothing to write.
                dseg
buffer:         ds      100
                end
```

MSX-DOS

```
A:\>tanren empty
...
1 modules, 4 records, ends at EOF.
Nothing to write: no module has any content.
```

これはエラーではありませんが、出力ファイルは作られず、以前のリンクで作った同じ名前のファイルは、そのまま残ります。

26.9 メッセージ

出力ファイルに関するメッセージを表 26.2 に示します。`ERROR: the linked image would run past FFFFh.` については 24 章で説明しています。

メッセージ	意味
`ERROR: cannot create NAME`	出力ファイルを作れません。たとえば、**`/o:nodir\x.com`** のように、そのディレクトリがないときです。名前は大文字で表示されます。
`ERROR: cannot write the output file - the disk may be full.`	ファイルは作られましたが、書き込みに失敗しました。
`ERROR: out of mapper memory.`	プログラムのイメージや TANREN の表のための場所が、メモリ・マッパーに残っていません。
`WARNING: NAME is entered at XXXX, not at YYYY.`	エラーではありません。`/B` なしで書き出したファイルは最初のアドレス XXXX から実行されますが、`END` のアドレスは YYYY です。
`Nothing to write: no module has any content.`	エラーではありません。内容のあるバイトを持つモジュールがなく、ファイルは作られません。

表 26.2: 出力ファイルに関するメッセージ。

第 27 章

リンカーのエラーメッセージ

27.1 メッセージの形 207
27.2 TANREN は最初のエラーで止まる 208
27.3 壊れたオブジェクト・ファイル 209
27.4 セグメント 210
27.5 メッセージ 212

TANREN は、誤りを見つけると、TATARA と同じようにエラーメッセージを 1 つ表示して止まります（20 章）。この章では、TANREN のメッセージの形、止まったときにディスクに何が残るか、そしてこれまでの章で説明していないメッセージ、つまり壊れたオブジェクト・ファイルとセグメントに関するメッセージを説明します。すべてのメッセージとその原因は、付録 D にあります。

27.1 メッセージの形

TANREN のメッセージは、`ERROR:` で始まる 1 行です。TANREN には指し示すソースの行がないので、メッセージに行番号はありません。誤りが 1 つのオブジェクト・ファイルにあるときは、TATARA のメッセージがソースの名前で始まるのと同じように、メッセージはそのファイルの名前で、大文字で始まります。

MSX-DOS

```
A:\>tanren main bad
...
BAD.TRO: ERROR: not a Tatara object file.
```

ほかのメッセージは、`ERROR: cannot open MAIN,SUB.TRO`（21 章）のように本文の中でファイルの名前を示すか、`ERROR: /D: would put the data on top of the code.`（24 章）のように、誤りがどのファイルのものでもないので名前を示し

ません。定義されていない名前の一覧のように、エラーの前に独自の行を表示するものもあります（25 章）。

メッセージがいつ表示されるかも、誤りについて教えてくれます。図 27.1 に、リンクの段階と、それぞれの段階で TANREN が見つける誤りを示します。コマンド・ラインとリンク・ファイルに関するメッセージはバナーの前に表示され（21.6 節）、それ以外はバナーのあとに表示されます。

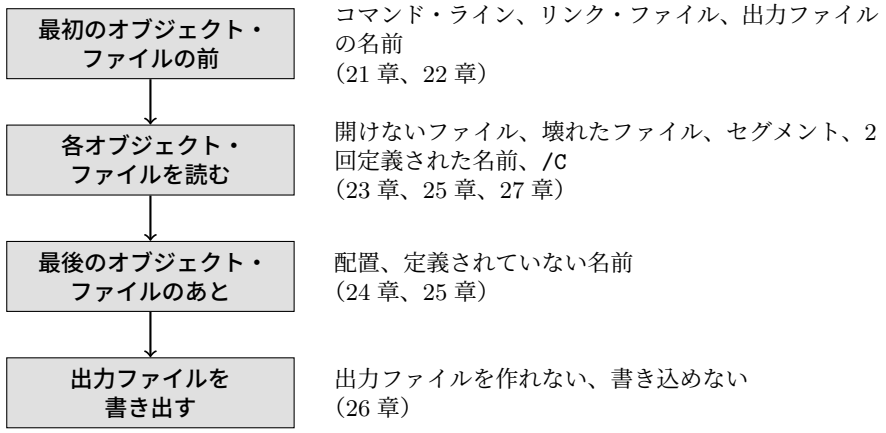

図 27.1: リンクの段階と、それぞれの段階で TANREN が見つける誤り。

/Q はバナーと要約行を表示しませんが、エラーメッセージは必ず表示されます。

27.2　TANREN は最初のエラーで止まる

TANREN はメッセージを 1 つ表示して止まります。誤りを直してもう一度リンクすると、次の誤りがあれば、TANREN はそこに進みます。コマンド・ラインに壊れたオブジェクト・ファイルが 2 つあっても、表示されるのは最初の 1 つだけです。

MSX-DOS

```
A:\>tanren sub bad trunc /o:x.com
...
BAD.TRO: ERROR: not a Tatara object file.
```

例外は、どのモジュールも定義していない名前です。TANREN は止まる前に、それをすべて表示します（25.3 節）。

TANREN は、すべてのオブジェクト・ファイルを 2 回読み終えてから出力ファイルを書き出すので、エラーがあると、ファイルが作られる前に止まります。以前のリンクで作った同じ名前の出力ファイルは、そのまま残ります。次の MAIN.COM は

tanren main sub が書き出したもので、同じ名前のリンクが失敗しても変わりません。

MSX-DOS

```
A:\>dir main.com
...
MAIN     COM        38 26-10-01  8:25
 38 bytes in 1 file   2336K free
A:\>tanren main bad
...
BAD.TRO: ERROR: not a Tatara object file.
A:\>dir main.com
...
MAIN     COM        38 26-10-01  8:25
 38 bytes in 1 file   2336K free
```

エラーのあとでディスクにある出力ファイルは、コマンド・ラインで指定したプログラムではなく、古いものです。TANREN がエラーなしでリンクするまで、それを実行しないでください。

> 警告. `ERROR: cannot write the output file - the disk may be full.` は、TANREN が出力ファイルを作ったあと、それを書いている途中で表示されます。TANREN はすぐに止まり、それ以上ディスクには触れません。ディスクのエラーのあとで何かをすると、被害を大きくするおそれがあるからです。出力ファイルは、プログラムの一部だけが入った状態でディスクに残ります。そのファイルを削除し、ディスクの空きを作るか別のディスクを使ってから、もう一度リンクしてください。

27.3 壊れたオブジェクト・ファイル

オブジェクト・ファイルは、TATARA のオブジェクト・ファイルであることを示し、その形式のバージョンを示す数バイトで始まります（付録 F）。TANREN は、ファイルを開くときにそれを確認し、ファイルのレコードを最後までたどります。次のメッセージは、そのファイルが TATARA の作るものではないことを示します。

MSX-DOS

```
A:\>tanren main bad
...
BAD.TRO: ERROR: not a Tatara object file.
A:\>tanren trunc sub /o:x.com
...
TRUNC.TRO: ERROR: the object file ends inside a record.
A:\>tanren oldver sub /o:x.com
...
OLDVER.TRO: ERROR: this object file was made by another version.
```

- `not a Tatara object file.` ファイルが、オブジェクト・ファイルのようには始まっていません。`BAD.TRO` は、まちがって `.tro` の拡張子を付けたソース・ファイルです。
- `the object file ends inside a record.` ファイルが、内容の途中で終わっています。`TRUNC.TRO` は正しいオブジェクト・ファイルの最初の 30 バイトで、ディスクがいっぱいになったり、コピーが途中で止まったりしたファイルと同じ状態です。
- `this object file was made by another version.` このファイルは、TATARA のオブジェクト・ファイルですが、この TANREN が読めない形式です。TANREN の各バージョンは、同時にリリースされた TATARA のオブジェクト・ファイルを読みます。

どの場合も、このバージョンの TATARA でモジュールをもう一度アセンブルし、リンクし直してください。

27.3.1　終わりのあとのバイト

オブジェクト・ファイルの最後のレコードは、ファイルの終わりを示します。そのレコードのあとに何かがあると、TANREN はそのファイルの名前を表示して、先に進みます。

MSX-DOS

```
A:\>tanren tail sub /o:tail.com
...
TAIL.TRO: bytes follow the END record.
2 modules, 16 records, AND BYTES AFTER THEM.
Wrote TAIL.COM, 0100-0125 (38 bytes), entry 0100.
```

これはエラーではありません。終わりのあとのバイトは読まれず、出力ファイルは書き出されます。しかし、TATARA はそのようなファイルを作らないので、アセンブルしたあとで何かがファイルを変えたことになります。そのモジュールをもう一度アセンブルしてください。`/Q` を付けると、どちらの行も表示されません。

27.4　セグメント

複数のモジュールで指定した同じ名前のセグメントは 1 つのセグメントになるので (24 章)、どのモジュールでも同じ種類のセグメントでなければなりません。`SEGA.AS` は `shared` をコード・セグメントに、`SEGB.AS` はデータ・セグメントにしています。

```
; SEGA.AS - shared is a code segment here.
                public  ina
                cseg    shared
ina:            nop
                end
```

```
; SEGB.AS - and a data segment here.
                public  inb
                dseg    shared
inb:            db      0
                end
```

MSX-DOS

```
A:\>tanren sega segb /o:x.com
...
SEGB.TRO: ERROR: segment shared is declared differently.
```

表示されるファイルは、2 つ目の、異なる宣言が見つかったファイルです。**tanren segb sega** では SEGA.TRO になります。

27.4.1　1 つのモジュールの限度

TANREN は、1 つのモジュールについて、セグメントを 16 個まで、グループを 16 個まで扱えます。TATARA はそれより多く受け付けるので、多すぎるモジュールは、リンクするときに初めてわかります。

MSX-DOS

```
A:\>tanren seg16 /o:x.com
...
SEG16.TRO: ERROR: too many segments or groups in one module.
```

SEG16.AS には、標準のコード・セグメントと 15 個の名前付きセグメントがあります。どのオブジェクト・ファイルにも、中身がなくても標準のデータ・セグメントがあるので、このモジュールのセグメントは 17 個になります。したがって、1 つのモジュールに置ける名前付きセグメントは、標準のコード・セグメントとデータ・セグメントのほかに 14 個までです。トランジェント・セグメントのグループ（10.6 節）は別に数え、1 つのモジュールに 16 個までです。

もっと多くのセグメントやグループが必要なプログラムは、それを複数のモジュールに分けられます。限度はモジュールごとのもので、プログラム全体のものではありません。

27.5　メッセージ

この章で説明したメッセージを表 27.1 に示します。ほかのメッセージは 21 章から 26 章で説明していて、付録 D にすべての一覧があります。

メッセージ	意味
`FILE: ERROR: not a Tatara object file.`	ファイルが、オブジェクト・ファイルのようには始まっていません。
`FILE: ERROR: the object file ends inside a record.`	ファイルが、内容の途中で終わっています。
`FILE: ERROR: this object file was made by another version.`	ファイルの形式が、この TANREN の読むものではありません。
`FILE: ERROR: segment NAME is declared differently.`	そのセグメントの種類が、このファイルと前のファイルで違います。
`FILE: ERROR: too many segments or groups in one module.`	モジュールのセグメントまたはグループが 16 個を超えています。
`FILE: bytes follow the END record.`	エラーではありません。ファイルの最後のレコードのあとにバイトがあります。
`ERROR: cannot write the output file - the disk may be full.`	出力ファイルの書き込みに失敗しました。その一部がディスクに残っています。

表 27.1: この章で説明したメッセージ。

第 IV 部

実例

28 複数のモジュールからなるプログラム 215
29 トランジェント・セグメントで共有する作業用変数 223
30 マクロとリスティングのモード 233
31 長いシンボル名と大文字・小文字を区別する名前 243
32 MSX-BASIC の BLOAD 用のバイナリ 251
33 16 KB の ROM カートリッジ 259
34 複数のディレクトリにまたがるプロジェクト 267
35 TATARA と TANREN を自身のソースからビルドする 275
A 疑似命令リファレンス 283

第 28 章

複数のモジュールからなるプログラム

28.1 この例 215
28.2 2 つのモジュール 216
28.3 リンク・ファイル 217
28.4 作って実行する 218
28.5 TANREN がしたこと 220
28.6 試してみること 220

この章では、A:\TATARA\EXAMPLES\TWOMOD にある例、TWOMOD を作ります。2 つのモジュールからなり、リンク・ファイルでリンクするプログラムです。ファイルを 1 つずつ見ていき、プログラムを作り、TANREN が作ったものを確かめ、最後に 3 通りに変えて、何がうまくいかないのか、なぜなのかを見ます。それぞれの部分のくわしい説明は、途中で示す参照の章にあります。

28.1 この例

TWOMOD は、2 行を表示して終わります。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>twomod
Two modules, one program.
The linker joined them.
```

表示は、独立したモジュールにある putstr というルーチンが行います。この例には、4 つのファイルがあります。

- MAIN.AS：プログラムが始まるモジュール
- PUTSTR.AS：putstr のあるモジュール
- TWOMOD.LNK：2 つのモジュールを指定するリンク・ファイル
- BUILD.BAT：それらをアセンブルしてリンクするバッチ・ファイル

28.2　2 つのモジュール

MAIN.AS は、1 行ごとに putstr を呼び、2 回呼んだあとでプログラムを終えます。

```
; MAIN.AS - one of two modules.
;
; It knows that putstr exists. It does not know where putstr is, and
; it does not need to: EXTRN says the name is somebody else's, and the
; linker fills the address in.

                extrn   putstr          ; defined in PUTSTR.AS

                include msxdos.inc      ; _TERM0 and "system"
                include ascii.inc       ; CHR_CR and CHR_LF

                cseg

start:          ld      hl,msg1
                call    putstr
                ld      hl,msg2
                call    putstr
                system  _TERM0

msg1:           db      "Two modules, one program.",CHR_CR,CHR_LF,0
msg2:           db      "The linker joined them.",CHR_CR,CHR_LF,0

                end     start
```

extrn putstr は、putstr がほかのどこかのモジュールで定義されていることを TATARA に伝えます（12.5 節）。TATARA にはそれがどこかわからないので、それぞれの call putstr のアドレスは、TANREN が入れるように空けておきます。END の行は、最初の命令である start を示しています。

PUTSTR.AS は putstr を定義しています。これは、0 のバイトに達するまで、文字列の文字を 1 つずつ表示します。

```
; PUTSTR.AS - the other module.
;
; PUBLIC is what makes the name visible outside this file. Without it
; the name would still assemble here and the link of MAIN.AS would
; stop with an undefined symbol - which is worth trying once.

                public  putstr

                include msxdos.inc      ; BDOS, _CONOUT and "system"

                cseg

; putstr - print a string.
;
;   HL is pushed around the BDOS call because MSX-DOS promises nothing
;   about the registers it hands back.
;
; Input:        HL -> the string, ending in a zero byte
; Output:       it is printed
; Modifies:     AF, BC, DE, HL

putstr:         ld      a,(hl)
                or      a
                ret     z
                push    hl
                ld      e,a
                system  _CONOUT
                pop     hl
                inc     hl
                jr      putstr

                end
```

public putstr は、この名前をほかのモジュールに提供します。どちらのモジュールも、相手についてそれ以上のことは知りません。MAIN.AS が知っているのは putstr という名前だけで、PUTSTR.AS は誰が自分を呼ぶかを知りません。それぞれが別々にアセンブルされ、それぞれのオブジェクト・ファイルになります。

28.3　リンク・ファイル

リンクのそれ以外の部分は、TWOMOD.LNK にあります（22 章）。

```
; TWOMOD.LNK - the link, in a file instead of on the command line.
;
; TANREN reads this when it is given @TWOMOD, and the words in it are
; the words it would have read from the command line. A ; starts a
; comment and the rest of the line is ignored.
;
; MSX-DOS gives a command line 127 characters and cuts a longer one
; without saying so. That is the whole reason this exists: the
; assembler is itself linked from a file of nineteen module names.

/o:twomod.com

main.tro putstr.tro
```

コメントのあとに、言葉の行が 2 つあります。出力ファイルの名前を示す/o:twomod.com と、2 つのオブジェクト・ファイルです。MAIN.TRO を先に書いているのは、MSX-DOS が.COM のプログラムを 0100h から実行し、TANREN が最初のモジュールのコードをそこに置くからです（21.1 節）。

28.4　作って実行する

BUILD.BAT がプログラムを作ります。

```
rem BUILD.BAT - EXAMPLE: TWOMOD
rem
rem Two modules, one program. MAIN.AS declares putstr EXTRN and
rem PUTSTR.AS declares it PUBLIC; the linker matches the two.
rem
rem The link itself is in TWOMOD.LNK rather than on this line. @NAME
rem reads a file of words, and .LNK is assumed when the name has no
rem extension. Everything in it could have been typed here instead.

rem The headers are in A:\TATARA\INCLUDE and are not copied here.
rem TATARA is rule 3 of the include search - see the DIRS example.
rem CHANGE THE PATH BELOW if you put the tree somewhere else.

set TATARA=a:\tatara\include

echo === Assembling both modules
tatara /q main.as main.tro
tatara /q putstr.as putstr.tro

echo === Linking from TWOMOD.LNK
tanren /q @twomod

echo === Done. Type TWOMOD to run it.

set TATARA=
```

このバッチ・ファイルは、TATARA が MSXDOS.INC と ASCII.INC を見つけられるように TATARA を設定し（13.3 節）、それぞれのモジュールをオブジェクト・ファイルにアセンブルし、**@twomod** でリンクしてから、TATARA を元に戻します。読むファイルと作るファイルを図 28.1 に示します。

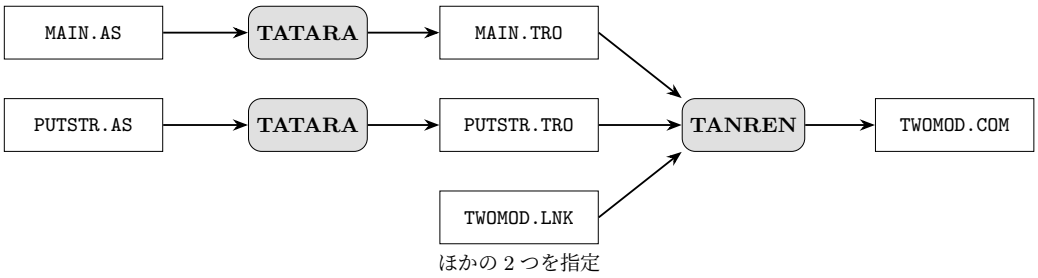

図 28.1: BUILD.BAT が TWOMOD.COM を作る流れ。

例のディレクトリで実行してください。/Q で TATARA と TANREN は何も表示しないので、バッチ・ファイル自身の行だけが表示されます。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>build
=== Assembling both modules
=== Linking from TWOMOD.LNK
=== Done. Type TWOMOD to run it.
A:\TATARA\EXAMPLES\TWOMOD>twomod
Two modules, one program.
The linker joined them.
```

同じリンクを/Q なしで行うと、TANREN の要約が表示されます。TATARA が必要なのはアセンブルのときだけなので、このリンクはそのまま入力できます。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tanren @twomod
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

2 modules, 14 records, ends at EOF.
Wrote TWOMOD.COM, 0100-0154 (85 bytes), entry 0100.
```

28.5　TANREN がしたこと

/M を付けると、TANREN がそれぞれをどこに置いたかがわかります（25.7 節）。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tanren /m @twomod /o:tm.com
...
Modules: 2
Groups:
Segments:
  flags 00 group FF base 0100 size 0055 end 0154  C
  flags 01 group FF base 0155 size 0000 end -----  D
Symbols:
  DR 0147 putstr
2 modules, 14 records, ends at EOF.
Wrote TM.COM, 0100-0154 (85 bytes), entry 0100.
```

どちらのモジュールもすべてを標準のコード・セグメントに置いているので、プログラムは 0100h から 0154h までの 55h バイトのコード・セグメント 1 つになります。最初に MAIN の 71 バイト、続いて PUTSTR の 14 バイトです。データ・セグメントは空です。putstr は PUTSTR が定義し（D）、MAIN が参照していて（R）、MAIN のすぐあとの 0147h にあります。TANREN は、2 つの call putstr の命令の両方に 0147h を書き込みました（図 28.2）。

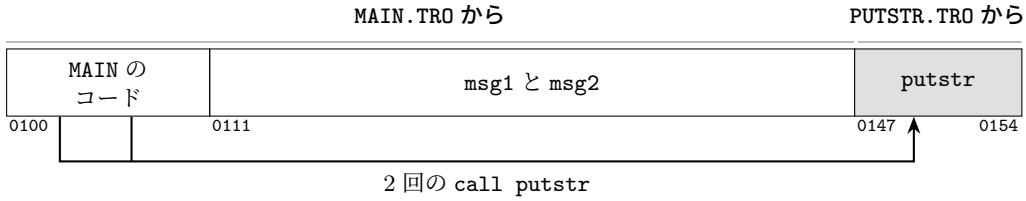

図 28.2: メモリの中の TWOMOD.COM。MAIN の 2 つの呼び出しは、どちらも 0147h の putstr に行きます。

28.6　試してみること

次のそれぞれの変更は、TANREN が見つける誤りを 1 つずつ示します。使うファイルは例には含まれていないので、例のファイルをコピーして、同じディレクトリに作ってください。アセンブルには、BUILD.BAT と同じように TATARA が必要です。

```
set tatara=a:\tatara\include
```

28.6.1 PUBLIC を外す

NOPUB.AS は、PUTSTR.AS から public putstr の行を除いたものです。これはアセンブルできます。putstr は、その中ではふつうのラベルです。しかし、どのモジュールもこの名前を提供しなくなるので、リンクは失敗します（25.3 節）。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tatara /q nopub.as nopub.tro
A:\TATARA\EXAMPLES\TWOMOD>tanren main nopub /o:np.com
...
Never defined:
  putstr
ERROR: the symbols above were never defined.
```

28.6.2 モジュールの順番を逆にする

SWAP.LNK は、TWOMOD.LNK の出力ファイルを/o:swap.com にし、最後の行を putstr.tro main.tro にしたものです。これはエラーなしでリンクできますが、警告が出ます。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tanren /m @swap
...
Symbols:
  DR 0100 putstr
WARNING: SWAP.COM is entered at 0100, not at 010E.
2 modules, 14 records, ends at EOF.
Wrote SWAP.COM, 0100-0154 (85 bytes), entry 010E.
```

しかし、putstr は 0100h に、start は 010Eh に置かれます。MSX-DOS は SWAP.COM を 0100h から実行するので、HL にたまたま入っている値で putstr を実行し、MAIN のコードには決してたどり着きません。実行しないでください。エントリー・アドレスの 010Eh は正しいのですが、.COM のプログラムはそこからは始まらず、TANREN は警告でそのことを伝えます（26.5 節）。

28.6.3　1 つのモジュールだけを /C でアセンブルする

/C を付けてアセンブルした MAIN.AS と、付けずにアセンブルした PUTSTR.AS は、一緒にリンクできません（25.5 節）。

MSX-DOS

```
A:\TATARA\EXAMPLES\TWOMOD>tatara /q /c main.as mainc.tro
A:\TATARA\EXAMPLES\TWOMOD>tanren mainc putstr /o:mix.com
...
ERROR: one module was assembled /C and another was not.
```

第 29 章

トランジェント・セグメントで共有する作業用変数

29.1 この例 223
29.2 ソース 224
29.3 作る 225
29.4 シンボル表を読む 226
29.5 マップを読む 227
29.6 試してみること 229

プログラムには、その一部だけが、しかもその部分が動いている間だけ使う変数がよくあります。読み込み中の行のバッファと、書き出し中の行のバッファなどです。2 つの部分が同時に動くことがなければ、2 つのバッファは同じメモリを共有できます。それを行うのがトランジェント・セグメント（10.6 節）です。この章では、`A:\TATARA\EXAMPLES\LAYOUT` にある例、`LAYOUT` を使って、ソース、TATARA が作るシンボル表、そして TANREN が 2 つの異なるアドレスでリンクしたときに作るマップを順に見ていきます。

29.1 この例

`LAYOUT` は、1 つのモジュール `LAYOUT.AS` と `BUILD.BAT` からなります。プログラム自体は 1 行を表示するだけです。

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>layout
Code here, data elsewhere.
```

この例の目的は、バイトがどこに置かれるかを見ることです。コード、2 つのふつうの変数、そしてトランジェント・セグメント SCRATCH があり、SCRATCH には同じメモリを共有する 2 つの変数のグループがあります。BUILD.BAT は、シンボル表と 2 つのマップをファイルに書き出すので、作ったあとでそれを読むことができます。

29.2　ソース

```
; LAYOUT.AS - where the bytes go.
;
; Three kinds of relocatable space, and the options that place them:
;
;   CSEG                        code            TANREN /P:<addr>
;   DSEG                        data            TANREN /D:<addr>
;   DSEG <name>,TRANSIENT       overlaid data   one GROUP at a time
;
; A SEGMENT MAY BE NAMED, and a named one keeps its own counter: two
; CSEG MUSIC lines in different files contribute to one segment.
;
; A TRANSIENT DSEG is space that is used by one part of a program at a
; time. Every GROUP in it starts at the same address, so the segment is
; as large as its largest group and not as large as their sum. Labels
; in two different groups cannot be subtracted from each other,
; because nothing says which of them is in memory.
;
; The absolute kind, ASEG, is in the BINARY and ROM examples - a fixed
; address in a .COM would make the file span everything from 0100h up
; to it.

                include msxdos.inc      ; _STROUT, _TERM0 and "system"
                include ascii.inc       ; CHR_CR and CHR_LF

                cseg

start:          ld      hl,runs         ; a byte in the plain DSEG
                inc     (hl)
                ld      de,msg
                system  _STROUT
                system  _TERM0

msg:            db      "Code here, data elsewhere.",CHR_CR,CHR_LF,"$"

                dseg                    ; the plain data segment: the
runs:           ds      1               ;   linker puts it after the code
lastkey:        ds      1               ;   unless /D: says where
```

```
                dseg    SCRATCH,TRANSIENT
                group   READING         ; 66 bytes...
inbuf:          ds      64
inlen:          ds      2
                group   WRITING         ; ...and 34, at the same address,
outbuf:         ds      32              ;   so SCRATCH is 66 bytes and not
outlen:         ds      2               ;   100

                end     start
```

ソースは、セグメントの種類ごとに3つの部分からなります。

- **コード**は標準のコード・セグメントにあります。runs に1を加え、MSX-DOSのファンクション_STROUT で msg を表示して終わります。
- **2つのふつうの変数** runs と lastkey は、標準のデータ・セグメントにあります。それぞれが自分の1バイトを持ちます。
- **トランジェント・セグメント** SCRATCH には、2つのグループがあります。READINGは、読み込み中の行のバッファとその長さで、66バイトです。WRITINGは、書き出し中の行のバッファとその長さで、34バイトです。どちらのグループも SCRATCHの先頭から始まるので、inbuf と outbuf は同じアドレスになり、SCRATCH は100バイトではなく66バイトになります。

プログラムは2つのグループを使いません。置かれる場所を見せるためにあるだけです。実際のプログラムなら、行を読み込んでいる間は READING を、行を書き出している間は WRITING を使い、両方を同時に使うことはありません。outbuf に書き込むと、inbuf が上書きされるからです（10.6節）。

29.3 作る

```
rem BUILD.BAT - EXAMPLE: LAYOUT
rem
rem This example is about the two maps it writes, not about the
rem program, which only prints a line.
rem
rem   TATARA /S   the symbol table: every name, by segment, with its
rem               offset inside that segment
rem   TANREN /M   the segment and group tables: where each segment
rem               ended up, how big it is, and which groups share it
rem
rem It links twice. The first is an ordinary .COM at 0100h. The second
rem puts the code at 8000h and the data at C000h - a real MSX layout,
rem code in the top pages with its variables above it, and not
rem something MSX-DOS can run as a command, so it goes to a .BIN that
```

```
rem nobody is asked to run.

rem The headers are in A:\TATARA\INCLUDE and are not copied here.
rem TATARA is rule 3 of the include search - see the DIRS example.
rem CHANGE THE PATH BELOW if you put the tree somewhere else.

set TATARA=a:\tatara\include

echo === Assembling, keeping the symbol table
tatara /q /s layout.as layout.tro > layout.sym

echo === Linking at 0100h, keeping the map
tanren /q /m /o:layout.com layout.tro > layout.map

echo === Linking again, code at 8000h and data at C000h
tanren /q /m /o:layout8.bin /p:8000 /d:c000 layout.tro > layout8.map

echo === Done. Read LAYOUT.SYM, LAYOUT.MAP and LAYOUT8.MAP.

set TATARA=
```

各コマンドは、表示する内容を>でファイルに送ります。/Q を付けると、TATARA と TANREN はバナーも要約も表示しないので、ファイルに入るのはシンボル表（/S、19 章）とマップ（/M、25.7 節）だけです。2 回目のリンクでは、コードを 8000h に、データを C000h に置きます（24.3 節）。その出力ファイル LAYOUT8.BIN は、MSX-DOS が実行できるプログラムではなく、実行するためのものでもありません。そのマップを見るためにあります。

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>build
=== Assembling, keeping the symbol table
=== Linking at 0100h, keeping the map
=== Linking again, code at 8000h and data at C000h
=== Done. Read LAYOUT.SYM, LAYOUT.MAP and LAYOUT8.MAP.
```

29.4　シンボル表を読む

LAYOUT.SYM は、アブソリュートのセクションで始まります。そこには MSXDOS.INC と ASCII.INC のすべての EQU が入っていて、100 行以上あるので、ここでは省略しています。そのあとに、プログラム自身の名前のセクションが続きます。

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>type layout.sym
ASEG - absolute

0000h             CHR_NUL
0000h             _TERM0
...
007Fh             CHR_DEL

CSEG - default code segment, 002Eh bytes

0000h             start
0011h             msg

DSEG - default data segment, 0002h bytes

0000h             runs
0001h             lastkey

SCRATCH - named data segment, transient, 0000h bytes

(no symbols)

SCRATCH - named data segment, transient, group READING, 0042h bytes

0000h             inbuf
0040h             inlen

SCRATCH - named data segment, transient, group WRITING, 0022h bytes

0000h             outbuf
0020h             outlen
```

それぞれの値は、そのセクションの中のオフセットです。SCRATCH には 3 つのセクションがあります。どのグループにも入らない部分のセクション（ここでは空です）と、グループごとのセクションです。inbuf と outbuf は、どちらも自分のグループの中の 0000h にあります。グループを置くのは TANREN なので、この時点のシンボル表には、2 つが同じアドレスになることはまだわかりません。

29.5 マップを読む

LAYOUT.MAP は、.COM のプログラムのマップです。

```
A:\TATARA\EXAMPLES\LAYOUT>type layout.map
Modules: 1
Groups:
  READING
  WRITING
Segments:
  flags 00 group FF base 0100 size 002E end 012D  C
  flags 01 group FF base 012E size 0002 end 012F  D
  flags 03 group 00 base 0130 size 0042 end 0171 SCRATCH
  flags 03 group 01 base 0130 size 0022 end 0151 SCRATCH
Symbols:
```

コードは 0100h から始まって 2Eh バイトあり、2 つのふつうの変数がそのあとの 012Eh に続き、最後に SCRATCH が、グループごとに 1 行ずつ並びます。2 行とも同じ 0130h から始まります。READING は 0171h まで、小さいほうの WRITING は 0151h までです。モジュールがパブリックな名前も外部の名前も宣言していないので、Symbols: は空です。

LAYOUT8.MAP は、2 回目のリンクのマップです。

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>type layout8.map
Modules: 1
Groups:
  READING
  WRITING
Segments:
  flags 00 group FF base 8000 size 002E end 802D  C
  flags 01 group FF base C000 size 0002 end C001  D
  flags 03 group 00 base C002 size 0042 end C043 SCRATCH
  flags 03 group 01 base C002 size 0022 end C023 SCRATCH
Symbols:
```

セグメントもその大きさも同じで、変わるのはアドレスだけです。コードは 8000h に、データはすべて、ふつうの変数も SCRATCH も C000h にあります。図 29.1 に、両方のリンクを示します。

どちらのファイルにも SCRATCH は入っていません。そのバイトは DS で確保されていて、DS はファイルに何も書きません（24.1.1 節）。LAYOUT.COM は、0100h から 012Dh までの 46 バイトです。

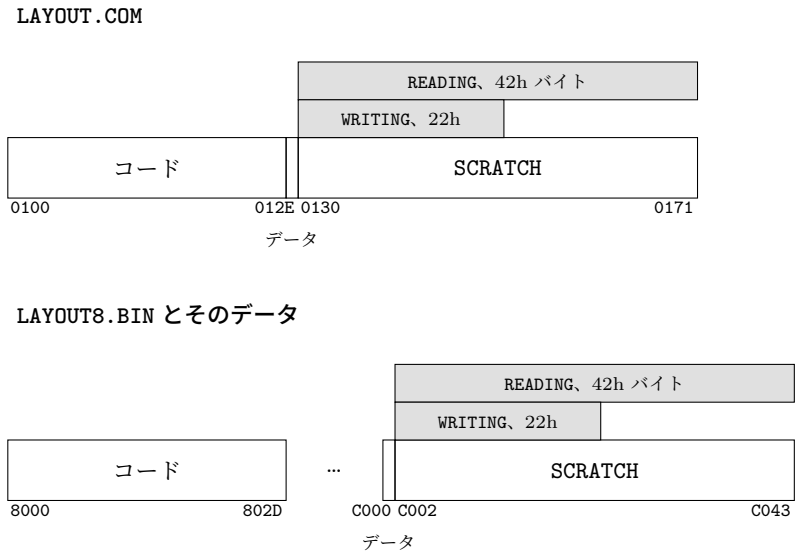

図 29.1: 0100h でリンクした LAYOUT と、コードを 8000h、データを C000h にしてリンクした LAYOUT。どちらでも、SCRATCH の 2 つのグループは同じアドレスから始まります。

29.6 試してみること

次のファイルは例には含まれていません。BUILD.BAT と同じように TATARA を設定して、例のディレクトリに作ってください。

29.6.1 別のグループのラベルどうしを引き算する

```
; SUBGRP.AS - a label from each group of one transient segment,
; in one expression.
                dseg    SCRATCH,TRANSIENT
                group   READING
inbuf:          ds      64
                group   WRITING
outbuf:         ds      32
                cseg
                ld      hl,outbuf-inbuf
                end
```

```
A:\TATARA\EXAMPLES\LAYOUT>tatara subgrp.as subgrp.tro
...
SUBGRP.AS(9): ERROR: relocation error - a segment-relative value is not allowed
    here.
```

どちらのラベルもオフセットは 0000h ですが、グループが違うので、TATARA は 2 つを関係のないアドレスとして扱います（10.6 節）。

29.6.2　別のモジュールでグループを加える

PARSE.AS は、80 バイトの 3 つ目のグループ PARSING を加えます。

```
; PARSE.AS - a third group of SCRATCH, larger than the other two.
                public  tokens
                dseg    SCRATCH,TRANSIENT
                group   PARSING
tokens:         ds      80
                end
```

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>tatara /q parse.as parse.tro
A:\TATARA\EXAMPLES\LAYOUT>tanren /m /o:lp.com layout parse
...
Groups:
  READING
  WRITING
  PARSING
Segments:
  flags 00 group FF base 0100 size 002E end 012D  C
  flags 01 group FF base 012E size 0002 end 012F  D
  flags 03 group 00 base 0130 size 0042 end 0171 SCRATCH
  flags 03 group 01 base 0130 size 0022 end 0151 SCRATCH
  flags 03 group 02 base 0130 size 0050 end 017F SCRATCH
Symbols:
  D- 0130 tokens
2 modules, 20 records, ends at EOF.
Wrote LP.COM, 0100-012D (46 bytes), entry 0100.
```

PARSING も 0130h から始まり、いちばん大きいグループなので、SCRATCH は 017Fh までの 80 バイトになります。トランジェント・セグメントのグループは、モジュールをまたいで重なります（24.4 節）。

29.6.3　別のモジュールからグループに加える

READ2.AS は、LAYOUT.AS にすでにあるグループ READING に変数を 1 つ加えます。

```
; READ2.AS - more variables for the group READING, from another module.
                public  linenum
                dseg    SCRATCH,TRANSIENT
                group   READING
linenum:        ds      2
                end
```

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>tatara /q read2.as read2.tro
A:\TATARA\EXAMPLES\LAYOUT>tanren /m /o:lr.com layout read2
...
Segments:
  flags 00 group FF base 0100 size 002E end 012D  C
  flags 01 group FF base 012E size 0002 end 012F  D
  flags 03 group 00 base 0130 size 0044 end 0173 SCRATCH
  flags 03 group 01 base 0130 size 0022 end 0151 SCRATCH
Symbols:
  D- 0172 linenum
2 modules, 20 records, ends at EOF.
Wrote LR.COM, 0100-012D (46 bytes), entry 0100.
```

ほかのモジュールにある同じ名前のグループは、同じグループです。その変数は、どのセグメントでも同じように前のものに続きます。`linenum` は `inbuf` と `inlen` のあとの 0172h に置かれ、`READING` は 44h バイトになります。大きなプログラムの各部分は、こうして作業用のメモリを共有します。読み込み中の行を扱うモジュールは、それぞれ自分の変数を `READING` に加え、ほかのモジュールについて知る必要はありません。

29.6.4 TRANSIENT を忘れる

`PLAIN.AS` は、`TRANSIENT` なしで `SCRATCH` を宣言しています。

```
; PLAIN.AS - SCRATCH again, but not transient.
                public  plainv
                dseg    SCRATCH
plainv:         ds      10
                end
```

MSX-DOS

```
A:\TATARA\EXAMPLES\LAYOUT>tatara /q plain.as plain.tro
A:\TATARA\EXAMPLES\LAYOUT>tanren /m /o:lpl.com layout plain
...
PLAIN.TRO: ERROR: segment SCRATCH is declared differently.
```

セグメントは、どのモジュールでも同じように宣言しなければなりません（27.4 節）。

第 30 章

マクロとリスティングのモード

30.1 この例 233
30.2 ソース 234
30.3 作って実行する 235
30.4 3 つのリスティングのモード 237
30.5 試してみること 240

マクロはソースの 1 行を複数の行に変え、繰り返しブロックは数行を多くの行に変えます。リスティング（18 章）は、それが何に変わったかを示し、そのうちどれだけを示すかはソースで決められます。この章では、A:\TATARA\EXAMPLES\MACROS にある例、MACROS を使って、そのソースにあるマクロと繰り返しブロック、それが作るプログラム、そして同じ行を 3 つのリスティングのモードで見ていきます。

30.1 この例

MACROS は 2 行を表示し、それぞれのあとで少し待ちます。

MSX-DOS

```
A:\TATARA\EXAMPLES\MACROS>macros
A macro is a line that becomes many.
LOCAL is why DELAY can be used twice.
```

この例には 3 つのファイルがあります。ソースの MACROS.AS、それをすべてリスティングする 2 行のソース LALL.AS、そしてプログラムと 2 つのリスティングを作る BUILD.BAT です。

30.2　ソース

```
; MACROS.AS - a line that becomes many.
;
; MACRO defines one, ENDM ends the definition, and the name is then
; used like a mnemonic. LOCAL, REPT and IRP are below.

                include msxdos.inc      ; _STROUT, _TERM0 and "system"
                include ascii.inc       ; CHR_CR and CHR_LF

; PRINT - three instructions from one line. The parameter is
; substituted wherever the name appears in the body.

print           macro   addr
                ld      de,addr
                system  _STROUT ; a macro inside a macro
                endm

; DELAY - and why LOCAL exists. The body has a label in it, and a
; macro used twice would define that label twice. LOCAL makes a fresh
; name for each expansion, so this may be used as often as you like.

delay           macro   n
                local   loop
                ld      bc,n
loop:           dec     bc
                ld      a,b
                or      c
                jr      nz,loop
                endm

                cseg

start:          print   msg1
                delay   20000
                print   msg2
                delay   20000
                system  _TERM0

msg1:           db      "A macro is a line that becomes many.",CHR_CR,CHR_LF,"$"
msg2:           db      "LOCAL is why DELAY can be used twice.",CHR_CR,CHR_LF,"$"

; REPT repeats a body a counted number of times, with no parameter.
; Neither it nor IRP takes a label: the body is what produces the
; bytes, and a label belongs on a line inside it.

                rept    8
                db      0ffh
                endm

; IRP repeats it once for each item in the list, and IRPC once for
; each character of a word. INSIDE A STRING the parameter is only
; substituted if an ampersand is put in front of it, which is how M80
; tells a parameter from two letters that happen to match.

                irp     n,<1,2,4,8>
                db      n
                endm
```

```
                irpc    c,TATARA
                db      '&c'
                endm
                db      0

                end     start
```

ソースには、2つのマクロ、それを使うプログラム、そして3つの繰り返しブロックがあります。

- print は、文字列のアドレスを DE に入れ、MSX-DOS のファンクション _STROUT で表示します。表示には system を使っていて、これ自体も MSXDOS.INC にあるマクロです。マクロは別のマクロを使えます（16.7 節）。
- delay は、BC を引数の値から 0 まで数え下げます。そのループにはラベルが必要ですが、ラベルは 1 回しか定義できないので、local loop で、マクロを使うたびにラベルに新しい名前を付けます（16.5 節）。
- **プログラム**は start から始まり、それぞれのマクロを 2 回ずつ使い、system _TERM0 で終わります。5 行のソースが 18 個の命令になります。
- **繰り返しブロック**は、プログラムが使わないバイトをプログラムのあとに加えます。リスティングのためにあるものです。REPT は FFh を 8 バイト、IRP は 1、2、4、8 のバイト、IRPC は TATARA の文字を 1 文字ずつ作ります。'&c' がその文字です。文字列の中では、パラメーターは前に&があるときだけ置き換えられます（17 章、16.3 節）。

LALL.AS は、同じプログラムの前に 1 行を加えたものです。

```
; LALL.AS - the same program, listed in full.
;
; .LALL asks for every line of every macro expansion, including the
; ones that produce no bytes. The default is .XALL, which keeps only
; the lines that emitted something. Nothing else here differs from
; MACROS.AS, so the two listings differ only in what they show.

                .lall
                include macros.as
```

30.3 作って実行する

```
rem BUILD.BAT - EXAMPLE: MACROS
rem
rem The listing is the example here. /L writes one, and the file it
rem goes to is named THIRD on the command line, after the source and
rem the object.
rem
rem A listing shows the bytes a line produced, and a line that came
rem out of a macro is marked with a +. HOW MUCH of an expansion is
rem listed is a choice, and the source makes it three times:
rem
rem   .LALL   every line of every expansion
rem   .XALL   only the lines that produced bytes (the default)
rem   .SALL   none of them, only the call
rem
rem It is assembled twice, to two listings, so the two can be read
rem side by side.

rem The headers are in A:\TATARA\INCLUDE and are not copied here.
rem TATARA is rule 3 of the include search - see the DIRS example.
rem CHANGE THE PATH BELOW if you put the tree somewhere else.

set TATARA=a:\tatara\include

echo === Assembling, listing under the default XALL
tatara /q /l macros.as macros.tro macxall.lst

echo === Assembling again, listing everything
tatara /q /l lall.as lall.tro maclall.lst

echo === Linking
tanren /q /o:macros.com macros.tro

echo === Done. Compare MACXALL.LST with MACLALL.LST, then type MACROS.

set TATARA=
```

/L は、コマンド・ラインの 3 番目に指定したファイルにリスティングを書き出します（18.1 節）。MACROS.AS は MACXALL.LST に、LALL.AS は MACLALL.LST にアセンブルされ、リンクされるのは MACROS.TRO だけです。

```
A:\TATARA\EXAMPLES\MACROS>build
=== Assembling, listing under the default XALL
=== Assembling again, listing everything
=== Linking
=== Done. Compare MACXALL.LST with MACLALL.LST, then type MACROS.
A:\TATARA\EXAMPLES\MACROS>macros
A macro is a line that becomes many.
LOCAL is why DELAY can be used twice.
```

プログラムは、0100h から 0186h までの 135 バイトです。

30.4 3つのリスティングのモード

どちらのリスティングも約 33 KB あります。プログラムの前に 2 つのインクルード・ファイルが 1 行ずつリスティングされ、プログラム自身の行は 339 行目から始まります。INCLUDE の行を.XLIST と.LIST ではさめば、それを省けます（18.3 節）。以下の例は、それぞれのリスティングのうちプログラムの部分です。

30.4.1 標準の.XALL

MACXALL.LST は、展開された行のうちバイトを作る行を、+を付けてリスティングします。

MSX-DOS

```
0000'                           start:          print   msg1
0000'   11 25 00        +                      ld      de,msg1
0003'   0E 09           +                      ld      c,_STROUT
0005'   CD 05 00        +                      call    BDOS
                                                delay   20000
0008'   01 20 4E        +                      ld      bc,20000
000B'   0B              +      ??0000:         dec     bc
000C'   78              +                      ld      a,b
000D'   B1              +                      or      c
000E'   20 FB           +                      jr      nz,??0000
                                                print   msg2
0010'   11 4C 00        +                      ld      de,msg2
0013'   0E 09           +                      ld      c,_STROUT
0015'   CD 05 00        +                      call    BDOS
                                                delay   20000
0018'   01 20 4E        +                      ld      bc,20000
001B'   0B              +      ??0001:         dec     bc
001C'   78              +                      ld      a,b
001D'   B1              +                      or      c
001E'   20 FB           +                      jr      nz,??0001
                                                system  _TERM0
0020'   0E 00           +                      ld      c,_TERM0
```

```
0022'   CD 05 00        +                       call    BDOS
```

ここには、ソースにない 2 つのことが表れています。delay のループのラベルは、1 回目は??0000、2 回目は??0001 です。これが LOCAL の作った名前です（16.5 節）。そして、print の system _STROUT の行は、その 2 つの命令はリスティングされているのに、行そのものはリスティングされていません。その行自体はバイトを作らず、展開された行がバイトを作るからです。

繰り返しブロックも同じようにリスティングされます。書いたとおりのブロックのあとに、それが作った各行が続きます。

MSX-DOS

```
                                        irpc    c,TATARA
                                        db      '&c'
                                        endm
0080'   54              +               db      'T'
0081'   41              +               db      'A'
0082'   54              +               db      'T'
0083'   41              +               db      'A'
0084'   52              +               db      'R'
0085'   41              +               db      'A'
0086'   00                              db      0
```

30.4.2　.LALL

MACLALL.LST は、すべての展開のすべての行をリスティングします。このプログラムで違うのは、system を使う行だけです。この行は自分ではバイトを作らないので、アドレスなしでリスティングされます。

MSX-DOS

```
0000'                           start:          print   msg1
0000'   11 25 00        +                      ld      de,msg1
                         +                      system  _STROUT ; a macro
  inside a macro
0003'   0E 09            +                      ld      c,_STROUT
0005'   CD 05 00         +                      call    BDOS
```

条件付きアセンブルを含むマクロでは、もっと違いが出ます。.LALL では、条件によって飛ばされた行もリスティングされます（18.4 節）。

30.4.3 .SALL

この例は3つ目のモードを使っていません。SALL.ASは、LALL.ASと同じく、1行とINCLUDEだけです。

```
; SALL.AS - the same program, with no expansion listed.

                .sall
                include macros.as
```

MSX-DOS

```
A:\TATARA\EXAMPLES\MACROS>tatara /l sall.as sall.tro macsall.lst
...
ended at MACROS.AS(63)
```

MACSALL.LSTは、どの展開の行もリスティングしません。マクロを使う各行には、リスティングに表れている以上のものがあることを示すため、+が付きます。

MSX-DOS

```
 0000'                  +     start:          print   msg1
                        +                     delay   20000
                        +                     print   msg2
                        +                     delay   20000
                        +                     system  _TERM0
```

繰り返しブロックは書いたとおりにリスティングされ、それが作るバイトはリスティングのどこにも表れません。

図30.1に、1つの呼び出しについて3つのモードを並べて示します。

.XALL（標準）

```
0000'                   start:  print msg1
0000'  11 25 00  +              ld    de,msg1
0003'  0E 09     +              ld    c,_STROUT
0005'  CD 05 00  +              call  BDOS
```

バイトを作る行

.LALL

```
0000'                   start:  print   msg1
0000'  11 25 00  +              ld      de,msg1
                 +              system  _STROUT
0003'  0E 09     +              ld      c,_STROUT
0005'  CD 05 00  +              call    BDOS
```

すべての行。自分ではバイトを作らないsystemの使用（網掛け）も含む

.SALL

```
0000'    +  start: print msg1
```

呼び出しの行だけ。+が付く

図30.1: 呼び出しprint msg1について、それぞれのモードがリスティングするもの。

ふつうは.XALL を使います。すべてのバイトが、それを作った行の隣にリスティングされるからです。.LALL は、マクロが何をするかを、条件付きアセンブルも含めて 1 行ずつ調べるためのものです。.SALL は、マクロを信頼できるようになってから、ソースのように読めるリスティングを作るためのものです。3 つは 1 つのソースの中で混ぜて使えます。それぞれ、その行から、3 つのうち次のものが現れるまで効きます。

30.5　試してみること

次のファイルは例には含まれていません。BUILD.BAT と同じように TATARA を設定して、例のディレクトリに作ってください。

30.5.1　LOCAL を外す

NOLOCAL.AS は、MACROS.AS から local loop の行を除いたものです。delay の 1 回目の使用で loop が定義され、2 回目でもう一度定義されます。

MSX-DOS

```
A:\TATARA\EXAMPLES\MACROS>tatara nolocal.as nolocal.tro
...
NOLOCAL.AS(23): ERROR: this name already has a value.
    in delay, called from NOLOCAL.AS(34)
```

23 行目は delay の定義の中の loop: で、34 行目は 2 回目の delay 20000 です（20.3 節）。

30.5.2　&を外す

NOAMP.AS は、MACROS.AS の IRPC の db '&c' を db 'c' にしたものです。エラーなしでアセンブルできますが、文字列の中の c はもうパラメーターではなく、ただの文字です。

MSX-DOS

```
A:\TATARA\EXAMPLES\MACROS>tatara /l noamp.as noamp.tro noamp.lst
...
ended at NOAMP.AS(63)
```

MSX-DOS

```
                                        irpc    c,TATARA
                                        db      'c'
                                        endm
0080'   63              +               db      'c'
0081'   63              +               db      'c'
0082'   63              +               db      'c'
0083'   63              +               db      'c'
0084'   63              +               db      'c'
0085'   63              +               db      'c'
```

この例では TATARA の 6 文字が入るところに、文字 c である 63h が 6 バイト入ります。このような誤りでは TATARA は止まらないので、見つける場所はリスティングです。

第 31 章

長いシンボル名と
大文字・小文字を区別する名前

31.1 この例 243
31.2 ソース 244
31.3 作る 245
31.4 2 つのファイルを読む 246
31.5 モジュールをまたぐ長いシンボル名 247
31.6 マクロの名前と/C 248

TATARA のシンボルの名前は必要なだけ長くでき、そのすべての文字が意味を持ちます。/C を付けると、大文字と小文字の違いも意味を持ちます（7 章）。この章では、A:\TATARA\EXAMPLES\NAMES にある例、NAMES を作ります。1 つのソースでその両方を示す例です。そのあとで、長いシンボル名と/C を 1 つのソースの外に広げ、複数のモジュールからなるプログラムで使います。最後に、/C がマクロの名前にどう働くかを見ます。

31.1 この例

NAMES はプログラムを作りません。その BUILD.BAT は、1 つのソースを/C なしと/C 付きで 2 回アセンブルし、それぞれのシンボル表をテキスト・ファイルに残します。この 2 つのファイルが結果で、TATARA がどの名前を 1 つとして、どの名前を 2 つとして扱ったかがわかります。

31.2　ソース

```
; NAMES.AS - names as long as you like, in either case.
;
; A SYMBOL MAY BE 255 CHARACTERS AND ALL OF THEM COUNT. M80 keeps the
; first six, which is why so much CP/M-era assembly is written in
; abbreviations. The two names below differ only in their last five
; characters and are two different symbols here. 255 is the limit, and
; the test suite has a 240-character name in it; these are 48, which is
; as long as a line can carry and still be read on an MSX screen.

first_pass_symbol_table_high_water_mark_in_bytes        defl    1024
first_pass_symbol_table_high_water_mark_in_words        defl    512

; THE SAME SIX LETTERS IN TWO CASES. Without /C they are ONE symbol,
; and the second line changes its value. With /C they are TWO, with
; values 1 and 2. The symbol table of each run says which - and DEFL
; is used rather than EQU because a redefinition is legal, so the
; source assembles cleanly in both modes.

Counter         defl    1
counter         defl    2

; DIRECTIVES, MNEMONICS AND REGISTER NAMES ARE CASE-INSENSITIVE IN
; BOTH MODES. /C is about the names you choose, not the ones the Z80
; already has.

                cseg

start:          LD      A,1
                ld      b,a
                DB      Counter
                db      counter
                ret

                end     start
```

ソースは 3 つの部分からなります。

- **2 つの長いシンボル名**は、どちらも 48 文字で、最後の 5 文字 `bytes` と `words` だけが違います。これは別々の名前で、それぞれ 1024 と 512 という値を持ちます。
- `Counter` と `counter` は、同じ 7 文字を 2 通りの大文字・小文字で書いたもので、どちらも `DEFL` で値を与えています。`DEFL` は名前に新しい値を与え直せるので（12.4 節）、2 つが 1 つの名前でも 2 つの名前でも、ソースはアセンブルできます。
- **いくつかの命令**は、大文字のものとそうでないものがあり、両方の名前を使い

ます。命令、レジスター、疑似命令は、/C の有無にかかわらず、大文字でも小文字でも書けます。

31.3 作る

```
rem BUILD.BAT - EXAMPLE: NAMES
rem
rem One source, assembled twice, and the two symbol tables are the
rem answer.
rem
rem   /S   print the symbol table when the assembly ends
rem   /P   assemble to the screen, producing no object file
rem   /C   make symbol and macro names case-SENSITIVE
rem
rem Without /C, Counter and counter are one symbol whose value ends up
rem 2, and the table has one entry. With /C they are two symbols, 1
rem and 2, and the table has both. The default is M80s behaviour.
rem
rem The two long names are 48 characters each and differ in the last
rem five. M80 would have made them one symbol.

echo === Case-insensitive, which is the default
tatara /q /p /s names.as > insens.txt

echo === Case-sensitive
tatara /q /p /c /s names.as > sens.txt

echo === Done. Compare INSENS.TXT with SENS.TXT.
```

/P はリスティングを画面に表示し、/S はそのあとにシンボル表を表示します（5.5 節、19 章）。コマンド・ラインにオブジェクト・ファイルの名前がないので、オブジェクト・ファイルは書き出されません。それぞれの>は、TATARA が表示する内容をファイルに送ります。この例はインクルード・ファイルを使わないので、バッチ・ファイルは TATARA を設定しません。

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>build
=== Case-insensitive, which is the default
=== Case-sensitive
=== Done. Compare INSENS.TXT with SENS.TXT.
```

31.4　2 つのファイルを読む

どちらのファイルも、ページの見出しと、書いたとおりのソースからなるリスティングで始まります。/L がないので、リスティングにはアドレスもバイトもありません。そのあとにシンボル表が続きます。INSENS.TXT のシンボル表全体は次のとおりです。

MSX-DOS

```
ASEG - absolute

0002h  var        Counter
0200h  var        first_pass_symbol_table_high_water_mark_in_words
0400h  var        first_pass_symbol_table_high_water_mark_in_bytes

CSEG - default code segment, 0006h bytes

0000h             start

DSEG - default data segment, 0000h bytes

(no symbols)
```

SENS.TXT のシンボル表は次のとおりです。

MSX-DOS

```
ASEG - absolute

0001h  var        Counter
0002h  var        counter
0200h  var        first_pass_symbol_table_high_water_mark_in_words
0400h  var        first_pass_symbol_table_high_water_mark_in_bytes

CSEG - default code segment, 0006h bytes

0000h             start

DSEG - default data segment, 0000h bytes

(no symbols)
```

2 つの長いシンボル名は、どちらのファイルでも 2 つのシンボルです。/C の有無にかかわらず、名前のすべての文字が意味を持ちます。2 つのファイルが違うのは Counter だけで、図 31.1 のとおりです。

- /C なしでは、Counter と counter は 1 つの名前です。counter defl 2 がそれに新しい値 2 を与え、表にはその名前の行が 1 行だけ、最初に書いたときの

綴りで表示されます。

- /C 付きでは、2 つの名前になり、値は 1 と 2 です。

INSENS.TXT：/C なし

```
0002h  var  Counter
0200h  var  first_pass_symbol_table_high_water_mark_in_words
0400h  var  first_pass_symbol_table_high_water_mark_in_bytes
```

シンボルは 1 つ。2 つ目の DEFL が値を変えた

SENS.TXT：/C 付き

```
0001h  var  Counter
0002h  var  counter
0200h  var  first_pass_symbol_table_high_water_mark_in_words
0400h  var  first_pass_symbol_table_high_water_mark_in_bytes
```

シンボルは 2 つ。大文字・小文字ごとに 1 つ

図 31.1: 2 つのシンボル表のアブソリュートのセクション。

コードは、どちらの実行でも同じ 6 バイトです。/C なしでは、db Counter と db counter はどちらも 2 で、/C 付きでは 1 と 2 です。

31.5 モジュールをまたぐ長いシンボル名

名前はオブジェクト・ファイルの中でも長さをすべて保ち（7.2 節）、TANREN はその全体で照合します。LONGUSE.AS は、NAMES.AS の 2 つの長いシンボル名を外部の名前として使います。

```
; LONGUSE.AS - uses two long names from another module.
                extrn   first_pass_symbol_table_high_water_mark_in_bytes
                extrn   first_pass_symbol_table_high_water_mark_in_words
                cseg
                ld      hl,first_pass_symbol_table_high_water_mark_in_bytes
                ld      de,first_pass_symbol_table_high_water_mark_in_words
                ret
                end
```

LONGDEF1.AS は、そのうち最初の名前だけを定義します。

```
; LONGDEF1.AS - defines only the first of the two.
                public  first_pass_symbol_table_high_water_mark_in_bytes
                cseg
first_pass_symbol_table_high_water_mark_in_bytes:
                nop
                end
```

2 つの名前は最初の 43 文字が同じですが、TANREN は一方を他方と取り違えません。足りない名前は、すべての文字で表示されます。

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>tatara /q longuse.as longuse.tro
A:\TATARA\EXAMPLES\NAMES>tatara /q longdef1.as longdef1.tro
A:\TATARA\EXAMPLES\NAMES>tanren longuse longdef1 /o:l1.com
...
Never defined:
  first_pass_symbol_table_high_water_mark_in_words
ERROR: the symbols above were never defined.
```

LONGDEF2.AS は、それぞれを nop 付きのラベルとして、両方を定義します。

```
; LONGDEF2.AS - defines both.
                public  first_pass_symbol_table_high_water_mark_in_bytes
                public  first_pass_symbol_table_high_water_mark_in_words
                cseg
first_pass_symbol_table_high_water_mark_in_bytes:
                nop
first_pass_symbol_table_high_water_mark_in_words:
                nop
                end
```

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>tatara /q longdef2.as longdef2.tro
A:\TATARA\EXAMPLES\NAMES>tanren /m longuse longdef2 /o:l2.com
...
Symbols:
  DR 0108 first_pass_symbol_table_high_water_mark_in_words
  DR 0107 first_pass_symbol_table_high_water_mark_in_bytes
2 modules, 15 records, ends at EOF.
Wrote L2.COM, 0100-0108 (9 bytes).
```

どちらの名前も、それぞれのアドレスで照合されます。そのため、プログラムは、print_string や keyboard_buffer_length のように、ルーチンや変数にその内容どおりの名前を、すべてのモジュールを通じて付けられます。限度は 255 文字と、1 行の長さだけです（7.2 節）。

31.6　マクロの名前と /C

/C は、シンボルだけでなくマクロの名前にも効きます（16.8 節）。MACCASE.AS は Twice という名前のマクロを定義し、2 通りの大文字・小文字で 1 回ずつ使います。

```
; MACCASE.AS - a macro defined as Twice and used as twice.
Twice           macro
                nop
                nop
                endm
                cseg
                Twice
                twice
                end
```

/C なしでは、どちらの行もマクロで、それぞれ 2 つの nop になります。

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>tatara /p /s maccase.as
...
                cseg
                Twice
                nop
                nop
                twice
                nop
                nop
                end
...
CSEG - default code segment, 0004h bytes
```

/C 付きでは、twice はマクロの名前でも、命令や疑似命令の名前でもありません。

MSX-DOS

```
A:\TATARA\EXAMPLES\NAMES>tatara /p /c /s maccase.as
...
MACCASE.AS(8): ERROR: not a directive, a macro or an instruction.
```

/C を前提に書くソースでは、インクルード・ファイルのマクロ、たとえば system も含め、どのマクロも定義とまったく同じ大文字・小文字で使う必要があります。

第 32 章

MSX-BASICのBLOAD用のバイナリ

32.1 この例 251
32.2 ソース 252
32.3 作る 253
32.4 BASIC から実行する 254
32.5 ファイルを置ける場所 254
32.6 試してみること 256

MSX-BASIC は、マシン語のファイルを BLOAD で読み込んで実行できます。この章では、A:\TATARA\EXAMPLES\BINARY にある例、BINARY を作ります。そのようなファイルを作って BASIC から実行し、そのあとで、BASIC のメモリのどこにマシン語のファイルを安全に置けるかを見ます。

32.1 この例

BINARY は、1 つのソース BEEP.AS と、その BUILD.BAT からなります。プログラムは 1 行を表示し、MSX のブザーを鳴らして、BASIC に戻ります。これは.COM のプログラムではありません。BASIC が実行するときには MSX-DOS がないので、MSX の ROM にあるルーチン、つまり BIOS を使って表示します。TANREN は、これを BLOAD のヘッダー付きで書き出し（26.4 節）、それによって.BIN のファイルになります。

32.2　ソース

```
; BEEP.AS - a machine-code file for MSX BASIC.
;
; BLOAD"BEEP.BIN",R loads this and runs it. The seven bytes BASIC
; reads first - FEh, the first address, the LAST address, and the
; address to call - are written by TANREN when /B is given, and
; nothing else makes a file a .BIN.
;
; ASEG and ORG: the code is at a fixed address and the linker moves
; nothing. It has to be, because BASIC loads it where the header says
; and does not relocate anything.
;
; There is no MSX-DOS here, so the printing is the BIOS.

                include bios.inc        ; CHPUT, BEEP, and every other
                                        ;   MAIN ROM entry
                include ascii.inc       ; CHR_CR and CHR_LF

                aseg
                org     8000h           ; above BASIC in a 64 KB machine

start:          ld      hl,msg
loop:           ld      a,(hl)
                or      a
                jr      z,done
                push    hl
                call    CHPUT
                pop     hl
                inc     hl
                jr      loop
done:           call    BEEP
                ret                     ; back to BASIC, which is waiting

msg:            db      "Loaded by BLOAD, run from BASIC.",CHR_CR,CHR_LF,0

                end     start           ; the address the header carries
```

- **ASEG と ORG** は、コードをアブソリュート・セグメントの 8000h に置きます（10 章）。ファイルはヘッダーにあるアドレスに読み込まれ、TANREN はソースが示すアドレスをそのまま使います。
- **ループ**は、最後の 0 のバイトに達するまで、A の文字を表示する BIOS のルーチン CHPUT で、メッセージを 1 文字ずつ表示します。CHPUT と BEEP のアドレスは、BIOS.INC にあります。
- BEEP は、ブザーを鳴らす別の BIOS のルーチンで、RET で BASIC に戻ります。
- **END の行**は start を示します。TANREN はこれを、呼び出すアドレスとしてヘッダーに入れます（26.5 節）。

32.3 作る

```
rem BUILD.BAT - EXAMPLE: BINARY
rem
rem A .BIN for MSX BASIC instead of a .COM for MSX-DOS. Two things
rem make the difference:
rem
rem   ASEG and ORG   the code is at a fixed address, not relocatable
rem   TANREN /B      the seven-byte BLOAD header goes in front of it
rem
rem The address BASIC calls is the one on the END line of the source.
rem Without /B the same link writes the bytes and no header, which is
rem what a ROM needs - see the ROM example.
rem
rem To run it, from BASIC:  BLOAD then the file name in quotes, ,R

rem The headers are in A:\TATARA\INCLUDE and are not copied here.
rem TATARA is rule 3 of the include search - see the DIRS example.
rem CHANGE THE PATH BELOW if you put the tree somewhere else.

set TATARA=a:\tatara\include

echo === Assembling
tatara /q beep.as beep.tro

echo === Linking, with a BLOAD header
tanren /q /b /o:beep.bin beep.tro

echo === Done. BEEP.BIN is 7 bytes of header and the rest is code.

set TATARA=
```

MSX-DOS

```
A:\TATARA\EXAMPLES\BINARY>build
=== Assembling
=== Linking, with a BLOAD header
=== Done. BEEP.BIN is 7 bytes of header and the rest is code.
```

/Q なしで実行すると、TANREN の要約で書き出したものがわかります。

MSX-DOS

```
A:\TATARA\EXAMPLES\BINARY>tanren /b /o:beep.bin beep.tro
...
1 modules, 6 records, ends at EOF.
Wrote BEEP.BIN, 8000-8035 (54 bytes), BLOAD header, entry 8000.
```

プログラムは 54 バイトで、ファイルは 61 バイトです。最初の 7 バイトがヘッダーです。

```
FE  00 80  35 80  00 80
```

FE、続いて最初のアドレス 8000h、最後のアドレス 8035h、呼び出すアドレス 8000h で、どれも下位バイトが先です。そのあとにプログラムが続きます。21 13 80 は ld hl,msg で、メッセージは 8013h にあります。

32.4　BASIC から実行する

MSX-BASIC で、例のディレクトリから、,R を付けて BLOAD すると、ファイルを読み込み、その開始アドレスを呼び出します。

MSX-DOS

```
Ok
BLOAD"BEEP.BIN",R
Loaded by BLOAD, run from BASIC.
Ok
```

行のあとにブザーが鳴り、RET で BASIC に戻って、BASIC が Ok と表示します。

32.5　ファイルを置ける場所

BASIC が動いている間、8000h より下のアドレスには BASIC の ROM と BIOS があるので、BLOAD 用のファイルは、その上の RAM に置く必要があります。しかし、BASIC もその RAM を使います。メモリの中の BASIC のプログラムは 8000h のすぐ上から始まり、そのあとに変数が続きます。上の例がうまくいったのは、メモリにプログラムがなかったからです。プログラムがあると、同じファイルがそれを上書きします。

MSX-DOS

```
list
10 PRINT "hello"
Ok
bload"beep.bin",r
Loaded by BLOAD, run from BASIC.
Ok
list
46974 INSTRMAXTRON
25697 ed by BLOAD, run from BASIC.501WAIT_VPOKE=^KILLL<0MERGEeMAXO<>41933;+;(3:(
...
```

マシン語はそれでも実行されて 1 行を表示しましたが、BASIC のプログラムは消えています。LIST は、マシン語をプログラムのように読んで表示し、そのあとマシンは反応しなくなります。

CLEAR を使うと、ファイルに必要なメモリから BASIC を遠ざけられます。CLEAR 200,&HC000 は、BASIC に文字列用の 200 バイトを与え、さらにここで大事なことに、C000h から上のメモリを使わないように BASIC に伝えます。そうすれば、図 32.1 のように、C000h に置いたファイルは BASIC の邪魔になりません。

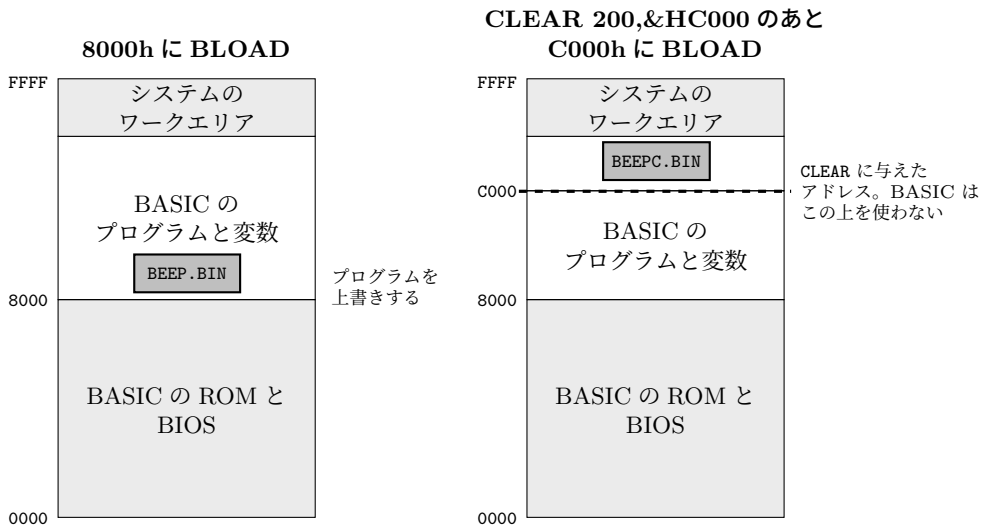

図 32.1: MSX-BASIC のメモリ。ファイルを 8000h に置いた場合と、C000h に置いた場合。

BEEPC.AS は、BEEP.AS の ORG を変えたものです。

```
                aseg
                org     0C000h          ; above the CLEAR address
```

MSX-DOS

```
Ok
clear 200,&hc000
Ok
bload"beepc.bin",r
Loaded by BLOAD, run from BASIC.
Ok
```

CLEAR は、最初の BLOAD の前、そして BASIC のプログラムを入力したり読み込んだりする前に実行してください。CLEAR は変数も消すからです。マシン語を使う BASIC のプログラムは、ふつう CLEAR の行で始まり、そのあとで自分のファイルを BLOAD します。マシン語が多いほど、CLEAR に与えるアドレスを低くする必要があり、BASIC に残るメモリは少なくなります。

32.6　試してみること

次のファイルは例には含まれていません。BUILD.BAT と同じように TATARA を設定して、例のディレクトリに作ってください。

32.6.1　コードを TANREN に置かせる

モジュールは ASEG と ORG を使わなくてもかまいません。BEEPR.AS は同じプログラムを標準のコード・セグメントに置いたもので、TANREN が/P: でそれを置きます（24.3 節）。

```
; BEEPR.AS - the same code, relocatable: TANREN /P: places it.
                include bios.inc
                include ascii.inc
                cseg
start:          ld      hl,msg
loop:           ld      a,(hl)
                or      a
                jr      z,done
                push    hl
                call    CHPUT
                pop     hl
                inc     hl
                jr      loop
done:           call    BEEP
                ret
msg:            db      "Relocated by TANREN /P:.",CHR_CR,CHR_LF,0
```

```
                end     start
```

MSX-DOS

```
A:\TATARA\EXAMPLES\BINARY>tatara /q beepr.as beepr.tro
A:\TATARA\EXAMPLES\BINARY>tanren /b /p:c000 /o:beepr.bin beepr.tro
...
Wrote BEEPR.BIN, C000-C02D (46 bytes), BLOAD header, entry C000.
```

こうすれば、同じソースをどのアドレス用にもリンクでき、複数のモジュールからなるプログラムを全体としてまとめて置けます。

32.6.2 USR で呼び出す

,R なしでは、BLOAD はファイルを読み込むだけです。DEFUSR がマシン語のルーチンのアドレスを示し、USR がそれを呼び出します。

MSX-DOS

```
Ok
clear 200,&hc000
Ok
bload"beepr.bin"
Ok
defusr=&hc000
Ok
a=usr(0)
Relocated by TANREN /P:.
Ok
```

こうすれば、ルーチンを読み込み直さずに、プログラムが必要なだけ何度でも呼び出せます。

32.6.3 /B を外す

/B なしでリンクすると、ファイルは同じ 54 バイトで、ヘッダーはありません。

MSX-DOS

```
A:\TATARA\EXAMPLES\BINARY>tanren /o:beepraw.bin beep.tro
...
Wrote BEEPRAW.BIN, 8000-8035 (54 bytes), entry 8000.
```

FE で始まらないので、BLOAD はこれを受け付けません。

MSX-DOS

```
Ok
bload"beepraw.bin",r
Bad file mode
Ok
```

第 33 章

16 KB の ROM カートリッジ

33.1 この例 259
33.2 ソース 260
33.3 作る 262
33.4 実行する 263
33.5 試してみること 263

カートリッジは、電源を入れたときに MSX が自分で見つけて起動する、ROM の中のプログラムです。この章では、`A:\TATARA\EXAMPLES\ROM` にある例、`ROM` を作ります。16 KB のカートリッジのイメージを作る例で、MSX が探すヘッダー、1 行を表示するルーチン、そしてファイルを正しい大きさにする詰め物からなります。そのあと、イメージをエミュレーターで実行し、3 通りに変えてみます。

33.1 この例

カートリッジは、MSX-DOS や MSX-BASIC よりも前に起動します。MSX は各スロットを調べてカートリッジを見つけ、まだ起動の途中でそれを呼び出します。プログラムはいつでもある BIOS を使えますが、MSX-DOS や BASIC が用意するものは使えません。`ROM` は、テキスト画面を用意して 1 行を表示し、そのまま止まります。

この例には、`ROM.AS` と `BUILD.BAT` の 2 つのファイルがあります。結果は、カートリッジの 16 KB をそのままバイト単位で持つファイル `ROM.ROM` です。これは MSX-DOS が実行できるプログラムではなく、エミュレーターにカートリッジとして入れるか、フラッシュ・カートリッジに書き込んで使います。

33.2　ソース

```
; ROM.AS - a 16 KB cartridge.
;
; A cartridge in page 1 begins at 4000h with a sixteen-byte header.
; The BIOS looks for the letters AB, and calls the INIT address if it
; is not zero - before there is an operating system of any kind.
;
; NOTHING HERE MAY CALL MSX-DOS. There is no MSX-DOS yet. The screen
; and the printing are the BIOS, and the code never returns, because
; there is nothing to return to.
;
; THE LAST TWO LINES ARE WHAT MAKES THE FILE 16 KB. TANREN writes the
; span from the lowest byte of content to the highest, and a gap
; inside that span comes out as zeros. One byte at 7FFFh therefore
; makes the file exactly 16384 bytes, which is what a cartridge has
; to be. A DS would not do it: the space a DS reserves after the last
; byte of content is deliberately not in the file.

                include bios.inc        ; INITXT, CHPUT, and the rest
                include ascii.inc       ; CHR_CR and CHR_LF

                aseg
                org     4000h

                db      "AB"            ; a cartridge, says the BIOS
                dw      init            ; INIT: called at once
                dw      0               ; STATEMENT: no CALL statements
                dw      0               ; DEVICE: no device
                dw      0               ; TEXT: no BASIC program in here
                dw      0,0,0           ; reserved, and zero

init:           call    INITXT          ; a text screen, from nothing
                ld      hl,msg
ini.lp:         ld      a,(hl)
                or      a
                jr      z,ini.end
                push    hl
                call    CHPUT
                pop     hl
                inc     hl
                jr      ini.lp
ini.end:        jr      ini.end         ; a cartridge has nowhere to go

msg:            db      "Assembled by Tatara. Linked by TANREN.",CHR_CR,CHR_LF,0

                org     7fffh           ; the last byte of the 16 KB...
                db      0               ;   ...so that the file is all of it

                end
```

33.2.1 ヘッダー

16 KB のカートリッジは 4000h から 7FFFh にあり、その最初の 16 バイトは、カートリッジに何が入っているかを MSX に伝えるヘッダーです（図 33.1）。

- **ID** は 2 バイトで、文字 `AB` です。これがないと、MSX はその ROM をカートリッジとして扱いません。
- **INIT** は、MSX が起動するときに呼び出すルーチンのアドレスで、なければ 0000h です。ここでは `init` です。
- **STATEMENT**、**DEVICE**、**TEXT** は、BASIC に `CALL` 文やデバイスを加えるルーチンと、ROM の中の BASIC プログラムのアドレスです。このカートリッジにはどれもないので、0000h です。
- **予約された 6 バイト**は、00h でなければなりません。

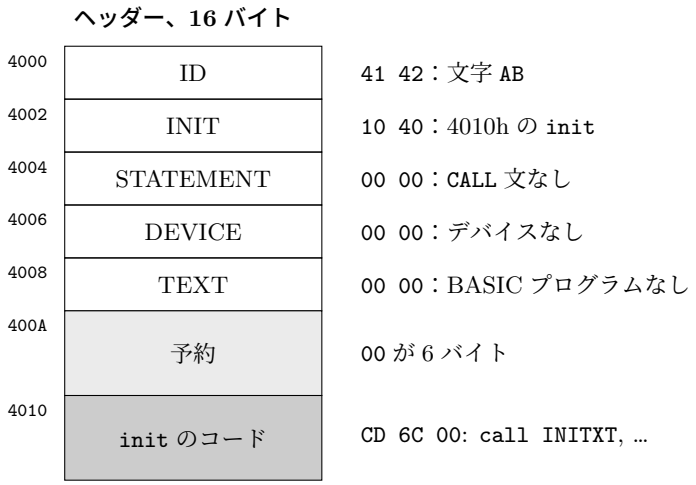

図 33.1: 4000h にある `ROM.ROM` のヘッダーと、各欄の値。

MSX はヘッダーを 4000h と 8000h で探します（26.6 節）。`ASEG` と `ORG 4000h` がヘッダーを 4000h に置き、TANREN が動かすものは何もありません。

33.2.2 INIT

`init` は、テキスト画面を用意する BIOS のルーチン `INITXT` を呼び、`CHPUT` でメッセージを表示してから、`ini.end: jr ini.end` で自分自身へ永久にジャンプし続けます。MSX は、カートリッジが残した画面のままになります。たとえばゲームは、このようにしてマシンを占有します。

33.2.3　最後の 2 行

カートリッジのイメージには、16 KB のすべてが入っていなければなりません。TANREN は内容のある最初のバイトから最後のバイトまでを書き出すので（24.1.1 節）、プログラムだけでは 77 バイトのファイルになります。`org 7fffh` と `db 0` が最後のアドレスに 1 バイトを置き、メッセージから 7FFFh までの間はすべて 0 として書き出されます。

33.3　作る

```
rem BUILD.BAT - EXAMPLE: ROM
rem
rem A 16 KB cartridge image. No MSX-DOS anywhere in it: the code runs
rem from a cartridge slot before there is an operating system, so it
rem calls the BIOS and never returns.
rem
rem   ASEG and ORG 4000h   page 1, where a 16 KB cartridge lives
rem   no /B                a ROM has no header of any kind
rem   ORG 7FFFH and DB 0   what makes the file exactly 16384 bytes
rem
rem THIS IS THE ONE EXAMPLE A SUCCESSFUL BUILD DOES NOT PROVE. Load
rem ROM.ROM as a cartridge in an emulator, or write it to a flash
rem cartridge. It is not a program whose name you can type.

rem The headers are in A:\TATARA\INCLUDE and are not copied here.
rem TATARA is rule 3 of the include search - see the DIRS example.
rem CHANGE THE PATH BELOW if you put the tree somewhere else.

set TATARA=a:\tatara\include

echo === Assembling
tatara /q rom.as rom.tro

echo === Linking, with no header of any kind
tanren /q /o:rom.rom rom.tro

echo === Done. ROM.ROM should be exactly 16384 bytes.

set TATARA=
```

リンクには/B を付けません。ファイルはカートリッジ自身のバイトで、その前に BLOAD のヘッダーはありません。ヘッダーは、ソースの中にあるものだけです。

```
A:\TATARA\EXAMPLES\ROM>build
=== Assembling
=== Linking, with no header of any kind
=== Done. ROM.ROM should be exactly 16384 bytes.
```

/Q なしで実行すると、TANREN の要約で大きさを確かめられます。

MSX-DOS

```
A:\TATARA\EXAMPLES\ROM>tanren /o:rom.rom rom.tro
...
1 modules, 6 records, ends at EOF.
Wrote ROM.ROM, 4000-7FFF (16384 bytes).
```

ファイルは 41 42 10 40 で始まります。AB と、下位バイトが先の init のアドレス 4010h で、そのあとにほかの欄の 10 バイトの 0 が続きます。init は 4010h から CD 6C 00、つまり call INITXT で始まります。

33.4 実行する

ROM.ROM をカートリッジとしてエミュレーターに入れ、MSX をリセットします。たとえば openMSX では、コンソールのコマンド carta が ROM イメージを 1 つ目のカートリッジ・スロットに入れ、reset がマシンを再起動します。画面には、テキスト画面にメッセージが表示されるだけです。

MSX-DOS

```
Assembled by Tatara. Linked by TANREN.
```

MSX は 4000h に AB を見つけて init を呼び、そこから戻ってきませんでした。MSX-DOS も BASIC も起動しません。

33.5 試してみること

次のファイルは例には含まれていません。BUILD.BAT と同じように TATARA を設定して例のディレクトリに作り、ROM.AS と同じようにリンクしてください。

33.5.1 INIT から戻る

ROMRET.AS は、ROM.AS の永久ループを ret にしたものです。

```
ini.end:        ret                     ; and the MSX goes on starting
```

メッセージが少しの間表示され、そのあと MSX は、カートリッジがないときと同じように起動を続けます。ディスク・ドライブのある MSX では、MSX-DOS が起動します。ROM.AS の先頭のコメントには戻る先がないとありますが、MSX は INIT が戻るのを待っています。システムに何かを加えるだけで、残りの起動はシステムに任せるカートリッジは、必要な準備をしてから戻ります。

33.5.2　ID を壊す

ROMBAD.AS は、ROM.AS の"AB"を"AC"にしたものです。

```
                db      "AC"            ; not a cartridge ID
```

ほかと同じように 16 KB のファイルにリンクできますが、MSX はこれをカートリッジとして扱いません。何も表示されず、MSX はスロットが空であるかのように MSX-DOS を起動します。カートリッジが何もしないときは、ファイルの最初の 2 バイトを確かめてください。

33.5.3　カートリッジを TANREN に置かせる

カートリッジは ASEG を使わなくてもかまいません。ROMREL.AS は同じカートリッジを標準のコード・セグメントに置いたもので、/P: で 4000h に置かれます（24.3 節）。

```
; ROMREL.AS - the cartridge as a relocatable module, placed by /P:4000.
                include bios.inc
                include ascii.inc
                cseg
header:         db      "AB"
                dw      init
                dw      0,0,0
                dw      0,0,0
init:           call    INITXT
                ld      hl,msg
ini.lp:         ld      a,(hl)
                or      a
                jr      z,ini.end
                push    hl
                call    CHPUT
                pop     hl
                inc     hl
                jr      ini.lp
ini.end:        jr      ini.end
msg:            db      "Placed by TANREN /P:4000.",CHR_CR,CHR_LF,0
                ds      3FFFh-($-header)
                db      0
```

```
                end
```

リロケータブルなセグメントを ORG で決まったアドレスまで埋めることはできないので、詰め物の量は大きさから計算します。$-header はそこまでのバイト数で、DS は 16 KB から 1 バイトを引いた残りを確保します。そのあとの DB が最後のバイトを埋めるので、DS はファイルの終わりのあとではなく、ファイルの中に入ります。

MSX-DOS

```
A:\TATARA\EXAMPLES\ROM>tatara /q romrel.as romrel.tro
A:\TATARA\EXAMPLES\ROM>tanren /p:4000 /o:romrel.rom romrel.tro
...
1 modules, 7 records, ends at EOF.
Wrote ROMREL.ROM, 4000-7FFF (16384 bytes).
```

カートリッジとして入れると、自分の行を表示してそのまま止まります。

MSX-DOS

```
Placed by TANREN /P:4000.
```

このように書けば、カートリッジのコードも、ほかのプログラムと同じように複数のモジュールに分けることができ、TANREN がそのすべてを置きます。

第 34 章

複数のディレクトリにまたがるプロジェクト

34.1 この例 267
34.2 ソース 268
34.3 作って実行する 270
34.4 試してみること 271

ある程度の大きさのプログラムは、すぐに 1 つのディレクトリに収まらなくなります。ソースはある場所に、それが取り込むファイルはその隣に、ほかのプログラムと共有するヘッダーは別の場所に、そしてオブジェクト・ファイルはそれらとは別に置くこともあります。この章では、`A:\TATARA\EXAMPLES\DIRS` にある例、`DIRS` を作ります。ファイルが 3 つのディレクトリに分かれている例で、TATARA がそれぞれをどのように見つけるかをたどります（13 章）。そのあとで、同じプログラムを別の方法でも作ります。別のディレクトリから作る方法、`TATARA` に相対パスのディレクトリを書く方法、そしてオブジェクト・ファイルとリンク・ファイルを専用のディレクトリに置く方法です。

34.1 この例

`DIRS` は 3 行を表示します。それぞれの行は別々のインクルード・ファイルにあり、そのファイルを見つけた探し方の規則を示しています。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>dirs
Rule 1: beside the file that asked.
Rule 1 again, one level deeper.
Rule 3: found through TATARA.
```

TATARA は、インクルード・ファイルを次の 3 つの場所でこの順に探します（13.2 節）。

1. INCLUDE の行があるファイルのディレクトリ
2. カレント・ディレクトリ
3. TATARA に書いたそれぞれのディレクトリ

図 34.1 に、この例のファイルと、DIRS から作るときにそれぞれを見つける規則を示します。

図 34.1: DIRS のファイルと、各インクルード・ファイルを見つける規則。

34.2　ソース

モジュールは SRC\MAIN.AS です。

```
; SRC\MAIN.AS - a program whose parts are in three directories.
;
; AN INCLUDE IS LOOKED FOR IN THREE PLACES, IN THIS ORDER:
;
;   1. the directory of the file doing the including
;   2. the current directory
;   3. each directory named in the TATARA environment variable
;
; Rule 1 comes first so that a file always finds the header sitting
; beside it, whatever directory you happen to be standing in when you
; run the build. Neither line below depends on where that is.

                include ascii.inc       ; CHR_CR and CHR_LF, for the messages
                include msxdos.inc      ; rule 3: TATARA names the
                                        ;   directory it is in

                cseg

start:          ld      de,text
                call    prtstr
                ld      de,deeper
                call    prtstr
                ld      de,sysmsg
                call    prtstr
                system  _TERM0

prtstr:         ld      c,_STROUT
                jp      BDOS            ; and BDOS returns to the caller

; THE MESSAGES COME AFTER THE CODE. Each include below holds a DB,
; and MSX-DOS starts a .COM at 0100h: text there would be run as
; instructions.

                include inc\text.inc    ; rule 1, with a separator:
                                        ;   relative to THIS file
                include sysmsg.inc      ; rule 3 as well, from the OTHER
                                        ;   directory in TATARA

                end     start
```

このソースは 4 つのファイルを、2 つのグループに分けて取り込んでいます。

- 先頭の ASCII.INC と MSXDOS.INC は、名前と system マクロを定義するだけで、バイトを作りません。どちらも MAIN.AS の隣にもカレント・ディレクトリにもないので、規則 3 によって、TATARA の最初のディレクトリ A:\TATARA\INCLUDE で見つかります。
- コードのあとの INC\TEXT.INC と SYSMSG.INC は、それぞれメッセージを持っています。これらがコードのあとにあるのは、バイトを作るからです。start より前に置くと、テキストが 0100h に置かれ、プログラムが始まるときに MSX-DOS がそれを命令として実行してしまいます（21.1 節）。

inc\text.inc は名前にディレクトリを含みますが、ドライブも先頭の\もないので相対パスです（13.4 節）。規則 1 はこれを MAIN.AS から見た位置、つまり SRC\INC

で探します。SRC\INC\TEXT.INC は最初のメッセージを持ち、MSG.INC を取り込みます。

```
; SRC\INC\TEXT.INC - reached as inc\text.inc from SRC\MAIN.AS, so
; rule 1 found it: a name with a separator in it is still relative to
; the file that asked for it.
;
; The include below has no directory at all, and rule 1 finds it
; beside THIS file rather than beside MAIN.AS.

                include msg.inc

text:           db      "Rule 1: beside the file that asked.",CHR_CR,CHR_LF,"$"
```

この INCLUDE についての規則 1 は TEXT.INC のディレクトリなので、MSG.INC も SRC\INC で見つかります。

```
; SRC\INC\MSG.INC - a bare name, included from TEXT.INC, which lives
; in this directory.
;
; IF THE SEARCH STARTED IN THE CURRENT DIRECTORY, THIS WOULD NOT BE
; FOUND: the build runs two levels up, and there is no MSG.INC there.

deeper:         db      "Rule 1 again, one level deeper.",CHR_CR,CHR_LF,"$"
```

LIB\SYSMSG.INC は、そのどちらのディレクトリにも、カレント・ディレクトリにもありません。見つかるのは、BUILD.BAT が LIB を TATARA に入れているからです。

```
; LIB\SYSMSG.INC - not beside MAIN.AS, and not in the directory the
; build runs from. BUILD.BAT puts this directory in TATARA, which is
; rule 3, and that is how a project keeps its shared headers in one
; place instead of a copy per program.

sysmsg:         db      "Rule 3: found through TATARA.",CHR_CR,CHR_LF,"$"
```

34.3　作って実行する

```
rem BUILD.BAT - EXAMPLE: DIRS
rem
rem Sources in SRC, headers beside them in SRC\INC, shared headers in
rem LIB. The build runs from HERE and nothing is copied anywhere.
rem
rem TATARA holds the directories rule 3 looks in - the same idea as
rem PATH, and the same shape: several may be separated by semicolons.
rem It is set for the build and cleared afterwards so that nothing
rem outside this example inherits it.
rem
rem THE PATH BELOW IS ABSOLUTE and names this example. MSX-DOS gives a
rem batch file no way to ask where it is, so change this line if you
rem put the tree somewhere else.
```

```
set TATARA=a:\tatara\include;a:\tatara\examples\dirs\lib

echo === Assembling SRC\MAIN.AS
tatara /q src\main.as dirs.tro

echo === Linking
tanren /q /o:dirs.com dirs.tro

set TATARA=

echo === Done. Type DIRS to run it.
```

TATARA には、セミコロンで区切った 2 つのディレクトリが入っています（13.3 節）。バッチ・ファイルは DIRS から実行し、ソースを src\main.as と指定します。オブジェクト・ファイルとプログラムは DIRS に書き出されます。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>build
=== Assembling SRC\MAIN.AS
=== Linking
=== Done. Type DIRS to run it.
A:\TATARA\EXAMPLES\DIRS>dirs
Rule 1: beside the file that asked.
Rule 1 again, one level deeper.
Rule 3: found through TATARA.
```

/Q なしで実行すると、プログラムが当然そうあるべきように、0100h のコードから始まることが TANREN の表示でわかります。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>tanren /o:dirs.com dirs.tro
...
1 modules, 7 records, ends at EOF.
Wrote DIRS.COM, 0100-0183 (132 bytes), entry 0100.
```

34.4 試してみること

次のどれも DIRS から始め、特に書いていなければ TATARA は BUILD.BAT と同じように設定します。

34.4.1 TATARA から LIB を外す

TATARA にインクルード用のディレクトリしかないと、SYSMSG.INC はどこからも見つかりません。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>set tatara=a:\tatara\include
A:\TATARA\EXAMPLES\DIRS>tatara src\main.as nolib.tro
...
ERROR: cannot open SYSMSG.INC
    included from SRC\MAIN.AS(36)
```

メッセージは、書いたとおりのファイル名と、それを求めた MAIN.AS の行を示します。

34.4.2　TATARA に相対パスのディレクトリを書く

TATARA のディレクトリも、ほかのパスと同じように相対パスで書けます。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>set tatara=a:\tatara\include;lib
A:\TATARA\EXAMPLES\DIRS>tatara src\main.as rel.tro
...
ended at SRC\MAIN.AS(39)
```

lib はカレント・ディレクトリから見た位置なので、これがうまくいくのは DIRS から作るときだけです。BUILD.BAT は絶対パスを使っているので、どこから作っても TATARA の意味は変わりません。

34.4.3　別のディレクトリから作る

規則 1 はカレント・ディレクトリに左右されません。SRC の中からアセンブルしても、MAIN.AS は INC\TEXT.INC と MSG.INC を見つけ、残りは TATARA が見つけます。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>cd src
A:\TATARA\EXAMPLES\DIRS\SRC>tatara main.as ..\dirs2.tro
...
ended at MAIN.AS(39)
A:\TATARA\EXAMPLES\DIRS\SRC>cd ..
A:\TATARA\EXAMPLES\DIRS>tanren /o:dirs2.com dirs2.tro
...
Wrote DIRS2.COM, 0100-0183 (132 bytes), entry 0100.
A:\TATARA\EXAMPLES\DIRS>dirs2
Rule 1: beside the file that asked.
Rule 1 again, one level deeper.
Rule 3: found through TATARA.
```

34.4.4　オブジェクト・ファイルを分けて置く

大きなプロジェクトでは、オブジェクト・ファイルと、それを指定するリンク・ファイルを、専用のディレクトリに置くことがよくあります。OBJ\DIRS.LNK はそのようなリンク・ファイルです。

```
; DIRS.LNK - the link, kept with the object files.
main
/o:dirs3.com
```

TATARA はオブジェクト・ファイルを OBJ に書き出し、TANREN はリンク・ファイルを与えられると、まずその隣で MAIN.TRO を探します（23.1 節）。

MSX-DOS

```
A:\TATARA\EXAMPLES\DIRS>tatara /q src\main.as obj\main.tro
A:\TATARA\EXAMPLES\DIRS>tanren @obj\dirs
...
1 modules, 7 records, ends at EOF.
Wrote DIRS3.COM, 0100-0183 (132 bytes), entry 0100.
A:\TATARA\EXAMPLES\DIRS>dirs3
Rule 1: beside the file that asked.
Rule 1 again, one level deeper.
Rule 3: found through TATARA.
```

DIRS3.COM は OBJ ではなく、カレント・ディレクトリの DIRS に書き出されます。/O: は名前を書いたとおりに使うからです（21.2 節）。プログラムを別の場所に置くには、/o:bin\dirs3.com のように、名前にパスを含めてください。

第 35 章

TATARA と TANREN を 自身のソースからビルドする

35.1 ソース 275
35.2 BUILD.BAT 276
35.3 実行する 279
35.4 ツール自身でツールをビルドする 280

TATARA と TANREN は、TATARA がアセンブルするソース言語で書かれていて、MSX の上で、TATARA と TANREN によってビルドされます。それぞれのリリースは、その前のリリースが作ったものです。この章では、ソースのある場所とその構成を示し、2 つのツールをビルドするバッチ・ファイルを順に見て、それを実行します。そのあとで、最初のビルドで作ったツールを使って、ツールをもう一度ビルドします。

ツールの内部のしくみはここでは扱いません。それぞれのモジュールは、ソースのコメントで説明されています。

35.1 ソース

3 章でインストールするアーカイブには、2 つのプログラム、インクルード・ファイル、サンプル、`README.TXT`、`LICENSE` が入っていますが、ソースは入っていません。ソースは公開リポジトリ `https://github.com/javilm/tatara` にあり、ZIP ファイルとしてダウンロードして MSX にコピーできます。

図 35.1 に、ツリーのうちビルドが使う部分を示します。この章では、ツリーを `A:\SRC\TATARA` に置きます。中のディレクトリの構成を変えなければ、ほかの場所でもかまいません。

リポジトリには、ビルドに必要なもの以外のものも入っています。配布物のイン

図 35.1: ソースのツリーと、そのうちビルドが使うもの。

クルード・ファイル、このマニュアルの例、2 つのツールのテスト一式です。BUILD\ には現在のリリースのプログラムが入っていて、ビルドはその上に自分の作ったものを書き込みます。

> 注. ソースは約 700 KB あり、ビルドにはそのほかにオブジェクト・ファイルと 2 つのプログラムを置く場所が必要です。これは 720 KB のフロッピー・ディスクに入る量を超えるので、ツールはハード・ディスクか、CF カードや SD カードの上でビルドしてください。

ビルドはすでにインストールされているツールで行うので、A:\TATARA\BIN、または TATARA.COM と TANREN.COM のあるディレクトリが PATH に入っていなければなりません（3.4 節）。

35.2　BUILD.BAT

BUILD.BAT は 27 のモジュールをアセンブルし、2 つのツールをリンクします。

```
rem BUILD.BAT - assemble and link both tools, with Tatara.

rem SILENCE IS SUCCESS. /Q prints nothing for a module that works and
rem errors print anyway, so anything on the screen between the ===
rem lines is a problem.

set TATARA=shared\


echo === Assembling TATARA modules...
tatara /q tatara\tatara.as tatara.tro
tatara /q tatara\cmdline.as cmdline.tro
tatara /q tatara\srcline.as srcline.tro
tatara /q tatara\fields.as fields.tro
```

```
tatara /q tatara\dirtab.as dirtab.tro
tatara /q tatara\macros.as macros.tro
tatara /q tatara\errs.as errs.tro
tatara /q tatara\expand.as expand.tro
tatara /q tatara\cond.as cond.tro
tatara /q tatara\expr.as expr.tro
tatara /q tatara\symtab.as symtab.tro
tatara /q tatara\optab.as optab.tro
tatara /q tatara\insn.as insn.tro
tatara /q tatara\emit.as emit.tro
tatara /q tatara\objout.as objout.tro

echo === Assembling TANREN modules...
tatara /q tanren\tanren.as tanren.tro
tatara /q tanren\lcmd.as lcmd.tro
tatara /q tanren\lobj.as lobj.tro
tatara /q tanren\lseg.as lseg.tro
tatara /q tanren\lsym.as lsym.tro
tatara /q tanren\lerrs.as lerrs.tro
tatara /q tanren\limg.as limg.tro
tatara /q tanren\arglist.as arglist.tro

echo === Assembling shared modules...
tatara /q shared\msxdos.as msxdos.tro
tatara /q shared\strutil.as strutil.tro
tatara /q shared\alloc.as alloc.tro
tatara /q shared\hash.as hash.tro

echo === Linking TATARA and TANREN binaries
tanren /o:build\tatara.com @tatara.lnk
tanren /o:build\tanren.com @tanren.lnk

echo === Cleaning up
del *.tro
```

これはツリーのルートから実行し、次の 4 つのことを行います。

- **TATARA を shared\にします。**モジュールはそれぞれの .INC ファイルを取り込みます。それらはモジュールの隣にあり、インクルードの探し方の規則 1 で見つかります。モジュールは MSXDOS.INC など SHARED にあるファイルも取り込み、それらは規則 3 によって TATARA を通して見つかります（13.2 節）。このディレクトリは相対パスなのでカレント・ディレクトリから見た位置になり、ビルドはツリーのルートから実行したときにしかうまくいきません（34 章）。
- **27 のモジュールをアセンブルします。**1 つのモジュールに **tatara** の行が 1 つずつあり、アセンブラーの 15 のモジュール、リンカーの 8 つ、両方が使う 4 つの、3 つのグループに分かれています。オブジェクト・ファイルは、どれもツリーのルートに書き出されます。

- **2 つのツールをリンクします。**それぞれをリンク・ファイルから（22 章）、/O: でプログラムを BUILD に書き出すようにしてリンクします。
- **オブジェクト・ファイルを削除します。**

tatara の行には、どれも /Q が付いています。エラーなしでアセンブルできたモジュールは何も表示せず、エラーは/Q の有無にかかわらず表示されます。そのため先頭のコメントにあるように、===の行の間に何かが表示されたら、それは問題があるということです。2 つのリンクには/Q がなく、TANREN の要約が表示されます。

TATARA.LNK はアセンブラーの 19 のオブジェクト・ファイルを指定するもので、22.2 節に示してあります。TANREN.LNK はリンカーの 12 のオブジェクト・ファイルを指定します。

```
; the linker's own modules.

tanren.tro                              ; the driver
lcmd.tro arglist.tro                    ; the command line
lobj.tro                                ; the object reader
lseg.tro lsym.tro limg.tro              ; tables and the image
lerrs.tro

msxdos.tro strutil.tro alloc.tro hash.tro       ; shared
```

どちらのファイルも同じ行で終わっています。共有の 4 つのモジュールは一度だけアセンブルし、両方のツールにリンクします。どちらのファイルもツールのドライバーを最初に指定しているので、そのコードは 0100h に置かれます。MSX-DOS は.COM プログラムを 0100h から実行するからです（21.1 節）。図 35.2 に、27 のモジュールが 2 つのプログラムになる流れを示します。

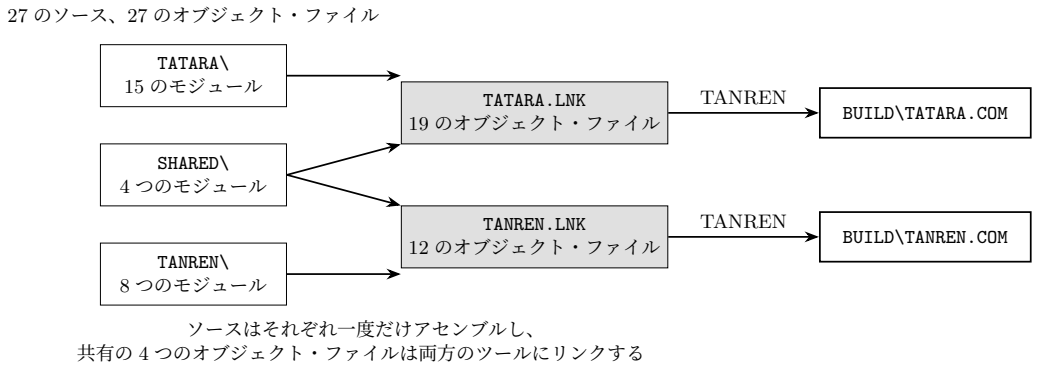

図 35.2: 27 のモジュール、2 つのリンク・ファイル、2 つのプログラム。

> 注. BUILD.BAT は終わるときに TATARA を元に戻さないので、ビルドのあとは shared\ が入ったままです。ほかのものをアセンブルする前に、インクルード用のディレクトリを設定し直してください（3.4 節）。

35.3 実行する

BUILD.BAT は、ツリーのルートから実行します。

MSX-DOS

```
A:\SRC\TATARA>build
=== Assembling TATARA modules...
=== Assembling TANREN modules...
=== Assembling shared modules...
=== Linking TATARA and TANREN binaries
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

19 modules, 1898 records, ends at EOF.
Wrote BUILD\TATARA.COM, 0100-7990 (30865 bytes).
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools

12 modules, 766 records, ends at EOF.
Wrote BUILD\TANREN.COM, 0100-3501 (13314 bytes).
=== Cleaning up
```

アセンブルの===の行の間には何も表示されていないので、27 のモジュールはどれもエラーなしでアセンブルできています。Wrote の行に entry の部分がないのは、どちらのツールのモジュールも END の行にアドレスを書いていないからです。どちらのプログラムも、最初のモジュールのコードから、0100h で始まります（21.4 節）。

ビルドにかかる時間は、マシンと、ファイルを置いたインターフェースやカードによって変わります。目安として、IDE インターフェースの CF カードや SD カードからビルドすると、MSX turbo R（パナソニック FS-A1GT）で約 4 分、MSX2+（ソニー HB-F1XV）で約 23 分かかります。turbo R が速いのは、R800 プロセッサーのためです。

openMSX などのエミュレーターは、エミュレートした MSX を、本物の MSX の速さではなく、動いているコンピューターで可能なかぎりの速さで動かすこともできます。そうするとビルドの時間は大きく短くなり、最近の Mac では約 14 秒です。

新しいプログラムが動くことは、/V で確かめられます。

MSX-DOS

```
A:\SRC\TATARA>build\tatara /v
Tatara MSX Macro-Assembler v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools


A:\SRC\TATARA>build\tanren /v
Tatara MSX Linker v1.2.0
Copyright (C) 2026 Javier Lavandeira
https://tatara.tools


A:\SRC\TATARA>
```

35.4　ツール自身でツールをビルドする

最初のビルドは、インストールされているツールで行いました。それで作ったツールでもう一度ビルドすると、ツールが自分自身をビルドできることがわかります。新しいツールのコピーを専用のディレクトリ STAGE1 に置き、そのディレクトリを PATH の先頭に入れます。こうすると、MSX-DOS はインストールされているものより先に、新しい TATARA.COM と TANREN.COM を見つけます。

MSX-DOS

```
A:\SRC\TATARA>md stage1
A:\SRC\TATARA>copy build\tatara.com stage1
A:\SRC\TATARA>copy build\tanren.com stage1
A:\SRC\TATARA>set path=a:\src\tatara\stage1;%path%
```

PATH には、どのディレクトリからでもツールが見つかるように、ディレクトリを完全なパスで書いています。**%path%**は、PATH にすでに入っていたディレクトリを、新しいディレクトリのあとに残します。

そして BUILD.BAT をもう一度実行します。1 回目と同じ行が表示され、2 つのプログラムが、1 回目のものの上に、もう一度 BUILD に書き出されます（図 35.3）。

2 回目のビルドのプログラムは、STAGE1 にあるものと 1 バイトも違いません。1 回目のビルドで作ったツールが、自分自身をそのまま作り直したわけです。1 回目のビルドの前に BUILD にあったプログラム、つまりリリースのプログラムのコピーを取っておけば、それとも同じです。

MSX-DOS には、2 つのファイルを比べるコマンドがありません。比べるには、ファイルをほかのコンピューターにコピーして、そこで比べてください。

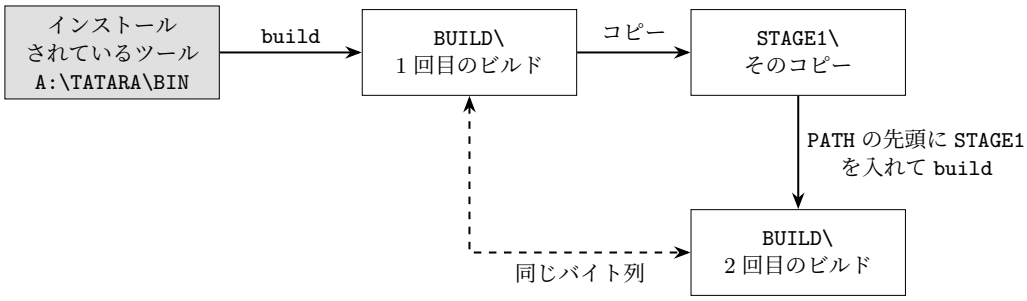

図 35.3: ツールを 2 回ビルドする。2 回目は、1 回目のビルドで作ったツールを使う。

作業が終わったら、PATH から STAGE1 を外すか、新しいセッションを始めて、インストールされているツールが実行されるようにしてください。

付録 A

疑似命令リファレンス

A.1 用途ごとの疑似命令 283
A.2 アルファベット順の疑似命令 286
A.3 疑似命令の行のラベル 292
A.4 別名 293

この付録では、TATARA が受け付けるすべての疑似命令を示します。A.1 節では用途ごとにまとめ、A.2 節ではそれぞれの書き方と短い説明をアルファベット順に示します。A.3 節では疑似命令の行に書いたラベルがどうなるかを、A.4 節では一部の疑似命令が持つ別名を示します。それぞれの項目には、その疑似命令を例とともにくわしく説明している節を示してあります。

書き方の中の斜体の語は、実際に書くものを表し、角かっこの中は省略できます。疑似命令は大文字でも小文字でも書けます。

A.1 用途ごとの疑似命令

A.1.1 データ

疑似命令	働き	節
DB	バイトと文字列を置く	11.1
DW	ワードを下位バイトから置く	11.3
DC	最後の文字のビット 7 を立てた文字列を置く	11.4
DS	何も置かずに場所を確保する	10.4

A.1.2　名前

疑似命令	働き	節
EQU	名前に値を一度だけ与える	12.3
DEFL	あとで変えられる値を名前に与える	12.4
PUBLIC	名前をほかのモジュールに提供する	12.5
EXTRN	ほかのモジュールが定義する名前を使う	12.5

A.1.3　セグメントとロケーション・カウンター

疑似命令	働き	節
CSEG	コード・セグメント、または名前付きのものを選ぶ	10.2, 10.5
DSEG	データ・セグメント、または名前付きのものを選ぶ	10.2, 10.5
ASEG	アブソリュート・セグメントを選ぶ	10.2
GROUP	トランジェント・データ・セグメントのグループを始める	10.6
ORG	ロケーション・カウンターを設定する	10.3

A.1.4　ファイルとソースの終わり

疑似命令	働き	節
INCLUDE	ほかのファイルをその場所で読む	13.1
END	ソースを終わらせ、開始アドレスを示すこともできる	12.6

A.1.5 条件付きアセンブル

疑似命令	働き	節
IF	値が 0 でなければ、その行をアセンブルする	15.1
IFE	値が 0 なら、その行をアセンブルする	15.1
IFDEF	名前が定義されていれば、その行をアセンブルする	15.3
IFNDEF	名前が定義されていなければ、その行をアセンブルする	15.3
IF1	第 1 パスでだけ、その行をアセンブルする	15.4
IF2	第 2 パスでだけ、その行をアセンブルする	15.4
IFB	テキストが空なら、その行をアセンブルする	15.5
IFNB	テキストが空でなければ、その行をアセンブルする	15.5
IFIDN	2 つのテキストが同じなら、その行をアセンブルする	15.5
IFDIF	2 つのテキストが違えば、その行をアセンブルする	15.5
ELSE	条件が偽のときにアセンブルする行を始める	15.1
ENDIF	条件付きアセンブルを終わらせる	15.1

A.1.6 マクロと繰り返しブロック

疑似命令	働き	節
MACRO	マクロの定義を始める	16.1
ENDM	マクロまたは繰り返しブロックを終わらせる	16.1
LOCAL	展開のたびに名前を新しい名前に置き換える	16.5
EXITM	展開を途中で終わらせる	16.6
REPT	ブロックを指定した回数だけ繰り返す	17.1
IRP	リストの項目ごとにブロックを繰り返す	17.2
IRPC	テキストの文字ごとにブロックを繰り返す	17.3

A.1.7　リスティング

疑似命令	働き	節
TITLE	ページ見出しのタイトルを設定する	18.2
SUBTTL	ページ見出しのサブタイトルを設定する	18.2
PAGE	新しいページを始め、ページの長さも設定できる	18.2
.XLIST	リスティングを止める	18.3
.LIST	リスティングを再開する	18.3
.XALL	展開のうちバイトを作る行を表示する	18.4
.LALL	展開のすべての行を表示する	18.4
.SALL	展開の行を表示しない	18.4
.LFCOND	条件付きアセンブルで省かれた行を表示する	18.5
.SFCOND	それらの行をリスティングから省く	18.5
.TFCOND	その 2 つを切り替える	18.5

A.2　アルファベット順の疑似命令

名前がドットで始まる疑似命令は、どれもリスティングを制御するもので、ほかの疑似命令のあとに別のグループとしてまとめてあります（A.2.1 節）。疑似命令の別名もアルファベット順の位置に置き、どの疑似命令と同じかを示しています。

ASEG

aseg
アブソリュート・セグメントを選びます。このセグメントのアドレスは書いたとおりのもので、TANREN はそれを動かしません。セグメントはそれぞれ自分のロケーション・カウンターを持つので、あとの ASEG は前回止まったところから続きます。10.2 節。

COND IF と同じです。

CSEG

cseg [*name*]
コード・セグメントを選びます。名前を書くと、その名前の名前付きコード・セグメントを選びます。ソースはコード・セグメントで始まります。同じ名前

の CSEG はどれも同じセグメントに追加され、TANREN はすべてのモジュールの同じ名前のセグメントをつなげます。10.2 節、10.5 節。

DB

db *item*[,*item*…]
それぞれの項目を置きます。式は 1 バイト（その下位バイト）として置き、値はアブソリュートでなければなりません。文字列は 1 文字を 1 バイトとして置きます。1 文字か 2 文字の文字列のあとに演算子が続くと、文字列ではなく式の中の文字定数になります。11.1 節、11.2 節。

DC

dc *string*
1 つの文字列を DB と同じように置き、最後の文字のビット 7 を立てます。文字列は空であってはなりません。11.4 節。

DEFB DB と同じです。

DEFL

name defl *expression*
名前に値を与えます。その値は、同じ名前のあとの DEFL で変えられます。名前はコロンを付けずに書き、ラベルや EQU の名前と同じにはできません。TATARA には SET がありません。Z80 のコードでは、set は命令だからです。12.4 節。

DEFM DB と同じです。

DEFS DS と同じです。

DEFW DW と同じです。

DS

ds *count*
ロケーション・カウンターを *count* バイト進めて場所を確保し、何も置きません。個数はアブソリュートで、第 1 パスでわかっていなければなりません。2 つ目のオペランドはありません。10.4 節。

DSEG

dseg [*name*[,transient]]
データ・セグメントを選びます。名前を書くと、その名前の名前付きデータ・セグメントを選びます。名前のあとに transient を書くと、そのセグメントはトランジェント・データ・セグメントになり、変数は GROUP でグループに分けます。10.2 節、10.5 節、10.6 節。

DW

dw *expression*[,*expression*…]
それぞれの式をワードとして、下位バイトから置きます。値は、ラベルや外部の名前のような、TANREN が仕上げるアドレスでもかまいません。1 文字か 2 文字の文字列は文字定数になり、それより長い文字列はエラーです。11.3 節。

ELSE

else
条件付きアセンブルのうち、条件が偽のときにアセンブルする行を始めます。まだ閉じていないいちばん内側の条件付きアセンブルに属し、省略できます。15.1 節。

END

end [*address*]
ソースを終わらせます。TATARA はそのあとを何も読みません。このファイルを取り込んだファイルの残りも読みません。アドレスがあれば、それはプログラムの開始アドレスで、TANREN は entry として表示します。END のないソースは、ファイルの終わりで終わります。12.6 節、13.7 節。

ENDIF

endif
まだ閉じていないいちばん内側の条件付きアセンブルを終わらせます。15.1 節。

ENDM

endm
マクロの本体、または REPT、IRP、IRPC のブロックを終わらせます。それぞれの ENDM は、開いているいちばん内側のものに属します。16.1 節、17.1 節。

ENTRY PUBLIC と同じです。

EQU

name equ *expression*
名前に値を与えます。その値は変えられません。名前はコロンを付けずに書きます。コロンを付けると、TATARA はそれをラベルと見なして止まります。12.3 節。

EXITM

exitm
マクロの展開、または繰り返しブロックの繰り返しを、すぐに終わらせます。条件付きアセンブルの中で使い、そこで開いたままの条件付きアセンブルも閉じます。16.6 節、17.4 節。

EXT EXTRN と同じです。

EXTRN

extrn *name*[,*name*…]
ほかのモジュールが定義する名前を宣言します。それを使う値はどれも TANREN が仕上げるので、2 バイトで値を持つ場所にしか書けません。EXTRN で宣言した名前は、同じモジュールでは定義できません。12.5 節。

GROUP

group *name*

トランジェント・データ・セグメントのグループを始めます。セグメントのグループはどれもセグメントの先頭から始まり、セグメントの大きさはいちばん大きいグループの大きさになります。GROUP はトランジェント・データ・セグメントの中でしか使えず、ラベルは付けられません。グループの名前はモジュールの間で共有されます。10.6 節。

IF

if *expression*
値が 0 でなければ、ELSE または ENDIF までの行をアセンブルします。式には、すでに定義されている名前しか使えません。条件付きアセンブルは 16 段まで入れ子にできます。15.1 節、15.2 節、15.6 節。

IF1

if1
第 1 パスでだけ、その行をアセンブルします。中でラベルを定義したりバイトを作ったりすると、2 つのパスが食い違い、エラーになります。15.4 節。

IF2

if2
第 2 パスでだけ、その行をアセンブルします。制限は IF1 と同じです。15.4 節。

IFB

ifb <*text*>
山かっこの中のテキストが空なら、その行をアセンブルします。15.5 節。

IFDEF

ifdef *name*
名前が定義されていれば、その行をアセンブルします。EXTRN で宣言した名前は、定義されているものと見なします。名前はこの行より前で定義してください。そうでないと、2 つのパスで答えが変わります。15.3 節。

IFDIF

ifdif <*text*>,<*text*>
2 つのテキストが違えば、その行をアセンブルします。比較は正確で、大文字と小文字も区別します。15.5 節。

IFE

ife *expression*
値が 0 なら、その行をアセンブルします。そのほかは IF と同じです。15.1 節。

IFF IFE と同じです。

IFIDN

ifidn <*text*>,<*text*>
2 つのテキストが、大文字と小文字も含めて同じなら、その行をアセンブルします。15.5 節。

IFNB

ifnb <*text*>
テキストが空でなければ、その行をアセンブルします。15.5 節。

IFNDEF

ifndef *name*
名前が定義されていなければ、その行をアセンブルします。そのほかは IFDEF と同じです。15.3 節。

IFT IF と同じです。

INCLUDE

include *filename*
そのファイルを、その行が書かれた場所で読みます。行がそこに書かれていたのと同じことになります。名前は書いたとおりに、引用符なしで使われ、何も付け加えられません。TATARA はファイルを、それを取り込むファイルの隣、カレント・ディレクトリ、環境変数 TATARA のそれぞれのディレクトリの順に探します。インクルード・ファイルとマクロの展開は 16 段まで入れ子にできます。13.1 節、13.2 節。

IRP

irp *parameter*,<*item*[,*item*…]>
ENDM までのブロックを、項目ごとに 1 回ずつ、パラメーターをその項目に置き換えてアセンブルします。項目はマクロの引数の規則に従い、山かっこは省略できます。17.2 節。

IRPC

irpc *parameter*,*text*
ENDM までのブロックを、テキストの文字ごとに 1 回ずつアセンブルします。山かっこが必要なのは、テキストに空白を含めるときだけです。17.3 節。

LOCAL

local *name*[,*name*…]
マクロまたは繰り返しブロックの中で、それぞれの名前を、展開のたびに新しい名前（??0000、??0001 など）に置き換えます。LOCAL の行は、本体の最初に書かなければなりません。16.5 節、17.4 節。

MACRO

name macro [*parameter*[,*parameter*…]]
マクロの定義を始めます。本体は、対応する ENDM までです。名前はコロンを付けずに書きます。マクロは使う前に定義しなければならず、定義し直すこともできます。16.1 節〜16.7 節。

ORG

org *expression*

いまのセグメントのロケーション・カウンターを設定します。アブソリュート・セグメントではアドレス、ほかのセグメントでは先頭からのオフセットです。値は第1パスでわかっていなければならず、アブソリュートか、いまのセグメントのアドレスでなければなりません。10.3節。

PAGE

`page [n]`
自分の行のあとで、リスティングの新しいページを始めます。10から255までの *n* を書くと、ページの長さ（ページ見出しを含む）も設定します。18.2節。

PUBLIC

`public name[,name…]`
名前をほかのモジュールに提供します。この行は定義の前にも後にも書けますが、名前はどれもモジュールのどこかで定義しなければなりません。ラベル、EQUの名前、DEFLの名前を公開できます。コロンを2つ付けて書いたラベルも公開されます。12.5節。

REPT

`rept count`
ENDMまでのブロックを *count* 回アセンブルします。回数はアブソリュートで、すでにわかっていなければなりません。回数が0なら、何もアセンブルしません。17.1節。

SUBTTL

`subttl text`
サブタイトルを設定します。サブタイトルは、次のページからページ見出しの下に表示されます。TATARAは最初の28文字を残します。18.2節。

TITLE

`title text`
タイトルを設定します。タイトルはページ見出しの左に表示されます。TATARAは最初の28文字を残します。18.2節。

A.2.1 ドットで始まる疑似命令

.LALL

`.lall`
ここから、マクロの展開や繰り返しブロックのすべての行を、条件付きアセンブルで省かれた行も含めて表示します。18.4節。

.LFCOND

`.lfcond`
条件付きアセンブルで省かれた行を、アドレスなしで表示します。これがデフォルトです。18.5節。

.LIST
.list
.XLIST のあとで、リスティングを再開します。18.3 節。

.SALL
.sall
ここから、展開の行を表示しません。マクロを使う行には+が付きます。18.4 節。

.SFCOND
.sfcond
条件付きアセンブルで省かれた行を、リスティングから省きます。IF、ELSE、ENDIF の行は表示されます。18.5 節。

.TFCOND
.tfcond
.LFCOND から.SFCOND へ、またはその逆に切り替えます。18.5 節。

.XALL
.xall
展開のうち、バイトを作る行を表示します。これがデフォルトです。18.4 節。

.XLIST
.xlist
リスティングを止めます。そのあとの行はふつうにアセンブルされますが、表示されません。18.3 節。

A.3　疑似命令の行のラベル

ラベルは、ほかの行と同じように、たいていの疑似命令の行に書けます（6.3 節）。ラベルは、TATARA がその行に来たときのロケーション・カウンターの値を、行が何かをする前のセグメントで持ちます。そのため、ORG のラベルは新しい値ではなく、ロケーション・カウンターの前の値を持ちます。セグメントを選ぶ疑似命令のラベルは、その行まで使っていたセグメントのアドレスを持ちます。INCLUDE のラベルは、そのファイルのバイトの前のアドレスを持ちます。

いくつかの疑似命令は違います。

- EQU、DEFL、MACRO は、ラベルではなく名前を取ります。名前のあとにコロンを付けると、TATARA は `EQU, DEFL and MACRO take a name, not a label.` で止まります。
- GROUP にはラベルを付けられません。ラベルを付けると、TATARA は `GROUP needs a name, no label, and a transient DSEG.` で止まります。

- IF、ELSE、ENDIF のラベルは、TATARA がその行に来たときにアセンブルしている場合にだけ定義されます（15.6 節）。
- REPT、IRP、IRPC のラベルは、最初の繰り返しの始まりのアドレスを持ちます。
- EXITM のラベルは、展開がそこまで進んだアドレスを持ちます。
- マクロの中の LOCAL と ENDM のラベルは、メッセージなしで無視されます。TATARA はこの 2 つの行をマクロの本体を保存するときに読み、アセンブルすることはないので、ラベルが持つロケーション・カウンターがないからです。M80 も同じです。

ラベルをいちばん安全に置けるのは、ラベルだけの行か、バイトを作る行です。

A.4 別名

一部の疑似命令には別名があります。TATARA は、M80 向けに書かれたソースがそのままアセンブルできるように、これらを受け付けます（*EJECT と $EJECT は 1 桁目に書いた場合だけ）。

別名	同じもの
COND, IFT	IF
DEFB, DEFM	DB
DEFS	DS
DEFW	DW
ENTRY	PUBLIC
EXT	EXTRN
IFF	IFE
*EJECT, $EJECT	PAGE

TATARA が受け付けない M80 の疑似命令は、付録 J に示してあります。

付録 B

命令一覧

B.1 表の読み方 296
B.2 8 ビットのロード 297
B.3 16 ビットのロード、PUSH と POP 298
B.4 交換、ブロック転送、サーチ 299
B.5 8 ビットの算術演算と論理演算 300
B.6 汎用と CPU 制御 301
B.7 16 ビットの算術演算 302
B.8 ローテートとシフト 303
B.9 ビットのセット、リセット、テスト 305
B.10 ジャンプ 305
B.11 コール、リターン、リスタート 306
B.12 入出力 307
B.13 未公開命令 308
B.14 R800 のかけ算 309
B.15 受け付けない書き方 309

この付録では、TATARA が受け付けるすべての命令の書き方を、ザイログのマニュアルと同じ分類でまとめ、TATARA がアセンブルするバイトと、命令の働きを簡単に示します。オペランドの書き方は 9 章で説明しています。それぞれの命令がくわしく何をするか、どのフラグを変えるか、どれだけ時間がかかるかは、Z80 のプログラミングを学ぶ本や講座で確かめてください。

TATARA は、公開されている Z80 の命令、9.5 節の未公開命令、R800 の 2 つのかけ算の命令を受け付けます。

B.1　表の読み方

表はどれも 3 つの列からなります。命令の書き方、TATARA がそれに対して書き出すバイト（16 進数）、そして命令の働きです。最初の 2 つの列で斜体になっている語は、使うたびに変わるものを表します。

語	表すもの
r, r′	レジスター B、C、D、E、H、L、A のどれか
n	1 バイト。どんな式でもよく、その下位バイトが使われる
nn	1 ワード。アドレスか 16 ビットの値
lo hi	*nn* の 2 バイト。下位バイトが先
d	インデックス付きオペランドのディスプレースメント。−128 から 127 まで
e	相対ジャンプの飛び先。TATARA が書き出すバイトは、命令の終わりから飛び先までの距離で、−128 から 127 まで
b	ビット番号。0 から 7 まで
x, x′	インデックス・ハーフ（B.13 節）

命令の中には、使うレジスターによってバイトが変わるものがあります。あるレジスターを別のレジスターにコピーする `ld r,r'` には、`ld b,b` から `ld a,a` まで 49 通りあります。表では、それぞれに 1 行ずつ使う代わりに、バイトを求める規則を示します。レジスターにはそれぞれ番号があり、これを**レジスター・コード**といいます。

B	C	D	E	H	L	(HL)	A
0	1	2	3	4	5	6	7

バイトの列で、`8r` は *r* のコードの 8 倍を表し、数は 16 進数で足します。`ld r,r'` の行には `40+8r+r'` とあります。`ld d,b` では、D のコードが 2、B のコードが 0 なので、バイトは 40h + 8×2 + 0 = 40h + 10h = **50h** です。同じように、`add a,r` は `80+r` なので、`add a,e` は 80h + 3 = 83h です。`bit b,r` は `CB 40+8b+r` なので、`bit 7,a` は `CB` のあとに 40h + 8×7 + 7 = 40h + 38h + 7 = 7Fh が続きます。

`push` のレジスター・ペアや `jp` の条件のように、数通りしかない命令は、それぞれに 1 行ずつ使い、バイトをそのまま示します。

IX や IY を使う命令は、HL を使う命令の前にプレフィックスのバイトを付けたものです。IX には DD、IY には FD を付けます。*d* のある書き方では、ディスプレースメントのバイトはオペコードのあとに続きます。DD CB や FD CB で始まるものでは、最後のバイトの前に入ります。

B.2 8 ビットのロード

書き方	バイト	働き
ld *r*,*r'*	40+8*r*+*r'*	*r′* を *r* にコピーする
ld *r*,*n*	06+8*r* *n*	*n* を *r* にロードする
ld *r*,(hl)	46+8*r*	HL の示すアドレスのバイトを *r* にロードする
ld *r*,(ix+*d*)	DD 46+8*r* *d*	IX+*d* のバイトを *r* にロードする
ld *r*,(iy+*d*)	FD 46+8*r* *d*	IY+*d* のバイトを *r* にロードする
ld (hl),*r*	70+*r*	*r* を HL の示すアドレスに書き込む
ld (ix+*d*),*r*	DD 70+*r* *d*	*r* を IX+*d* に書き込む
ld (iy+*d*),*r*	FD 70+*r* *d*	*r* を IY+*d* に書き込む
ld (hl),*n*	36 *n*	*n* を HL の示すアドレスに書き込む
ld (ix+*d*),*n*	DD 36 *d* *n*	*n* を IX+*d* に書き込む
ld (iy+*d*),*n*	FD 36 *d* *n*	*n* を IY+*d* に書き込む
ld a,(bc)	0A	BC の示すアドレスから A をロードする
ld a,(de)	1A	DE の示すアドレスから A をロードする
ld a,(*nn*)	3A *lo* *hi*	アドレス *nn* から A をロードする
ld (bc),a	02	A を BC の示すアドレスに書き込む
ld (de),a	12	A を DE の示すアドレスに書き込む
ld (*nn*),a	32 *lo* *hi*	A をアドレス *nn* に書き込む
ld a,i	ED 57	割り込みベクター・レジスターを A にコピーする
ld a,r	ED 5F	リフレッシュ・レジスターを A にコピーする
ld i,a	ED 47	A を割り込みベクター・レジスターにコピーする
ld r,a	ED 4F	A をリフレッシュ・レジスターにコピーする

B.3　16 ビットのロード、PUSH と POP

書き方	バイト	働き
`ld bc,`*nn*	`01` *lo hi*	*nn* を BC にロードする
`ld de,`*nn*	`11` *lo hi*	*nn* を DE にロードする
`ld hl,`*nn*	`21` *lo hi*	*nn* を HL にロードする
`ld sp,`*nn*	`31` *lo hi*	*nn* を SP にロードする
`ld ix,`*nn*	`DD 21` *lo hi*	*nn* を IX にロードする
`ld iy,`*nn*	`FD 21` *lo hi*	*nn* を IY にロードする
`ld bc,(`*nn*`)`	`ED 4B` *lo hi*	アドレス *nn* から BC をロードする
`ld de,(`*nn*`)`	`ED 5B` *lo hi*	アドレス *nn* から DE をロードする
`ld hl,(`*nn*`)`	`2A` *lo hi*	アドレス *nn* から HL をロードする
`ld sp,(`*nn*`)`	`ED 7B` *lo hi*	アドレス *nn* から SP をロードする
`ld ix,(`*nn*`)`	`DD 2A` *lo hi*	アドレス *nn* から IX をロードする
`ld iy,(`*nn*`)`	`FD 2A` *lo hi*	アドレス *nn* から IY をロードする
`ld (`*nn*`),bc`	`ED 43` *lo hi*	BC をアドレス *nn* に書き込む
`ld (`*nn*`),de`	`ED 53` *lo hi*	DE をアドレス *nn* に書き込む
`ld (`*nn*`),hl`	`22` *lo hi*	HL をアドレス *nn* に書き込む
`ld (`*nn*`),sp`	`ED 73` *lo hi*	SP をアドレス *nn* に書き込む
`ld (`*nn*`),ix`	`DD 22` *lo hi*	IX をアドレス *nn* に書き込む
`ld (`*nn*`),iy`	`FD 22` *lo hi*	IY をアドレス *nn* に書き込む
`ld sp,hl`	`F9`	HL を SP にコピーする
`ld sp,ix`	`DD F9`	IX を SP にコピーする
`ld sp,iy`	`FD F9`	IY を SP にコピーする
`push bc`	`C5`	BC をスタックに積む
`push de`	`D5`	DE をスタックに積む
`push hl`	`E5`	HL をスタックに積む
`push af`	`F5`	AF をスタックに積む
`push ix`	`DD E5`	IX をスタックに積む
`push iy`	`FD E5`	IY をスタックに積む
`pop bc`	`C1`	スタックから BC を取り出す
`pop de`	`D1`	スタックから DE を取り出す
`pop hl`	`E1`	スタックから HL を取り出す

書き方	バイト	働き
pop af	F1	スタックから AF を取り出す
pop ix	DD E1	スタックから IX を取り出す
pop iy	FD E1	スタックから IY を取り出す

B.4 交換、ブロック転送、サーチ

書き方	バイト	働き
ex de,hl	EB	DE と HL を交換する
ex af,af'	08	AF と裏の AF を交換する
exx	D9	BC、DE、HL を裏のレジスターと交換する
ex (sp),hl	E3	HL とスタックの先頭のワードを交換する
ex (sp),ix	DD E3	IX とスタックの先頭のワードを交換する
ex (sp),iy	FD E3	IY とスタックの先頭のワードを交換する
ldi	ED A0	(HL) から (DE) へ 1 バイトコピーし、先へ進む
ldir	ED B0	BC が 0 になるまで LDI を繰り返す
ldd	ED A8	(HL) から (DE) へ 1 バイトコピーし、後ろへ戻る
lddr	ED B8	BC が 0 になるまで LDD を繰り返す
cpi	ED A1	A と (HL) を比べ、先へ進む
cpir	ED B1	一致するか BC が 0 になるまで CPI を繰り返す
cpd	ED A9	A と (HL) を比べ、後ろへ戻る
cpdr	ED B9	一致するか BC が 0 になるまで CPD を繰り返す

B.5　8 ビットの算術演算と論理演算

書き方	バイト	働き
add a,*r*	80+*r*	*r* を A に加える
add a,*n*	C6 *n*	*n* を A に加える
add a,(hl)	86	(HL) のバイトを A に加える
add a,(ix+*d*)	DD 86 *d*	IX+*d* のバイトを A に加える
add a,(iy+*d*)	FD 86 *d*	IY+*d* のバイトを A に加える
adc a,*r*	88+*r*	*r* とキャリーを A に加える
adc a,*n*	CE *n*	*n* とキャリーを A に加える
adc a,(hl)	8E	(HL) のバイトとキャリーを A に加える
adc a,(ix+*d*)	DD 8E *d*	IX+*d* のバイトとキャリーを A に加える
adc a,(iy+*d*)	FD 8E *d*	IY+*d* のバイトとキャリーを A に加える
sub *r*	90+*r*	A から *r* を引く
sub *n*	D6 *n*	A から *n* を引く
sub (hl)	96	A から (HL) のバイトを引く
sub (ix+*d*)	DD 96 *d*	A から IX+*d* のバイトを引く
sub (iy+*d*)	FD 96 *d*	A から IY+*d* のバイトを引く
sbc a,*r*	98+*r*	A から *r* とキャリーを引く
sbc a,*n*	DE *n*	A から *n* とキャリーを引く
sbc a,(hl)	9E	A から (HL) のバイトとキャリーを引く
sbc a,(ix+*d*)	DD 9E *d*	A から IX+*d* のバイトとキャリーを引く
sbc a,(iy+*d*)	FD 9E *d*	A から IY+*d* のバイトとキャリーを引く
and *r*	A0+*r*	A と *r* の AND
and *n*	E6 *n*	A と *n* の AND
and (hl)	A6	A と (HL) のバイトの AND
and (ix+*d*)	DD A6 *d*	A と IX+*d* のバイトの AND
and (iy+*d*)	FD A6 *d*	A と IY+*d* のバイトの AND
xor *r*	A8+*r*	A と *r* の XOR
xor *n*	EE *n*	A と *n* の XOR
xor (hl)	AE	A と (HL) のバイトの XOR
xor (ix+*d*)	DD AE *d*	A と IX+*d* のバイトの XOR
xor (iy+*d*)	FD AE *d*	A と IY+*d* のバイトの XOR

書き方	バイト	働き
`or r`	`B0+r`	A と *r* の OR
`or n`	`F6 n`	A と *n* の OR
`or (hl)`	`B6`	A と (HL) のバイトの OR
`or (ix+d)`	`DD B6 d`	A と IX+*d* のバイトの OR
`or (iy+d)`	`FD B6 d`	A と IY+*d* のバイトの OR
`cp r`	`B8+r`	A と *r* を比べる
`cp n`	`FE n`	A と *n* を比べる
`cp (hl)`	`BE`	A と (HL) のバイトを比べる
`cp (ix+d)`	`DD BE d`	A と IX+*d* のバイトを比べる
`cp (iy+d)`	`FD BE d`	A と IY+*d* のバイトを比べる
`inc r`	`04+8r`	*r* に 1 を加える
`inc (hl)`	`34`	(HL) のバイトに 1 を加える
`inc (ix+d)`	`DD 34 d`	IX+*d* のバイトに 1 を加える
`inc (iy+d)`	`FD 34 d`	IY+*d* のバイトに 1 を加える
`dec r`	`05+8r`	*r* から 1 を引く
`dec (hl)`	`35`	(HL) のバイトから 1 を引く
`dec (ix+d)`	`DD 35 d`	IX+*d* のバイトから 1 を引く
`dec (iy+d)`	`FD 35 d`	IY+*d* のバイトから 1 を引く

B.6 汎用と CPU 制御

書き方	バイト	働き
`daa`	`27`	BCD の加算や減算のあとで A を補正する
`cpl`	`2F`	A のすべてのビットを反転する
`neg`	`ED 44`	A の符号を反転する
`ccf`	`3F`	キャリー・フラグを反転する
`scf`	`37`	キャリー・フラグをセットする
`nop`	`00`	何もしない
`halt`	`76`	割り込みを待つ

書き方	バイト	働き
di	F3	割り込みを禁止する
ei	FB	割り込みを許可する
im 0	ED 46	割り込みモード 0 にする
im 1	ED 56	割り込みモード 1 にする
im 2	ED 5E	割り込みモード 2 にする

B.7　16 ビットの算術演算

書き方	バイト	働き
add hl,bc	09	BC を HL に加える
add hl,de	19	DE を HL に加える
add hl,hl	29	HL を HL に加える
add hl,sp	39	SP を HL に加える
adc hl,bc	ED 4A	BC とキャリーを HL に加える
adc hl,de	ED 5A	DE とキャリーを HL に加える
adc hl,hl	ED 6A	HL とキャリーを HL に加える
adc hl,sp	ED 7A	SP とキャリーを HL に加える
sbc hl,bc	ED 42	HL から BC とキャリーを引く
sbc hl,de	ED 52	HL から DE とキャリーを引く
sbc hl,hl	ED 62	HL から HL とキャリーを引く
sbc hl,sp	ED 72	HL から SP とキャリーを引く
add ix,bc	DD 09	BC を IX に加える
add ix,de	DD 19	DE を IX に加える
add ix,ix	DD 29	IX を IX に加える
add ix,sp	DD 39	SP を IX に加える
add iy,bc	FD 09	BC を IY に加える
add iy,de	FD 19	DE を IY に加える
add iy,iy	FD 29	IY を IY に加える
add iy,sp	FD 39	SP を IY に加える

書き方	バイト	働き
inc bc	03	BC に 1 を加える
inc de	13	DE に 1 を加える
inc hl	23	HL に 1 を加える
inc sp	33	SP に 1 を加える
inc ix	DD 23	IX に 1 を加える
inc iy	FD 23	IY に 1 を加える
dec bc	0B	BC から 1 を引く
dec de	1B	DE から 1 を引く
dec hl	2B	HL から 1 を引く
dec sp	3B	SP から 1 を引く
dec ix	DD 2B	IX から 1 を引く
dec iy	FD 2B	IY から 1 を引く

B.8 ローテートとシフト

書き方	バイト	働き
rlca	07	A を左に回転する
rla	17	A をキャリーを通して左に回転する
rrca	0F	A を右に回転する
rra	1F	A をキャリーを通して右に回転する
rlc *r*	CB 00+*r*	*r* を左に回転する
rlc (hl)	CB 06	(HL) のバイトを左に回転する
rlc (ix+*d*)	DD CB *d* 06	IX+*d* のバイトを左に回転する
rlc (iy+*d*)	FD CB *d* 06	IY+*d* のバイトを左に回転する
rl *r*	CB 10+*r*	*r* をキャリーを通して左に回転する
rl (hl)	CB 16	(HL) のバイトをキャリーを通して左に回転する
rl (ix+*d*)	DD CB *d* 16	IX+*d* のバイトをキャリーを通して左に回転する

書き方	バイト	働き
rl (iy+*d*)	FD CB *d* 16	IY+*d* のバイトをキャリーを通して左に回転する
rrc *r*	CB 08+*r*	*r* を右に回転する
rrc (hl)	CB 0E	(HL) のバイトを右に回転する
rrc (ix+*d*)	DD CB *d* 0E	IX+*d* のバイトを右に回転する
rrc (iy+*d*)	FD CB *d* 0E	IY+*d* のバイトを右に回転する
rr *r*	CB 18+*r*	*r* をキャリーを通して右に回転する
rr (hl)	CB 1E	(HL) のバイトをキャリーを通して右に回転する
rr (ix+*d*)	DD CB *d* 1E	IX+*d* のバイトをキャリーを通して右に回転する
rr (iy+*d*)	FD CB *d* 1E	IY+*d* のバイトをキャリーを通して右に回転する
sla *r*	CB 20+*r*	*r* を左にシフトする
sla (hl)	CB 26	(HL) のバイトを左にシフトする
sla (ix+*d*)	DD CB *d* 26	IX+*d* のバイトを左にシフトする
sla (iy+*d*)	FD CB *d* 26	IY+*d* のバイトを左にシフトする
sra *r*	CB 28+*r*	*r* をビット 7 を保って右にシフトする
sra (hl)	CB 2E	(HL) のバイトをビット 7 を保って右にシフトする
sra (ix+*d*)	DD CB *d* 2E	IX+*d* のバイトをビット 7 を保って右にシフトする
sra (iy+*d*)	FD CB *d* 2E	IY+*d* のバイトをビット 7 を保って右にシフトする
srl *r*	CB 38+*r*	*r* をビット 7 に 0 を入れて右にシフトする
srl (hl)	CB 3E	(HL) のバイトをビット 7 に 0 を入れて右にシフトする
srl (ix+*d*)	DD CB *d* 3E	IX+*d* のバイトをビット 7 に 0 を入れて右にシフトする
srl (iy+*d*)	FD CB *d* 3E	IY+*d* のバイトをビット 7 に 0 を入れて右にシフトする
rld	ED 6F	A と (HL) の間で数字を左に回転する
rrd	ED 67	A と (HL) の間で数字を右に回転する

B.9 ビットのセット、リセット、テスト

書き方	バイト	働き
`bit b,r`	`CB 40+8b+r`	*r* のビット *b* を調べる
`bit b,(hl)`	`CB 46+8b`	(HL) のバイトのビット *b* を調べる
`bit b,(ix+d)`	`DD CB d 46+8b`	IX+*d* のバイトのビット *b* を調べる
`bit b,(iy+d)`	`FD CB d 46+8b`	IY+*d* のバイトのビット *b* を調べる
`set b,r`	`CB C0+8b+r`	*r* のビット *b* をセットする
`set b,(hl)`	`CB C6+8b`	(HL) のバイトのビット *b* をセットする
`set b,(ix+d)`	`DD CB d C6+8b`	IX+*d* のバイトのビット *b* をセットする
`set b,(iy+d)`	`FD CB d C6+8b`	IY+*d* のバイトのビット *b* をセットする
`res b,r`	`CB 80+8b+r`	*r* のビット *b* をクリアする
`res b,(hl)`	`CB 86+8b`	(HL) のバイトのビット *b* をクリアする
`res b,(ix+d)`	`DD CB d 86+8b`	IX+*d* のバイトのビット *b* をクリアする
`res b,(iy+d)`	`FD CB d 86+8b`	IY+*d* のバイトのビット *b* をクリアする

B.10 ジャンプ

書き方	バイト	働き
`jp nn`	`C3 lo hi`	*nn* へジャンプする
`jp nz,nn`	`C2 lo hi`	ゼロでなければ *nn* へジャンプする
`jp z,nn`	`CA lo hi`	ゼロなら *nn* へジャンプする
`jp nc,nn`	`D2 lo hi`	キャリーがなければ *nn* へジャンプする
`jp c,nn`	`DA lo hi`	キャリーがあれば *nn* へジャンプする
`jp po,nn`	`E2 lo hi`	パリティが奇数なら *nn* へジャンプする
`jp pe,nn`	`EA lo hi`	パリティが偶数なら *nn* へジャンプする
`jp p,nn`	`F2 lo hi`	正なら *nn* へジャンプする
`jp m,nn`	`FA lo hi`	負なら *nn* へジャンプする
`jr e`	`18 e`	*e* へジャンプする
`jr nz,e`	`20 e`	ゼロでなければ *e* へジャンプする

書き方	バイト	働き
`jr z,`*e*	`28` *e*	ゼロなら *e* へジャンプする
`jr nc,`*e*	`30` *e*	キャリーがなければ *e* へジャンプする
`jr c,`*e*	`38` *e*	キャリーがあれば *e* へジャンプする
`jp (hl)`	`E9`	HL の示すアドレスへジャンプする
`jp (ix)`	`DD E9`	IX の示すアドレスへジャンプする
`jp (iy)`	`FD E9`	IY の示すアドレスへジャンプする
`djnz` *e*	`10` *e*	B から 1 を引き、0 でなければ *e* へジャンプする

B.11　コール、リターン、リスタート

書き方	バイト	働き
`call` *nn*	`CD` *lo hi*	*nn* のルーチンを呼び出す
`call nz,`*nn*	`C4` *lo hi*	ゼロでなければ *nn* のルーチンを呼び出す
`call z,`*nn*	`CC` *lo hi*	ゼロなら *nn* のルーチンを呼び出す
`call nc,`*nn*	`D4` *lo hi*	キャリーがなければ *nn* のルーチンを呼び出す
`call c,`*nn*	`DC` *lo hi*	キャリーがあれば *nn* のルーチンを呼び出す
`call po,`*nn*	`E4` *lo hi*	パリティが奇数なら *nn* のルーチンを呼び出す
`call pe,`*nn*	`EC` *lo hi*	パリティが偶数なら *nn* のルーチンを呼び出す
`call p,`*nn*	`F4` *lo hi*	正なら *nn* のルーチンを呼び出す
`call m,`*nn*	`FC` *lo hi*	負なら *nn* のルーチンを呼び出す
`ret`	`C9`	ルーチンから戻る
`ret nz`	`C0`	ゼロでなければ戻る
`ret z`	`C8`	ゼロなら戻る
`ret nc`	`D0`	キャリーがなければ戻る
`ret c`	`D8`	キャリーがあれば戻る

書き方	バイト	働き
`ret po`	`E0`	パリティが奇数なら戻る
`ret pe`	`E8`	パリティが偶数なら戻る
`ret p`	`F0`	正なら戻る
`ret m`	`F8`	負なら戻る
`reti`	`ED 4D`	割り込みルーチンから戻る
`retn`	`ED 45`	マスク不可能割り込みのルーチンから戻る
`rst 00h`	`C7`	00h のルーチンを呼び出す
`rst 08h`	`CF`	08h のルーチンを呼び出す
`rst 10h`	`D7`	10h のルーチンを呼び出す
`rst 18h`	`DF`	18h のルーチンを呼び出す
`rst 20h`	`E7`	20h のルーチンを呼び出す
`rst 28h`	`EF`	28h のルーチンを呼び出す
`rst 30h`	`F7`	30h のルーチンを呼び出す
`rst 38h`	`FF`	38h のルーチンを呼び出す

B.12 入出力

書き方	バイト	働き
`in a,(`*n*`)`	`DB` *n*	ポート *n* から A へ読み込む
`in` *r*`,(c)`	`ED 40+8`*r*	C のポートから *r* へ読み込む
`ini`	`ED A2`	C のポートから (HL) へ読み込み、先へ進む
`inir`	`ED B2`	B が 0 になるまで INI を繰り返す
`ind`	`ED AA`	C のポートから (HL) へ読み込み、後ろへ戻る
`indr`	`ED BA`	B が 0 になるまで IND を繰り返す
`out (`*n*`),a`	`D3` *n*	A をポート *n* へ書き出す
`out (c),`*r*	`ED 41+8`*r*	*r* を C のポートへ書き出す
`outi`	`ED A3`	(HL) を C のポートへ書き出し、先へ進む

書き方	バイト	働き
otir	ED B3	B が 0 になるまで OUTI を繰り返す
outd	ED AB	(HL) を C のポートへ書き出し、後ろへ戻る
otdr	ED BB	B が 0 になるまで OUTD を繰り返す

B.13　未公開命令

9.5 節の未公開命令です。SLL の行では、*r* は 7 つのレジスターのどれでもかまいません。ほかの行では、*x* と *x′* は IXH か IXL で、そのコードは H と L のコードと同じ 4 と 5 です。*r* は B、C、D、E、A のどれかで、*x′* は *x* と同じレジスターの半分です。表は IX のバイトを示します。IYH と IYL では、DD の代わりに FD を使います。ld iyl,a は FD 6F です。

書き方	バイト	働き
sll *r*	CB 30+*r*	*r* を左へシフトし、ビット 0 を 1 にする
sll (hl)	CB 36	HL のアドレスのバイトに同じことをする
sll (ix+*d*)	DD CB *d* 36	IX+*d* のバイトに同じことをする
sll (iy+*d*)	FD CB *d* 36	IY+*d* のバイトに同じことをする
ld *x*,*r*	DD 40+8*x*+*r*	*r* を *x* にコピーする
ld *r*,*x*	DD 40+8*r*+*x*	*x* を *r* にコピーする
ld *x*,*x'*	DD 40+8*x*+*x'*	IX の一方の半分を他方か、同じ半分にコピーする
ld *x*,*n*	DD 06+8*x* *n*	*n* を *x* に入れる
inc *x*	DD 04+8*x*	*x* に 1 を足す
dec *x*	DD 05+8*x*	*x* から 1 を引く
add a,*x*	DD 80+*x*	A に *x* を足す
adc a,*x*	DD 88+*x*	A に *x* とキャリーを足す
sub *x*	DD 90+*x*	A から *x* を引く
sbc a,*x*	DD 98+*x*	A から *x* とキャリーを引く
and *x*	DD A0+*x*	A と *x* の AND
xor *x*	DD A8+*x*	A と *x* の XOR

書き方	バイト	働き
`or` *x*	`DD B0+`*x*	A と *x* の OR
`cp` *x*	`DD B8+`*x*	A と *x* を比べる
`in f,(c)`	`ED 70`	C のポートを読んでフラグを設定する。A は変わらない
`out (c),0`	`ED 71`	C のポートに 0 か 0FFh を書く（9.5 節）

B.14 R800 のかけ算

書き方	バイト	働き
`mulub a,`*r*	`ED C1+8`*r*	A×*r*、結果は HL
`muluw hl,bc`	`ED C3`	HL×BC、結果は DE:HL
`muluw hl,sp`	`ED F3`	HL×SP、結果は DE:HL

B.15 受け付けない書き方

ほかのアセンブラーでは受け付けられたり、うまくいきそうに見えたりしても、ザイログの一覧にも、TATARA が受け付ける未公開命令にもない書き方があります。TATARA はそれらを受け付けません（表 B.14）。

`SUB`、`AND`、`OR`、`XOR`、`CP` はいつも A に対して働くので、ザイログは A を書きません。`ADD`、`ADC`、`SBC` には 16 ビットの書き方もあるので、8 ビットの書き方では A を書きます。かっこがないと、`jp hl` は `jp nn` と読まれ、`hl` はアドレスの名前と見なされますが、その名前は定義されていません。

インデックス・ハーフを使う書き方が書けないのは、1 つのプレフィックスが命令全体にかかるからです（9.5 節）。`rlc (ix+2),b` は、結果をメモリとレジスターの両方に入れる未公開の書き方の 1 つで、TATARA は受け付けません。`in (c)`、`sl1`、`hx` は、TATARA では 1 通りにしか書かない命令の、ほかのアセンブラーでの綴りです。

ディスプレースメントのない `(IX)` と `(IY)` は受け付けられ、`(IX+0)` と `(IY+0)` の意味になります。`ld a,(ix)` は `DD 7E 00` です。

書いたもの	メッセージ	代わりに書くもの
`sub a,b`	`not a form this instruction has.`	`sub b`
`and a,b`	`not a form this instruction has.`	`and b`
`cp a,b`	`not a form this instruction has.`	`cp b`
`add b`	`not a form this instruction has.`	`add a,b`
`ex af,af`	`not a form this instruction has.`	`ex af,af'`
`jp hl`	`undefined symbol in an expression.`	`jp (hl)`
`ld h,ixl`	`not a form this instruction has.`	なし
`ld ixh,iyl`	`not a form this instruction has.`	なし
`ld ixh,(hl)`	`not a form this instruction has.`	なし
`rlc ixh`	`not a form this instruction has.`	なし
`rlc (ix+2),b`	`not a form this instruction has.`	`rlc (ix+2)` と `ld b,(ix+2)`
`in (c)`	`not a form this instruction has.`	`in f,(c)`
`sl1 b`	`not a directive, a macro or an instruction.`	`sll b`
`ld a,hx`	`undefined symbol in an expression.`	`ld a,ixh`

表 B.14: TATARA が受け付けない書き方。

付録C

演算子と優先順位

C.1 演算子 311
C.2 範囲の端 313
C.3 値 313

この付録では、8章で説明した式についてのことを、参照しやすいようにいくつかの表にまとめます。演算子とその優先順位、16ビットの範囲の端での結果、そして式を作る値です。

C.1 演算子

表C.1に、すべての演算子を優先順位のレベルごとに示します。レベル1がいちばん強く結び付きます。演算子が2つ以上ある式では、レベルの数が小さい演算子から先に計算されます。

どの演算子にも、次の規則があります（8.5節）。

- 同じレベルの演算子は左から右へ計算されます。`10-4-3`は`(10-4)-3`、`1 shl 3 * 2`は10h、`1 or 2 xor 3`は`(1 or 2) xor 3`で0です。
- かっこで順序を変えられます。`2+3*4`は14、`(2+3)*4`は20です。
- `HIGH`と`LOW`は、すぐあとの値にだけ働きます。`high 1234h+1`は`(high 1234h)+1`で、13hです。
- 符号反転は`*`、`/`、`MOD`より弱く結び付きます。`-2*3`は`-(2*3)`、`-3/2`は`-(3/2)`で、−1です。
- `NOT`は比較より弱く結び付きます。`not 1 eq 1`は`not (1 eq 1)`で、0です。
- `AND`は`OR`や`XOR`より強く結び付きます。`2 or 1 and 0`は`2 or (1 and 0)`で、2です。

レベル	演算子	働き	例と値
1	HIGH	上位バイト	high 1234h = 12h
	LOW	下位バイト	low 1234h = 34h
2	*	乗算	6*7 = 42
	/	除算（符号なし）	17/5 = 3
	MOD	除算の余り（符号なし）	17 mod 5 = 2
	SHL	左シフト	1 shl 4 = 10h
	SHR	右シフト	8000h shr 15 = 1
3	-	符号反転（値の前）	-1 = 0FFFFh
4	+	加算	2+3 = 5
	-	減算	10-4-3 = 3
5	EQ NE	等しい、等しくない	1 eq 1 = 0FFFFh
	LT LE	より小さい、以下	2 le 1 = 0
	GT GE	より大きい、以上	2 gt 1 = 0FFFFh
6	NOT	すべてのビットを反転	not 0 = 0FFFFh
7	AND	ビットごとの論理積	0f0h and 3ch = 30h
8	OR	ビットごとの論理和	0f0h or 0fh = 0FFh
	XOR	ビットごとの排他的論理和	0ffh xor 0fh = 0F0h

表 C.1: 優先順位のレベルごとの演算子（結び付きの強い順）

- 比較の演算子は、真なら 0FFFFh、偽なら 0 を返し、符号なしの数として比べます。-1 gt 1 は真です。
- 語の演算子は大文字でも小文字でも書けますが、両側に空白かかっこが必要です。aandb ではなく a and b と書きます。

これは M80 のレベルと同じです。迷ったら、かっこを付けてください。かっこを付けても損はありません。

C.2 範囲の端

値はどれも 0 から 65535 までの 16 ビットの数で、負の数は 16 ビットの補数として保持されます（8.1 節）。表 C.2 に、その範囲の端で演算子が返す値を示します。

式	値	理由
`65535+1`	0	値が一周する
`0-1`	0FFFFh	値が一周する
`17/0`	0FFFFh	0 で割ってもエラーにならない
`17 mod 0`	0	その余りもエラーにならない
`1 shl 16`	0	すべてのビットが押し出される
`1 shl 0ffffh`	0	同じ
`(-3)/2`	7FFEh	−3 は 0FFFDh で、除算は符号なし
`(-7) mod 2`	1	−7 は 0FFF9h で、余りは符号なし
`-3/2`	0FFFFh	`-(3/2)` のこと。符号反転は/のあと
`high -1`	0FFh	−1 は 0FFFFh

表 C.2: 16 ビットの範囲の端での結果

`ld a,n` や `DB` のオペランドのように、行が 1 バイトしか必要としない場所では、TATARA は値の下位バイトを使い、収まるかどうかは確かめません。

C.3 値

C.3.1 数、文字、ロケーション・カウンター

数は、あとに別の基数を示す文字がなければ 10 進数です（8.2 節）。数は数字で始まらなければならないので、文字で始まる 16 進数には前に `0` を付けます。

基数	接尾辞	例	値
10 進数	なし、または `D`	`100`、`100d`	100
16 進数	`H`	`100h`、`0ffh`	256、255
8 進数	`O` または `Q`	`100o`、`100q`	64
2 進数	`B`	`100b`	4

B と D は 16 進数の数字でもあり接尾辞でもあるので、16 進数は必ず H で終えてください。10b は 2 です。

文字定数は、アポストロフィか引用符で囲んだ 1 文字か 2 文字で、その値は文字のコードです。'A' は 41h、"AB"は 4142h で、最初の文字が上位バイトに入ります（8.3 節）。

$だけを書くと、ロケーション・カウンター、つまりそれを書いた行のいまのセグメントでのアドレスになります（8.4 節）。

C.3.2　値の種類

式は、**アブソリュート**、**リロケータブル**、**外部**のどれかです（8.6 節）。アブソリュートな値は TATARA がアセンブルするときに完成しています。リロケータブルな値はセグメントの中のアドレスで、TANREN が仕上げます。外部の値は EXTRN で宣言した名前です。受け付けられるのは、次の組み合わせだけです。

式	結果	例
アブソリュート＋リロケータブル、またはその逆	リロケータブル	a1+2
リロケータブル – アブソリュート	リロケータブル	a1-2
リロケータブル – リロケータブル（同じセグメント）	アブソリュート	a2-a1
外部＋アブソリュート、またはその逆	外部	far+2
外部 – アブソリュート	外部	far-2

そのほかの演算子は、符号反転、NOT、HIGH、LOW も含めて、アブソリュートな値しか受け付けません。外部の名前は、1 つの式に 1 回しか使えません。リロケータブルな値や外部の値は、ld hl,a1 や DW のように 2 バイトで値を持つ場所にしか書けません。1 バイトのオペランドはアブソリュートでなければなりません。

付録 D

エラーメッセージ

D.1 メッセージの形 315
D.2 TATARA のメッセージ 316
D.3 TANREN のメッセージ 320
D.4 TANREN の警告とお知らせ 322

この付録では、TATARA と TANREN が表示するすべてのメッセージについて、その意味、対処のしかた、例を示している節を示します。D.1 節ではメッセージの形を説明し、D.2 節と D.3 節ではそれぞれのツールのメッセージを、表示されるとおりにアルファベット順で示します。D.4 節では、エラーではない TANREN の警告とお知らせを示します。

D.1 メッセージの形

TATARA のエラーは、ファイルと行を示し、そのあとの `ERROR:` に続けてメッセージを表示します（20.1 節）。

MSX-DOS

```
LERR.AS(5): ERROR: not a form this instruction has.
```

その行がインクルード・ファイル、マクロ、繰り返しブロックの中にあると、メッセージのあとの行、つまり**経路**が、TATARA がそこにどうたどり着いたかを示します。インクルード・ファイルでは `included from FILE(line)`、マクロでは `in NAME, called from FILE(line)` です（20.3 節）。コマンド・ラインの誤りや、MSX-DOS2 やメモリ・マッパーがない場合は、示す行がないので、バナーの前に `ERROR:` とメッセージだけを表示します。

TANREN のエラーは行を示しません。誤りが 1 つのオブジェクト・ファイルにあるときは、`BAD.TRO: ERROR: not a Tatara object file.` のようにメッセー

ジがファイルの名前で始まります。そうでなければ、ERROR: とメッセージだけです（27.1 節）。

どちらのツールも、エラーを 1 つ表示して止まります（20.2 節、27.2 節）。誤りを直してもう一度実行すると、次の誤りが見つかります。/Q でエラーが隠れることはありません。メッセージの中のファイル名は大文字で表示されます。

下の一覧では、それぞれのメッセージを ERROR: のあとに表示されるとおりに示し、斜体の語は変わる名前を表します。一覧は、表示されるメッセージのアルファベット順なので、a PUBLIC name was never defined. は A の位置にあります。

D.2 TATARA のメッセージ

a bit number must be 0 to 7.
: BIT、SET、RES のビット番号が 0 から 7 の範囲にありません。9.6 節。

a JR or DJNZ can only reach -128 to 127.
: 相対ジャンプの飛び先が遠すぎます。どのアドレスにも届く JP を使ってください。9.3 節。

a macro name may be at most 64 characters.
: MACRO の前の名前が 64 文字を超えています。7.2 節。

a name may hold only letters, digits, ? @ . _ and the dollar sign.
: foo*bar の*のように、ラベルに名前に使えない文字が含まれています。7.1 節。

a PAGE length must be 10 to 255.
: PAGE のあとの数がその範囲にありません。18.2 節。

a PUBLIC name was never defined.
: PUBLIC で宣言した名前が、ソースのどこにも定義されていません。オブジェクト・ファイルを書き出すときに見つかるので、オブジェクト・ファイルを作らなければソースは受け付けられます。12.5 節。

a segment or group name may be at most 16 characters.
: CSEG、DSEG、GROUP のあとのセグメントやグループの名前が 16 文字を超えています。7.2 節。

an external symbol may not be used here.
: EXTRN で宣言した名前が、TANREN が仕上げられない場所で使われています。EQU の中、1 つの式で 2 回、または 2-far のように数を足すか引く以外の使い方です。8.6 節。

an index displacement must be -128 to 127.
: (IX+d) や (IY+d) のディスプレースメント *d* がその範囲にありません。9.2 節。

bad ASEG, CSEG or DSEG line.
: セグメントの疑似命令に、名前のあとの TRANSIENT 以外の語のような、受け付

けないオペランドがあります。10.7 節。

`bad DB, DW or DC operand.`
: `db 1,,2` のように項目が抜けているか、`DW` の文字列が 3 文字以上あるか、`DC` に文字列がないか空です。11.4 節。

`bad expression.`
: 式が読めません。基数にない数字、名前に使えない文字、オペランドを書けない場所のオペランド、または値が必要な場所に何もない場合です。8.2 節。

`bad macro parameter list.`
: `MACRO` のパラメーター、または `IRP` や `IRPC` のパラメーターとリストが、コンマで区切った名前になっていません。16.9 節。

`bad PUBLIC or EXTRN list.`
: `PUBLIC` や `EXTRN` に名前がないか、名前でないものがあります。12.7 節。

`cannot create` *name*
: オブジェクト・ファイルやリスティングを作れません。たとえば、ディレクトリがない、ディスクがいっぱい、書き込み禁止の場合です。5.6 節。

`cannot open` *name*
: ソース、または `INCLUDE` で指定したファイルが見つかりません。インクルード・ファイルの場合は、メッセージのあとの行がそれを求めた行を示します。TATARA は 13.2 節の 3 つの場所を探しています。13.8 節。

`cannot write to the output file.`
: ディスクがいっぱいか書き込み禁止のため、オブジェクト・ファイルやリスティングを書き込めませんでした。第 2 パスでのほかのエラーと同じように、TATARA はオブジェクト・ファイルを削除します。ディスクに空きを作るか別のディスクを使って、もう一度アセンブルしてください。20.2 節。

`conditional not closed by ENDIF.`
: `IF` に `ENDIF` がありません。メッセージは `IF` の行を示します。15.7 節。

`conditionals nested too deeply.`
: 同時に開いている条件付きアセンブルが 16 を超えています。15.7 節。

`ELSE or ENDIF without a matching IF.`
: `IF` が開いていないのに `ELSE` や `ENDIF` があるか、1 つの `IF` に 2 つ目の `ELSE` があります。15.7 節。

`ENDM without a macro definition.`
: マクロも繰り返しブロックも開いていないのに `ENDM` があります。16.9 節。

`EQU, DEFL and MACRO take a name, not a label.`
: `EQU`、`DEFL`、`MACRO` の前の名前にコロンが付いています。コロンを取ってください。12.3 節。

`EXITM outside a macro or repeat block.`
: `EXITM` がマクロや繰り返しブロックの中にありません。16.9 節。

`GROUP needs a name, no label, and a transient DSEG.`
: GROUP に名前がないか、ラベルが付いているか、トランジェント・データ・セグメントの中にありません。10.7 節。

`IM takes 0, 1 or 2.`
: IM のオペランドが 0、1、2 のどれでもありません。9.6 節。

`INCLUDE without a filename.`
: INCLUDE のあとに何もありません。13.8 節。

`internal - an instruction emitted nothing.`
: 内部エラーです。TATARA が自分自身の不具合を見つけました。報告してください（20.5 節）。20.5 節。

`internal - this line emitted more bytes than fit.`
: 上と同じく内部エラーです。20.5 節。

`IXH, IXL, IYH and IYL are registers, not names.`
: ラベル、PUBLIC、EXTRN が、インデックス・ハーフの 4 つの名前のどれかを使っています。これらはバージョン 1.2.0 からレジスターです。M80 はこれらを名前として受け付けます。9.5 節。

`LOCAL must come before the macro body.`
: LOCAL の行が、本体のほかの行のあとにあります。16.5 節。

`macro definition not closed by ENDM.`
: マクロや繰り返しブロックに ENDM がありません。メッセージはそれを始めた行を示します。16.9 節。

`macro line too long.`
: マクロや繰り返しブロックの行が、パラメーターを引数に置き換えると 255 文字を超えます。経路がマクロを示します。16.2 節。

`MACRO without a name.`
: MACRO の前に名前がありません。16.9 節。

`not a directive, a macro or an instruction.`
: オペレーションの語が、そのどれでもありません。つづりの誤り、TATARA にない M80 の疑似命令（付録 J）、まだ定義されていないマクロ、または 1 桁目から始まらずコロンもないラベルかもしれません。6.3 節。

`not a form this instruction has.`
: 命令はありますが、`ld hl,a` のように、そのオペランドの形はありません。書き方は付録 B に一覧があります。9.6 節。

`only *EJECT and its dollar spelling are control lines.`
: 1 桁目の語が*か$で始まっているので、M80 と同じように制御行になりますが、それが*EJECT や$EJECT ではありません。M80 の*TITLE と$TITLE は、TATARA では SUBTTL と書きます。付録 J。

`out of mapper memory.`

TATARA の表のための空きがメモリ・マッパーにありません。内部エラーではなく、ソースが MSX のメモリーより多くを必要としています。20.5 節。

`phase error - this label had a different value on pass 1.`
ラベルが第 2 パスで別のアドレスになりました。たいていは、条件付きアセンブルの答えがパスごとに違ったためです。15.2 節。

`relocation error - a segment-relative value is not allowed here.`
TANREN が仕上げるアドレスや外部の名前が、使えない場所で使われています。1 バイトの中、かけ算、`REPT` の回数、ほかのセグメントのアドレスを使った `ORG`、ほかのセグメントのラベルとの引き算です。8.6 節。

`RST takes 0, 8, 10h and so on to 38h.`
`RST` のオペランドが、8 つのリスタート・アドレスのどれでもありません。9.6 節。

`source line too long.`
ソース・ファイルの行が 255 文字を超えています。6.6 節。

`sources nested too deeply.`
インクルード・ファイル、マクロの展開、繰り返しブロックの入れ子が 16 段を超えています。自分自身を終わりなく使うマクロが原因のことがよくあります。16.7 節。

`Tatara needs a memory mapper.`
MSX にメモリ・マッパーがありません。バナーの前に表示されます。5.6 節。

`Tatara needs MSX-DOS2 or Nextor.`
MSX が MSX-DOS1 で動いています。バナーの前に表示されます。5.6 節。

`the output file is the input file.`
オブジェクト・ファイルがソースと同じ名前です。5.2 節。

`the TATARA variable is too long.`
`TATARA` の値が 127 文字を超えています。MSX-DOS はコマンドに 127 文字しか与えないので、プロンプトやバッチ・ファイルの **`set`** ではこれほど長い値は設定できません。設定できるのはプログラムだけです。13.3 節。

`this label was defined on pass 1 and not on pass 2 - a conditional skipped it.`
条件付きアセンブルが、第 1 パスではラベルをアセンブルし、第 2 パスでは飛ばしました。たとえば `IF1` の中のラベルや、2 回読まれたインクルード・ファイルです。15.2 節。

`this name already has a value.`
名前が 2 回定義されています。ラベルや `EQU` の名前をもう一度定義した、`DEFL` の名前がラベルや `EQU` の名前でもある、`EXTRN` で宣言した名前を定義した、のどれかです。マクロの中のラベルには `LOCAL` を使ってください。12.2 節。

`this segment was declared differently before.`
あるセグメントが、ある場所ではトランジェントとして、別の場所ではそうで

なく選ばれています。10.7 節。

`this transient DSEG needs a GROUP first.`
: トランジェント・データ・セグメントで、最初の `GROUP` より前に場所を確保している行があります。10.7 節。

`too many different source files.`
: TATARA が名前を覚えておけるのは、ソースを含めて 32 回のファイルのオープンまでで、33 回目が求められました。同じファイルを 2 回取り込むと 2 回と数えます。付録 E。

`too many filenames.`
: コマンド・ラインにファイル名が 4 つ以上あります。5.6 節。

`too many macro parameters.`
: マクロのパラメーターが 16 を超えています。16.9 節。

`too many segments, groups or externals.`
: 名前付きセグメント、グループ、外部の名前が、オブジェクト・ファイルに入りきらないほどあります（付録 E）。10.7 節。

`too many source files open at once.`
: 同時に開いているファイルが 4 つを超えています。インクルード・ファイルがさらにほかのファイルを取り込んでいるか、ファイルが自分自身を取り込んでいます。13.5 節。

`undefined symbol in an expression.`
: 式の中の名前がどこにも定義されていないか、`IF` や `REPT` ではその行でまだ定義されていません。文字で始まる 16 進数には前に 0 が必要です。15.2 節。

`unknown option.`
: スラッシュのあとが、TATARA のオプションにない文字か、何もありません。バナーの前に表示されます。5.6 節。

D.3　TANREN のメッセージ

`/D: would put the data on top of the code.`
: `/D:` で指定したアドレスがコードの中にあります。24.3 節。

`a file list may not name another one.`
: リンク・ファイルの中の語が`@`で始まっています。22.4 節。

`cannot create` *`NAME`*
: 出力ファイルを作れません。たとえば、そのディレクトリがない場合です。26.9 節。

`cannot open` *`NAME`*`.TRO`

その名前のオブジェクト・ファイルが、TANREN の探す場所に見つかりません。23.1 節。

`cannot open the file list` *`NAME`*`.LNK`
その名前のリンク・ファイルがないか、`@`のあとに名前がありません。22.4 節。

`cannot write the output file - the disk may be full.`
出力ファイルは作られましたが、書き込みに失敗しました。TANREN はすぐに止まり、書いた部分はそのまま残ります。もう一度リンクする前に削除してください。27.2 節。

`not a Tatara object file.`
メッセージはオブジェクト・ファイルの名前で始まります。ファイルがオブジェクト・ファイルの形で始まっていません。27.1 節。

`one module was assembled /C and another was not.`
モジュールはすべて`/C`付きか、すべて`/C`なしでアセンブルしなければなりません。25.5 節。

`only one file list may be given.`
コマンド・ラインに 2 つ以上のリンク・ファイルがあります。22.4 節。

`out of mapper memory.`
プログラムのイメージや TANREN の表のための空きがメモリ・マッパーにありません。26.7 節。

`segment` *`NAME`* `is declared differently.`
メッセージはオブジェクト・ファイルの名前で始まります。そのセグメントの種類が、このファイルと前のファイルで違います。27.4 節。

`Tanren needs a memory mapper.`
MSX にメモリ・マッパーがありません。バナーの前に表示されます。21.6 節。

`Tanren needs MSX-DOS2 or Nextor.`
MSX が MSX-DOS1 で動いています。バナーの前に表示されます。21.6 節。

`that public symbol is already defined.`
2 つのモジュールが同じ公開の名前を定義しています。メッセージの前の行、`NAME is defined again in module MOD` がその名前を示します。25.4 節。

`the linked image would run past FFFFh.`
プログラムが、始まるアドレスから FFFFh までに収まりません。24.6 節。

`the object file ends inside a record.`
メッセージはオブジェクト・ファイルの名前で始まります。ファイルが内容の途中で終わっています。モジュールをアセンブルし直してください。27.3 節。

`the output file` *`NAME`* `is also an input file.`
出力ファイルがオブジェクト・ファイルの 1 つを上書きしてしまいます。21.2 節。

`the symbols above were never defined.`

`Never defined:` の下に並んだ名前は、どこかのモジュールが外部と宣言していますが、どのモジュールも定義していません。25.3 節。

`the TANREN variable is too long.`
`TANREN` が 127 文字を超えています。23.5 節。

`this object file was made by another version.`
メッセージはオブジェクト・ファイルの名前で始まります。ファイルの形式が、この TANREN の読むものではありません。同じバージョンの TATARA でモジュールをアセンブルし直してください。27.3 節。

`too many external symbols in one module.`
1 つのモジュールが 512 を超える外部の名前を宣言しています。TATARA 自身は 254 で止まるので、TATARA が作ったオブジェクト・ファイルではこうなりません。付録 E。

`too many segments or groups in one module.`
メッセージはオブジェクト・ファイルの名前で始まります。モジュールのセグメントかグループが、2 つのデフォルトのセグメントを含めて 16 を超えています。27.4 節。

`too many words on the command line or in the file list.`
語が全部で 4096 文字を超えるか、1 つの語が 128 文字を超えています。22.4 節。

`unknown option.`
TANREN にないオプションか、コロンのない `/D`、`/O`、`/P` です。バナーの前に表示されます。21.3 節。

D.4 TANREN の警告とお知らせ

これらはエラーではありません。TANREN は続けて、リンクは成功します。

`FILE``: bytes follow the END record.`
エラーではありません。オブジェクト・ファイルの最後のレコードのあとにバイトがあります。このとき要約行は `ends at EOF.` ではなく `AND BYTES AFTER THEM.` で終わります。`/Q` を付けるとどちらも表示されません。
TATARA はこのようなファイルを作らないので、モジュールをアセンブルし直してください。27.3 節。

`Nothing to write: no module has any content.`
エラーではありません。内容のあるバイトを持つモジュールがないので、ファイルは作られません。26.8 節。

`WARNING: `*`NAME`*` is entered at `*`XXXX`*`, not at `*`YYYY`*`.`
ファイルは `/B` なしで書き出されたので、最初のアドレス *XXXX* から実行され

ますが、END の行のアドレスは *YYYY* です。.COM プログラムでは、END で示すコードを先頭に置いてください。/Q を付けても表示されます。26.5 節。

WARNING: the command line is 127 characters, which is all MSX-DOS gives - anything past it was dropped in silence.
3 行目に Use @<file> if a module is missing. が続きます。コマンド・ラインがちょうど 127 文字なので、一部が切り捨てられたかもしれません。リンクをリンク・ファイルに書いてください。21.5 節。

付録 E

制限値

E.1 TATARA 325
E.2 TANREN 326
E.3 コマンド・ライン 327
E.4 決まった制限のないもの 328

この付録では、TATARA と TANREN の決まった制限値を、それぞれの上限、それを超えたときのメッセージ、説明している節とともに示します。E.4 節では、決まった制限のないものを示します。

E.1 TATARA

対象	上限	メッセージ	節
ソースの 1 行	255 文字	`source line too long.`	6.6
シンボルの名前	255 文字。すべて区別される		7.2
マクロの名前	64 文字	`a macro name may be at most 64 characters.`	7.2
セグメントやグループの名前	16 文字	`a segment or group name may be at most 16 characters.`	7.2
コマンド・ラインのファイル名	3 つ	`too many filenames.`	5.6
同時に開いているファイル	4 つ	`too many source files open at once.`	13.5

対象	上限	メッセージ	節
インクルード・ファイル、マクロの展開、繰り返しブロックの入れ子	16 段	`sources nested too deeply.`	16.7
1 回のアセンブルで開くファイル（ソースを含む）	32 回	`too many different source files.`	D.2
メッセージの先頭のファイル名	最初の 31 文字を表示		20.1
同時に開いている条件付きアセンブル	16	`conditionals nested too deeply.`	15.6
マクロのパラメーター	16	`too many macro parameters.`	16.9
展開したあとのマクロや繰り返しブロックの 1 行	255 文字	`macro line too long.`	16.2
外部の名前	254	`too many segments, groups or externals.`	12.5
タイトルとサブタイトル	最初の 28 文字を残す		18.2
ページの長さ	10 から 255 行	`a PAGE length must be 10 to 255.`	18.2
環境変数 `TATARA`	127 文字	`the TATARA variable is too long.`	13.3

長すぎるマクロ、セグメント、グループの名前は、切り詰められるのではなく、拒否されます。同じファイルを 2 回開くと、1 回のアセンブルの 32 回のうちに 2 回と数えます。メッセージの先頭のファイル名は最初の 31 文字に切り詰められるので、長いパスではファイルそのものの名前が見えなくなることがあります。`included from` の行と `cannot open` のメッセージは、名前をすべて表示します。

TATARA は TANREN より多くの名前付きセグメントとグループを受け付けるので、実際には E.2 節の制限が適用されます。

E.2　TANREN

対象	上限	メッセージ	節
1 つのモジュールのセグメント	16。デフォルトのコード・セグメントとデータ・セグメントを含む	`too many segments or groups in one module.`	27.4
1 つのモジュールのグループ	16	`too many segments or groups in one module.`	27.4
1 つのモジュールの外部の名前	512	`too many external symbols in one module.`	D.3
リンクの語（コマンド・ラインとリンク・ファイルの合計）	4096 文字	`too many words on the command line or in the file list.`	22.4
1 つの語	128 文字	同上	22.4
リンク・ファイル	1 つ。ほかのリンク・ファイルは指定できない	`only one file list may be given.`、`a file list may not name another one.`	22.4
環境変数 `TANREN`	127 文字	`the TANREN variable is too long.`	23.5
プログラム	FFFFh 以下で終わる	`the linked image would run past FFFFh.`	24.6

そのため、1 つのモジュールには、デフォルトのコード・セグメントとデータ・セグメントのほかに、名前付きセグメントを 14 まで置けます。制限はプログラム全体ではなくモジュールごとのものなので、それより多く必要なプログラムは、いくつかのモジュールに分ければすみます。TATARA は外部の名前 254 で止まるので、TATARA が作ったオブジェクト・ファイルが 512 の制限に達することはありません。

E.3 コマンド・ライン

MSX-DOS がプログラムに渡すコマンド・ラインは 127 文字までで、残りはメッセージなしで捨てられます。これはツールではなく MSX-DOS の制限で、どちらのツールにもあてはまります。TANREN は、コマンド・ラインがちょうど 127 文字のときには一部が欠けているかもしれないので警告します。リンク・ファイルにはこの制限はありません（21.5 節）。

E.4　決まった制限のないもの

シンボルの数、マクロの数と大きさ、プログラムの大きさは、メモリ・マッパーによってだけ制限されます。TATARA はそこに表を置き、TANREN はそこでプログラムを組み立てます。マッパーがいっぱいになると、ツールは `out of mapper memory.` で止まります。ツールのメモリの使い方は付録 G で説明します。

付録F

.troオブジェクト・ファイルの形式

F.1 オブジェクト・ファイルが伝えること 329
F.2 例 330
F.3 ファイルの構成 333
F.4 レコード 334
F.5 TANREN がオブジェクト・ファイルを読む方法 337

TATARA が書き出し、TANREN が読む**オブジェクト・ファイル**の拡張子は.troです。この付録では、そのファイルに何が入っていて、どのように並んでいるかを説明します。これを読み書きするプログラムを作りたい人や、TANREN の/Rが何を表示しているかを知りたい人のためのものです。説明するのは、バージョン1.2.0のTATARA と TANREN が使う、形式のバージョン2です。まずオブジェクト・ファイルが何を伝えなければならないかを説明し、次に実際のファイルを1バイトずつたどり、最後にリファレンスを示します。

F.1 オブジェクト・ファイルが伝えること

2章で、アセンブラーだけではプログラムを完成できない理由を説明しました。リンカーがそれぞれの部分をどこに置くかも、ほかのモジュールが定義する名前のアドレスもわからないからです。そこでオブジェクト・ファイルは、自分のモジュールについて次の4つのことをリンカーに伝えます。

1. **部分**　バイトを置くセグメント、それぞれのバイト数、そしてバイトそのもの。
2. **提供する名前**　公開するそれぞれの名前と、その値。値はあるセグメントの中のオフセットか、ただの数です。
3. **必要な名前**　それぞれの外部の名前。

4. **穴**　バイトの中の、セグメントを置く場所や外部の名前によって値が決まるすべてのワード。

穴は空のままにはなりません。ワードには値のうちすでにわかっている部分、つまり**アデンド**（addend）が入っていて、別の**フィックスアップ**（fixup）が、それに何を足すかを示します。フィックスアップは自分では数を持たず、何を足すかを示すだけで、数はいつもワードに入っています。`count` がデータ・セグメントのオフセット 0000h にあるとき、`ld hl,count` のワードは 0000h で、フィックスアップは「データ・セグメントのアドレスを足す」と示します。`ld hl,count+2` では、ワードは `count` のオフセットに 2 を足した 0002h で、フィックスアップは同じです。外部の名前を呼び出すワードは 0000h で、そのフィックスアップは「その名前の値を足す」と示します。TANREN はバイトを所定の場所にコピーし、そのあとで、それぞれのフィックスアップが示すものを足します。

これらはどれも**レコード**として保存されます。レコードは、タイプのバイト、そのあとに続くものの長さ、そして内容からなります。ファイルを読むプログラムは、必要のないレコードを長さだけで読み飛ばせます。

F.2　例

次のモジュールには、上のものが 1 つずつ入っています。

```
; OBJ.AS - a module with one of each thing an object file holds.

                public  start,count
                extrn   putc

                cseg
start:          ld      hl,count        ; a word in the data segment
                ld      a,(hl)
                call    putc            ; an external
                ret

                dseg
count:          db      3

                dseg    scratch,transient
                group   one
buf1:           ds      4
                group   two
buf2:           ds      2

                end     start
```

`tatara obj.as obj.tro` で 169 バイトのファイルができます。図 F.1 はそれを

レコードの並びとして示し、表 F.1 は 1 バイトずつたどります。オフセットとバイトは 16 進数で、文字列のバイトはその文字で示します。

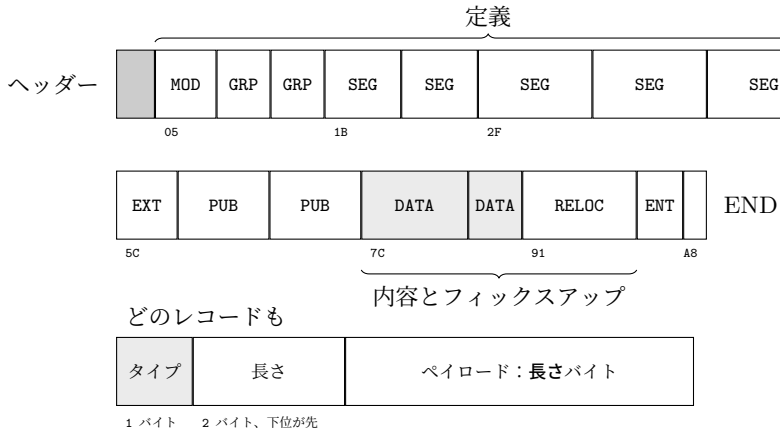

図 F.1: レコードの並びとしての OBJ.TRO と、1 つのレコードの形。箱の幅はバイト数を表す。

表 F.1: OBJ.TRO の 1 バイトずつ

オフセット	バイト	意味
00	54 52 4F 1A 02	ヘッダー：TRO、1Ah、バージョン 2
05	01 05 00	**MODNAME**、5 バイト：
	00 03 "obj"	フラグ 00（/C なし）、名前 obj
0D	02 04 00 03 "one"	**GRPDEF**：グループ 00 は one
14	02 04 00 03 "two"	**GRPDEF**：グループ 01 は two
1B	03 07 00	**SEGDEF**、7 バイト：セグメント 00
	00 FF 08 00 02 " C"	コード、グループなし、8 バイト、デフォルトのコード・セグメント
25	03 07 00	**SEGDEF**：セグメント 01
	01 FF 01 00 02 " D"	データ、グループなし、1 バイト、デフォルトのデータ・セグメント
2F	03 0C 00	**SEGDEF**：セグメント 02
	03 FF 00 00 07 "scratch"	トランジェントなデータ、グループなし、0 バイト：セグメントそのもの
3E	03 0C 00	**SEGDEF**：セグメント 03
	03 00 04 00 07 "scratch"	トランジェントなデータ、グループ 00、4 バイト
4D	03 0C 00	**SEGDEF**：セグメント 04

オフセット	バイト	意味
	03 01 02 00 07 "scratch"	トランジェントなデータ、グループ 01、2 バイト
5C	05 05 00 04 "putc"	**EXTDEF**：外部 0000 は putc
64	04 09 00	**PUBDEF**、9 バイト：
	01 00 00 05 "count"	セグメント 01、オフセット 0000h：count
70	04 09 00	**PUBDEF**：
	00 00 00 05 "start"	セグメント 00、オフセット 0000h：start
7C	06 0B 00	**DATA**、11 バイト：
	00 00 00	セグメント 00、オフセット 0000h
	21 00 00 7E CD 00 00 C9	コード：ld hl,count、ld a,(hl)、call putc、ret。2 つの穴は 0000h
8A	06 04 00 01 00 00 03	**DATA**：セグメント 01、オフセット 0000h、バイト 03
91	07 0C 00	**RELOC**、12 バイト、フィックスアップ 2 つ：
	00 00 01 00 01 00	セグメント相対：セグメント 00 のオフセット 0001h のワードに、セグメント 01 のアドレスを足す
	01 00 05 00 00 00	外部：セグメント 00 のオフセット 0005h のワードに、外部 0000 の値を足す
A0	08 03 00 00 00 00	**ENTRY**：セグメント 00、オフセット 0000h
A6	FF 00 00	**END**

次の点に注目してください。

- デフォルトのコード・セグメントとデータ・セグメントには、" C"と" D"という名前があります。空白と 1 文字で、ソースではセグメントの名前として書けない名前です。どのオブジェクト・ファイルにも、片方が空でも、両方があります。
- トランジェント・セグメント scratch には、グループもバイトもない自分自身の **SEGDEF** があり、そのあとにグループごとに 1 つずつ、そのグループの大きさを持つ **SEGDEF** があります。
- モジュールの名前は、コマンド・ラインで入力したとおりのソースの名前です。

putc を ret1 つとして定義する PUTC.TRO とリンクすると、TANREN は OBJ のコードを 0100h に、PUTC のコードをそのあとの 0108h に、データ・セグメントを

0109h に置きます。バイトを所定の場所にコピーし、2 つのフィックスアップを適用します。`ld hl,count` には 0000h + 0109h、`call putc` には 0000h + 0108h です。プログラム `OBJ.COM` は、次の 10 バイトです。

```
21 09 01 7E CD 08 01 C9 C9 03
```

トランジェント・セグメントは 010Ah にあり、2 つのグループはどちらもそこから始まります。バイトを持たないので、ファイルには入りません。

F.3 ファイルの構成

ファイルには 1 つのモジュールが入ります。ファイルは 5 バイトのヘッダーで始まります。

オフセット	バイト	意味
0	`54 52 4F`	文字 `TRO`
3	`1A`	MSX-DOS のファイルの終わりの印。**`type obj.tro`** は `TRO` を表示して止まる
4	`02`	形式のバージョン

そのあとに、**END** レコードまでレコードが続きます。レコードは、タイプのバイト、ペイロードの長さのワード、ペイロードからなります。全体を通して、次の規則があります。

- **ワード**は 2 バイトで、Z80 と同じように下位バイトが先です。
- **文字列**は、1 から 255 までの長さのバイトと、その数の文字です。名前は書いたとおりに保存されます。大文字と小文字を区別したかどうかは、**MODNAME** のフラグが示します。
- セグメント、グループ、外部の名前には、レコードの順に 0 から番号が付きます。セグメントとグループの番号は 1 バイトで、FFh は「なし」を意味します。セグメントではアブソリュートな値、グループではグループなしです。外部の名前の番号はワードです。
- レコードが使えるのは、それより前のレコードで定義されたセグメント、グループ、外部の名前の番号だけです。
- **END** レコードのあとのものは無視されます。TANREN は `bytes follow the END record.` でそれを知らせます（27.3 節）。

TATARA は、**MODNAME**、**GRPDEF**、**SEGDEF**、**EXTDEF**、**PUBDEF**、**DATA**、**RELOC**、END にアドレスがあれば **ENTRY**、**END** の順にレコードを書き出します。

F.4　レコード

それぞれのレコードについて、ペイロードの形と、TANREN の/R が表示する行を示します（21.3 節）。/R の行の値は 16 進数です。

01h MODNAME：モジュール

フィールド	大きさ	意味
flags	バイト	ビット 0 が 1 なら/C 付きでアセンブルした。ほかのビットは 0
name	文字列	モジュールの名前

ファイルの最初のレコードで、この種類のものは 1 つだけです。TANREN はモジュールの名前を、大文字と小文字を区別せずに比べます。/R は `MODNAME  flags 00 obj` と表示します。

02h GRPDEF：グループ

ペイロードはグループの名前の文字列です。グループには次の番号が付きます。同じ名前のグループは、このモジュールでもほかのモジュールでも同じグループです（10.6 節）。/R は `GRPDEF    one` と表示します。

03h SEGDEF：セグメント

フィールド	大きさ	意味
flags	バイト	ビット 0 が 1 ならデータ、0 ならコード。ビット 1 が 1 ならトランジェント
group	バイト	グループの番号。なしなら FFh
size	ワード	このモジュールがセグメントに置くバイト数
name	文字列	名前。デフォルトのセグメントは`" C"`と`" D"`

セグメントには次の番号が付きます。TANREN は、すべてのモジュールの同じ名前とグループのセグメントを、モジュールの順につなげ、トランジェント・セグメントのグループは重ねて置きます。あるモジュールではコードで別のモジュールではデータのセグメントや、一方ではトランジェントで他方ではそうでないセグメントはエラーです。/R は `SEGDEF  flags 03 group 00 size 0004 scratch` と表示します。

04h PUBDEF：公開する名前

フィールド	大きさ	意味
segment	バイト	値のセグメント。ただの数なら FFh
offset	ワード	値。そのセグメントの中のオフセット、または数
name	文字列	名前

この 3 つのフィールドを繰り返してペイロードを埋めることもできますが、TATARA は 1 つのレコードに 1 つの名前を書きます。/R は `PUBDEF  seg 01 offset 0000 count` と表示します。

05h EXTDEF：外部の名前

ペイロードは 1 つ以上の文字列で、それぞれが外部の名前です。名前には次の外部の番号が付きます。/R は `EXTDEF  putc` と表示します。

06h DATA：バイト

フィールド	大きさ	意味
segment	バイト	セグメント。アブソリュート・セグメントなら FFh
offset	ワード	セグメントの中でバイトが始まる位置、またはアドレス
bytes	残り	バイト。穴にはアデンドが入っている

レコードの長さには、バイトの前の 3 バイトも含まれるので、8 バイトなら 11 です。1 つのセグメントの **DATA** レコードはいくつあってもよく、順序も自由です。/R

はバイト数そのものとバイトを表示します。`DATA　　seg 00 offset 0000 len 0008 21 00 00 7E CD 00 00 C9`

07h RELOC：フィックスアップ

フィールド	大きさ	意味
kind	バイト	00h：セグメントのアドレスを足す。01h：外部の値を足す
segment	バイト	ワードのあるセグメント
offset	ワード	そのセグメントの中のワードの位置
target	ワード	kind が 00h ならセグメントの番号、01h なら外部の番号

この 6 バイトがフィックスアップごとに繰り返されます。フィックスアップはどれもワード全体に対するもので、1 バイトに対するフィックスアップはありません。1 バイトのオペランドがアブソリュートでなければならないのはそのためです（8.6 節）。`/R` はその数を表示します。`RELOC　　2 fixups`

08h ENTRY：開始アドレス

セグメントのバイトとオフセットのワードで、`END` の行のアドレスです。1 つのモジュールに 1 つまでで、TANREN はモジュールの中で最初に見つけたものを使います（26.5 節）。`/R` は `ENTRY　　seg 00 offset 0000` と表示します。

7Fh COMMENT

どんなバイトでもよく、読むプログラムは読み飛ばします。TATARA はこのレコードを書きません。

FFh END

ペイロードは空です。モジュールの最後のレコードです。`/R` は `END` と表示します。

F.5 TANREN がオブジェクト・ファイルを読む方法

TANREN は、どのオブジェクト・ファイルも始めから終わりまで 2 回読みます（27.2 節）。図 F.2 に、F.2 節の例の `OBJ.TRO` と `PUTC.TRO` についての 2 つのパスを示します。

1. **第 1 パス**では定義を読みます。モジュールの名前、グループとセグメントとその大きさ、公開する名前、外部の名前で、**DATA** と **RELOC** は長さで読み飛ばします。すべてのファイルを読み終えると、セグメントを配置し（24 章）、公開するそれぞれの名前の値を求め、外部の名前がどれも定義されていることを確かめます。
2. **第 2 パス**では、**DATA** レコードのバイトを所定の場所にコピーし、それぞれのワードにフィックスアップが示すものを足します。そのあと、TANREN は出力ファイルを書き出します。

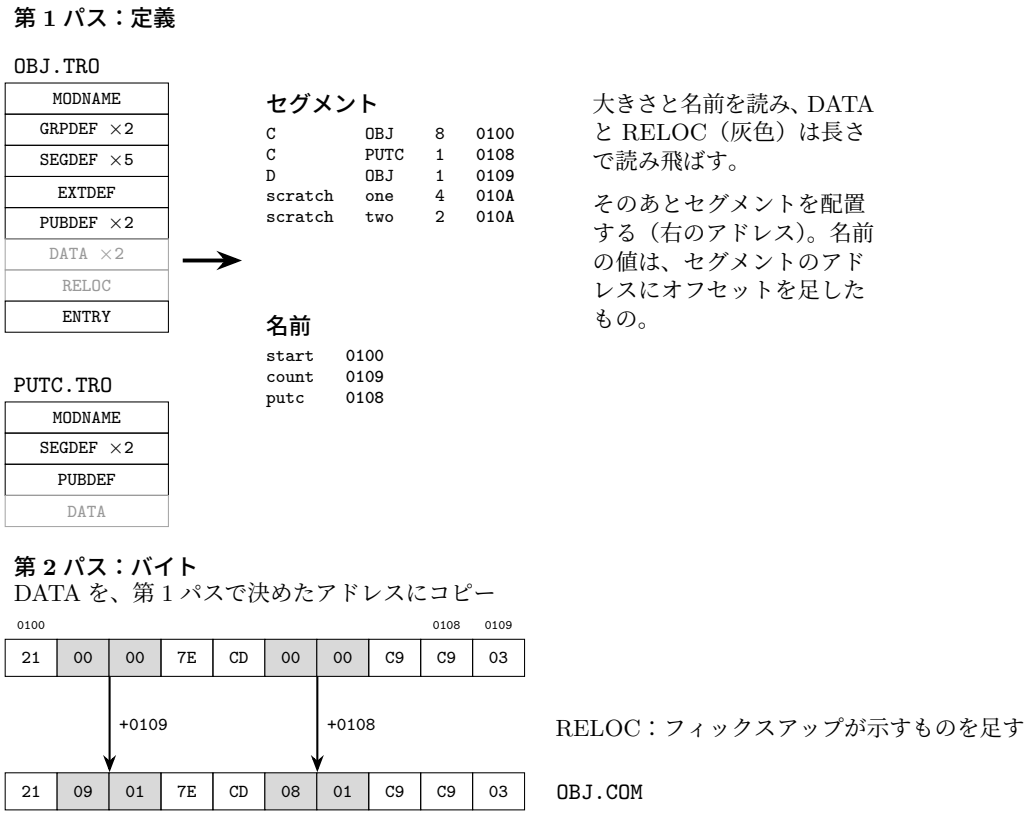

図 F.2: TANREN が `OBJ.TRO` と `PUTC.TRO` で行うこと。上は第 1 パスで、セグメントと名前の表を作る。下は第 2 パスで、バイトをコピーし、穴（網掛け）を埋める。

付録G

メモリ

G.1 メモリ・マップ 339
G.2 マッパーに置くもの 340
G.3 プログラムを作るのにツールが必要とするメモリの量 340
G.4 マッパーがいっぱいになったとき 342

TATARA と TANREN は、表を、そして TANREN は作成中のプログラムも、Z80 から見える 64 KB の中ではなく、MSX のメモリ・マッパーに置きます。この付録では、ツールのメモリの使い方、プログラムを作るのに必要なメモリの量、そして足りないときにどうすればよいかを説明します。ここで扱うのは、ツールが動いている間に必要なメモリで、できあがったプログラムが動くときに必要なメモリではありません。

G.1 メモリ・マップ

メモリ・マッパーは、16 KB のセグメントに分けられた RAM です。Z80 はそのうちの 4 つを同時に見ることができ、64 KB の 4 分の 1 ずつに 1 つずつ置かれます。この 4 分の 1 を**ページ**と呼びます。図 G.1 に、TATARA や TANREN が動いている間の 4 つのページの使い方を示します。

- **0000h〜3FFFh と 4000h〜7FFFh**：0100h より下は MSX-DOS が使い、0100h からはツールそのもの、つまりそのコードと、いつも必要なデータが置かれます。
- **8000h〜BFFFh**：窓です。ツールは、マッパーに置いたものを読み書きするために、マッパーのセグメントを 1 つずつこのページに持ってきます。MSX-DOS を呼ぶ前には、毎回このページを MSX-DOS に返します。
- **C000h〜FFFFh**：MSX-DOS とスタックです。

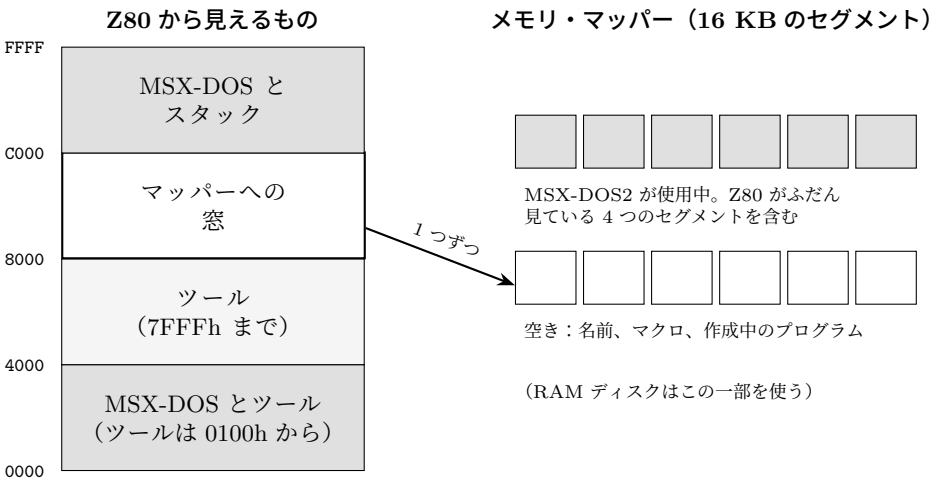

図 G.1: TATARA や TANREN が動いている間の Z80 の 64 KB と、窓の後ろにあるマッパーのセグメント

G.2　マッパーに置くもの

MSX-DOS2 がほかのものに割り当てていないセグメントは**空き**で、ツールは必要になるたびに、それを MSX-DOS2 に求めます。

- **TATARA** は、名前とその値、マクロの本体、オブジェクト・ファイルに書き出すレコードを置きます。
- **TANREN** は、各モジュールのセグメントと名前、そして作成中のプログラムを置きます。プログラムは、それが占めるアドレス 16 KB ごとに 1 つのセグメントを使います。0100h から A0FFh までのプログラムなら 3 つです。何も書き込まれないアドレスはセグメントを使いません。

つまり、プログラムの大きさを制限するのはマッパーの空きメモリで、MSX-DOS が C000h より下にプログラム用に残すメモリではありません。

G.3　プログラムを作るのにツールが必要とするメモリの量

問題になるのは全体の量ではなく、空きメモリの量です。MSX-DOS2 はマッパーの一部を自分で使います。以下のテストでは、RAM ディスクなしで、MSX2 と MSX2+ では 112 KB、MSX turbo R では 176 KB でした。RAM ディスクがあると、さらに使われます。

表 G.1 に、いくつかのマシンの空きメモリに何が収まったかを示します。どのソースも、16 文字の名前を EQU で定義したもので、プログラムは 40 KB のデータを .COM プログラムとしてリンクしたものです。

MSX	全体	MSX-DOS2	空き	名前 2000	名前 8000	40 KB
MSX2	128 KB	112 KB	16 KB	×（495）	×（495）	×
MSX2	192 KB	112 KB	80 KB	○	×（2479）	○
turbo R	256 KB	176 KB	80 KB	○	×（2479）	○
turbo R	384 KB	176 KB	208 KB	○	×（6447）	○
turbo R	512 KB	176 KB	336 KB	○	○	○
turbo R	1024 KB	176 KB	848 KB	○	○	○

表 G.1: マッパーの空きメモリに収まるもの。40 KB は 40 KB のプログラム。かっこの中の数は、収まった名前の数

ここから、次の目安が得られます。

- 16 KB の空きメモリには、16 文字の名前がおよそ 495 個入ります。1 つの名前がおよそ 33 バイトを使うということです。短い名前はもっと少なくてすみます。
- TANREN には、表のための場所のほかに、プログラムが占めるアドレス 16 KB ごとに 1 つの空きセグメントが必要です。
- 128 KB の MSX で MSX-DOS2 が残す最小の 16 KB の空きでも、ツールは動きますが、作れるのは小さなプログラムだけです。
- 80 KB の空きがあれば、ツールは自分自身を作れます（35 章）。27 個のモジュールからなる、およそ 30 KB のプログラムです。192 KB の MSX2 や 256 KB の turbo R には、それだけの空きがあります。
- 336 KB 以上の空きがあれば、表のすべてが収まります。512 KB のマッパーがあれば、MSX に収まるどんなプログラムでも余裕を持って作れます。

MSX2 に MSX-DOS2 のカーネルを加えるカートリッジの多くは、自分のマッパーを持っています。アスキーの MSX-DOS2 カートリッジは 128 KB、SD-512 と MegaFlashRom SCC+ SD は 512 KB、Carnivore2 は 1 MB です。最大 4 MB の MEGA-MAPPER や 16 MB メモリーカートリッジ V3 のような、マッパーだけのカートリッジは、どの MSX にもメモリを加えます。これらのカートリッジとその入手先は、付録 H に示します。

MSX のメモリの量を調べる

MSX の起動画面にはメモリの量が表示されますが、機種によっては、コンピューター本体の中のマッパーのメモリだけを表示し、カートリッジのマッパーのメモリは表示しません。`https://github.com/javilm/msx-mapper` から入手できる `MAPPER.COM` は、MSX-DOS2 が見つけたすべてのマッパーを、それぞれのうち MSX-DOS2 が使っている量と空いている量とともに表示します。

MSX-DOS

```
A:\>mapper
Total 256Kb (176Kb sys, 0Kb user, 80Kb free).
Memory mappers detected by MSX-DOS2:
[PRIMARY] Expanded slot 3-0: 176Kb sys, 0Kb user, 80Kb free.
```

ツールが使えるのは、`free` の量です。

G.4　マッパーがいっぱいになったとき

空きセグメントがなくなると、ツールは `out of mapper memory.` で止まります。空きを増やすには、次の方法があります。

- **RAM ディスクをなくす**：RAM ディスクがあれば、**`ramdisk 0`** でそのメモリを MSX-DOS2 に返します。メモリに常駐するプログラムも、マッパーのセグメントを使います。
- **プログラムを分ける**：より多くのモジュールに分けます（28 章）。TATARA が一度に場所を必要とするのは、1 つのモジュールの名前だけです。
- **メモリを加える**：上に書いたようなマッパーのカートリッジを使います。

付録 H

ハードウェアの入手先

H.1 ツールに必要なもの 343
H.2 カートリッジ 344
H.3 メーカーと購入先 345
H.4 入手のしやすさ 346

この付録は、TATARA と TANREN を実機の MSX で動かしたい読者のためのもので、MSX に何を加えればよいかを説明します。いまでもいくつかの小さなメーカーが MSX 用のカートリッジを作っていて、その中には、ツールに必要なものをすべて MSX に与えるものもあります。

H.1 ツールに必要なもの

MSX2、MSX2+、MSX turbo R のほかに、ツールには次の 3 つが必要です。

- **MSX-DOS2**、または MSX-DOS2 用のプログラムを実行できる *Nextor*。ツールは MSX-DOS2 を通してマッパーのメモリを使います（付録 G）。
- 十分な空きメモリのある**メモリ・マッパー**。付録 G では 512 KB をすすめています（G.3 節）。
- **ファイルを置く場所**。小さなプログラムならフロッピー・ディスクで足りますが、ツールそのもの（35 章）のような大きなプログラムは、SD カードやコンパクトフラッシュ・カード、ハード・ディスクのほうがずっと扱いやすくなります。

MSX2 や MSX2+の多くはマッパーが 64 KB か 128 KB しかなく、MSX-DOS2 のカーネルを内蔵しているのは MSX turbo R だけです。カートリッジ 1 本で 3 つすべてをまかなうこともできます。

H.2　カートリッジ

表 H.1 に、これらをまかなうカートリッジと、それぞれが MSX に加えるものを示します。

表 H.1: ツールに必要なものを MSX に与えるカートリッジ

カートリッジ	マッパー	ほかに持っているもの	メーカー
SD-512	512 KB	Nextor、SD カードと microSD カードのスロット	8bits4ever
MegaFlashRom SCC+ SD	512 KB（オプション）	Nextor、microSD カードのスロット 2 つ、ROM 用の 8 MB のフラッシュ・メモリ、SCC+音源	MSX Cartridge Shop
Carnivore2	1 MB	Nextor、コンパクトフラッシュ・カードのスロット、ROM 用の 8 MB のフラッシュ・メモリ、SCC と FM-PAC の音源	RBSC
MEGA-MAPPER	最大 4 MB	なし。マッパーだけ	8bits4ever
16 MB メモリーカートリッジ V3	16 MB（4×4 MB）	なし。マッパーだけ	クラシック PC 研究会
MSX-DOS2 カートリッジ	128 KB	MSX-DOS2	アスキー

SD-512、MegaFlashRom SCC+ SD（マッパー・モジュール付き）、Carnivore2 は、それぞれ 1 本で MSX2 に 3 つすべてを与えます。MEGA-MAPPER と 16 MB のカートリッジが加えるのはメモリだけなので、MSX-DOS2 とファイルを置く場所がすでにある MSX か、それらを別のカートリッジでまかなう MSX のためのものです。

16 MB のカートリッジは中にスロット拡張器を持っていて、MSX-DOS2 には、それぞれが別のスロットにある 4 MB のマッパー 4 つとして見えます。MSX-DOS2 はそのすべてを使うので、ツールからは、16 MB から MSX-DOS2 が自分で使う分を引いた量が見えます。

アスキーの MSX-DOS2 カートリッジはもう作られていませんが、いまでも中古で見つかります。その 128 KB で作れるのは小さなプログラムだけです（表 G.1）。

H.3 メーカーと購入先

8bits4ever SD-512 と MEGA-MAPPER を作っていて、Carnivore2 も販売しています。
https://www.8bits4ever.net/product-page/sd-512

MSX Cartridge Shop MegaFlashRom SCC+ SD を作っています。
https://www.msxcartridgeshop.com

RBSC （Russian Bear Service Crew）Carnivore2 と、その新しい版で、もう 1 つの選択肢である **Carnivore2+**を設計しています。
https://rbsc.su/en/completed_projects/
https://rbsc.su/en/projects_in_development/

クラシック PC 研究会 「MSX2/2+/turboR 用 16 MB メモリーカートリッジ V3」を作っています。チームは小林、花島、井上、鈴木、小豆野の各氏です。ソニーの HB-F1XDJ と HB-F1XV、パナソニックの FS-A1、FS-A1ST、FS-A1GT で動作が確認されています。
https://classicpc.org

RBSC は Carnivore2 を直接は販売せず、いくつかの国の販売店を通して販売しています。

- 8bits4ever （スペイン）：https://www.8bits4ever.net/product-page/carnivore2
- HobbyRetro（スペイン）：https://hobbyretro.com/en/retro/carnivore-2
- Retro Game Restore（台湾）：https://retrogamerestore.com/store/carnivore2/
- The Retro Hacker Store（ブラジル）：https://www.theretrohacker.com/product/rbsc-carnivore2-msx-flashrom-scc-fmpac-ide-cf-ram/
- Denjhang's Retro Hardware（中国）：https://denjhang.com/products/carnivore2-msx-multifunctional-cartridge
- Maxiol（ロシア）。Carnivore2+も販売しています：https://radioshop.maxiol.com/product/carnivore-2-for-msx

H.4　入手のしやすさ

これらのカートリッジは小さなメーカーが少量ずつ作っているもので、ロットが作られては売り切れるたびに、店に並んだり消えたりします。このマニュアルを書いた時点の MEGA-MAPPER のように、いまは載っていないカートリッジも、次のロットで戻ってくるかもしれません。また、ウェブのアドレスが変わることもあります。いちばん確かなのは、メーカー自身のサイトで尋ねることです。

これらがなくても、openMSX のようなエミュレーターで、MSX-DOS2、好きな大きさのマッパー、ハード・ディスクを持つ MSX をエミュレートでき、ツールは実機と同じように動きます。

付録 I

環境変数

I.1 環境変数の設定 347
I.2 PATH 348
I.3 TATARA 348
I.4 TANREN 349
I.5 両方に共通の規則 350

ツールは 3 つの環境変数を使います。PATH は MSX-DOS2 がツールを見つけるため、TATARA は TATARA がインクルード・ファイルを探す場所、TANREN は TANREN がオブジェクト・ファイルを探す場所を示します。これらは、最初に必要になる 3 章、13 章、23 章で紹介しています。この付録では、その規則を 1 か所にまとめます。ツールがほかの環境変数を読むことはありません。

I.1 環境変数の設定

環境変数は、MSX-DOS2 の SET コマンドで、プロンプトかバッチ・ファイルで設定します。

MSX-DOS

```
A:\>set tatara="a:\tatara\include"
```

- **プロンプトでは**、\を含む値を、上のように引用符で囲みます。囲まないと、MSX-DOS2 は*** Invalid environment string と答え、環境変数を設定しません。
- **バッチ・ファイルでは**、引用符は必要ありません。書いても SET が取り除くので、値は同じになります。

- **環境変数のいまの値**は%name%と書きます。下の AUTOEXEC.BAT の例の 1 行目では、%path%を使って、PATH を置き換えずに、その最後にディレクトリを加えています。
- **一覧を見るには、set** だけを入力します。すべての環境変数とその値が表示されます。
- **削除するには**、何も書かずに設定します。**set tanren=** です。
- **名前**は大文字でも小文字でも書けます。MSX-DOS2 は大文字で保持するので、**set Tatara=inc** は TATARA を設定します。

環境変数は、MSX をリセットするか電源を切るまで残ります。MSX を起動するたびに設定するには、SET コマンドを起動ドライブのルート・ディレクトリにある AUTOEXEC.BAT に書きます（3.4 節）。3 つとも設定すると、次のようになります。

```
set path=%path%;a:\tatara\bin
set tatara=a:\tatara\include;a:\project\inc
set tanren=a:\project\lib
```

コマンド・ラインは、SET コマンドも含めて、それぞれの%name%を値に置き換えたあとで 127 文字までです。それより長いと、MSX-DOS2 は*** Command too long で拒否し、環境変数はもとのままです。

I.2　PATH

PATH はツールのものではなく、MSX-DOS2 のものです。コマンドが入力されると、MSX-DOS2 はその名前のプログラムを、カレント・ディレクトリ、次に PATH のそれぞれのディレクトリの順に探します。そのため、TATARA.COM と TANREN.COM のディレクトリを PATH に書いておくと、どのディレクトリからでもツールを実行できます。ツール自身は PATH を読まないので、インクルード・ファイルやオブジェクト・ファイルを探すのには関係しません。

I.3　TATARA

TATARA は、INCLUDE で指定されたファイルを、次の 3 か所で探します（13.2 節、図 13.1）。

1. インクルードする側のファイルのディレクトリ
2. カレント・ディレクトリ

3. TATARA のそれぞれのディレクトリ（左から順に）

環境変数が使われるのは、ファイルが最初の 2 か所のどちらにもないときだけで、ドライブや\で始まる絶対パスの名前には使われません（13.4 節）。TATARA は、最初に必要になったときに環境変数を読み、アセンブルが終わるまでその値を使います。

sub\mid.inc のように、ディレクトリを含む相対パスの名前も、ふつうの名前と同じように環境変数のそれぞれのディレクトリとつなげられます。図 I.1 に、値と名前から、TATARA が試す名前ができる様子を示します。

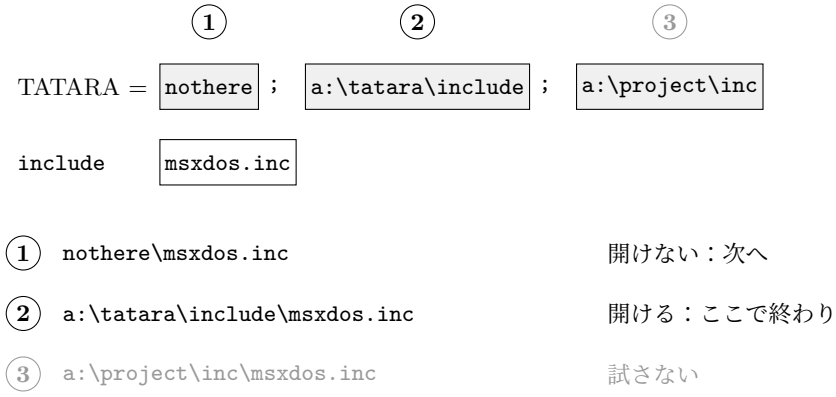

図 I.1: TATARA の値と INCLUDE の行の名前から、試す名前ができる様子（試す順）

I.4 TANREN

TANREN も、オブジェクト・ファイルを 3 か所で探します（23.1 節、図 23.1）。

1. そのファイルを指定したリンク・ファイルのディレクトリ（リンク・ファイルで指定した場合）
2. カレント・ディレクトリ
3. TANREN のそれぞれのディレクトリ（左から順に）

TATARA と同じように、環境変数が使われるのはファイルが最初の 2 か所のどちらにもないときだけで、絶対パスの名前には使われず、リンク全体で 1 回だけ読まれます。

/O: がなければ、出力ファイルは最初のオブジェクト・ファイルが見つかったディレクトリに書き出されます。そのオブジェクト・ファイルが TANREN のディレクトリにあったなら、出力ファイルもそこに書き出されます（23.4 節）。別の場所に置くには、/O: で出力ファイルの名前を指定します。

I.5　両方に共通の規則

TATARA と TANREN は同じように読まれます。値のそれぞれのディレクトリに\と名前を続けたものを順に試し、最初に開けたものを使います。詳しくは表 I.1 に示します。

表 I.1: TATARA と TANREN の読まれ方

対象	どうなるか
区切り	PATH と同じく、ディレクトリの間に; を書きます。
順序	左から右へ。ファイルが開けた最初のディレクトリが使われ、残りは試されません。
最後の\	必要ありませんが、あっても問題ありません。a:\inc と a:\inc\は同じです。
ドライブだけ	b: は、MSX-DOS2 でと同じように、ドライブ B: のカレント・ディレクトリです。
相対パスのディレクトリ	ソース・ファイルやリンク・ファイルではなく、カレント・ディレクトリからの相対です。inc は、ツールを起動したディレクトリの下の INC です。
ルートからのディレクトリ	\project\inc はカレント・ドライブのものです。
存在しないディレクトリ	メッセージなしで飛ばされ、次のディレクトリが試されます。
空の項目	;; や先頭の; はカレント・ディレクトリを意味します。そこはすでに試されているので、問題ありません。
空白	ディレクトリの名前の一部になります。a:\inc; a:\lib では、2 つ目のディレクトリが空白で始まるので存在せず、飛ばされます。; のあとに空白を入れないでください。
大文字と小文字	MSX-DOS のどこでもそうであるように、区別されません。
長いディレクトリ	ディレクトリ、\、名前を合わせて 63 文字に収まらなければなりません。名前に対して長すぎるディレクトリは、ファイルがそこにあっても、メッセージなしで飛ばされます。
長い値	127 文字まで。それより長い値は、短くなったリストで探すのではなく、the TATARA variable is too long. または the TANREN variable is too long. でツールを止めます。SET のコマンド・ラインそのものが 127 文字までなので、SET でこれほど長い値は作れません。このメッセージが出るのは、ほかのプログラムが環境変数を設定した場合だけです。
環境変数がない	最初の 2 か所だけが探されます。

どこにもファイルが見つからないとき、メッセージは試した名前ではなく、書かれたとおりのファイルの名前を示します。

```
A:\>tatara which.as
...
ERROR: cannot open ENVT.INC
    included from WHICH.AS(2)
```

あるはずのファイルが見つからないときは、まず SET だけを入力して、環境変数の値を確かめてください。

日本の MSX では、\は¥と表示されます。同じ文字です。

付録J

M80とのソースの互換性

J.1 そのまま使えるもの 353
J.2 TATARAが受け付けないM80の疑似命令 354
J.3 そのほかの書き方 355
J.4 TATARAが加えるもの 356
J.5 そのほかの違い 357
J.6 ソースの変換 357

この付録は、M80向けに書いたソースをTATARAでアセンブルしたい読者のためのものです。ここでは、TATARAをMSX-M80、つまりアスキーのMSX-DOS2 TOOLSに入っているM80と比べます。そのリスティングには`MSX.M-80 2.00`と表示されます。

M80向けに書かれたZ80のソースのほとんどは、そのままTATARAでアセンブルできます。変更が必要な疑似命令や書き方はわずかで、以下に、代わりに何を書けばよいかとともに示します。オブジェクト・ファイルには互換性がありません。TATARAが書き出すのは独自のオブジェクト・ファイル（付録F）で、M80の`.REL`ファイルではありません。

J.1 そのまま使えるもの

次のものは、TATARAでもM80と同じように使えます。

- 文の形。ラベル、オペレーション、オペランド、コメントの順で、ラベルは1桁目に書くかコロンを付けます。`name::`は`PUBLIC`でもあるラベルです（6章）。
- 接尾辞`B`、`D`、`O`、`Q`、`H`の付いた数、2文字までの文字定数（`''`は0）、どちらの引用符でも囲める文字列、ロケーション・カウンターを表す`$`。16ビットを

超える数は、M80 と同じように下位 16 ビットが残ります。

- 演算子とその優先順位（付録 C）。
- ASEG、CSEG、DSEG、ORG（10 章）。
- DB、DW、DS、DC、そして EQU と DEFL。
- PUBLIC、EXTRN、END。
- IF から IFDIF までの条件付きアセンブルと、ELSE、ENDIF（15 章）。
- MACRO、REPT、IRP、IRPC、LOCAL、EXITM、ENDM、そしてマクロの特別な書き方。&、;;、!、山かっこ、値で渡す引数の%です（16 章、17 章）。
- TITLE、SUBTTL、PAGE と、.LIST から.TFCOND までのリスティングの制御（18 章）。

M80 の疑似命令の別名も受け付けます。IF の別名の COND と IFT、IFE の別名の IFF、DB の別名の DEFB と DEFM、DS の別名の DEFS、DW の別名の DEFW、PUBLIC の別名の ENTRY、EXTRN の別名の EXT、そして 1 桁目に書いた PAGE の別名の*EJECT と $EJECT です（A.4 節）。

J.2　TATARA が受け付けない M80 の疑似命令

表 J.1 の疑似命令では、TATARA は `not a directive, a macro or an instruction.` で止まります。2 つの制御行では、`only *EJECT and its dollar spelling are control lines.` で止まります。

表 J.1: TATARA が受け付けない M80 の疑似命令と、代わりに書くもの

M80	TATARA では
SET	同じ働きをする DEFL（12.4 節）。
GLOBAL	PUBLIC。
EXTERNAL	EXTRN。
ENDC	ENDIF。
MACLIB、$INCLUDE、*INCLUDE	INCLUDE。
*TITLE、$TITLE	SUBTTL。テキストはかっこと引用符を付けずに書きます。
.Z80	削除します。TATARA はいつも Z80 の命令をアセンブルします。

M80	TATARA では
.8080	TATARA には 8080 モードがありません。ソースを Z80 のニーモニックで書き直す必要があります。
.RADIX	それぞれの数に接尾辞を付けて書きます。.radix 16 のあとの 0FF ではなく、0FFh です。
COMMON	変数を 1 つのモジュールで定義し、PUBLIC と EXTRN で共有します（12.5 節）。同じメモリを交代で使うブロックには、トランジェント・セグメントを使います（29 章）。
NAME	削除します。モジュールの名前はソース・ファイルの名前です（F.1 節）。
.COMMENT	コメントの各行を; で始めるか、ブロックを if 0 と ENDIF の間に置きます。
.PRINTX	相当するものはありません。削除します。
.REQUEST	ファイルを TANREN のコマンド・ラインかリンク・ファイルで指定します（21 章）。
.PHASE、.DEPHASE	相当するものはありません。
.CREF、 .XCREF	削除します。TATARA はクロス・リファレンスのファイルを作りません。

J.3 そのほかの書き方

表 J.2 に、ソースのそのほかの部分のうち、違いのあるものや、M80 を知っている読者が気にしそうなものを比べます。

表 J.2: M80 のソースのそのほかの書き方と、TATARA での扱い

M80 の書き方	TATARA では
foo::	同じ。PUBLIC でもあるラベルです。
far##	受け付けません（undefined symbol in an expression.）。名前を EXTRN で宣言し、##を付けずに書きます。
X'41'	受け付けません（undefined symbol in an expression.）。41h と書きます。
NUL	受け付けません。引数は IFB や IFNB で調べます（16.6 節）。
TYPE	受け付けず、相当するものもありません。名前が定義されているかどうかは IFDEF でわかります。
マクロの引数の%1b	同じ。1b の値が 10 進数で渡されます。
a!;b と<a;b>	同じ。どちらもセミコロンを含む a;b を渡します。
マクロの中の;;	同じ。そのコメントは展開に残りません。

M80 の書き方	TATARA では
1 行に式だけ	受け付けません。M80 は `41h` だけの行を `db 41h` として扱いますが、TATARA では `DB` を書きます。
8080 のニーモニック	受け付けません。`mvi a,1` は `ld a,1` です。
1 桁目の*か$で始まる語	M80 と同じく制御行です。受け付けるのは`*EJECT` と`$EJECT` だけで、それ以外では TATARA は止まります。そのため、`$loop:` はラベルにできません。
ほかの文字を含むラベル	拒否されます（`a name may hold only letters, digits, ? @ . _ and the dollar sign.`）。
名前の長さ	M80 は名前の最初の 6 文字しか使いませんが、TATARA は 255 文字まですべてを使います（7.2 節）。`buffer1` と `buffer2` のように、M80 が 1 つの名前とみなした 2 つの名前は、TATARA では別の名前です。
大文字と小文字	デフォルトでは同じで、名前では区別されません。`/C` で区別されるようになります（7.3 節）。
行の長さ	M80 は 132 文字、TATARA は 255 文字まで読みます。
`IXH`、`IXL`、`IYH`、`IYL` という名前のラベル	M80 では使えますが、TATARA ではこの 4 つはレジスターなので拒否されます（`IXH, IXL, IYH and IYL are registers, not names.`）。

J.4　TATARA が加えるもの

TATARA は、次の点で MSX-M80 より多くのことができます。

- **未公開命令**。`SLL`、インデックス・ハーフの `IXH` から `IYL`、`IN F,(C)`、`OUT (C),0` です（9.5 節）。MSX-M80 にはどれもありません。
- **長いシンボル名**。すべての文字が意味を持ちます（7.2 節、31 章）。
- **名前付きセグメント**と `GROUP`、そして変数でメモリを共有する**トランジェント・セグメント**（10.5 節、29 章）。
- **インクルード・ファイルの広い検索**。インクルードする側のファイルのディレクトリ、カレント・ディレクトリ、そして `TATARA` のそれぞれのディレクトリを探します（13 章、付録 I）。MSX-M80 が探すのは 1 か所だけで、MSX-C も使う環境変数 `INCLUDE` が設定されていればそのディレクトリ、設定されていなければカレント・ディレクトリです。
- すべてのファイル名で使える **MSX-DOS2 のディレクトリ**（34 章）。

- **大きなプログラム**。表を、Z80 から見える 64 KB ではなく、メモリ・マッパーに置きます（付録 G）。
- それぞれの名前のセグメントと種類を示す**シンボル表**。`/S` で表示します（19 章）。
- **何が悪いのかを示すメッセージ**。ファイルと行、そしてそこに至ったインクルード・ファイルとマクロのつながりを示します（20 章）。

J.5 そのほかの違い

コマンド・ライン MSX-M80 は **`m80 =hello/l`** のように、出力ファイルを=の前に書くコマンドを受け付けます。TATARA ではソースを先に書きます。**`tatara /l hello.as`** です（5 章）。

オブジェクト・ファイル TATARA は、TANREN がリンクする `.TRO` ファイルを書き出します（付録 F）。`.REL` ファイルは書き出せず、TANREN も読めないので、プログラムのすべてのモジュールを TATARA でアセンブルします。

リスティング レイアウトは M80 のものに近く、違いは 18 章で説明しています。

メッセージ M80 は U や V のような文字で行に印を付け、ソースの最後まで続けます。TATARA は文でメッセージを表示し、最初のエラーで止まります（20.2 節、付録 D）。

J.6 ソースの変換

MSX-M80 向けに書かれたソースを TATARA でアセンブルするには、次のようにします。

1. `.Z80`、`NAME`、`.PRINTX`、`.CREF`、`.XCREF` を削除します。
2. `SET` を `DEFL` に、`GLOBAL` を `PUBLIC` に、`EXTERNAL` を `EXTRN` に、`ENDC` を `ENDIF` に、`MACLIB`、`$INCLUDE`、`*INCLUDE` を `INCLUDE` に、`$TITLE` や`*TITLE` を `SUBTTL` に変えます。
3. `name##`を `EXTRN` の宣言に、`X'nn'` を `nnH` に、`.RADIX` のあとに書いた数を接尾辞の付いた数に置き換えます。
4. `.COMMENT` のブロック、`COMMON` のブロック、1 行に式だけの行を、表 J.1 と表 J.2 のように書き直します。

5. インクルード・ファイルがソースと同じディレクトリにもカレント・ディレクトリにもなければ、TATARA にそのディレクトリを設定します。TATARA は INCLUDE を読みません。

6. IXH、IXL、IYH、IYL という名前のラベルを付け直します。

7. 7 文字目以降だけが違う名前を探します。M80 はそれらを 1 つの名前とみなしましたが、TATARA はそうしません。

8. アセンブルします。TATARA は受け付けられない最初の行で止まり、メッセージがそのファイルと行を示すので、残りも 1 つずつ見つけられます。

9. プログラムのすべてのモジュールを TATARA でアセンブルし、TANREN でリンクします。

付録 K

インクルード・ファイル

K.1 MSXDOS.INC 360
K.2 ERRORS.INC 360
K.3 BIOS.INC 361
K.4 WORKAREA.INC 362
K.5 SUBROM.INC 362
K.6 HOOKS.INC 362
K.7 PORTS.INC 362
K.8 ASCII.INC 363
K.9 EXTBIO.INC 363
K.10 参考文献 363

`A:\TATARA\INCLUDE` にある 9 つのインクルード・ファイルには、903 の名前と 1 つのマクロが入っています。これらはアドレスや数を手で書く手間を省くためのもので、その説明はしていません。この付録では、それぞれのファイルに何が入っているかと、その名前がどこで説明されているかを示します。使い方は 14 章で説明しています。

名前は MSX-Datapack から取ったもので、ファイルに断りがない限り、その綴りのままです。それぞれの名前にはファイルの中に 1 行のコメントがあり、それぞれのファイルの先頭には、中身の使い方を説明するコメントがあります。ルーチンがレジスターに何を必要とし、何を返すかは、K.10 節に挙げた本に書かれています。

表 K.1 に、この付録の順、つまりプログラムが必要とすることの多い順に、ファイルを示します。

MSX-DOS のプログラムには `MSXDOS.INC` が必要で、たいていは `ERRORS.INC` も使います。カートリッジから動くプログラムや、BASIC から読み込むプログラムには、代わりに `BIOS.INC` と `WORKAREA.INC` が必要で、MSX2 で加わった機能には

ファイル	中身	名前	範囲
MSXDOS.INC	MSX-DOS のファンクション番号と system マクロ	93	00h–70h
ERRORS.INC	MSX-DOS のエラー・コード	76	0FFh–81h
BIOS.INC	メイン ROM のエントリー	92	0000h–017Dh
WORKAREA.INC	システムのワーク・エリア	311	0F323h–0FFFFh
SUBROM.INC	サブ ROM のエントリー	36	0089h–01F9h
HOOKS.INC	フック	117	0FD9Ah–0FFD4h
PORTS.INC	I/O ポート	56	07Ch–0FCh
ASCII.INC	制御コード	37	0–127
EXTBIO.INC	拡張 BIOS と FM BIOS	85	

表 K.1: インクルード・ファイル（この付録の順）

SUBROM.INC も使います。残りのファイルは、ハードウェアを直接扱うプログラムのためのものです。

図 K.1 に、ファイルが名前を付けているアドレスが、Z80 の 64 KB のどこにあるかを示します。

K.1　MSXDOS.INC

MSX-DOS と MSX-DOS2 のファンクション番号です。_TERM0（00h）から_REDIR（70h）までと、0005h のエントリーである BDOS、そして C に番号を入れて BDOS を呼ぶ system マクロが入っています（14.3 節）。名前は MSX-DOS2 のソースで使われている公式のもので、先頭のアンダースコアも含みます。31h より小さい番号は MSX-DOS1 にもあり、31h 以上は MSX-DOS2 だけのものです。

MSX-DOS2 のファンクションの仕様書と、MSX-Datapack の第 3 巻で説明されています。

K.2　ERRORS.INC

MSX-DOS と MSX-DOS2 のエラー・コードです。.NCOMP（0FFh）から.OVDEST（81h）まで、ディスクのエラー、ファンクションが返すエラー、プログラムを終わらせるエラー、コマンド・インタープリターのエラーの 4 つのグループに分かれています。

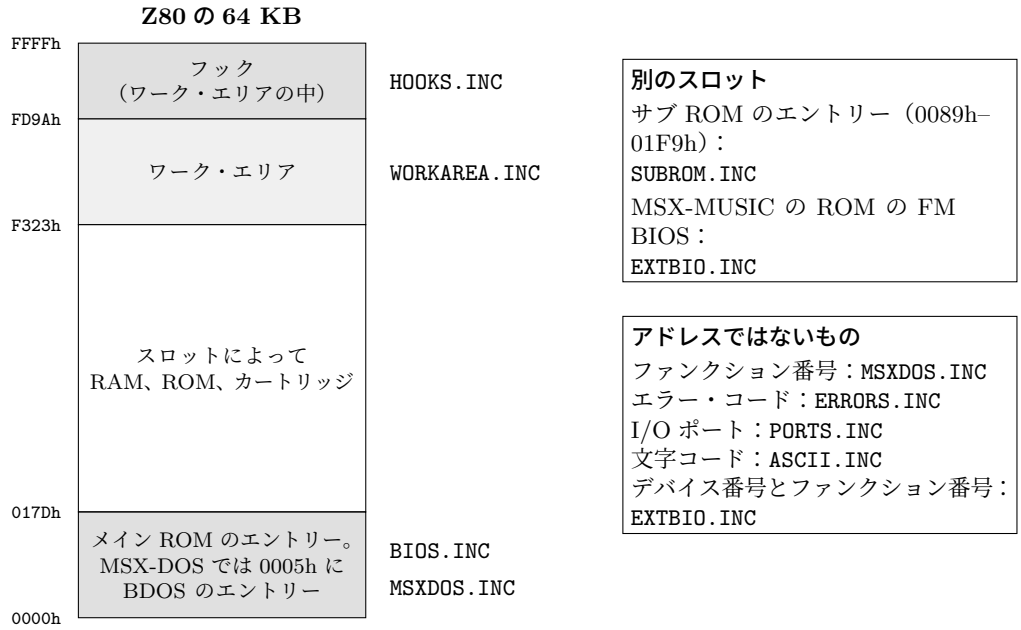

図 K.1: インクルード・ファイルが名前を付けているアドレスの位置

名前は、マニュアルの表記どおりドットで始まります。コードは 0FFh から下へ順に割り当てられたので、ファイルも下へ向かって並んでいます。

MSX-DOS2 のファンクションの仕様書と、MSX-Datapack の第 3 巻で説明されています。

K.3 BIOS.INC

メイン ROM のエントリーです。CHKRAM（0000h）から WRRES（017Dh）まで、アドレス順に並んでいます。MSX2 と MSX2+で加わったエントリーには、ファイルの中で印が付いています。MSX turbo R で加わった、CPU の切り替えと PCM のためのエントリーは入っていません。

MSX-DOS から直接呼べるのは、このうちの 5 つだけです。ほかはスロット間コールが必要です（14.4 節）。

MSX2 テクニカル・ハンドブックと、MSX-Datapack の第 2 巻で説明されています。

K.4　WORKAREA.INC

システムのワーク・エリア、つまり BIOS と BASIC が RAM に置く変数です。DISKVE（0F323h）から EXPSLT（0FFFFh）まであります。それぞれの名前のコメントはバイト単位の長さで始まります。ワーク・エリアは連続した 1 つの領域ではなく、グループの間のすき間はシステムのもので、公開されていません。

MSX2 テクニカル・ハンドブックと、MSX-Datapack の第 2 巻で説明されています。

K.5　SUBROM.INC

MSX2 で加わったサブ ROM のエントリーです。SUBROM_GRPPRT（0089h）から SUBROM_WRTCLK（01F9h）まであります。サブ ROM は別のスロットにあるので、どれも直接は呼べず、BIOS.INC にある EXTROM か SUBROM を通して呼びます。サブ ROM はメイン ROM の名前の一部を、別のアドレスの別のルーチンに使っているので、名前はすべて SUBROM_で始まります。

MSX2 テクニカル・ハンドブックと、MSX-Datapack の第 2 巻で説明されています。

K.6　HOOKS.INC

フック、つまりシステムが決まったときに呼ぶ RAM の中の 5 バイトのエントリーです。H.KEYI（0FD9Ah）から ENAINT（0FFD4h）まであります。フックはワーク・エリアの中にあり、拡張 BIOS のエントリーである EXTBIO もその中にあります。2 つのフックには、古い名前と MSX-MIDI が付けた名前の 2 つの名前があります。

MSX2 テクニカル・ハンドブックと、MSX-Datapack の第 2 巻と第 3 巻で説明されています。

K.7　PORTS.INC

I/O ポートです。IO_MUSREG（07Ch）から IO_MAPSEG（0FCh）までと、MSX-MUSIC や MSX-AUDIO のカートリッジのアドレスのように、ポートのように働くいくつかのメモリのアドレスが入っています。MSX にはポートの標準の名前がないので、これらの名前は Tatara 独自のものです。ポートは IO_と名前で、3 桁の 16 進数で書きます。

ポートは、MSX2 テクニカル・ハンドブック、MSX turbo R テクニカル・ハンドブック、MSX-Datapack の第 2 巻と第 3 巻で説明されています。

K.8 ASCII.INC

CHR_NUL（0）から CHR_DEL（127）までの制御コードと、CHR_SPACE、テキスト・ファイルの終わりを示す CHR_EOF、2 種類の引用符です。値はほかのファイルと違って 10 進数で、それぞれのコメントは、そのコードを表示したときに MSX が何をするかを示します。

MSX-Datapack の第 2 巻で説明されています。

K.9 EXTBIO.INC

拡張 BIOS です。MSX-DOS2 のマッパー・ルーチン、RS-232C、MSX-AUDIO、MSX-MIDI、MSX-JE のように、拡張 BIOS を通して答えるデバイスの、デバイス番号、ファンクション番号、ジャンプ・テーブルの中のオフセットが入っています。名前はすべて EXTBIO_とデバイスの名前で始まります。ファイルの最後には MSX-MUSIC の FM BIOS があり、その名前は FMBIOS_で始まります。これらは別のスロットにある MSX-MUSIC の ROM の中のアドレスです。

MSX-Datapack の第 2 巻と第 3 巻で説明されています。

K.10 参考文献

上の説明で挙げた本と資料です。MSX-Datapack と MSX turbo R テクニカル・ハンドブックは、日本語でだけ出版されました。

MSX2 テクニカル・ハンドブック （アスキー）。MSX2 の BIOS、サブ ROM、ワーク・エリア、フック、VDP、I/O ポート。英語版も出版されました。

MSX-Datapack （アスキー）、全 3 巻。すべてをまとめた資料で、BIOS とサブ ROM は第 2 巻、MSX-DOS2 は第 3 巻にあります。インクルード・ファイルはこれをもとに作りました。

MSX turbo R テクニカル・ハンドブック （アスキー）。MSX turbo R で加わったもの。R800、CPU の切り替え、PCM、MIDI です。

MSX-DOS2 のファンクションの仕様書 と**プログラム・インターフェースの仕様書**
（アスキー）。ファンクション、そのレジスターとエラー・コード、MSX-DOS2 でプログラムが動く仕組みです。英語の原題は *MSX-DOS2 Function Specification* と *MSX-DOS2 Program Interface Specification* です。

付録 L

用語集

この用語集には、マニュアルが定義している言葉と、定義せずに使っているいくつかの名前を、読みの五十音順に挙げます。メッセージや英語の資料と照らし合わせられるように、かっこの中に英語を示します。それぞれの項目の最後には、その言葉を紹介している節と、「参照」として関連する項目を示します。

アーカイブ（archive）いくつかのファイルをまとめて詰めたファイル。Tatara は `TATARxyz.ZIP` という 1 つのアーカイブで配布される。3.1 節。

空き（free）メモリ・マッパーのセグメントのうち、MSX-DOS2 がほかのものに割り当てていないもの。G.2 節。参照：**メモリ・マッパー**。

アセンブル（assembling）ソース・ファイルのそれぞれの行を、マシン語のバイトに変えること。2.2 節。

アデンド（addend）TATARA がオブジェクト・ファイルを書くときにすでにわかっている値の部分。穴に入れておき、何を足すかはフィックスアップが示す。F.1 節。参照：**フィックスアップ**、**穴**。

アドレス（address）メモリの位置の番号。ラベルは、その行が作る最初のバイトのアドレスを表す。2.2 節。

穴（hole）TATARA が値を完成できなかったマシン語の中の場所。オブジェクト・ファイルにそこに入るものが記録され、TANREN が埋める。2.3 節。参照：**フィックスアップ**、**リロケーション**。

アブソリュート（absolute）`EQU` で定義した名前や数のように、TATARA がアセンブルするときに完成している値のこと。8.6 節。参照：**リロケータブル**、**外部**。

アブソリュート・セグメント（absolute segment）`ASEG` で選ぶセグメント。すべてのアドレスが書いたとおりで、TANREN は何も動かさない。10.2 節。参照：**セグメント**。

R800 MSX turbo R の CPU。Z80 の命令と、いくつかの独自の命令を実行する。9.4 節。

インクルード・ガード（include guard）ファイルを 2 回取り込んでも安全にするための、ファイルの中身を囲む条件付きアセンブル。第 1 パスで定義した名前は第 2 パスでも定義されたままなので、TATARA では働かない。13.6 節。

インクルード・ファイル（include file）別のソースが `INCLUDE` で、その行がそこに書かれているかのように読むファイル。13.1 節。

インデックス・ハーフ（index half）IXH、IXL、IYH、IYL のこと。IX や IY の上位か下位のバイトで、未公開命令が 8 ビットのレジスターとして使う。9.5 節。参照：**未公開命令**。

MSX-DOS1　最初の MSX-DOS。ツールはこの上では動かない。1.4 節。参照：*MSX-DOS2*。

MSX-DOS2　サブディレクトリ、環境変数、メモリ・マッパーのルーチンを持つ MSX-DOS の版。ツールにはこれが必要。1.4 節。参照：*Nextor*。

M80　Microsoft のアセンブラー MACRO-80。TATARA はそのソースの形式にならっている。MSX では、アスキーの MSX-DOS2 TOOLS に入っていた。付録 J。

エラーメッセージ（error message）ツールが間違いを見つけたときに、止まる前に表示するメッセージ。20 章。参照：**経路、内部エラー**。

エントリー・アドレス（entry address）プログラムが実行を始めるアドレス。`END` の行で指定し、TANREN は `entry` として表示する。26.5 節。参照：**開始アドレス**。

オブジェクト・ファイル（object file）TATARA がソース・ファイルから書き出す、拡張子 `.TRO` のファイル。穴のあるマシン語と、それぞれの穴についての記録が入っている。2.1 節。参照：**モジュール**。

オペコード（opcode）命令の最初の 1 バイトか 2 バイトで、どの命令かを示す部分。9.4 節。参照：**ニーモニック**。

オペランド（operand）`de,msg` のように、オペレーションが扱うもの。行の 3 つ目のフィールド。6.2 節。

オペレーション（operation）行の命令、疑似命令、マクロ。行の 2 つ目のフィールド。6.2 節。

開始アドレス（start address）`END` のオペランドで指定する、プログラムが実行を始めるアドレス。12.6 節。参照：**エントリー・アドレス**。

拡張 BIOS（extended BIOS）あるデバイスのルーチンをどのカートリッジが持っているかをプログラムが尋ね、それを見つける仕組み。14.6 節。

カレント・ディレクトリ（current directory）コマンドを入力したときにいたディレクトリ。13.2 節。

環境変数（environment variable）MSX-DOS2 がメモリに保持し、プログラムが読める、文字列を持つ名前。ツールが使うものは付録 I で説明する。3.4 節。

外部（external）別のモジュールで定義された名前によって決まる値のこと。TANREN が仕上げる。8.6 節。参照：**アブソリュート**、**リロケータブル**、**外部シンボル**。

外部シンボル（external symbol）モジュールが使うが、別のモジュールが定義する名前。`EXTRN` で宣言する。2.3 節。参照：**パブリック・シンボル**。

繰り返しブロック（repeat block）`REPT`、`IRP`、`IRPC` で、TATARA がその場で繰り返す行のまとまり。17 章。

グループ（group）トランジェント・データ・セグメントの中の、`GROUP` の行で始まる変数のまとまり。1 つのセグメントのグループは同じメモリを共有する。10.6 節。参照：**トランジェント・データ・セグメント**。

経路（trail）エラーメッセージのあとの行で、その行に至ったインクルード・ファイル、マクロ、繰り返しブロックを示す。20 章。参照：**エラーメッセージ**。

コード・セグメント（code segment）`CSEG` で選ぶ、プログラムの命令を入れるセグメント。2.6 節。参照：**データ・セグメント**、**セグメント**。

.COM ファイル（.COM file）MSX-DOS が 0100h に読み込み、名前を入力すると実行するプログラム・ファイル。TANREN がデフォルトで書き出すもの。2.1 節。参照：**プログラム・ファイル**。

コメント（comment）セミコロンから行の終わりまでの、読む人のための文。TATARA は無視する。6.4 節。

参照（referenced）モジュールが外部の名前を使うこと。参照された名前は、どこかのモジュールが定義していなければならない。25.1 節。参照：**定義**。

式（expression）数が必要な場所に書く、数、名前、またはそれらを使った計算。8 章。

シンボル（symbol）数を表す名前。2.2 節。参照：**ラベル**、**名前**。

シンボル表（symbol table）ソースが定義したすべての名前と、その値の一覧。`/S` で表示する。19 章。

条件付きアセンブル（conditional assembly）TATARA がソースを読みながら求める条件によって、ソースの一部の行をアセンブルし、ほかの行を飛ばすこと。15 章。

ステートメント（statement）命令、疑似命令、マクロの名前のどれかを持つ行。6.1 節。

スロット（slot）MSX の中で、ROM、RAM、カートリッジが置かれる場所。Z80 の 64 KB のそれぞれのページが、1 つのスロットに切り替えられる。14.4 節。参照：**スロット間コール**。

スロット間コール（inter-slot call）CALSLT を通して行う、別のスロットにあるルーチンの呼び出し。14.4 節。参照：**スロット**。

セグメント（segment）TANREN がひとまとまりとして置く、プログラムの部分。コード・セグメント、データ・セグメント、名前付きセグメント、アブソリュート・セグメント。2.6 節。参照：**コード・セグメント**、**データ・セグメント**。

絶対パス（absolute path）ドライブか\で始まり、ファイルの場所を正確に示すファイル名。13.4 節。参照：**相対パス**。

相対ジャンプ（relative jump）飛び先のアドレスではなく、飛び先までの距離を持つ JR や DJNZ。9.3 節。

相対パス（relative path）ドライブや\で始まらず、いくつかの場所で探されるファイル名。13.4 節。参照：**絶対パス**。

即値（immediate value）かっこのないオペランド。その数そのものが命令に入る。9.2 節。

ソース・ファイル（source file）TATARA が読む、ふつう拡張子.AS の、アセンブリ言語のテキスト・ファイル。2.1 節。

第 1 パス（first pass）ソースを読む 1 回目。TATARA はすべてのラベルのアドレスと、すべての名前の値を求める。15.2 節。参照：**パス**、**第 *2* パス**。

第 2 パス（second pass）ソースを読む 2 回目。すべての値がわかっていて、TATARA はオブジェクト・ファイルとリスティングを書き出す。15.2 節。参照：**第 *1* パス**、**パス**。

使い方の画面（usage screen）ツールが/?で表示する、コマンド・ラインの要約。5.4 節。

定義（defined）名前に値があること。TANREN では、パブリックな名前は、それを宣言したモジュールが定義する。25.1 節。参照：**参照**。

展開（expansion）マクロや繰り返しブロックを 1 回使ったときにできる行。16 章。参照：**マクロ**。

ディスプレースメント（displacement）(IX+d) や (IY+d) の、−128 から 127 までの符号付きの数。インデックス・レジスターに足される。9.2 節。

データ・セグメント（data segment）DSEG で選ぶ、プログラムの変数を入れるセグメント。2.6 節。参照：**コード・セグメント**、**トランジェント・データ・セグメント**。

トランジェント・データ・セグメント（transient data segment）変数のグループが同じメモリを共有する、名前付きのデータ・セグメント。10.6 節。参照：**グループ**。

.TRO ファイル（.TRO file） TATARA が書き出し、TANREN が読むオブジェクト・ファイル。形式は付録 F で説明する。2.1 節。参照：**オブジェクト・ファイル**。

内部エラー（internal error） `internal -`で始まるメッセージ。TATARA がソースではなく、自分自身の不具合を見つけたことを示す。20.5 節。

名前（name） ラベル、`EQU` や `DEFL` で定義した値、マクロ、セグメントのこと。7 章。参照：**シンボル**。

名前付きセグメント（named segment） `CSEG` や `DSEG` のあとに書いた名前で選ぶ、コード・セグメントやデータ・セグメント。10.5 節。参照：**セグメント**。

ニーモニック（mnemonic） `LD` や `DJNZ` のように、人がオペコードに付けた名前。9.4 節。参照：**オペコード**。

Nextor MSX-DOS2 用のプログラムを実行できるシステム。そのためツールも動く。H.1 節。参照：*MSX-DOS2*。

BIOS プログラムがハードウェアを使うために呼ぶ、メイン ROM の中のルーチン。`BIOS.INC` がそのエントリーに名前を付けている。14.4 節。参照：**スロット間コール**。

バナー（banner） ツールが最初に表示する行。名前とバージョン、著作権表示、ウェブのアドレス。5.4 節。参照：**要約行**。

パス（pass） ソースを 1 回読むこと。TATARA はソースを 2 回読む。15.2 節。参照：**第 *1* パス**、**第 *2* パス**。

パブリック・シンボル（public symbol） モジュールが定義し、ほかのモジュールに使わせる名前。`PUBLIC` で宣言する。2.3 節。参照：**外部シンボル**。

パラメーター（parameter） マクロの本体の中で、使うたびに引数に置き換えられる名前。16.1 節。参照：**引数**。

引数（argument） マクロを使うときに名前のあとに書く値。本体の中の対応するパラメーターと置き換わる。16.1 節。参照：**パラメーター**。

BDOS プログラムが MSX-DOS のファンクションを呼ぶための 0005h のエントリー。`MSXDOS.INC` が名前を付けている。14.3 節。

ビルド・ファイル（build file） 例をビルドするコマンドを実行するバッチ・ファイル、`BUILD.BAT`。4.6 節。

フィックスアップ（fixup） オブジェクト・ファイルの中の、穴に何を足すかを示す記録。セグメントが置かれた場所か、外部の名前の値。F.1 節。参照：**アデンド**、**穴**。

フィールド（field）行の 4 つの部分、つまりラベル、オペレーション、オペランド、コメントのそれぞれ。6.2 節。

フック（hook）システムが決まったときに呼ぶ RAM の中の 5 バイト。プログラムが乗っ取ることができる。14.6 節。

BLOAD　マシン語のファイルを読み込む MSX-BASIC のコマンド。そのファイルは拡張子が `.BIN` で、BLOAD ヘッダーで始まる。26.4 節。参照：*BLOAD* **ヘッダー**。

BLOAD ヘッダー（BLOAD header）`BLOAD` 用のファイルの先頭の 7 バイト。読み込む場所と開始する場所を示す。TANREN は `/B` でこれを書く。26.4 節。

プログラム・ファイル（program file）`.COM` ファイルのような、TANREN が書き出す完成したプログラム。2.1 節。

変数シンボル（variable symbol）`DEFL` で定義した、値を変えられる名前。12.4 節。

ページ（page）Z80 の 64 KB の 4 分の 1。0000h、4000h、8000h、C000h からの範囲。G.1 節。参照：**マッパーのセグメント**。

ページ見出し（page heading）リスティングの各ページの先頭の行。タイトル、バージョン、日付、ページ番号、サブタイトル。18.2 節。

本体（body）`MACRO` と `ENDM` の間の行。マクロを使うたびに、その場所に置かれる。16.1 節。参照：**マクロ**。

マクロ（macro）行のまとまりに付けた名前。名前を使うたびに、その行に置き換えられる。16 章。参照：**本体**、**展開**。

マシン語（machine code）Z80 が読んで実行するバイト。2.2 節。

マッパーのセグメント（mapper segment）メモリ・マッパーの 16 KB の 1 区画。Z80 はそれぞれのページに 1 つずつ見る。G.1 節。参照：**メモリ・マッパー**、**ページ**。

未公開命令（undocumented instructions）ザイログが公表しなかったが、どの Z80 でも動く命令。TATARA は SLL、インデックス・ハーフ、IN F,(C)、OUT (C),0 を受け付ける。9.1 節。参照：**インデックス・ハーフ**。

メモリ・マッパー（memory mapper）16 KB のセグメントに分けられた RAM で、Z80 は同時に 4 つを見る。ツールは表をここに置く。G.1 節。参照：**マッパーのセグメント**。

文字定数（character constant）引用符で囲んだ 1 文字か 2 文字で、その文字のコードを数として使うもの。8.3 節。

モジュール（module）1 つのソース・ファイルのコードを、オブジェクト・ファイルに入っている形で呼ぶもの。2.3 節。

文字列（string）引用符で囲んだ文字で、1 文字を 1 バイトとして格納する。11.2 節。

優先順位（precedence）式の演算子を計算する順序。8.5 節。

要約行（summary line）ツールが最後に表示する行。ソースがどこで終わったか、何を書き出したかを示す。5.4 節。参照：**バナー**。

ラベル（label）行の先頭に書き、その行が作る最初のバイトのアドレスを表す名前。2.2 節。参照：**シンボル**。

リスティング（listing）TATARA がアセンブルしたとおりのソースを、各行のバイトとともにページに分けて印刷したもの。5.5 節。参照：**ページ見出し**。

リロケーション（relocation）アドレスが決まったあとで、マシン語の穴を埋めること。2.4 節。参照：**穴**。

リロケータブル（relocatable）セグメントが置かれる場所によって決まるアドレスのこと。TANREN が仕上げる。2.3 節。参照：**アブソリュート**、**外部**。

リンク（linking）オブジェクト・ファイルをまとめてプログラム・ファイルにすること。モジュールを置き、穴を埋め、結果を書き出す。2.4 節。参照：**リロケーション**。

リンク・ファイル（link file）TANREN のコマンド・ラインに入力する内容を書いておくテキスト・ファイル。4.7 節。

レコード（record）オブジェクト・ファイルの 1 つの部分。タイプのバイト、長さ、中身からなる。F.1 節。

レジスター・コード（register code）命令のバイトの中でのレジスターの番号。B の 0 から A の 7 まで。B.1 節。

ロケーション・カウンター（location counter）セグメントでそれまでにアセンブルしたバイトの数。ラベルはその値を取り、$はそれを読む。10.1 節。

ワーク・エリア（work area）システムが変数を置く、0F323h から上の RAM。14.6 節。参照：**フック**。

ワード（word）2 バイトの値。Z80 がすべての 16 ビットの値を格納するのと同じく、下位バイトが先。11.3 節。

索引

.LALL (疑似命令), 135, 136, 238, 240, 286, 291
.LFCOND (疑似命令), 136, 286, 291, 292
.LIST (疑似命令), 134, 237, 286, 292, 354
.SALL (疑似命令), 135, 136, 240, 286, 292
.SFCOND (疑似命令), 136, 286, 292
.TFCOND (疑似命令), 136, 286, 292, 354
.XALL (疑似命令), 135, 240, 286, 292
.XLIST (疑似命令), 134, 237, 286, 292
/? (オプション), 36, 38, 163, 164, 368
/B (オプション), 161–164, 197–199, 202, 206, 257, 262, 322, 370
/C (オプション), 4, 36, 44, 49–51, 99, 110, 120, 142, 192, 193, 196, 222, 243, 245–249, 321, 331, 334, 356
/D (オプション), 166, 322
/D: (オプション), 162, 163, 183, 186, 320
/F (オプション), 37, 139, 144, 146, 155
/H (オプション), 37, 139, 144, 147
/L (オプション), 36, 39, 129, 144, 151, 236, 246
/M (オプション), 37, 139, 144, 146, 147, 155, 163, 180, 187–189, 193–195, 220, 226
/O (オプション), 166, 322
/O: (オプション), 26, 162, 163, 170, 176, 197–199, 273, 278, 349
/P (オプション), 36, 39, 129, 134, 146, 151, 166, 245, 322
/P: (オプション), 162, 163, 182, 198, 203, 256, 264
/Q (オプション), 27, 33, 36, 37, 141, 150, 161, 163–165, 202, 208, 210, 219, 226, 253, 263, 271, 278, 316, 322, 323
/R (オプション), 163, 329, 334–336
/S (オプション), 36, 50, 139, 143, 144, 226, 245, 357, 367
/V (オプション), 21, 22, 36, 37, 163, 164, 279

ASEG (疑似命令), 71, 131, 198, 252, 256, 261, 264, 284, 286, 354, 365

COND (疑似命令), 106, 286, 293, 354
CSEG (疑似命令), 14, 71, 72, 284, 286, 287, 316, 354, 367, 369

DB (疑似命令), 55, 57, 77–80, 115, 131, 181, 265, 283, 287, 293, 313, 354, 356
DC (疑似命令), 77, 79, 283, 287, 317, 354
DEFB (疑似命令), 80, 287, 293, 354
DEFL (疑似命令), 48, 49, 57, 83, 84, 124, 142, 143, 244, 284, 287, 291, 292, 317, 319, 354, 357, 369, 370
DEFM (疑似命令), 80, 287, 293, 354
DEFS (疑似命令), 72, 80, 287, 293, 354
DEFW (疑似命令), 80, 287, 293, 354
DS (疑似命令), 72, 74, 77, 80, 181, 183, 184, 204, 205, 228, 265, 283, 287, 293, 354
DSEG (疑似命令), 14, 71, 72, 284, 287, 316, 354, 368, 369
DW (疑似命令), 55, 77–80, 181, 283, 287, 293, 314, 317, 354

ELSE (疑似命令), 105, 110, 111, 136, 285, 288, 289, 292, 293, 317, 354

END (疑似命令), 25, 37, 81, 85, 86, 94, 161, 164, 201, 206, 216, 252, 279, 284, 288, 323, 334, 336, 354, 366
ENDIF (疑似命令), 105, 106, 110, 111, 136, 285, 288, 289, 292, 293, 317, 354, 355, 357
ENDM (疑似命令), 114, 119, 120, 123, 127, 128, 147, 285, 288, 290, 291, 293, 317, 318, 354, 370
ENTRY (疑似命令), 83, 288, 293, 354
EQU (疑似命令), 47, 48, 57, 58, 73, 82–84, 93, 97, 114, 131, 139, 142, 143, 190, 226, 284, 287, 288, 291, 292, 316, 317, 319, 341, 354, 365, 369
EXITM (疑似命令), 113, 118, 120, 123, 127, 285, 288, 293, 317, 354
EXT (疑似命令), 83, 288, 293, 354
EXTRN (疑似命令), 10, 59, 68, 83, 85, 86, 90, 108, 131, 142, 187, 191, 284, 288, 289, 293, 314, 316–319, 354, 355, 357, 367

GROUP (疑似命令), 73, 74, 76, 284, 287–289, 292, 316, 318, 320, 356, 367

IF (疑似命令), 105, 106, 108, 110, 111, 116, 136, 150, 285, 286, 289, 290, 292, 293, 317, 320, 354
IF1 (疑似命令), 109, 285, 289, 319
IF2 (疑似命令), 109, 285, 289
IFB (疑似命令), 109, 115, 118, 285, 289, 355
IFDEF (疑似命令), 108, 285, 289, 290, 355
IFDIF (疑似命令), 109, 118, 285, 289, 354
IFE (疑似命令), 105, 106, 285, 289, 293, 354
IFF (疑似命令), 106, 289, 293, 354
IFIDN (疑似命令), 109, 118, 285, 289
IFNB (疑似命令), 109, 115, 118, 285, 290, 355
IFNDEF (疑似命令), 93, 94, 108, 109, 285, 290
IFT (疑似命令), 106, 290, 293, 354
INCLUDE (環境変数), 356, 358
INCLUDE (疑似命令), 5, 12, 89–92, 94, 95, 97, 152, 173, 237, 239, 268, 270, 284, 290, 292, 317, 318, 348, 349, 354, 357, 366
IRP (疑似命令), 113, 123–125, 152, 235, 285, 288, 290, 293, 317, 354, 367
IRPC (疑似命令), 113, 123, 126, 152, 235, 240, 285, 288, 290, 293, 317, 354, 367

LOCAL (疑似命令), 52, 53, 113, 117, 118, 120, 123, 126, 127, 142, 146, 147, 238, 285, 290, 293, 318, 319, 354

MACRO (疑似命令), 114, 119, 120, 147, 150, 285, 290, 292, 316–318, 354, 370

ORG (疑似命令), 71, 72, 131, 198, 252, 255, 256, 284, 290, 292, 319, 354
ORG 4000h (疑似命令), 261

PAGE (疑似命令), 132, 133, 137, 286, 291, 293, 316, 354
PATH (環境変数), 18, 20–23, 276, 280, 281, 347, 348, 350
PUBLIC (疑似命令), 10, 44, 68, 83–85, 90, 141–143, 151, 187, 284, 288, 291, 293, 316–318, 353–355, 357, 369

REPT (疑似命令), 113, 123, 124, 147, 152, 235, 285, 288, 291, 293, 319, 320, 354, 367

SUBTTL (疑似命令), 131, 133, 286, 291, 318, 354, 357

TANREN (環境変数), 5, 21, 173–177, 322, 327, 347, 349, 350
TATARA (環境変数), 5, 18, 20–25, 27, 89, 91, 92, 175, 219, 220, 229, 240, 245, 256, 263, 267–272, 277, 279, 290, 319, 326, 347–350, 356, 358
TITLE (疑似命令), 131, 133, 286, 291, 354

空き, 340
アセンブル, 8
アデンド, 330
アドレス, 9
穴, 9
アブソリュート, 59, 190, 314
アブソリュート・セグメント, 71
アーカイブ, 17
インクルード・ガード, 93
インクルード・ファイル, 89
インデックス・ハーフ, 65
エラーメッセージ, 149
エントリー・アドレス, 201
オブジェクト・ファイル, 7
オブジェクト・ファイル, 329
オペコード, 64
オペランド, 42
オペレーション, 42
開始アドレス, 85
拡張 BIOS, 103
カレント・ディレクトリ, 91
環境変数, 20
外部, 314
外部シンボル, 10
繰り返しブロック, 123
グループ, 73
経路, 149, 315
コメント, 44
コード・セグメント, 14
参照, 188
式, 55
シンボル, 8
シンボル表, 139
条件付きアセンブル, 105
ステートメント, 41
スロット間コール, 100
セグメント, 14
絶対パス, 92, 176
相対ジャンプ, 63
相対パス, 92, 176
即値, 62
ソース・ファイル, 7
第 1 パス, 107
第 2 パス, 107
使い方の画面, 38
定義, 188
展開, 113
ディスプレースメント, 63
データ・セグメント, 14
トランジェント・データ・セグメント, 73
内部エラー, 155
名前, 47
名前付きセグメント, 72
ニーモニック, 64
Nextor, 343
バナー, 37
パス, 94, 107
パブリック・シンボル, 10
パラメーター, 114
引数, 114
ビルド・ファイル, 27
フィックスアップ, 330
フィールド, 42
フック, 102
BLOAD ヘッダー, 199
プログラム・ファイル, 7
変数シンボル, 83
ページ, 339
ページ見出し, 131
本体, 114
マクロ, 113
マシン語, 8
未公開命令, 61
メモリ・マッパー, 339
文字定数, 56
モジュール, 9
文字列, 78, 333
優先順位, 58
要約行, 37
ラベル, 9
リスティング, 39
リロケーション, 10
リロケータブル, 9, 314
リンク, 10

リンク・ファイル, 28
リンク・ファイル, 165, 167
レコード, 330
レジスター・コード, 296
ロケーション・カウンター, 57, 69
ワーク・エリア, 103
ワード, 78, 333