ARM® コンパイラツールチェーン

バージョン 5.01

fromelf イメージ変換ユーティリティの使用

ARM DUI 0477GJ (ID ０５１６１２)

ARM コンパイラツールチェーン
fromelf イメージ変換ユーティリティの使用

リリース情報

本書には以下の変更が加えられています。

変更履歴

日付	発行	機密保持ステータス	変更点
2010 年 5 月 28 日	A	非機密扱い	ARM コンパイラツールチェーン v4.1 リリース
2010 年 9 月 30 日	B	非機密扱い	ARM コンパイラツールチェーン v4.1 のアップデート 1
2011 年 1 月 28 日	C	非機密扱い	ARM コンパイラツールチェーン v4.1 パッチ 3 のアップデート 2
2011 年 4 月 30 日	D	非機密扱い	ARM コンパイラツールチェーン v5.0 リリース
2011 年 7 月 29 日	E	非機密扱い	ARM コンパイラツールチェーン v5.0 のアップデート 1
2011 年 9 月 30 日	F	非機密扱い	ARM コンパイラツールチェーン v5.01 リリース
2012 年 2 月 29 日	G	非機密扱い	ARM コンパイラツールチェーン v5.01 リリースマニュアルの更新

著作権

本書に記載されている製品は、今後も継続的に開発・改良の対象となります。本書に含まれる製品およびその利用方法についての情報は、ARM が利用者の利益のために提供するものです。したがって当社では、製品の市販性または利用の適切性を含め、暗示的・明示的に関係なく一切の任を負いません。

本書は、本製品の利用者をサポートすることだけを目的としています。本書に記載されている情報の使用、情報の誤りまたは省略、あるいは本製品の誤使用によって発生したいかなる損失・損傷についても、ARM は一切責任を負いません。

ARM という用語が使用されている場合、"ARM または必要に応じてその子会社 " を指します。

機密保持ステータス

本書は非機密扱いであり、本書を使用、複製、および開示する権利は、ARM および ARM が本書を提供した当事者との間で締結した契約の条項に基づいたライセンスの制限により異なります。

製品ステータス

本書の情報は最終版であり、開発済み製品に対応しています。

Web アドレス

http://www.arm.com

目次
ARM コンパイラツールチェーン fromelf イメージ変換ユーティリティの使用

第 1 章
表記規則とフィードバック

以下では、表記規則とフィードバックの方法について説明します。

表記規則

以下の表記規則を使用しています。

`monospace` コマンド、ファイル名、プログラム名、ソースコードなど、キーボードから入力可能なテキストを示しています。

`monospace` コマンドまたはオプションに使用可能な略語を示します。コマンド名またはオプション名をすべて入力する代わりに、下線部分の文字だけを入力することができます。

`monospace italic`

コマンドまたは関数の引数で、特定の値に置き換えることが可能なものを示しています。

`monospace bold`

サンプルコード以外に使用される言語キーワードを示しています。

italic 重要事項、重要用語、相互参照、引用箇所を斜体で記載しています。

bold メニュー名などのユーザインタフェース要素を太字で記載しています。また、適宜記述リスト内の重要箇所と ARM® プロセッサの信号名にも太字を用いています。

本製品に関するフィードバック

本製品についてのご意見やご提案がございましたら、以下の情報を添えて購入元までお寄せ下さい。

- お名前と会社名
- 製品のシリアル番号
- 製品のリリース情報
- ご使用のプラットフォームの詳細（ハードウェアプラットフォーム、オペレーティングシステムの種類とバージョンなど）
- 問題を再現するサイズの小さな独立したサンプルコード
- 操作の目的と実際の動作に関する詳しい説明
- 使用したコマンド（コマンドラインオプションを含む）
- 問題を例示するサンプル出力
- ツールのバージョン情報（バージョン番号、ビルド番号を含む）

内容に関するフィードバック

内容に関するご意見につきましては、電子メールを errata@arm.com まで送信して下さい。その際には、以下の内容を記載して下さい。

- タイトル
- 文書番号（ARM DUI 0477GJ）
- オンラインでご覧の場合は、該当するトピック名
- PDF 版の文書をご覧の場合は、問題のあるページ番号
- 問題点の簡潔な説明

また、補足すべき点や改善すべき点についての全般的なご提案もお待ちしております。

ARM では、技術情報記事や *FAQ* の拡充と共に、ドキュメントに対する更新と訂正を ARM Information Center にて定期的に行っております。

その他の情報

- ARM Information Center, http://infocenter.arm.com/help/index.jsp
- ARM Technical Support Knowledge Articles, http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html
- ARM サポートおよびメンテナンス , http://www.arm.com/support/services/support-maintenance.php
- ARM 用語集 , http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html

第 2 章
fromelf イメージ変換ユーティリティの概要

以下の各トピックでは、ARM コンパイラツールチェーンに付属の `fromelf` イメージ変換ユーティリティの概要について説明します。

タスク

- *fromelf コマンドのヘルプの取得*（2-5 ページ）

概念

- *fromelf イメージ変換ユーティリティについて*（2-2 ページ）
- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *fromelf コマンドラインオプションの目的別一覧*（2-7 ページ）.

2.1 fromelf イメージ変換ユーティリティについて

イメージ変換ユーティリティ fromelf では、以下の操作を実行できます。

- コンパイラ、アセンブラ、およびリンカにより作成された ARM ELF オブジェクトおよびイメージファイルを処理します。
- armar によって作成されたアーカイブ内のすべての ELF ファイルを処理し、処理したファイルを必要に応じて別のアーカイブに出力します。
- ELF イメージを、ROM ツールで使用したり直接メモリにロードしたりできる他の形式に変換します。使用できる形式は以下のとおりです。
  - プレーンバイナリ
  - Motorola 32 ビット S レコード形式
  - Intel Hex 32 ビット形式
  - バイト指向（Verilog メモリモデル）16 進形式
  - ELF。例えば、ELF イメージからデバッグ情報を削除するために、ELF 形式で保存し直すことができます。
- サードパーティに配信されるイメージとオブジェクトの*知的所有権*（IP）を保護します。
- 逆アセンブルの出力やシンボルリストなどの入力ファイルに関する情報を標準出力（stdout）やテキストファイルに出力します。

注

デバッグ情報を持たないイメージを生成した場合、fromelf ユーティリティには、以下の制限があります。

- イメージを別の形式のファイルに変換できません。
- 分かりやすい逆アセンブルリストを生成できません。

2.1.1 関連項目

概念

- *fromelf の実行モード*（2-3 ページ）
- *fromelf 使用時の注意事項*（2-4 ページ）
- *fromelf によるイメージおよびオブジェクトに含まれるコードの保護*（3-10 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *fromelf コマンドラインオプションの目的別一覧*（2-7 ページ）

2.2 fromelf の実行モード

fromelf には、以下の実行モードがあります。

- ファイルを ELF として保存し直すための ELF モード（--elf）
- オブジェクトまたはイメージファイルに関する情報を出力するためのテキストモード（--text など）
- 形式変換モード（--bin、--m32、--i32、--vhx）

2.2.1 関連項目

参照

- *--bin*（4-6 ページ）
- *--elf*（4-31 ページ）
- *--i32*（4-45 ページ）
- *--m32*（4-57 ページ）
- *--text*（4-76 ページ）
- *--vhx*（4-79 ページ）.

2.3 fromelf 使用時の注意事項

以下の点に注意して下さい。

- fromelf を使用して複数のロード領域を含んでいる ELF イメージを --bin、--m32、--i32、または --vhx の任意のオプションでバイナリ形式に変換する場合は、fromelf によって *destination* という名前の出力ディレクトリが作成され、入力イメージの各ロード領域に対する 1 つのバイナリ出力ファイルが生成されます。出力ファイルは fromelf によって *destination* ディレクトリに配置されます。

  注

  複数のロード領域の場合、対応するロード領域内の最初の空でない実行領域の名前がファイル名に使用されます。

  --m32combined または --i32combined オプションのいずれかを使用して、複数のロード領域を含んでいる ELF イメージを変換する場合は、fromelf によって *destination* という名前の出力ディレクトリが作成され、入力イメージのすべてのロード領域に対する 1 つのバイナリ出力ファイルが生成されます。出力ファイルは *destination* ディレクトリに配置されます。

  ELF イメージは、複数のロード領域を定義しているスキャッタファイルを使用してビルドされた場合などに、複数のロード領域を保持します。

- fromelf を使用する場合、以下の制限があります。
  - — --base オプションを使用して Motorola S レコード形式または Intel Hex 形式の出力のベースアドレスを変更する場合を除き、イメージの構造やアドレスを変できません。
  - — 分散ロードされる ELF イメージを、別の形式の非分散ロードのイメージに変換することはできません。構造やアドレスに関する情報は、すべてリンク時にリンカに渡す必要があります。

2.3.1 関連項目

参照

- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--bin*（4-6 ページ）
- *--i32*（4-45 ページ）
- *--i32combined*（4-46 ページ）
- *--m32*（4-57 ページ）
- *--m32combined*（4-58 ページ）
- *--vhx*（4-79 ページ）.

2.4 fromelf コマンドのヘルプの取得

主なコマンドラインオプションの一覧を表示するには、--help オプションを使用します。

これは、オプションやファイルを指定しない場合のデフォルトの動作です。

2.4.1 例

ヘルプ情報を表示するには、以下のように入力します。

```
fromelf --help
```

2.4.2 関連項目

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--help*（4-42 ページ）.

2.5 fromelf のコマンドライン構文

fromelf コマンドラインの構文を以下に示します。

```
fromelf [options] input_file
```

options fromelf コマンドラインのオプション。

input_file 処理対象の ELF ファイルまたはライブラリファイル。いくつかのオプションを使用すると、複数の入力ファイルを指定できます。

2.5.1 関連項目

概念

『ARM コンパイラツールチェーンの概要』:

- 第 2 章 *ARM コンパイラツールチェーンの概要*.

参照

- *fromelf コマンドラインオプションの目的別一覧*（2-7 ページ）
- *input_file*（4-51 ページ）.

2.6 fromelf コマンドラインオプションの目的別一覧

fromelf コマンドラインのオプションを以下に示します。

ビルド属性の出力形式の制御

- *--decode_build_attributes*（4-21 ページ）
- *--dump_build_attributes*（4-30 ページ）
- *--extract_build_attributes*（4-36 ページ）.

出力ファイル内にデバッグ情報を含めるかどうかの制御

- *--debugonly*（4-20 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

出力ファイルの診断情報の制御

以下のオプションを使用して、出力ファイルの診断情報を制御します。

- *--compare=option[,option,...]*（4-14 ページ）
- *--continue_on_error*（4-16 ページ）
- *--diag_error=tag[,tag,...]*（4-24 ページ）
- *--diag_remark=tag[,tag,...]*（4-25 ページ）
- *--diag_style={arm|ide|gnu}*（4-26 ページ）
- *--diag_suppress=tag[,tag,...]*（4-27 ページ）
- *--diag_warning=tag[,tag,...]*（4-28 ページ）
- *--ignore_section=option[,option,...]*（4-47 ページ）
- *--ignore_symbol=option[,option,...]*（4-48 ページ）
- *--relax_section=option[,option,...]*（4-65 ページ）
- *--relax_symbol=option[,option,...]*（4-66 ページ）
- *--show_cmdline*（4-71 ページ）.

コマンドラインヘルプ

- *--help*（4-42 ページ）
- *--version_number*（4-78 ページ）
- *--vsn*（4-81 ページ）.

ファイルからのコマンドライン引数の取得

- *--via=file*（4-80 ページ）.

イメージの内容に影響するさまざまな要因の制御

- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--cad*（4-11 ページ）
- *--cadcombined*（4-13 ページ）
- *--cpu=list*（4-17 ページ）
- *--cpu=name*（4-18 ページ）
- *--device=list*（4-22 ページ）
- *--device=name*（4-23 ページ）
- *--disassemble*（4-29 ページ）
- *--emit=option[,option,...]*（4-32 ページ）

- *--expandarrays*（4-35 ページ）
- *--fieldoffsets*（4-37 ページ）
- *--fpu=list*（4-39 ページ）
- *--fpu=name*（4-40 ページ）
- *--globalize=option[,option,...]*（4-41 ページ）
- *--hide=option[,option,...]*（4-43 ページ）
- *--hide_and_localize=option[,option,...]*（4-44 ページ）
- *--in_place*（4-49 ページ）
- *--interleave=option*（4-53 ページ）
- *--linkview*、*--no_linkview*（4-55 ページ）
- *--localize=option[,option,...]*（4-56 ページ）
- *--qualify*（4-63 ページ）
- *--rename=option[,option,...]*（4-67 ページ）
- *--select=select_options*（4-68 ページ）
- *--show=option[,option,...]*（4-69 ページ）
- *--show_and_globalize=option[,option,...]*（4-70 ページ）
- *--source_directory=path*（4-72 ページ）
- *--strip=option[,option,...]*（4-73 ページ）
- *--symbolversions*、*--no_symbolversions*（4-75 ページ）.

フローティングライセンスの取得

- *--licretry*（4-54 ページ）.

出力形式の制御

- *--bin*（4-6 ページ）
- *--bincombined*（4-7 ページ）
- *--bincombined_base=address*（4-8 ページ）
- *--bincombined_padding=size,num*（4-9 ページ）
- *--elf*（4-31 ページ）
- *--i32*（4-45 ページ）
- *--i32combined*（4-46 ページ）
- *--m32*（4-57 ページ）
- *--m32combined*（4-58 ページ）
- *--output=destination*（4-60 ページ）
- *--vhx*（4-79 ページ）
- *--widthxbanks*（4-83 ページ）.

情報の表示制御

- *--info=topic[,topic,...]*（4-50 ページ）
- *--only=section_name*（4-59 ページ）
- *--text*（4-76 ページ）
- *-w*（4-82 ページ）.

イメージおよびオブジェクトに含まれる知的所有権の保護

- *--privacy*（4-61 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

プロジェクトテンプレートの使用の制御

- *--project=filename?--no_project*（4-62 ページ）
- *--reinitialize_workdir*（4-64 ページ）
- *--workdir=directory*（4-85 ページ）.

第 3 章
fromelf ユーティリティの使用

以下の各トピックでは、ARM コンパイラツールチェーンに付属の `fromelf` イメージ変換ユーティリティの使用方法について説明します。

タスク

3.1 ELF イメージから Intel Hex 32 ビット形式への変換

Intel Hex 32 ビット形式の出力を生成するには、以下のいずれかのオプションを使用します。

- `--i32`
- `--i32combined`

`--i32` を指定すると、イメージ内のロード領域ごとに 1 つの出力ファイルが生成されます。`--i32combined` を指定すると、複数のロード領域を含むイメージ用に 1 つの出力ファイルが生成されます。

注

これらのオプションは `--output` と組み合わせて使用する必要があります。

この出力のベースアドレスは、`--base` オプションを使用して指定できます。

3.1.1 例

ELF ファイル `infile.axf` を Intel Hex-32 形式のファイル（`outfile.bin` など）に変換するには、以下のように入力します。

```
fromelf --i32 --output=outfile.bin infile.axf
```

2 つのロード領域を持つイメージファイル（`infile2.axf`）から、開始アドレスを `0x1000` とし、単一の出力ファイル（`outfile2.bin`）を作成するには、以下のように入力します。

```
fromelf --i32combined --base=0x1000 --output=outfile2.bin infile2.axf
```

3.1.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--i32*（4-45 ページ）
- *--i32combined*（4-46 ページ）
- *--output=destination*（4-60 ページ）.

3.2 ELF イメージから Motorola 32 ビット形式への変換

Motorola 32 ビット形式（32 ビット S レコード形式）の出力を作成するには、以下のいずれかのオプションを使用します。

- `--m32`
- `--m32combined`

`--m32` を指定すると、イメージ内のロード領域ごとに 1 つの出力ファイルが生成されます。`--m32combined` を指定すると、複数のロード領域を含むイメージ用に 1 つの出力ファイルが生成されます。

注

これらのオプションは `--output` と組み合わせて使用する必要があります。

この出力のベースアドレスは、`--base` オプションを使用して指定できます。

3.2.1 例

ELF ファイル `infile.axf` を Motorola 32 ビット形式のファイル（`outfile.bin` など）に変換するには、以下のように入力します。

```
fromelf --m32 --output=outfile.bin infile.axf
```

2 つのロード領域を持つイメージファイル（`infile2.axf`）から、開始アドレスを `0x1000` とし、Motorola 32 ビット形式の単一の出力ファイル（`outfile2.bin`）を作成するには、以下のように入力します。

```
fromelf --m32combined --base=0x1000 --output=outfile2.bin infile2.axf
```

3.2.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--m32*（4-57 ページ）
- *--m32combined*（4-58 ページ）
- *--output=destination*（4-60 ページ）.

3.3 ELF イメージからプレーンバイナリ形式への変換

各ロード領域に対して 1 つのプレーンバイナリ形式の出力ファイルを生成するには、--bin オプションを使用します。*--widthxbanks* オプションを使用して、このオプションによって生成される出力を複数ファイルに分割できます。

プレーンバイナリ形式の出力を生成するには、--bincombined オプションを使用します。複数のロード領域を含むイメージ用に 1 つの出力ファイルが生成されます。デフォルトでは、メモリ内の最初のロー領域の開始アドレスがベースアドレスとして使用されます。fromelf ユーティリティにより、ロード領域間にパディングが挿入され、お互いに対して正しい相対オフセットになることが保証されます。この方法でード領域を分けることにより、メモリに出力ファイルをロードし、ベースアドレスから始まるように正しく整列することができます。

ベースアドレスおよびパディングのデフォルト値を変更するには、--bincombined オプションを --bincombined_base および --bincombined_padding と組み合わせて使用します。

これらのオプションを使用する場合は、以下の点に注意して下さい。

- --output オプションは、--bin および --bincombined と組み合わせて使用する必要があります。
- --bincombined では、大きなアドレス空間を挟んで存在する 2 つのロード領域を定義するスキャッタファイルを使用した場合、パディングが大部分を占めることになって、最終的なバイナリが非常に大きくなる場合があります。例えば、0x00000000 というアドレスに存在するサイズ 0x100 バイトのロード領域と、0x30000000 というアドレスに存在するロード領域がある場合、パディングのサイズは 0x2FFFFFF00 バイトとなります。

3.3.1 サンプル

ELF ファイルをプレーンバイナリファイル（outfile.bin など）に変換するには、以下のように入力します。

```
fromelf --bin --output=out.bin in.axf
```

開始アドレス 0x1000 でロードできるバイナリファイルを生成するには、以下のように入力します。

```
fromelf --bincombined --bincombined_base=0x1000 --output=out.bin in.axf
```

プレーンバイナリ形式の出力を生成し、32 ビットのワード 0x12345678 のコピーでロード領域の間のスペースを埋めるには、以下のように入力します。

```
fromelf --bincombined --bincombined_padding=4,0x12345678 --output=out.bin in.axf
```

3.3.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--bin*（4-6 ページ）
- *--bincombined*（4-7 ページ）

- *--bincombined_base=address*（4-8 ページ）
- *--bincombined_padding=size,num*（4-9 ページ）
- *--output=destination*（4-60 ページ）
- *--widthxbanks*（4-83 ページ）.

3.4 ELF イメージからバイト指向（Verilog メモリモデル）16 進形式への変換

バイト指向（Verilog メモリモデル）16 進形式の出力を生成するには、--vhx オプションを使用します。この形式は、*ハードウェア記述言語*（HDL）シミュレータのメモリモデルへのロードに適しています。*--widthxbanks* オプションを使用して、このオプションによって生成される出力を複数ファイルに分割できます。

注

これらのオプションは --output と組み合わせて使用する必要があります。

3.4.1 サンプル

ELF ファイル infile.axf をバイト指向 16 進形式のファイル（outfile.bin など）に変換するには、以下のように入力します。

```
fromelf --vhx --output=outfile.bin infile.axf
```

8 ビットのメモリバンクを 2 つ持つイメージファイル multiload.axf から、regions ディレクトリに複数の出力ファイルを作成するには、以下のように入力します。

```
fromelf --vhx --8x2 multiload.axf --output=regions
```

3.4.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--output=destination*（4-60 ページ）
- *--vhx*（4-79 ページ）
- *--widthxbanks*（4-83 ページ）.

3.5 出力ファイル内にデバッグ情報を含めるかどうかの制御

コードセクションまたはデータセクションの内容を削除するには、--debugonly オプションを使用します。これにより、出力ファイルにデバッグに必要な情報（デバッグセクション、シンボルテーブル、ストリングテーブルど）のみが含まれるようになります。セクションヘッダは、シンボルのターゲットとして機能する必要があるため保持されます。

注

このオプションは --elf と組み合わせて使用する必要があります。--elf を使用する関係上、--output も必要となります。

3.5.1 例

ELF ファイル infile.axf から、デバッグ情報のみを含んでいる ELF ファイル debugout.axf を作成するには、以下のように入力します。

```
fromelf --elf --debugonly --output=debugout.axf infile.axf
```

3.5.2 関連項目

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--debugonly*（4-20 ページ）
- *--elf*（4-31 ページ）
- *--output=destination*（4-60 ページ）.

3.6 ELF 形式のファイルの逆アセンブル

逆アセンブルされたイメージを stdout に表示するには、--disassemble オプションを使用します。このオプションを --output *destination* オプションと組み合わせて使用した場合、armasm により出力ファイルを再アセンブルできます。

このオプションを使用すると、ELF イメージファイルまたは ELF オブジェクトファイルを逆アセンブルできます。

注

この出力は、--emit=code および --text -c の出力とは異なります。

3.6.1 例

ARM1176JZF-S™ プロセッサ用の ELF ファイル infile.axf を逆アセンブルしてソースファイル outfile.asm を作成するには、以下のように入力します。

```
fromelf --cpu=ARM1176JZF-S --disassemble --output=outfile.asm infile.axf
```

3.6.2 関連項目

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--cpu=name*（4-18 ページ）
- *--disassemble*（4-29 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--interleave=option*（4-53 ページ）
- *--output=destination*（4-60 ページ）
- *--text*（4-76 ページ）.

『アセンブラの使用』:

- *第 7 章 アセンブラの使用*.

3.7 アーカイブ内の ELF ファイルの処理

アーカイブ内のすべての ELF ファイルまたはこれらのファイルのサブセットを処理できます。処理されたファイルは、処理されていないファイル共に別のアーカイブに出力されます。

以降の例に、test.a アーカイブ内の ELF ファイルを処理する方法を示します。このアーカイブには、以下のファイルが含まれています。

```
bmw.o
bmw1.o
call_c_code.o
newtst.o
shapes.o
strmtst.o
```

3.7.1 アーカイブ内のすべてのファイルを処理する例

この例では、アーカイブ内のすべてのファイルから、すべてのデバッグ、コメント、メモ、およびシンボルが削除されます。

```
fromelf --elf --strip=all test.a -o strip_all/
```

この例を実行すると、test.a という名前の出力アーカイブが、サブディレクトリ strip_all に作成されます。

3.7.2 アーカイブ内のファイルのサブセットを処理する例

アーカイブ内の shapes.o ファイルと strmtst.o ファイルからのみ、すべてのデバッグ、コメント、メモ、およびシンボルを削除するには、以下のように入力します。

```
fromelf --elf --strip=all test.a(s*.o) -o subset/
```

この例を実行すると、test.a という名前の出力アーカイブが、サブディレクトリ subset に作成されます。このアーカイブには、処理されたファイルが処理されていない残りのファイルと共に格納されます。

アーカイブ内の bmw.o、bmw1.o、および newtst.o ファイルを処理するには、以下のように入力します。

```
fromelf --elf --strip=all test.a(??w*) -o subset/
```

3.7.3 アーカイブ内のファイルの逆アセンブルされたバージョンを表示する例

アーカイブ内の call_c_code.o の逆アセンブルされたバージョンを表示するには、以下のように入力します。

```
fromelf --disassemble test.a(c*)
```

3.7.4 関連項目

参照

- *--disassemble*（4-29 ページ）
- *--elf*（4-31 ページ）
- *input_file*（4-51 ページ）
- *--output=destination*（4-60 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

3.8 fromelf によるイメージおよびオブジェクトに含まれるコードの保護

イメージおよびオブジェクトをサードパーティに配布する場合、そこに含まれるコードを保護する必要があります。fromelf には、コードを保護するための --strip オプションと --privacy オプションが用意されています。これらのオプションを使用すると、オブジェクトまたはイメージ内のシンボル名を削除したりあいまいにしたできます。どちらのオプションを選択するかは、削除する情報の量次第です。これらのオプションの効果は、オブジェクトファイルとイメージで異なります。

注

これらのオプションは --elf と組み合わせて使用する必要があります。--elf を使用する関係上、--output も必要となります。

3.8.1 イメージファイル内のコードの保護

イメージファイルの場合：

表 3-1 イメージファイルに対する fromelf の --privacy オプションと --strip オプションの効果

オプション	ローカルシンボル	セクション名	マッピングシンボル	ビルド属性
fromelf --elf --privacy	シンボルテーブル全体が削除されます。 .comment セクション名が削除されます。fromelf --text の出力結果には、これが [Anonymous Section] としてマークされます。 セクション名にデフォルト値が付けられます。例えば、コードセクション名は '.text' に変わります。			
fromelf --elf --strip=symbols	シンボルテーブル全体が削除されます。 セクション名は変更されません。			
fromelf --elf --strip=localsymbols	削除されます。	そのまま残ります。	削除されます。	そのまま残ります。

3.8.2 オブジェクトファイル内のコードの保護

オブジェクトファイルの場合：

表 3-2 オブジェクトファイルに対する fromelf の --privacy オプションと --strip オプションの効果

オプション	ローカルシンボル	セクション名	マッピングシンボル	ビルド属性
`fromelf --elf --privacy`	機能を損なうことなく削除できるローカルシンボルは削除されます。 再配置のターゲットなど、削除できないシンボルは維持されます。このようなシンボルについては、名前が削除されます。`fromelf --text` の出力結果には、これらが `[Anonymous Symbol]` としてマークされます。	セクション名にデフォルト値が付けられます。例えば、コードセクション名は `'.text'` に変わります。	そのまま残ります。	そのまま残ります。
`fromelf --elf --strip=symbols`	機能を損なうことなく削除できるローカルシンボルは削除されます。 再配置のターゲットなど、削除できないシンボルは維持されます。このようなシンボルについては、名前が削除されます。`fromelf --text` の出力結果には、これらが `[Anonymous Symbol]` としてマークされます。	セクション名は変更されません。	そのまま残ります。	そのまま残ります。
`fromelf --elf --strip=localsymbols`	機能を損なうことなく削除できるローカルシンボルは削除されます。 再配置のターゲットなど、削除できないシンボルは維持されます。このようなシンボルについては、名前が削除されます。`fromelf --text` の出力結果には、これらが `[Anonymous Symbol]` としてマークされます。	セクション名は変更されません。	そのまま残ります。	そのまま残ります。

3.8.3 例

シンボルテーブル全体が削除され、各セクション名が変更された新しい ELF 実行可能イメージを生成するには、以下のように指定します。

```
fromelf --elf --privacy --output=outfile.axf infile.axf
```

3.8.4 関連項目

概念

『リンカの使用』:

- *マッピングシンボルについて*（7-3 ページ）.

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--elf*（4-31 ページ）
- *--output=destination*（4-60 ページ）
- *--privacy*（4-61 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

『リンカリファレンス』:

- *--list_mapping_symbols*、*--no_list_mapping_symbols*（2-104 ページ）
- *--locals*、*--no_locals*（2-106 ページ）
- *--privacy*（2-128 ページ）.

3.9 ELF 形式ファイルの詳細の出力

--emit オプションを使用すると、テキスト出力に含める ELF オブジェクトの要素を指定できます。出力には、ELF ヘッダおよびセクション情報が含まれます。各要素はコンマで区切って指定できます。

注

--emit のいくつかのオプションは、--text オプションを使用して指定できます。

3.9.1 データセクションの出力例

ELF ファイル infile.axf のデータセクションの内容を出力するには、以下のように入力します。

```
fromelf --emit=data infile.axf
```

3.9.2 再配置情報の出力例

ELF ファイル infile2.axf の再配置情報およびダイナミックセクションの内容を出力するには、以下のように入力します。

```
fromelf --emit=relocation_tables,dynamic_segment infile2.axf
```

3.9.3 関連項目

参照

- *fromelf のコマンドライン構文*（2-6 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--text*（4-76 ページ）.

3.10 実行可能な ELF イメージ内のシンボルの場所を fromelf を使用して調べる方法

ELF イメージファイル内のシンボルの場所を調べるには、--text -s -v オプションを使用して、各セグメントとセクションヘッダのシンボルテーブルおよび詳細情報を表示します。以下に例を示します。

```
fromelf --text -s -v s.axf
```

シンボルテーブルを見ると、シンボルが配置されているセクションを確認できます。

3.10.1 例

以下の手順に従います。

1. 以下のソースコードを含む s.c という名前のファイルを作成します。

```
long long altstack[10] __attribute__ ((section ("STACK"), zero_init));

int main()
{
    return sizeof(altstack);
}
```

2. ソースをコンパイルします。

```
armcc -c s.c -o s.o
```

3. オブジェクト s.o をリンクします。STACK シンボルは残して下さい。

```
armlink --keep=s.o(STACK) s.o --output=s.axf
```

4. fromelf コマンドを実行して、各セグメントおよびセクションヘッダのシンボルテーブルと詳細情報を表示します。

```
fromelf --text -s -v s.o
```

5. 以下のような fromelf の出力結果から、STACK シンボルと altstack シンボルを探します。

```
...
** Section #9

    Name        : .symtab
    Type        : SHT_SYMTAB (0x00000002)
    Flags       : None (0x00000000)
    Addr        : 0x00000000
    File Offset : 2792 (0xae8)
    Size        : 2896 bytes (0xb50)
    Link        : Section 10 (.strtab)
    Info        : Last local symbol no = 115
    Alignment   : 4
    Entry Size  : 16    Symbol table .symtab (180 symbols, 115 local)

      #  Symbol Name                Value      Bind  Sec  Type  Vis  Size
==============================================================================
...
     16  STACK                      0x00008228   Lc    2  Sect  De   0x50
...

    179  altstack                   0x00008228   Gb    2  Data  Hi   0x50
...
```

スタックが配置されているセクションが、Sec 列に表示されています。この例では、セクション 2 です。

6. 確認したシンボルのセクションを fromelf の出力結果から探します。以下に例を示します。

```
...
==================================

** Section #2

    Name         : ER_ZI
    Type         : SHT_NOBITS (0x00000008)
    Flags        : SHF_ALLOC + SHF_WRITE (0x00000003)
    Addr         : 0x000081c8
    File Offset  : 508 (0x1fc)
    Size         : 176 bytes (0xb0)
    Link         : SHN_UNDEF
    Info         : 0
    Alignment    : 8
    Entry Size   : 0

==================================
...
```

以上の結果から、シンボルは ZI 実行領域に存在することがわかります。

3.10.2 関連項目

タスク

- *リンク時のシンボルの場所を調べる方法*（6-6 ページ）.

参照

- *--text*（4-76 ページ）.

『コンパイラリファレンス』:

- *-c*（3-36 ページ）
- *-o filename*（3-163 ページ）.

『リンカリファレンス』:

- *--keep=section_id*（2-89 ページ）
- *--output=filename*（2-115 ページ）.

第 4 章
fromelf コマンドリファレンス

以下の各トピックでは、ARM コンパイラツールチェーンに付属の `fromelf` イメージ変換ユーティリティのコマンドラインオプションの使用方法について説明します。

- *--symbolversions、--no_symbolversions*（4-75 ページ）
- *--text*（4-76 ページ）
- *--version_number*（4-78 ページ）
- *--vhx*（4-79 ページ）
- *--via=file*（4-80 ページ）
- *--vsn*（4-81 ページ）
- *-w*（4-82 ページ）
- *--widthxbanks*（4-83 ページ）
- *--workdir=directory*（4-85 ページ）.

fromelf のコマンドライン構文（2-6 ページ）も参照して下さい。

4.1 --base [[*object_file*::]*load_region_ID*=]*num*

このオプションにより、Motorola S レコード形式または Intel Hex ファイル形式の 1 つ以上のロード領域に指定されたベースアドレスを変更できます。

4.1.1 制約条件

このオプションは、--i32、--i32combined、--m32、または --m32combined のいずれかの出力形式と組み合わせて使用する必要があります。

4.1.2 構文

--base [[*object_file*::]*load_region_ID*=]*num*

各項目には以下の意味があります。

object_file オプションの ELF 入力ファイル。

load_region_ID

オプションのロード領域。これには、ロード領域に属する実行領域のシンボル名か、最初の領域を示す場合には #0 などのゼロベースのロード領域番号のいずれかを指定できます。

num 10 進数値または 16 進数値。

以下のことができます。

- *object_file* 引数および *load_region_ID* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --base オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

出力ファイル内にエンコードされるすべてのアドレスは、ベースアドレス *num* から開始されます。--base オプションが指定されていない場合、ベースアドレスはロード領域のアドレスから取得されます。

表 4-1 --base の使用例

--base 0	10 進数値
--base 0x8000	16 進数値
--base #0=0	最初のロード領域のベースアドレス
--base foo.o::*=0	foo.o のすべてのロード領域のベースアドレス
--base #0=0,#1=0x8000	最初および 2 番目のロード領域のベースアドレス

4.1.3 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--i32*（4-45 ページ）
- *--i32combined*（4-46 ページ）

- *--m32*（4-57 ページ）
- *--m32combined*（4-58 ページ）.

4.2 --bin

このオプションは、各ロード領域に対して 1 つのプレーンバイナリ形式の出力ファイルを作成します。*--widthxbanks* オプションを使用して、このオプションによって生成される出力を複数ファイルに分割できます。

4.2.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは --output と組み合わせて使用する必要があります。

4.2.2 例

ELF ファイルをプレーンバイナリファイル（outfile.bin など）に変換するには、以下のように入力します。

```
fromelf --bin --output=outfile.bin infile.axf
```

4.2.3 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--output=destination*（4-60 ページ）
- *--widthxbanks*（4-83 ページ）.

4.3 --bincombined

このオプションは、プレーンバイナリ形式の出力を作成します。複数のロード領域を含むイメージ用に 1 つの出力ファイルが生成されます。デフォルトでは、メモリ内の最初のロード領域の開始アドレスがベースアドレスとして使用されます。fromelf ユーティリティにより、ロード領域間にパディングが挿入され、お互いに対して正しい相対オフセットになることが保証されます。この方法でード領域を分けることにより、メモリに出力ファイルをロードし、ベースアドレスから始まるように正しく整列することができます。

ベースアドレスおよびパディングのデフォルト値を変更するには、このオプションを --bincombined_base および --bincombined_padding と組み合わせて使用します。

4.3.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは --output と組み合わせて使用する必要があります。

4.3.2 --bincombined 使用時の注意事項

ベースアドレスのデフォルト値を変更するには、このオプションを --bincombined_base と組み合わせて使用します。

パディングのデフォルト値は 0xFF です。パディングのデフォルト値を変更するには、このオプションを --bincombined_padding と組み合わせて使用します。

大きなアドレス空間を挟んで存在する 2 つのロード領域を定義するスキャッタファイルを使用した場合、パディングが大部分を占めることになって、最終的なバイナリが非常に大きくなる場合があります。例えば、0x00000000 というアドレスに存在するサイズ 0x100 バイトのロード領域と、0x30000000 というアドレスに存在するロード領域がある場合、パディングのサイズは 0x2FFFFF00 バイトとなります。

ロード領域の間隔が大きい場合は、*--widthxbanks* オプションでバイナリファイルを複数のファイルに分割するなど、何か他の方法を用いることを推奨します。

4.3.3 関連項目

概念

『リンカの使用』:

- *入力セクション、出力セクション、領域、およびプログラムセグメント*（4-5 ページ）.

参照

- *--bincombined_base=address*（4-8 ページ）
- *--bincombined_padding=size,num*（4-9 ページ）
- *--output=destination*（4-60 ページ）
- *--widthxbanks*（4-83 ページ）.

4.4 --bincombined_base=*address*

このオプションにより、--bincombined 出力モードで使用するベースアドレスを下げることができます。生成される出力ファイルは、指定されたアドレスから始まるメモリにロードできます。

4.4.1 制約条件

このオプションは --bincombined と組み合わせて使用する必要があります。
--bincombined を省略した場合は、警告メッセージが表示されます。

4.4.2 構文

--bincombined_base=*address*

各項目には以下の意味があります。

address イメージをロードする開始アドレス。

- 指定されたアドレスが最初のロード領域の始めよりも低い場合は、fromelf ユーティリティにより、出力ファイルの始めにパディングが追加されます。
- 指定されたアドレスが最初のロード領域の始めよりも高い場合は、fromelf ユーティリテによりエラーが出力されます。

4.4.3 デフォルト

デフォルトでは、メモリ内の最初のロード領域の開始アドレスがベースアドレスとして使用されます。

4.4.4 例

```
--bincombined --bincombined_base=0x1000
```

4.4.5 関連項目

概念

『リンカの使用』:

- *入力セクション、出力セクション、領域、およびプログラムセグメント*（4-5 ページ）.

参照

- *--bincombined*（4-7 ページ）
- *--bincombined_padding=size,num*（4-9 ページ）.

4.5 --bincombined_padding=*size*,*num*

--bincombined 出力モードで使用されるパディングに、デフォルトとは異なる値を指定できます。

4.5.1 制約条件

このオプションは --bincombined と組み合わせて使用する必要があります。
--bincombined を省略した場合は、警告メッセージが表示されます。

4.5.2 構文

--bincombined_padding=*size*,*num*

各項目には以下の意味があります。

size バイト、ハーフワード、またはワードを指定するための 1 バイト、2 バイト、または 4 バイト。

num パディングに使用する値。指定されたサイズに収まりきらない大きな値を指定すると、警告メッセージが表示されます。

注

fromelf ユーティリティは、入力ファイルの適切なエンディアンに 2 バイトおよび 4 バイトのパディング値が指定されていると想定します。例えば、ビグエンディアンの ELF ファイルをバイナリに変換する場合は、指定されたパディング値がビッグエンディアンのワードまたはハーフワードとしてわれます。

4.5.3 デフォルト

デフォルトは --bincombined_padding=1,0xFF です。

4.5.4 例

以下に、--bincombined_padding の使用例を示します。

--bincombined --bincombined_padding=4,0x12345678

この例では、プレーンバイナリ形式の出力が作成され、ロード領域の間のスペースが 32 ビットのワード 0x12345678 のコピーで埋められます。

--bincombined --bincombined_padding=2,0x1234

この例では、プレーンバイナリ形式の出力が作成され、ロード領域の間のスペースが 16 ビットのハーフワード 0x1234 のコピーで埋められます。

--bincombined --bincombined_padding=2,0x01

ビッグエンディアンメモリに対してこの例を指定すると、ロード領域の間のスペースが 0x0100 で埋められます。

4.5.5 関連項目

参照

- *--bincombined*（4-7 ページ）

- *--bincombined_base=address*（4-8 ページ）.

4.6 --cad

このオプションは、バイナリ出力を含む C 配列定義または C++ 配列定義を作成します。別のアプリケーションのソースコードには、それぞれの配定義を使用できます。例えば、組み込みオペレーティングシステムなど、別のアプリケーションのアドレス空間にはイメージを組み込むことがきます。

イメージ内のロード領域が 1 つの場合、デフォルトで出力が標準出力（stdout）に送られます。ファイルに出力を保存するには、--output オプションをファイル名と共に使用します。

イメージに複数のロード領域がある場合は、--output オプションをディレクトリ名と共に使用する必要があります。完全パス名を指定しない限り、パスは現在のディレクトリに対する相対パスになます。指定したディレクトリに、各ロード領域用のファイルが作成されます。各ファイルの名前は、対応する実行領域の名前です。

イメージ内のロード領域ごとに 1 つの出力ファイルを生成するには、このオプションを --output と組み合わせて使用します。

4.6.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

4.6.2 例

以下に、--cad の使用例を示します。

- 1 つのロード領域を持つイメージの配列定義を作成するには、以下のように入力します。

```
fromelf --cad myimage.axf
unsigned char LR0[] = {
    0x00,0x00,0x00,0xEB,0x28,0x00,0x00,0xEB,0x2C,0x00,0x8F,0xE2,0x00,0x0C,0x90,0xE8,
    0x00,0xA0,0x8A,0xE0,0x00,0xB0,0x8B,0xE0,0x01,0x70,0x4A,0xE2,0x0B,0x00,0x5A,0xE1,
    0x00,0x00,0x00,0x1A,0x20,0x00,0x00,0xEB,0x0F,0x00,0xBA,0xE8,0x18,0xE0,0x4F,0xE2,
    0x01,0x00,0x13,0xE3,0x03,0xF0,0x47,0x10,0x03,0xF0,0xA0,0xE1,0xAC,0x18,0x00,0x00,
    0xBC,0x18,0x00,0x00,0x00,0x30,0xB0,0xE3,0x00,0x40,0xB0,0xE3,0x00,0x50,0xB0,0xE3,
    0x00,0x60,0xB0,0xE3,0x10,0x20,0x52,0xE2,0x78,0x00,0xA1,0x28,0xFC,0xFF,0xFF,0x8A,
    0x82,0x2E,0xB0,0xE1,0x30,0x00,0xA1,0x28,0x00,0x30,0x81,0x45,0x0E,0xF0,0xA0,0xE1,
    0x70,0x00,0x51,0xE3,0x66,0x00,0x00,0x0A,0x64,0x00,0x51,0xE3,0x38,0x00,0x00,0x0A,
    0x00,0x00,0xB0,0xE3,0x0E,0xF0,0xA0,0xE1,0x1F,0x40,0x2D,0xE9,0x00,0x00,0xA0,0xE1,
.
.
.
    0x3A,0x74,0x74,0x00,0x43,0x6F,0x6E,0x73,0x74,0x72,0x75,0x63,0x74,0x65,0x64,0x20,
    0x41,0x20,0x23,0x25,0x64,0x20,0x61,0x74,0x20,0x25,0x70,0x0A,0x00,0x00,0x00,0x00,
    0x44,0x65,0x73,0x74,0x72,0x6F,0x79,0x65,0x64,0x20,0x41,0x20,0x23,0x25,0x64,0x20,
    0x61,0x74,0x20,0x25,0x70,0x0A,0x00,0x00,0x0C,0x99,0x00,0x00,0x0C,0x99,0x00,0x00,
    0x50,0x01,0x00,0x00,0x44,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
```

- 複数のロード領域を持つイメージには、以下のコマンドを使用すると、ディレクトリ *root*\myprojects\multiload\load_regions にロード領域ごとのファイルが作成されます。

```
cd root\myprojects\multiload
fromelf --cad image_multiload.axf --output load_regions
```

image_multiload.axf に実行領域 EXEC_ROM および RAM が含まれる場合、ファイル EXEC_ROM および RAM が load_regions サブディレクトリに作成されます。

4.6.3 関連項目

タスク

『リンカの使用』:

- 第 8 章 *スキャッタファイルの使用*.

概念

『リンカの使用』:

- *入力セクション、出力セクション、領域、およびプログラムセグメント*（4-5 ページ）.

参照

- *--cadcombined*（4-13 ページ）
- *--output=destination*（4-60 ページ）.

4.7 --cadcombined

このオプションは、バイナリ出力を含む C 配列定義または C++ 配列定義を作成します。別のアプリケーションのソースコードには、それぞれの配定義を使用できます。例えば、組み込みオペレーティングシステムなど、別のアプリケーションのアドレス空間にはイメージを組み込むことがきます。

デフォルトでは、出力が標準出力（stdout）に送られます。ファイルに出力を保存するには、--output オプションをファイル名と共に使用します。

4.7.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

4.7.2 例

以下のコマンドを使用すると、ファイル load_regions.c がディレクトリ *root*\myprojects\multiload に作成されます。

```
cd root\myprojects\multiload
fromelf --cadcombined image_multiload.axf --output load_regions.c
```

4.7.3 関連項目

タスク

『リンカの使用』:

- 第 8 章 *スキャッタファイルの使用*.

参照

- *--cad*（4-11 ページ）
- *--output=destination*（4-60 ページ）

4.8 --compare=*option*[,*option*,...]

このオプションは、2 つの入力ファイルを比較し、その違いをテキスト形式のリストに出力します。入力ファイルは、2 つの ELF ファイルまたは 2 つのライブラリファイルの、同じ種類とする必要があります。ライブラリファイルはメンバ単位で比較され、その違いが出力で連結して示されます。

2 つの入力ファイルのすべての違いは、--relax_section オプションを使用して警告に格下げされている場合を除いて、エラーとして報告されます。

4.8.1 構文

--compare=*option*[,*option*,...]

option には以下のいずれかを指定できます。

section_sizes

ライブラリファイルの各 ELF ファイルまたは各 ELF メンバのすべてのセクションのサイズを比較します。

section_sizes::*object_name*

object_name と一致する名前の ELF オブジェクト内のすべてのセクションのサイズを比較します。

section_sizes::*section_name*

section_name と一致する名前のすべてのセクションのサイズを比較します。

セクション ライブラリファイルの各 ELF ファイルまたは各 ELF メンバのすべてのセクションのサイズおよび内容を比較します。

sections::*object_name*

object_name と一致する名前の ELF オブジェクト内のすべてのセクションのサイズと内容を比較します。

sections::*section_name*

section_name と一致する名前のすべてのセクションのサイズおよび内容を比較します。

function_sizes

ライブラリファイルの各 ELF ファイルまたは各 ELF メンバのすべての関数のサイズを比較します。

function_sizes::*object_name*

object_name と一致する名前の ELF オブジェクト内のすべての関数のサイズを比較します。

function_size::*function_name*

function_name と一致する名前のすべての関数のサイズを比較します。

global_function_sizes

ライブラリファイルの各 ELF ファイルまたは各 ELF メンバのすべてのグローバル関数のサイズを比較します。

global_function_sizes::*function_name*

function_name と一致する名前の ELF オブジェクト内のすべてのグローバル関数のサイズを比較します。

以下のことができます。

- *section_name* 引数、*function_name* 引数、および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --compare オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.8.2 関連項目

参照

- *--ignore_section=option[,option,...]*（4-47 ページ）
- *--ignore_symbol=option[,option,...]*（4-48 ページ）
- *--relax_section=option[,option,...]*（4-65 ページ）
- *--relax_symbol=option[,option,...]*（4-66 ページ）.

4.9 --continue_on_error

すべてのエラーを報告した上で実行を続行します。

このオプションの代わりに --diag_warning=error を使用して下さい。

4.9.1 関連項目

参照

- *--diag_warning=tag[,tag,...]*（4-28 ページ）.

4.10 --cpu=list

このオプションは、--cpu=*name* と一緒に使用可能なサポートされている ARM アーキテクチャとプロセッサ名を一覧にします。

4.10.1 関連項目

参照

- *--cpu=name*（4-18 ページ）.

4.11 --cpu=*name*

このオプションは、特定の ARM アーキテクチャまたはプロセッサの逆アセンブリを選択します。これにより、fromelf が入力ファイルで見つかった命令を処理する方法に影響が出ます。

4.11.1 構文

```
--cpu=name
```

name は ARM アーキテクチャまたはプロセッサの名前です。

4.11.2 例

ARM1176JZF-S™ プロセッサの逆アセンブリを選択するには、以下のように入力します。

```
--cpu=ARM1176JZF-S
```

4.11.3 関連項目

参照

- *--cpu=list*（4-17 ページ）
- *--device=list*（4-22 ページ）
- *--device=name*（4-23 ページ）
- *--disassemble*（4-29 ページ）
- *--info=topic[,topic,...]*（4-50 ページ）
- *--text*（4-76 ページ）.

『アセンブラリファレンス』:

- *--cpu=name*（2-9 ページ）
- *--device=name*（2-11 ページ）.

『コンパイラリファレンス』:

- *--cpu=name*（3-55 ページ）
- *--device=name*（3-76 ページ）.

『リンカリファレンス』:

- *--cpu=name*（2-38 ページ）
- *--device=name*（2-43 ページ）.

4.12 --datasymbols

このオプションは、シンボル定義をインターリーブするようにデータセクションの出力情報を変更します。

このオプションは、--text -d と共に指定する必要があります。

4.12.1 関連項目

参照

- *--text*（4-76 ページ）.

4.13 --debugonly

このオプションは、コードセクションまたはデータセクションの内容を削除します。これにより、出力ファイルにデバッグに必要な情報（デバッグセクション、シンボルテーブル、ストリングテーブルなど）のみが含まれるようになります。セクションヘッダは、シンボルのターゲットとして機能する必要があるため保持されます。

4.13.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.13.2 関連項目

参照

- *--elf*（4-31 ページ）.

4.14 --decode_build_attributes

このオプションは、標準のビルド属性の場合は人間が読める形式でビルド属性セクションの内容を出力し、非標準のビルド属性の場合は生の 16 進形式で出力します。

注

標準のビルド属性については、『*Application Binary Interface for the ARM Architecture*』を参照して下さい。

4.14.1 制約条件

このオプションは、テキストモードでのみ使用できます。

4.14.2 例

--decode_build_attributes の出力例を以下に示します。

```
** Section #12 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)
    Size   : 69 bytes

    'aeabi' file build attributes:
    0x000000:   05 41 52 4d 37 54 44 4d 49 00 06 02 08 01 11 01    .ARM7TDMI.......
    0x000010:   12 02 14 02 17 01 18 01 19 01 1a 01 1e 03 20 02    .............. .
    0x000020:   41 52 4d 00                                        ARM.
        Tag_CPU_name = "ARM7TDMI"
        Tag_CPU_arch = ARM v4T (=2)
        Tag_ARM_ISA_use = ARM instructions were permitted to be used (=1)
        Tag_ABI_PCS_GOT_use = Data are imported directly (=1)
        Tag_ABI_PCS_wchar_t = Size of wchar_t is 2 (=2)
        Tag_ABI_FP_denormal = This code was permitted to require that the sign of a flushed-to-zero number be
preserved in the sign of 0 (=2)
        Tag_ABI_FP_number_model = This code was permitted to use only IEEE 754 format FP numbers (=1)
        Tag_ABI_align8_needed = Code was permitted to depend on the 8-byte alignment of 8-byte data items (=1)
        Tag_ABI_align8_preserved = Code was required to preserve 8-byte alignment of 8-byte data objects (=1)
        Tag_ABI_enum_size = Enum values occupy the smallest container big enough to hold all values (=1)
        Tag_ABI_optimization_goals = Optimized for small size, but speed and debugging illusion preserved (=3)
        Tag_compatibility = 2, "ARM"

    'ARM' file build attributes:
    0x000000:   04 01 12 01                                        ....
```

4.14.3 関連項目

参照

- *--dump_build_attributes*（4-30 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--extract_build_attributes*（4-36 ページ）.

その他の情報

- 『*Application Binary Interface for the ARM Architecture*』, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0036-/index.html

4.15 --device=list

このオプションは、--device=*name* オプションと一緒に使用可能なサポートされているデバイス名を一覧表示します。

注

このオプションの使用は廃止される予定です。

4.15.1 関連項目

参照

- *--device=name*（4-23 ページ）.

4.16 --device=*name*

このオプションにより、CPU 名の代わりにマイクロコントローラまたは SoC（System-on-Chip）デバイス名を指定できます。これにより、fromelf が入力ファイルで見つかった命令を処理する方法に影響が出ます。これは、コンパイラでサポートされている形式と同じです。

各デバイスには、CPU および浮動小数点ユニット（FPU）のデフォルト値があります。しかし、--device オプションの後に --fpu オプションを指定することにより、コマンドラインから FPU をオーバーライドできます。

CPU および FPU の実装の詳細については、デバイスのマニュアルを参照して下さい。

注

このオプションの使用は廃止される予定です。

4.16.1 構文

--device=*name*

name は特定のデバイス名です。

使用可能なデバイスの完全なリストを取得するには、--device=list オプションを使用します。

4.16.2 関連項目

参照

- *--cpu=list*（4-17 ページ）
- *--cpu=name*（4-18 ページ）
- *--device=list*（4-22 ページ）
- *--fpu=list*（4-39 ページ）
- *--fpu=name*（4-40 ページ）.

『アセンブラリファレンス』:

- *--cpu=name*（2-9 ページ）
- *--device=name*（2-11 ページ）.

『コンパイラリファレンス』:

- *--cpu=name*（3-55 ページ）
- *--device=name*（3-76 ページ）.

『リンカリファレンス』:

- *--cpu=name*（2-38 ページ）
- *--device=name*（2-43 ページ）.

4.17 --diag_error=*tag[,tag,...]*

このオプションを使用して、特定のタグがある診断メッセージにエラーの重大度を設定します。

4.17.1 構文

```
--diag_error=tag[,tag,...]
```

tag には以下のいずれかを指定できます。

- エラーの重大度を設定する診断メッセージ番号
- `warning`（すべての警告をエラーとして扱う場合）

4.17.2 関連項目

参照

- *--diag_remark=tag[,tag,...]*（4-25 ページ）
- *--diag_style={arm|ide|gnu}*（4-26 ページ）
- *--diag_suppress=tag[,tag,...]*（4-27 ページ）
- *--diag_warning=tag[,tag,...]*（4-28 ページ）.

4.18 --diag_remark=*tag[,tag,...]*

このオプションを使用して、特定のタグがある診断メッセージに注釈の重大度を設定します。

4.18.1 構文

--diag_remark=*tag[,tag,...]*

tag は診断メッセージ番号のコンマ区切りのリストです。

4.18.2 関連項目

参照

- *--diag_error=tag[,tag,...]*（4-24 ページ）
- *--diag_style={arm|ide|gnu}*（4-26 ページ）
- *--diag_suppress=tag[,tag,...]*（4-27 ページ）
- *--diag_warning=tag[,tag,...]*（4-28 ページ）.

4.19 --diag_style={arm|ide|gnu}

このオプションを使用すると、診断メッセージの表示に使用する形式を指定できます。

4.19.1 構文

`--diag_style=`*`string`*

`string` には以下のいずれかを指定できます。

`arm` ARM 形式でメッセージを表示します。

`ide` エラーのある行の行番号と文字数を表示します。これらの値は括弧に囲まれて表示されます。

`gnu` GNU で使用される形式でメッセージを表示します。

4.19.2 デフォルト

デフォルトは `--diag_style=arm` です。

4.19.3 関連項目

参照

- *--diag_error=tag[,tag,...]*（4-24 ページ）
- *--diag_remark=tag[,tag,...]*（4-25 ページ）
- *--diag_suppress=tag[,tag,...]*（4-27 ページ）
- *--diag_warning=tag[,tag,...]*（4-28 ページ）.

4.20 --diag_suppress=*tag*[,*tag*,...]

このオプションを指定すると、指定されたタグの診断メッセージが無効になります。

4.20.1 構文

--diag_suppress=*tag*[,*tag*,...]

tag には以下のいずれかを指定できます。

- 非表示にする診断メッセージ番号
- error（すべてのエラーを非表示にする場合）
- warning（すべての警告を非表示にする場合）

4.20.2 関連項目

参照

- *--diag_error=tag[,tag,...]*（4-24 ページ）
- *--diag_remark=tag[,tag,...]*（4-25 ページ）
- *--diag_style={arm|ide|gnu}*（4-26 ページ）
- *--diag_warning=tag[,tag,...]*（4-28 ページ）.

4.21 --diag_warning=*tag[,tag,...]*

このオプションを使用して、特定のタグがある診断メッセージに警告の重大度を設定します。

4.21.1 構文

```
--diag_warning=tag[,tag,...]
```

tag には以下のいずれかを指定できます。

- 警告の重大度を設定する診断メッセージ番号
- `error`（すべてのエラーを警告に降格する場合）

4.21.2 関連項目

参照

- *--diag_error=tag[,tag,...]*（4-24 ページ）
- *--diag_remark=tag[,tag,...]*（4-25 ページ）
- *--diag_style={arm|ide|gnu}*（4-26 ページ）
- *--diag_warning=tag[,tag,...]*.

4.22 --disassemble

このオプションは、逆アセンブルされたイメージを標準出力（stdout）に表示します。このオプションを --output *destination* と組み合わせて使用した場合、armasm により出力ファイルを再アセンブルできます。

このオプションを使用すると、ELF イメージファイルまたは ELF オブジェクトファイルを逆アセンブルできます。

注

この出力は、--emit=code および --text -c の出力とは異なります。

4.22.1 関連項目

参照

- *--cpu=name*（4-18 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--interleave=option*（4-53 ページ）
- *--output=destination*（4-60 ページ）
- *--text*（4-76 ページ）.

4.23 --dump_build_attributes

このオプションは、ビルド属性セクションの内容を生の 16 進形式で出力します。

4.23.1 制約条件

このオプションは、テキストモードでのみ使用できます。

4.23.2 例

--dump_build_attributes の出力例を以下に示します。

```
...
** Section #12 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)
    Size   : 69 bytes

    0x000000:   41 33 00 00 00 61 65 61 62 69 00 01 29 00 00 00    A3...aeabi..)...
    0x000010:   05 41 52 4d 37 54 44 4d 49 00 06 02 08 01 11 01    .ARM7TDMI.......
    0x000020:   12 02 14 02 17 01 18 01 19 01 1a 01 1e 03 20 02    .............. .
    0x000030:   41 52 4d 00 11 00 00 00 41 52 4d 00 01 09 00 00    ARM.....ARM.....
    0x000040:   00 04 01 12 01                                     .....
```

4.23.3 関連項目

参照

- *--decode_build_attributes*（4-21 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--extract_build_attributes*（4-36 ページ）
- *--text*（4-76 ページ）.

4.24 --elf

このオプションは ELF 出力モードを選択します。

ELF イメージからデバッグ情報を削除するには、--strip=debug,symbols と組み合わせて使用します。

4.24.1 制約条件

このオプションは --output と組み合わせて使用する必要があります。

4.24.2 関連項目

参照

- *--in_place*（4-49 ページ）
- *--output=destination*（4-60 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

4.25 --emit=*option*[,*option*,...]

このオプションにより、テキスト出力に表示する ELF オブジェクトの要素を指定できます。出力には、ELF ヘッダおよびセクション情報が含まれます。

4.25.1 制約条件

このオプションは、テキストモードでのみ使用できます。

4.25.2 構文

--emit=*option*[,*option*,...]

option には以下のいずれかを指定できます。

addresses　このオプションは、グローバルデータアドレスとスタティックデータアドレス（構造体とユニオンの内容のアドレスも含む）を出力します。これは、--text -a と同じ効果があります。

このオプションは、デバッグ情報を含むファイルに対してのみ使用できます。デバッグ情報が含まれない場合、警告メッセージが生成されます。

データアドレスのサブセットを出力する場合は、--select オプションを使用します。

構造体内外で展開された配列のデータアドレスを参照するには、このテキストカテゴリと共に --expandarrays オプションを使用します。

build_attributes

このオプションは、標準のビルド属性の場合は人間が読める形式でビルド属性セクションの内容を出力し、非標準のビルド属性の場合は生の 16 進形式で出力します。生成される出力は、--decode_build_attributes オプションの出力と同じです。

code　このオプションは、逆アセンブル対象の元のバイナリデータのダンプおよび命令のアドレスと共に、コードを逆アセンブルします。これは、--text -c と同じ効果があります。

注

--disassemble とは異なり、逆アセンブリをアセンブラに入力することはできません。

data　このオプションはデータセクションの内容を出力します。これは、--text -d と同じ効果があります。

data_symbols

このオプションは、シンボル定義をインターリーブするようにデータセクションの出力情報を変更します。

debug_info　このオプションはデバッグ情報を出力します。これは、--text -g と同じ効果があります。

dynamic_segment

このオプションは、ダイナミックセグメントの内容を出力します。これは、--text -y と同じ効果があります。

exception_tables

このオプションは、オブジェクトの例外テーブル情報をデコードします。これは、--text -e と同じ効果があります。

frame_directives

このオプションは、オブジェクトモジュールに組み込まれたデバッグ情報に指定されているように、逆アセンブルされたコードに FRAME ディレクティブの内容を出力します。

このオプションは --disassemble と組み合わせて使用します。

got このオプションは、グローバルオフセットテーブル（GOT）オブジェクトの内容を出力します。

heading_comments

このオプションは、.comment セクションのツール情報およびコマンドライン情報を含む、逆アセンブリの冒頭にある見出しコメントを出力します。

このオプションは --disassemble と組み合わせて使用します。

raw_build_attributes

このオプションは、生の 16 進形式で、つまりデータと同じ形式でビルド属性セクションの内容を出力します。

relocation_tables

このオプションは再配置情報を出力します。これは、--text -r と同じ効果があります。

string_tables

このオプションはストリングテーブルを出力します。これは、--text -t と同じ効果があります。

summary このオプションは、ファイルのセグメントおよびセクションの概要を出力します。これは fromelf --text のデフォルト出力です。ただし、概要は --info オプションにより出力されません。必要に応じて --emit summary を使用して明示的に概要を再度有効にします。

symbol_annotations

このオプションは、それぞれのプロパティ情報を含むコメントの注釈を付けて、逆アセンブルされるコードおよびデータのシンボルを出力します

このオプションは --disassemble と組み合わせて使用します。

symbol_tables

このオプションは、シンボルテーブルとバージョン管理テーブルを出力します。これは、--text -s と同じ効果があります。

vfe このオプションは、使用されていない仮想関数の情報を出力します。

whole_segments

このオプションは、リンクビューがある場合でも、逆アセンブルされた実行可能ファイルまたは共有ライブラリをセグメントごとに出力します。

このオプションは --disassemble と組み合わせて使用します。

1 つの --emit オプションにコンマ区切りの引数リストを続けることで、複数のオプションを指定できます。

4.25.3 関連項目

参照

- *--disassemble*（4-29 ページ）
- *--decode_build_attributes*（4-21 ページ）
- *--expandarrays*（4-35 ページ）
- *--text*（4-76 ページ）.

4.26 --expandarrays

このオプションを指定すると、データのアドレスを出力できます。これには、構造体内外で展開された配列が含まれます。

4.26.1 制約条件

このオプションは、--text -a と共に指定する必要があります。

4.26.2 関連項目

参照

- *--text*（4-76 ページ）.

4.27 --extract_build_attributes

このオプションは、以下のいずれかの形式でビルド属性のみを出力します。

- 標準のビルド属性の場合は人間が読める形式
- 非標準のビルド属性の場合は生の 16 進形式で

4.27.1 制約条件

このオプションは、テキストモードでのみ使用できます。

4.27.2 例

--extract_build_attributes の出力例を以下に示します。

```
==============================================================================

** Object/Image Build Attributes

    'aeabi' file build attributes:
    0x000000:   05 41 52 4d 37 54 44 4d 49 00 06 02 08 01 11 01    .ARM7TDMI.......
    0x000010:   12 02 14 02 17 01 18 01 19 01 1a 01 1e 03 20 02    .............. .
    0x000020:   41 52 4d 00                                        ARM.
        Tag_CPU_name = "ARM7TDMI"
        Tag_CPU_arch = ARM v4T (=2)
        Tag_ARM_ISA_use = ARM instructions were permitted to be used (=1)
        Tag_ABI_PCS_GOT_use = Data are imported directly (=1)
        Tag_ABI_PCS_wchar_t = Size of wchar_t is 2 (=2)
        Tag_ABI_FP_denormal = This code was permitted to require that the sign of a flushed-to-zero number be
preserved in the sign of 0 (=2)
        Tag_ABI_FP_number_model = This code was permitted to use only IEEE 754 format FP numbers (=1)
        Tag_ABI_align8_needed = Code was permitted to depend on the 8-byte alignment of 8-byte data items (=1)
        Tag_ABI_align8_preserved = Code was required to preserve 8-byte alignment of 8-byte data objects (=1)
        Tag_ABI_enum_size = Enum values occupy the smallest container big enough to hold all values (=1)
        Tag_ABI_optimization_goals = Optimized for small size, but speed and debugging illusion preserved (=3)
        Tag_compatibility = 2, "ARM"

    'ARM' file build attributes:
    0x000000:   04 01 12 01                                        ....
```

4.27.3 関連項目

参照

- *--decode_build_attributes*（4-21 ページ）
- *--dump_build_attributes*（4-30 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--text*（4-76 ページ）.

4.28 --fieldoffsets

このオプションは、アセンブリ言語の EQU ディレクティブのリストを出力します。このディレクティブによって、C++ のクラスや C の構造体のフィールド名は、そのクラスまたは構造体のースからのオフセットを表すようになります。入力 ELF ファイルは、再配置可能なオブジェクトまたはイメージです。

--output を使用すると、出力がファイルに転送されます。armasm で INCLUDE コマンドを使用すると、生成されたファイルをロードし、C++ クラスおよび C 構造体メンバにアセンブリ言語から名前でアクセスできます。

このオプションを指定すると、構造体に関するすべての情報が出力されます。構造体のサブセットに関する情報を出力するには、--select *select_options* を使用します。

armasm で入力ファイルとして指定できるファイルを生成する必要がない場合は、--text -a オプションを使用して、表示されるアドレスを読みやすい形式にすることができます。-a オプションを指定すると、アドレスは再配置可能なオブジェクト内には存在しないため、イメージ内の構造体とスタティックデータのアドレス情報のみが出力されます。

4.28.1 制約条件

このオプションには以下の制限があります。

- ソースファイルにデバッグ情報がない場合は使用できません。
- テキストモードでのみ使用できます。

4.28.2 例

以下に、--fieldoffsets ユーティリティの使用例を示します。

- inputfile.o ファイル内にあるすべての構造体のすべてのフィールドオフセットが含まれた一覧を stdout に出力するには、以下のように入力します。

  fromelf --fieldoffsets inputfile.o

- inputfile.o ファイル内で、名前が p で始まる構造体のすべてのフィールドオフセットが含まれた一覧を outputfile.a に出力するには、以下のように入力します。

  fromelf --fieldoffsets --select=p* --output=outputfile.a inputfile.o

- inputfile.o ファイル内で、tools または moretools という名前の構造体のすべてのフィールドオフセットが含まれた一覧を outputfile.a に出力するには、以下のように入力します。

  fromelf --fieldoffsets --select=tools.*,moretools.* --output=outputfile.a inputfile.o

- inputfile.o ファイル内の構造体 tools の構造体フィールド top の中にあり、名前が number で始まる構造体フィールドのすべてのフィールドオフセットが含まれた一覧を outputfile.a に出力するには、以下のように入力します。

  fromelf --fieldoffsets --select=tools.top.number* --output=outputfile.a inputfile.o

4.28.3 関連項目

概念

『ARM® プロセッサをターゲットとしたソフトウェア開発』:

- *言語間の呼び出しの例*（4-11 ページ）.

参照

- *--qualify*（4-63 ページ）
- *--select=select_options*（4-68 ページ）
- *--text*（4-76 ページ）

『アセンブラリファレンス』:

- *EQU*（6-80 ページ）
- *GET、INCLUDE*（6-84 ページ）.

4.29 --fpu=list

このオプションを使用すると、--fpu=*name* オプションと組み合わせて使用可能なサポートされている FPU アーキテクチャ名が一覧表示されます。

4.29.1 関連項目

参照

- *--fpu=name*（4-40 ページ）.

4.30 --fpu=*name*

このオプションは、特定の FPU アーキテクチャの逆アセンブリを選択します。これにより、fromelf が入力ファイルで見つかった命令を処理する方法に影響が出ます。

4.30.1 構文

--fpu=*name*

name はサポートされる FPU アーキテクチャの名前です。

4.30.2 例

VFPv2 アーキテクチャの逆アセンブリを選択するには、以下のように入力します。

```
--fpu=VFPv2
```

4.30.3 関連項目

参照

- *--device=list*（4-22 ページ）
- *--device=name*（4-23 ページ）
- *--disassemble*（4-29 ページ）
- *--fpu=list*（4-39 ページ）
- *--info=topic[,topic,...]*（4-50 ページ）
- *--text*（4-76 ページ）.

4.31 --globalize=*option*[,*option*,...]

このオプションは、選択されたシンボルをグローバルシンボルに変換します。

4.31.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.31.2 構文

--globalize=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボルがグローバルシンボルに変換されます。

object_name::*symbol_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボルがグローバルシンボルに変換されます。

symbol_name *symbol_name* と一致するシンボル名のすべてのシンボルがグローバルシンボルに変換されます。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --globalize オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.31.3 関連項目

参照

- *--elf*（4-31 ページ）
- *--hide=option[,option,...]*（4-43 ページ）.

4.32 --help

このオプションを選択すると、主なコマンドラインオプションの一覧が表示されます。

これは、オプションやソースファイルを指定しない場合のデフォルトの動作です。

4.32.1 関連項目

参照

- *--show_cmdline*（4-71 ページ）
- *--version_number*（4-78 ページ）
- *--vsn*（4-81 ページ）.

4.33 --hide=*option*[,*option*,...]

このオプションは、シンボルの可視性プロパティを変更して、選択されたシンボルを非表示としてマークします。

4.33.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.33.2 構文

--hide=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::
: *object_name* と一致する名前の ELF オブジェクト内のすべてのシンボル。

object_name::*symbol_name*
: *object_name* と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボル。

symbol_name
: *symbol_name* と一致するシンボル名のすべてのシンボル。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --hide オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.33.3 関連項目

参照

- *--elf*（4-31 ページ）
- *--show=option[,option,...]*（4-69 ページ）.

4.34 --hide_and_localize=*option*[,*option*,...]

このオプションは、シンボルの可視性プロパティを変更して、選択されたシンボルを非表示としてマークし、選択されたシンボルをローカルシンボルに変換します。

4.34.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.34.2 構文

--hide_and_localize=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボルが非表示としてマークされ、ローカルシンボルに変換されます。

object_name::*symbol_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボルが非表示としてマークされ、ローカルシンボルに変換されます。

symbol_name *symbol_name* と一致するシンボル名のすべてのシンボルが非表示としてマークされ、ローカルシンボルに変換されます。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --hide_and_localize オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.34.3 関連項目

参照

- *--elf*（4-31 ページ）.

4.35 --i32

このオプションは、Intel Hex32 ビット形式の出力を作成します。これによりイメージ内のロード領域ごとに 1 つの出力ファイルを生成できます。この出力のベースアドレスは、--base オプションを使用して指定できます。

4.35.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは --output と組み合わせて使用する必要があります。

4.35.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--i32combined*（4-46 ページ）
- *--output=destination*（4-60 ページ）.

4.36 --i32combined

このオプションは、Intel Hex32 ビット形式の出力を作成します。このオプションを指定すると、複数のロード領域を含むイメージ用に 1 つの出力ファイルを生成できます。この出力のベースアドレスは、--base オプションを使用して指定できます。

4.36.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは --output と組み合わせて使用する必要があります。

4.36.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--i32*（4-45 ページ）
- *--output=destination*（4-60 ページ）.

4.37 --ignore_section=*option*[,*option*,...]

このオプションは、比較時に無視するセクションを指定します。これらのセクションに比較する入力ファイル間の違いが含まれる場合はそれが無視されます。

4.37.1 制約条件

このオプションは --compare と組み合わせて使用する必要があります。

4.37.2 構文

--ignore_section=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのセクション。

object_name::*section_name*

object_name と一致する名前の ELF オブジェクト内のすべてのセクション、および *section_name* と一致するセクション名のすべてのセクション。

section_name *section_name* と一致する名前のすべてのセクション。

以下のことができます。

- *section_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --ignore_section オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.37.3 関連項目

参照

- *--compare=option[,option,...]*（4-14 ページ）
- *--ignore_symbol=option[,option,...]*（4-48 ページ）
- *--relax_section=option[,option,...]*（4-65 ページ）.

4.38 --ignore_symbol=*option*[,*option*,...]

このオプションは、比較中に無視するシンボルを指定します。比較する入力ファイル間でのこれらのシンボルに関連する違いが無視されます。

4.38.1 制約条件

このオプションは --compare と組み合わせて使用する必要があります。

4.38.2 構文

--ignore_symbol=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル。

object_name::*symbol_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボル。

symbol_name *symbol_name* と一致する名前のすべてのシンボル。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --ignore_symbol オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.38.3 関連項目

参照

- *--compare=option[,option,...]*（4-14 ページ）
- *--ignore_section=option[,option,...]*（4-47 ページ）
- *--relax_symbol=option[,option,...]*（4-66 ページ）.

4.39 --in_place

このオプションにより、入力ファイルの ELF メンバを変換して以前の内容を上書きできます。

4.39.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.39.2 例

ライブラリファイル test.a のメンバからデバッグ情報を削除するには、以下のように入力します。

```
fromelf --elf --in_place --strip=debug test.a
```

4.39.3 関連項目

参照

- *--elf*（4-31 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

4.40 --info=*topic*[,*topic*,...]

このオプションは特定のトピックに関する情報を出力します。

4.40.1 制約条件

このオプションは、テキストモードでのみ使用できます。

4.40.2 構文

--info=*topic*[,*topic*,...]

topic は、以下のトピックキーワードからのコンマ区切りのリストです。

instruction_usage
: 各入力ファイルのコードセクションで定義された ARM 命令と Thumb 命令を分類して一覧表示します。

function_sizes
: 1 つ以上の入力ファイルで定義されたグローバル関数の名前と、その分類（ARM 関数か Thumb 関数か）を一覧表示します。

function_sizes_all
: 1 つ以上の入力ファイルで定義されたローカル関数およびグローバル関数の名前と、その分類（ARM 関数か Thumb 関数か）を一覧表示します。

sizes
: イメージ内の入力オブジェクトおよびライブラリのメンバごとに、Code、RO Data、RW Data、ZI Data、および Debug のサイズが一覧表示されます。このオプションを使用すると、--info=sizes,totals を指定したことになります。

totals
: 入力オブジェクトとライブラリの Code、RO Data、RW Data、ZI Data、および Debug の合計サイズが一覧表示されます。

--info=sizes,totals の出力には、常に入力オブジェクトとライブラリの合計にパディング値が含まれます。

注

リスト内のトピックキーワードの間にはスペースを挿入しないで下さい。例えば、「--info=sizes,totals」と入力することはできますが、「--info=sizes, totals」と入力することはできません。

4.40.3 関連項目

参照

- *--text*（4-76 ページ）.

4.41 *input_file*

このオプションは、処理する ELF ファイルまたは ELF ファイルを含むアーカイブを指定します。以下の場合は、複数の入力ファイルを指定できます。

- `--text` 形式に出力する
- `--compare` オプションを使用する
- `--elf` を `--in_place` と組み合わせて使用する
- `--output` を使用して出力ディレクトリを指定する

4.41.1 使用法

input_file が複数のロード領域を含む分散ロードイメージであり、その出力形式に `--bin`、`--cad`、`--m32`、`--i32`、`--vhx` のいずれかが指定されている場合には、`fromelf` によって各ロード領域用に個別のファイルが生成されます。

input_file が複数のロード領域を含む分散ロードイメージであり、その出力形式に `--cadcombined`、`--m32combined`、`--i32combined` のいずれかが指定されている場合には、`fromelf` によりすべてのロード領域を含む 1 つのファイルが作成されます。

input_file がアーカイブの場合は、アーカイブ内のすべてのファイルまたはファイルのサブセットを処理できます。アーカイブ内のファイルのサブセット処理するには、以下に示すようにアーカイブ名に続けてフィルタを指定します。

`archive.a(`*filter_pattern*`)`

ここで、*filter_pattern* は、メンバファイルを指定します。ファイルのサブセットを指定する場合、以下のワイルドカード文字を使用できます。

`*` 0 個以上の文字に一致します。

`?` 任意の 1 文字と一致します。

注

Unix システムの一般的なシェルでは、かっこを使用し、バックスラッシュでこれらの文字をエスケープする必要があります。あるいは、以下に示すように、アーカイブ名とフィルタを単一引用符で囲みます。

`'archive.a(??str*)'`

アーカイブ内の処理されていないファイルは、処理されたファイルと共に出力アーカイブに格納されます。

4.41.2 例

アーカイブ内の s で始まるすべてのファイルからデバッグ情報を削除した後、新しいアーカイブ `my_archive.a` を作成して、処理されたファイルと処理されていないファイルを格納するには、以下のように入力します。

```
fromelf --elf --strip=debug archive.a(s*.o) --output=my_archive.a
```

4.41.3 関連項目

タスク

- *アーカイブ内の ELF ファイルの処理*（3-9 ページ）

参照

- *--bin*（4-6 ページ）
- *--cad*（4-11 ページ）
- *--cadcombined*（4-13 ページ）
- *--compare=option[,option,...]*（4-14 ページ）
- *--elf*（4-31 ページ）
- *--i32*（4-45 ページ）
- *--i32combined*（4-46 ページ）
- *--in_place*（4-49 ページ）
- *--m32*（4-57 ページ）
- *--m32combined*（4-58 ページ）
- *--output=destination*（4-60 ページ）
- *--text*（4-76 ページ）
- *--vhx*（4-79 ページ）.

4.42 --interleave=*option*

このオプションは、デバッグ情報が存在する場合に、元のソースコードを逆アセンブル結果にコメントとして挿入します。

このオプションは --emit=code、--text -c、または --disassemble と組み合わせて使用します。

このオプションは、ソースコードの検索パスを追加指定したい場合に、--source_directory と組み合わせて使用します。

4.42.1 構文

--interleave=*option*

option は次のいずれかになります。

line_directives
: 逆アセンブルされた命令のファイル名および行番号を含む #line ディレクティブをインターリーブします。

line_numbers
: 逆アセンブルされた命令のファイル名および行番号を含むコメントをインターリーブします。

none
: インターリーブが無効になります。これは、作成されたメイクファイルで、fromelf コマンドに --interleave に加えて複数のオプションがある場合に役立ちます。この場合は、--interleave=none を最後のオプションとして指定することで、fromelf コマンド全体を再度指定しなくてもインターリーブを無効にすることができます。

source
: ソースコードを含むコメントをインターリーブします。ソースコードが使用できなくなった場合は、fromelf ユーティリティが line_numbers と同じようにインターリーブします。

source_only
: ソースコードを含むコメントをインターリーブします。ソースコードが使用できなくなった場合は、fromelf ユーティリティはそのコードをインターリーブしません。

4.42.2 デフォルト

デフォルトは --interleave=none です。

4.42.3 関連項目

参照

- *--disassemble*（4-29 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--source_directory=path*（4-72 ページ）
- *--text*（4-76 ページ）.

4.43 --licretry

フローティングライセンスを使用している場合、fromelf を起動したときに、10 回までライセンスの取得を試みます。

4.43.1 使用法

このオプションは、ライセンスサーバからライセンスを取得できなかった場合に、ネットワークまたはライセンスサーバの設定に関する他の問題ないことを確認した後でのみ使用して下さい。

このオプションは、ARMCC*n*_FROMELFOPT 環境変数で指定することを推奨します。これにより、ビルドファイルを修正する必要がなくなります。

4.43.2 関連項目

参照

『ARM コンパイラツールチェーンの概要』:

- *ツールチェーンの環境変数*（2-16 ページ）

『コンパイラリファレンス』:

- *--licretry*（3-139 ページ）.

『リンカリファレンス』:

- *--licretry*（2-100 ページ）.

『アセンブラリファレンス』:

- *--licretry*（2-19 ページ）.

その他の情報

- *ARM® DS-5™ ライセンス管理ガイド*，
  http://infocenter.arm.com/help/topic/com.arm.doc.dui0577-/index.html
  .

4.44 --linkview、--no_linkview

このオプションの使用は廃止される予定です。

ELF イメージのセクションレベルのビューを制御します。

--no_linkview を指定すると、セクションレベルのビューが破棄され、セグメントレベルのビュー（ロード時のビュー）のみが保持されます。セクションレベのビューを破棄することによって、以下が削除されます。

- セクションのヘッダテーブル
- セクションのヘッダストリングテーブル
- ストリングテーブル
- シンボルテーブル
- すべてのデバッグセクション

出力に含まれるのは、プログラムのヘッダテーブルとプログラムのセグメントのみです。*『System V Application Binary Interface』*仕様によれば、プログラムローダが ELF ファイルに存在すると想定してよいのはこれらのみです。

4.44.1 制約条件

次の使用制限があります。

- --linkview および --no_linkview は --elf と組み合わせて使用する必要があります。
- SysV イメージには --no_linkview オプションを使用しないで下さい。

4.44.2 例

image.axf の ELF 形式の出力を生成するには、以下のように入力します。

```
fromelf --no_linkview --elf image.axf --output=image_nlk.axf
```

4.44.3 関連項目

参照

- *--elf*（4-31 ページ）
- *--privacy*（4-61 ページ）
- *--strip=option[,option,...]*（4-73 ページ）.

『リンカリファレンス』:

- *--privacy*（2-128 ページ）.

その他の情報

- *『System V Application Binary Interface』（草案、2003 年 12 月 17 日）*仕様

4.45 --localize=*option*[,*option*,...]

このオプションは、選択されたシンボルをローカルシンボルに変換します。

4.45.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.45.2 構文

--localize=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボルがローカルシンボルに変換されます。

object_name::*symbol_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボルがローカルシンボルに変換されます。

symbol_name *symbol_name* と一致するシンボル名のすべてのシンボルがローカルシンボルに変換されます。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --localize オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.45.3 関連項目

参照

- *--elf*（4-31 ページ）
- *--hide=option[,option,...]*（4-43 ページ）.

4.46 --m32

このオプションは、Motorola 32 ビット形式（32 ビット S レコード形式）の出力を作成します。これによりイメージ内のロード領域ごとに 1 つの出力ファイルを生成できます。この出力のベースアドレスは、--base オプションを使用して指定できます。

4.46.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは --output と組み合わせて使用する必要があります。

4.46.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--m32combined*（4-58 ページ）
- *--output=destination*（4-60 ページ）.

4.47 --m32combined

このオプションは、Motorola 32 ビット形式（32 ビット S レコード形式）の出力を作成します。このオプションを指定すると、複数のロード領域を含むイメージ用に 1 つの出力ファイルを生成できます。この出力のベースアドレスは、--base オプションを使用して指定できます。

4.47.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは --output と組み合わせて使用する必要があります。

4.47.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--base [[object_file::]load_region_ID=]num*（4-4 ページ）
- *--m32*（4-57 ページ）
- *--output=destination*（4-60 ページ）.

4.48 --only=*section_name*

指定されたセクションのみを出力結果として表示します。

4.48.1 構文

--only=*section_name*

section_name は、表示させるセクションの名前です。

以下のことができます。

- セクションの名前には、ワイルドカード文字 ? および * を使用できます。
- 複数の --only オプションを使用することによって、表示するセクションを追加指定できます。

4.48.2 例

以下に、--only の使用例を示します。

- シンボルテーブル（.symtab）のみを表示するには、以下のように入力します。

  fromelf --only=.symtab --text -s test.axf

- すべての ER*n* セクションを表示するには、以下のように入力します。

  fromelf --only=ER? test.axf

- HEAP セクションと、すべてのシンボルテーブルセクションおよびすべてのストリングテーブルセクションを表示するには、以下のように入力します。

  fromelf --only=HEAP --only=.*tab --text -s -t test.axf

4.48.3 関連項目

参照

- *--text*（4-76 ページ）.

4.49 --output=*destination*

このオプションは、出力ファイルの名前、または複数の出力ファイルを作成する場合は出力ディレクトリの名前を指定します。

4.49.1 構文

`--output=`*destination*

`--o=`*destination*

destination はファイルまたはディレクトリのいずれかです。以下に例を示します。

`--output=foo` 出力ファイルの名前。

`--output=foo/`
出力ディレクトリの名前。

4.49.2 使用法

`--bin` または `--elf` での使用法を以下に示します。

- 1 つの入力ファイルと 1 つの出力ファイル名を指定できます。
- `--elf` を使用して複数の入力ファイルを指定する場合、`--in_place` を使用して、入力ファイルに各ファイル処理の出力を上書きできます。
- 多くの入力ファイル名と 1 つの出力ディレクトリを指定した場合、各ファイル処理の出力が出力ディレクトリに書き込まれます。各出力ファイル名は、対応する入力ファイルから付けられます。したがって、この方法で出力ディレクトリを指定することが、`fromelf` を 1 回実行して多くの ELF ファイルをバイナリ形式または 16 進形式に変換するための唯一の手段になります。
- アーカイブファイルを入力として指定した場合は、出力ファイルもアーカイブになります。例えば、以下のコマンドは、`output.o` という名前のアーカイブファイルを作成します。

  ```
  fromelf --elf --strip=debug mylib.a --output=output.o
  ```

- アーカイブ内のオブジェクトのサブセットを選択するパターンを括弧で囲んで指定した場合、そのサブセットのみが fromelf によって変換されます その他のすべてのオブジェクトは、変更されずにそのまま出力アーカイブに渡されます。

4.49.3 関連項目

参照

- *--bin*（4-6 ページ）
- *--elf*（4-31 ページ）
- *--text*（4-76 ページ）.

4.50 --privacy

これらのオプションの効果は、イメージとオブジェクトファイルとで異なります。

イメージの場合：

- セクション名をデフォルト値に変更します。例えば、コードセクションの名前は .text に変更されます。
- --strip symbols の場合と同様に、シンボルテーブル全体を削除します。
- .comment セクション名を削除し、この部分を fromelf --text 出力内で [Anonymous Section] とマークします。

オブジェクトファイルの場合：

- セクション名をデフォルト値に変更します。例えば、コードセクションの名前は .text に変更されます。
- マッピングシンボルおよびビルド属性はシンボルテーブルに維持されます。
- 機能を損なうことなく削除できるローカルシンボルは削除されます。
  再配置のターゲットなど、削除できないシンボルは維持されます。このようなシンボルについては、名前が削除されます。fromelf --text の出力結果には、これらが [Anonymous Symbol] としてマークされます。

サードパーティに配布されるイメージとオブジェクトに含まれるコードを保護するには、このオプションを使用します。

4.50.1 関連項目

タスク

- *fromelf によるイメージおよびオブジェクトに含まれるコードの保護*（3-10 ページ）

参照

- *--strip=option[,option,...]*（4-73 ページ）

『リンカリファレンス』:

- *--locals、--no_locals*（2-106 ページ）
- *--privacy*（2-128 ページ）.

4.51 --project=*filename*?--no_project

指定したプロジェクトテンプレートファイルのロードを制御します。

注

このオプションの使用は廃止される予定です。

4.51.1 構文

--project=*filename*

filename はプロジェクトテンプレートファイルの名前です。

注

filename をデフォルトのプロジェクトファイルとして使用するには、RVDS_PROJECT 環境変数を *filename* に設定します。

--no_project を指定すると、環境変数 RVDS_PROJECT で指定されたデフォルトのプロジェクトテンプレートファイルは使用されません。

4.51.2 制約条件

プロジェクトテンプレートファイルのオプションは、コマンドラインで設定済みのオプションと競合しない場合にのみ設定できます。プロジェクトテンプレートファイルのオプションが既存のコマンドラインオプションと競合する場合は、コマンドラインオプションが優先されます。

4.51.3 例

以下のプロジェクトテンプレートファイルについて考えてみます。

```
<!-- suiteconf.cfg -->
<suiteconf name="Platform Baseboard for ARM926EJ-S">
    <tool name="fromelf">
        <cmdline>
            --cpu=ARM926EJ-S
            --fpu=vfpv2
        </cmdline>
    </tool>
</suiteconf>
```

RVDS_PROJECT 環境変数がこのファイルを示すように設定されている場合に、次のコマンドを実行したとします。

```
fromelf foo.o
```

このコマンドの実際のコマンドラインは、以下のようになります。

```
fromelf --cpu=ARM926EJ-S --fpu=vfpv2 foo.o
```

4.51.4 関連項目

参照

- *--reinitialize_workdir*（4-64 ページ）
- *--workdir=directory*（4-85 ページ）.

4.52 --qualify

このオプションは、各出力シンボル名に、関連する構造体を含むソースファイルが示されるように、--fieldoffsets オプションの効果を変更します。これにより、2 つのソースファイルが同じ名前の異なる構造体を定義している場合でも、--fieldoffsets オプションで機能的な出力が作成されます。

4.52.1 例

foo という構造体は、例えば、one.h および two.h の 2 つのヘッダで定義されます。

fromelf --fieldoffsets を使用して、リンカで以下のようなシンボルを定義できます。

- foo.a、foo.b、および foo.c
- foo.x、foo.y、および foo.z

fromelf --qualify --fieldoffsets を使用して、リンカで以下のシンボルを定義します。

- oneh_foo.a、oneh_foo.b、および oneh_foo.c
- twoh_foo.x、twoh_foo.y、および twoh_foo.z

4.52.2 関連項目

参照

- *--fieldoffsets*（4-37 ページ）.

4.53 --reinitialize_workdir

このオプションを使用すると、--workdir を使用してプロジェクトテンプレートの作業ディレクトリを再初期化するかどうかを指定できます。

--workdir を使用して設定されたディレクトリが、変更されたプロジェクトテンプレートファイルが含まれている既存の作業ディレクトリを参照している場合、このオプションを指定すると、作業ディレクトリは削除され、元のプロジェクトテンプレートファイルの新しいコピーで再作成されます。

注

このオプションの使用は廃止される予定です。

4.53.1 制約条件

このオプションは --workdir と組み合わせて使用する必要があります。

4.53.2 関連項目

参照

- *--project=filename?--no_project*（4-62 ページ）
- *--workdir=directory*（4-85 ページ）.

4.54 --relax_section=*option*[,*option*,...]

このオプションは、指定されたセクションの比較レポートの重大度をエラーから警告に変更します。

4.54.1 制約条件

このオプションは --compare と組み合わせて使用する必要があります。

4.54.2 構文

--relax_section=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのセクション。

object_name::*section_name*

object_name と一致する名前の ELF オブジェクト内のすべてのセクション、および *section_name* と一致するセクション名のすべてのセクション。

section_name *section_name* と一致する名前のすべてのセクション。

以下のことができます。

- *section_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --relax_section オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.54.3 関連項目

参照

- *--compare=option[,option,...]*（4-14 ページ）
- *--ignore_section=option[,option,...]*（4-47 ページ）
- *--relax_symbol=option[,option,...]*（4-66 ページ）.

4.55 --relax_symbol=*option*[,*option*,...]

このオプションは、指定されたシンボルの比較レポートの重大度をエラーから警告に変更します。

4.55.1 制約条件

このオプションは --compare と組み合わせて使用する必要があります。

4.55.2 構文

--relax_symbol=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル。

object_name::*section_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボル。

symbol_name *symbol_name* と一致する名前のすべてのシンボル。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --relax_symbol オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.55.3 関連項目

参照

- *--compare=option[,option,...]*（4-14 ページ）
- *--ignore_symbol=option[,option,...]*（4-48 ページ）
- *--relax_section=option[,option,...]*（4-65 ページ）.

4.56 --rename=*option*[,*option*,...]

このオプションは、出力 ELF オブジェクトに指定されたシンボルの名前を変更します。

4.56.1 制約条件

このオプションは --elf および --output と組み合わせて使用する必要があります。

4.56.2 構文

--rename=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::*old_symbol_name=new_symbol_name*

ELF オブジェクト *object_name* 内の *old_symbol_name* と一致するシンボル名のすべてのシンボルを置換します。

old_symbol_name=new_symbol_name

old_symbol_name と一致するシンボル名のすべてのシンボルを置換します。

以下のことができます。

- *old_symbol_name* 引数、*new_symbol_name* 引数、および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --rename オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.56.3 例

この例では、timer.axf イメージ内の clock シンボルの名前を myclock に変更し、mytimer.axf という新しいファイルを作成します。

```
fromelf --elf --rename=clock=myclock --output=mytimer.axf timer.axf
```

4.56.4 関連項目

参照

- *--elf*（4-31 ページ）
- *--output=destination*（4-60 ページ）.

4.57 --select=*select_options*

このオプションは、指定されたパターンリストに一致するフィールドのみを選択します。

このオプションは、--fieldoffsets または --text -a と組み合わせて使用します。

4.57.1 構文

--select=*select_options*

select_options は一致させるパターンのリストです。複数のフィールドを選択する場合は、以下のように特殊文字を使用します。

- 複数のフィールドを指定するには、コンマ区切りのリストを使用します。例えば、以下のように入力します。

  a*,b*,c*

- 任意の名前と一致させるには、ワイルドカード文字 * を使用します。
- 任意の 1 文字と一致させるには、ワイルドカード文字 ? を使用します。
- インクルードするフィールドを指定するには、*select_options* 文字列の前に + を付けます。これはデフォルトの動作です。
- 除外するフィールドを指定するには、*select_options* 文字列の前に ~ を付けます。

UNIX プラットフォームで特殊文字を使用する場合は、シェルによって文字列展開がされないように、これらのオプションを引用符で囲む必要があります。

4.57.2 関連項目

参照

- *--fieldoffsets*（4-37 ページ）
- *--text*（4-76 ページ）.

4.58 --show=*option*[,*option*,...]

このオプションは、選択されたシンボルの可視性プロパティを変更して、デフォルトの可視性でマークします。

4.58.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.58.2 構文

--show=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボルにデフォルトの可視性がマークされます。

object_name::*symbol_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボルにデフォルトの可視性がマークされます。

symbol_name *symbol_name* と一致するシンボル名のすべてのシンボルにデフォルトの可視性がマークされます。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --show オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.58.3 関連項目

参照

- *--elf*（4-31 ページ）
- *--hide=option[,option,...]*（4-43 ページ）.

4.59 --show_and_globalize=*option*[,*option*,...]

このオプションは、選択されたシンボルの可視性プロパティを変更して、デフォルトの可視性でマークし、選択されたシンボルをグローバルシンボルに変換します。

4.59.1 制約条件

このオプションは --elf と組み合わせて使用する必要があります。

4.59.2 構文

--show_and_globalize=*option*[,*option*,...]

option には以下のいずれかを指定できます。

object_name::

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル。

object_name::*symbol_name*

object_name と一致する名前の ELF オブジェクト内のすべてのシンボル、および *symbol_name* と一致するシンボル名のすべてのシンボル。

symbol_name *symbol_name* と一致するシンボル名のすべてのシンボル。

以下のことができます。

- *symbol_name* 引数および *object_name* 引数のシンボル名には、ワイルドカード文字 ? および * を使用できます。
- 1 つの --show_and_globalize オプションにコンマ区切りの引数リストを続けることで、オプションを複数指定できます。

4.59.3 関連項目

参照

- *--elf*（4-31 ページ）.

4.60 --show_cmdline

このオプションは、fromelf がコマンドラインをどのように処理したかを表示します。fromelf によって処理された後のコマンドラインを表示することによって、以下の点を確認できます。

- ビルドシステムによって使用されているコマンドライン
- 指定されたコマンドラインが fromelf によってどのように解釈されているか（コマンドラインオプションの順序など）

コマンドは適切な形式で表示されます。また、via ファイルの内容は展開されます。

4.60.1 関連項目

参照

- *--via=file*（4-80 ページ）
- 第 4 章 *fromelf コマンドリファレンス* .

4.61 --source_directory=*path*

このオプションは、ソースコードのディレクトリを明示的に指定します。デフォルトでは、ELF 入力ファイルの相対ディレクトリにソースコード配置されているものと想定されます。このオプションを複数回使用して、複数のディレクトリの検索パスを指定できます。

このオプションは --interleave と組み合わせて使用します。

4.61.1 関連項目

参照

- *--interleave=option*（4-53 ページ）.

4.62 --strip=*option*[,*option*,...]

このオプションを使用すると、サードパーティに配布されるイメージとオブジェクトに含まれるコードを保護できます。さらに、出力イメージのサイズを減らすためにも使用できます。

4.62.1 制約条件

このオプションは --elf および --output と組み合わせて使用する必要があります。

4.62.2 構文

--strip=*option*[,*option*,...]

option には以下のいずれかを指定できます。

all
: オブジェクトモジュールの場合、このオプションによって ELF ファイルからすべてのデバッグ、コメント、メモ、およびシンボルが削除されます 実行可能ファイルの場合、このオプションは --no_linkview と同じように機能します。

  注
  SysV イメージには --strip=all オプションを使用しないで下さい。

debug
: ELF ファイルから、すべてのデバッグセクションを削除します。

comment
: ELF ファイルから、**.comment** セクションを削除します。

filesymbols
: STT_FILE シンボルが ELF ファイルから削除されます。

localsymbols
: これらのオプションの効果は、イメージとオブジェクトファイルとで異なります。

  イメージの場合、マッピングシンボルを含むすべてのローカルシンボルが出力シンボルテーブルから削除されます。

  オブジェクトファイルの場合：

  - マッピングシンボルおよびビルド属性はシンボルテーブルに維持されます。
  - 機能を損なうことなく削除できるローカルシンボルは削除されます。

    再配置のターゲットなど、削除できないシンボルは維持されます。このようなシンボルについては、名前が削除されます。fromelf --text の出力結果には、これらが [Anonymous Symbol] としてマークされます。

notes
: ELF ファイルから、**.notes** セクションを削除します。

pathnames
: タイプが STT_FILE であるすべてのシンボルからパス情報を削除します。例えば、C:\work\myobject.o という名前の STT_FILE シンボルは、myobject.o という名前に変更されます。

  注
  デバッグ情報に含まれるパス名は、このオプションでは排除されません。

symbols　これらのオプションの効果は、イメージとオブジェクトファイルとで異なります。

イメージの場合、シンボルテーブル全体とすべてのスタティックシンボルが削除されます。そのようなスタティックシンボルがスタティックな再配置ターゲットとして使用されている場合、再配置情報も削除されます。どの場合でも、STT_FILE シンボルは削除されます。

オブジェクトファイルの場合：

- マッピングシンボルおよびビルド属性はシンボルテーブルに維持されます。
- 機能を損なうことなく削除できるローカルシンボルは削除されます。

  再配置のターゲットなど、削除できないシンボルは維持されます。このようなシンボルについては、名前が削除されます。fromelf --text の出力結果には、これらが [Anonymous Symbol] としてマークされます。

注

シンボル、パス名、ファイルシンボルを外すと、ファイルのデバッグがしにくくなる可能性があります。

4.62.3 例

デバッグ情報を含めて生成した ELF ファイル infile.axf からデバッグ情報を含まない output.axf ファイルを生成するには、以下のように入力します。

```
fromelf --strip=debug,symbols --elf --output=outfile.axf infile.axf
```

4.62.4 関連項目

概念

『リンカの使用』:

- *マッピングシンボルについて*（7-3 ページ）.

参照

- *--elf*（4-31 ページ）
- *--linkview、--no_linkview*（4-55 ページ）
- *--privacy*（4-61 ページ）.

『リンカリファレンス』:

- *--locals、--no_locals*（2-106 ページ）
- *--privacy*（2-128 ページ）.

4.63 --symbolversions、--no_symbolversions

このオプションを指定すると、シンボルバージョン管理テーブルのデコードが無効になります。

4.63.1 制約条件

このオプションと共に --elf を使用する場合は、--output も使用する必要があります。

4.63.2 関連項目

参照

『リンカリファレンス』:

- *シンボルバージョン管理について*（10-27 ページ）.

その他の情報

- *『ARM アーキテクチャ向けベースプラットフォーム ABI』*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0037-/index.html.

.

4.64 --text

このオプションは、イメージ情報をテキスト形式で出力します。このオプションを使用すると、ELF イメージファイルまたは ELF オブジェクトファイルをデコードできます。

コードの出力形式を指定しない場合は、--text が想定されます。つまり、--text を指定しなくてもオプション（複数可）を指定することができます。例えば、fromelf -a は fromelf --text -a と同じ意味です。

コードの出力形式（--bin など）を指定した場合、--text オプションはすべて無視されます。

destination が --output オプションと組み合わせて指定されていない場合、または --output が指定されていない場合は、情報が標準出力（stdout）に出力されます。

4.64.1 構文

--text [*options*]

options は表示する内容です。以下のいずれかを指定できます。

-a
: グローバルデータアドレスとスタティックデータアドレス（構造体とユニオンの内容のアドレスも含む）を出力します。

  このオプションは、デバッグ情報を含むファイルに対してのみ使用できます。デバッグ情報が含まれない場合、警告が表示されます。

  データアドレスのサブセットを出力する場合は、--select オプションを使用します。

  構造体内外で展開された配列のデータアドレスを参照するには、このテキストカテゴリと共に --expandarrays オプションを使用します。

-c
: このオプションは、逆アセンブル対象の元のバイナリデータのダンプおよび命令のアドレスと共に、コードを逆アセンブルします。

  注

  --disassemble とは異なり、逆アセンブリをアセンブラに入力することはできません。

-d
: データセクションの内容を出力します。

-e
: オブジェクトの例外テーブル情報をデコードします。イメージを逆アセンブルするときに、-c と組み合わせて使用します。

-g
: デバッグ情報を出力します。

-r
: 再配置情報を出力します。

-s
: シンボルテーブルとバージョン管理テーブルを出力します。

-t
: ストリングテーブルを出力します。

-v
: イメージの各セグメントヘッダとセクションヘッダに関する詳細情報を出力します。

-w
: 行を折り返しません。

-y
: ダイナミックセグメントの内容を出力します。

-z　　コードサイズとデータサイズを出力します。

これらのオプションはテキストモードでのみ認識されます。

4.64.2 例

以下に、--text の使用例を示します。

- 逆アセンブルされた ELF イメージとシンボルテーブルを含むプレーンテキスト出力ファイルを生成するには、以下のように入力します。

  ```
  fromelf --text -c -s --output=outfile.lst infile.axf
  ```

- すべてのグローバルデータ変数とスタティックデータ変数、すべての構造体フィールドのアドレス一覧を stdout に出力するには、以下のように入力します。

  ```
  fromelf -a --select=* infile.axf
  ```

- inputfile.axf に含まれるすべての構造体のアドレスを保持し、グローバルデータ変数またはスタティックデータ変数に関する情報は保持しないテキストファイルを生成するには、以下のように入力します。

  ```
  fromelf --text -a --select=*.* --output=structaddress.txt infile.axf
  ```

- ネストされた構造体のアドレスのみを含むテキストファイルを生成するには、以下のように入力します。

  ```
  fromelf --text -a --select=*.*.* --output=structaddress.txt infile.axf
  ```

- inputfile.axf に含まれるすべてのグローバル変数またはスタティックデータ変数の情報を保持し、構造体のアドレスは保持しないテキストファイルを生成するには、以下のように入力します。

  ```
  fromelf --text -a --select=*,~*.* --output=structaddress.txt infile.axf
  ```

4.64.3 関連項目

タスク

- *実行可能な ELF イメージ内のシンボルの場所を fromelf を使用して調べる方法*（3-14 ページ）.

『リンカの使用』:

- *イメージに関する情報を取得するためのリンカオプション*（6-2 ページ）.

参照

- *--cpu=name*（4-18 ページ）
- *--disassemble*（4-29 ページ）
- *--emit=option[,option,...]*（4-32 ページ）
- *--expandarrays*（4-35 ページ）
- *--info=topic[,topic,...]*（4-50 ページ）
- *--interleave=option*（4-53 ページ）
- *--only=section_name*（4-59 ページ）
- *--output=destination*（4-60 ページ）
- *--select=select_options*（4-68 ページ）
- *-w*（4-82 ページ）.

4.65 --version_number

このオプションは、使用されている fromelf のバージョンを表示します。

4.65.1 構文

```
fromelf --version_number
```

fromelf は、nnnbbbb 形式のバージョン番号を表示します。各項目には以下の意味があります。

- nnn はバージョン番号です。
- bbbb はビルド番号を示します。

4.65.2 例

バージョン 5.01 ビルド 0019 は 5010019 と表示されます。

4.65.3 関連項目

参照

- *--help*（4-42 ページ）
- *--vsn*（4-81 ページ）

4.66 --vhx

このオプションは、バイト指向（Verilog メモリモデル）16 進形式の出力を作成します。この形式は、*ハードウェア記述言語*（HDL）シミュレータのメモリモデルへのロードに適しています。*--widthxbanks* オプションを使用して、このオプションによって生成される出力を複数ファイルに分割できます。

4.66.1 制約条件

オブジェクトファイルに対してはこのオプションを使用できません。

このオプションは `--output` と組み合わせて使用する必要があります。

4.66.2 関連項目

概念

- *fromelf 使用時の注意事項*（2-4 ページ）.

参照

- *--output=destination*（4-60 ページ）
- *--widthxbanks*（4-83 ページ）

4.67 --via=*file*

file に指定されたオプションを使用するよう fromelf に指示します。

4.67.1 関連項目

参照

『コンパイラリファレンス』:

- 付録 B *via ファイルの構文*.

4.68 --vsn

このオプションは、fromelf のバージョン情報（使用されているライセンスの種類など）を表示します。以下に例を示します。

```
>fromelf --vsn
ARM FromELF, N.nn [Build num]
license_type
ソフトウェアの提供元： ARM Limited
```

4.68.1 関連項目

参照

- *--help*（4-42 ページ）
- *--version_number*（4-78 ページ）.

4.69 -w

このオプションを指定した場合、通常なら複数行で表示されるテキスト出力情報が 1 行で表示されます。

Perl などのテキスト処理ユーティリティで解析する際に、出力結果を処理しやすい形式にすることができます。

以下に例を示します。

```
> fromelf --text -w -c test.axf
==============================================================================

** ELF ヘッダ情報
.
.
.
==============================================================================

** Section #1 '.text' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR]    Size   : 36 bytes (alignment 4)    Address:
0x00000000    $a
     .text
.
.
.
** Section #7 '.rel.text' (SHT_REL)    Size   : 8 bytes (alignment 4)    Symbol table #6 '.symtab'    1
relocations applied to section #1 '.text'
** Section #2 '.ARM.exidx' (SHT_ARM_EXIDX) [SHF_ALLOC + SHF_LINK_ORDER]    Size   : 8 bytes (alignment 4)
Address: 0x
00000000    Link to section #1 '.text'
** Section #8 '.rel.ARM.exidx' (SHT_REL)    Size   : 8 bytes (alignment 4)    Symbol table #6 '.symtab'    1
relocations applied to section #2 '.ARM.exidx'
** Section #3 '.arm_vfe_header' (SHT_PROGBITS)    Size   : 4 bytes (alignment 4)
** Section #4 '.comment' (SHT_PROGBITS)    Size   : 74 bytes
** Section #5 '.debug_frame' (SHT_PROGBITS)    Size   : 140 bytes
** Section #9 '.rel.debug_frame' (SHT_REL)    Size   : 32 bytes (alignment 4)    Symbol table #6 '.symtab'    4
relocations applied to section #5 '.debug_frame'
** Section #6 '.symtab' (SHT_SYMTAB)    Size   : 176 bytes (alignment 4)    String table #11 '.strtab'    Last
local symbol no. 5
** Section #10 '.shstrtab' (SHT_STRTAB)    Size   : 110 bytes
** Section #11 '.strtab' (SHT_STRTAB)    Size   : 223 bytes
** Section #12 '.ARM.attributes' (SHT_ARM_ATTRIBUTES)    Size   : 69 bytes
```

4.69.1 関連項目

参照

- *--text*（4-76 ページ）.

4.70 --widthxbanks

このオプションは、複数のメモリバンク用に複数のファイルを出力します。

複数の設定が指定されている場合、fromelf は最後に指定された設定を使用します。

4.70.1 制約条件

このオプションは --output と組み合わせて使用する必要があります。

4.70.2 構文

--widthxbanks

各項目には以下の意味があります。

banks ターゲットメモリシステム内のメモリバンクの数を指定します。これにより、各ロード領域に生成される出力ファイルの数が決まります。

width ターゲットメモリシステムにおけるメモリの幅を指定します（8 ビット、16 ビット、32 ビット、または 64 ビット）。

有効な設定は以下のとおりです。

```
--8x1
--8x2
--8x4
--16x1
--16x2
--32x1
--32x2
--64x1
```

4.70.3 使用法

イメージに 1 つのロード領域がある場合は、fromelf によって、*banks* で指定された数のファイルが生成されます。ファイル名は、以下の命名規則に基づいて --output=*destination* 引数から付けられます。

- メモリバンクが 1 つの場合は（*banks*=1）、出力ファイルの名前は *destination* です。
- 複数のメモリバンクがある場合（*banks*>1）、fromelf は、*destinationN* に指定された *banks* 数のファイルを生成します。*N* は、0 から *banks*-1 までの範囲です。出力ファイル名のファイル拡張子を指定すると、ファイル拡張子の前に番号 *N* が付けられます。例えば、

  ```
  fromelf --vhx --8x2 test.axf --output=test.txt
  ```

  これにより、test0.txt および test1.txt という名前の 2 つのファイルが生成されます。

イメージに複数のロード領域がある場合、fromelf は、*destination* という名前のディレクトリを作成し、そのディレクトリに各ロード領域用の *banks* ファイルを生成します。各ロード領域用のファイルには *load_regionN* という名前が付けられます。*load_region* はロード領域の名前で、*N* は 0 から *banks*-1 までの範囲です。以下に例を示します。

```
fromelf --vhx --8x2 multiload.axf --output=regions/
```

これにより regions ディレクトリに以下のようなファイルが生成されます。

```
EXEC_ROM0
EXEC_ROM1
RAM0
RAM1
```

width によって指定されたメモリ幅によって、各出力ファイルの 1 行に格納されるメモリ量が制御されます。各出力ファイルのサイズは、読み取るメリのサイズを、作成されるファイル数で分割したものです。以下に例を示します。

- fromelf --vhx --8x4 test.axf --output=file により 4 つのファイル（file0、file1、file2、および file3）が生成されます。各ファイルには、以下のような 1 バイトの行が含まれます。

  ```
  00
  00
  2D
  00
  2C
  8F
  ...
  ```

- fromelf --vhx --16x2 test.axf --output=file により 2 つのファイル（file0 および file1）が生成されます。各ファイルには、以下のような 2 バイトの行が含まれます。

  ```
  0000
  002D
  002C
  ...
  ```

4.70.4 関連項目

参照

- *--bin*（4-6 ページ）
- *--output=destination*（4-60 ページ）
- *--vhx*（4-79 ページ）.

4.71 --workdir=*directory*

このオプションを使用すると、プロジェクトテンプレートに作業ディレクトリを提供するかどうかを指定できます。

注

プロジェクトテンプレートに作業ディレクトリが必要となるのは、作業ディレクトリにデバッガコンフィギュレーションファイルなどのファイル含まれている場合のみです。

注

このオプションの使用は廃止される予定です。

4.71.1 構文

--workdir=*directory*

directory はプロジェクトディレクトリの名前です。

4.71.2 制約条件

--workdir を使用してプロジェクト作業ディレクトリを指定する場合、--project を使用してプロジェクトファイルを指定する必要があります。

4.71.3 エラー

--workdir を指定せずに --project を使用するときに、--workdir が要求された場合、エラーメッセージが表示されます。

4.71.4 関連項目

参照

- *--project=filename?--no_project*（4-62 ページ）
- *--reinitialize_workdir*（4-64 ページ）.

付録 A
『fromelf イメージ変換ユーティリティの使用』に対する改訂

『fromelf イメージ変換ユーティリティの使用』 に対して、以下の技術的変更が加えられました。

表 A-1 発行 F と発行 G の相違点

変更点	関連するトピック
トピックのタイトルにある誤ったアンダースコアを削除し、代替の構文を追加しました。	*--output=destination*（4-60 ページ）

表 A-2 発行 D と発行 F の相違点

変更点	関連するトピック
--device オプションが廃止された旨のメモを追加しました。	• *--device=list*（4-22 ページ） • *--device=name*（4-23 ページ）
`--version_number` および `--vsn` で報告されたバージョン番号を変更しました。	• *--version_number*（4-78 ページ） • *--vsn*（4-81 ページ）.

表 A-3 発行 C と発行 D の相違点

変更点	関連するトピック
`--project=filename`、`--no_project` のトピックタイトルを修正しました。 `--project`、`--reinitialize_workdir`、および `--workdir` オプションの説明についてのメモを追加しました。	• *--project=filename?--no_project*（4-62 ページ） • *--reinitialize_workdir*（4-64 ページ） • *--workdir=directory*（4-85 ページ）.

表 A-4 発行 A と発行 B の相違点

変更点	関連するトピック
fromelf でアーカイブ内のすべてのファイルを処理できるようになった旨のリスト項目を追加しました。	*fromelf イメージ変換ユーティリティについて*（2-2 ページ）
アーカイブ内の ELF ファイルを処理する方法を説明する新しいトピックを追加しました。	*アーカイブ内の ELF ファイルの処理*（3-9 ページ）
fromelf を使用してイメージおよびオブジェクトに含まれるコードを保護する方法および `--privacy` と `--strip` のコマンドラインオプションの使用に関する説明を明確にしました。	*fromelf によるイメージおよびオブジェクトに含まれるコードの保護*（3-10 ページ）
`--decode_build_attributes` コマンドラインオプションの例を追加しました。	*--decode_build_attributes*（4-21 ページ）
`--dump_build_attributes` コマンドラインオプションの例を追加しました。	*--dump_build_attributes*（4-30 ページ）
`--emit` コマンドラインオプションの `build_attributes` オプションの説明を変更しました。	*--emit=option[,option,...]*（4-32 ページ）
`--extract_build_attributes` コマンドラインオプションの例を追加しました。	*--extract_build_attributes*（4-36 ページ）
input_file の説明を変更して、アーカイブ内の ELF ファイルの処理に関する説明を追加しました。	*input_file*（4-51 ページ）
`--[no_]linkview` コマンドラインオプションが廃止された旨を明記しました。	*--linkview、--no_linkview*（4-55 ページ）
`--output` コマンドラインオプションで入力アーカイブファイルを使用する方法に関する情報を追加しました。	*--output=destination*（4-60 ページ）

表 A-4 発行 A と発行 B の相違点 （続き）

変更点	関連するトピック
--privacy コマンドラインオプションの説明を明確にしました。	*--privacy*（4-61 ページ）
--strip コマンドラインオプションの localsymbols オプションと symbols オプションの説明を明確にしました。	*--strip=option[,option,...]*（4-73 ページ）
オブジェクトファイルで使用できない以下のコマンドラインオプションに関する制限を追加しました。--bin、--bincombined、--cad、--cadcombined、--i32、--i32combined、--m32、--m32combined、および --vhx。	• *--bin*（4-6 ページ） • *--bincombined*（4-7 ページ） • *--cad*（4-11 ページ） • *--cadcombined*（4-13 ページ） • *--i32*（4-45 ページ） • *--i32combined*（4-46 ページ） • *--m32*（4-57 ページ） • *--m32combined*（4-58 ページ） • *--vhx*（4-79 ページ）.