ARM® コンパイラツールチェーン

バージョン **4.1**

ARM コンパイラツールチェーンの概要

ARM®

ARM DUI 0529BJ（ID ０１１８１１）

ARM コンパイラツールチェーン
ARM コンパイラツールチェーンの概要

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

著作権

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 ARM コンパイラツールチェーンの概要

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 0529BJ）
- 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 Support and Maintenance, http://www.arm.com/support/services/support-maintenance.php

第 2 章
ARM コンパイラツールチェーンの概要

以下のトピックは、ARM コンパイラツールチェーンの一般的な情報について説明しています。

タスク

概念

- *「ホスト間でのソースファイルの移植性」*（ページ 2-28）
- *「一時ファイルディレクトリの TMP および TMPDIR 環境変数」*（ページ 2-29）
- *「Rogue Wave のマニュアル」*（ページ 2-32）

参照

- *「ARM コンパイラツールチェーンのホストプラットフォームサポート」*（ページ 2-6）
- *「ツールチェーンの環境変数」*（ページ 2-15）
- *「参考資料」*（ページ 2-33）

2.1 ARM コンパイラツールチェーンについて

ARM コンパイラツールチェーンを使用すると、ARM プロセッサファミリ用のアプリケーションを C、C++、または ARM アセンブリ言語ソースからビルドできます。ツールチェーンは以下のもので構成されています。

armcc ARM と Thumb® に対応しているコンパイラ。C コードと C++ コードをコンパイルします。インラインアセンブラおよび組み込みアセンブラをサポートしており、NEON™ ベクトル化コンパイラも含まれています。ベクトル化コンパイラを呼び出すには、以下のコマンドを使用します。

```
armcc --vectorize
```

注

NEON ベクトル化コンパイラは、別途ライセンスされる機能です。

armasm ARM と Thumb に対応しているアセンブラ。ARM および Thumb アセンブリ言語によるソースファイルをアセンブルします。

armlink リンカ。1 つまたは複数のオブジェクトファイルの内容と、1 つまたは複数のオブジェクトライブラリから選択された部分を結合し、実行可能プログラムを生成します。

64 ビットコンピュータ上でより大容量のメモリを使用可能な armlink の 64 ビット版も用意されています。64 ビット版は、このリリースの armlink の 32 ビット版でサポートされているすべての機能をサポートします。

デフォルトでは 32 ビット版が使用されます。

armar ライブラリアン。ELF 形式のオブジェクトファイルをアーカイブやライブラリにまとめて保存するために使用します。これにより、複数の ELF ファイルの代わりにライブラリやアーカイブをリンカに渡すことができます。アーカイブをサードパーティに配布して、さらに高度なアプリケーションの開発に使用してもらうこともできます。

fromelf イメージ変換ユーティリティ。逆アセンブリおよびコードサイズやデータサイズなど、入力イメージに関するテキスト情報を生成することもできます。

C++ ライブラリ

ARM C++ ライブラリには、以下の関数が用意されています。

- C++ コンパイル時に使用するヘルパ関数
- Rogue Wave ライブラリでサポートされていないその他の C++ 関数

C ライブラリ ARM C ライブラリには、以下の機能が用意されています。

- C 標準および C++ 標準で定義されているライブラリ機能の実装
- コンパイラ固有の拡張機能（_fisatty()、__heapstats()、__heapvalid() など）
- GNU 拡張機能
- 多くの C ライブラリで一般的な非標準拡張
- POSIX 拡張機能
- POSIX で標準化されている関数

C マイクロライブラリ

ARM C *マイクロライブラリ*（Microlib）には、高度に最適化された一連の関数が用意されています。これらの関数は、ディープエンベデッドアプリケーション用に、極めて小容量のメモリに収める必要があるアプリケーションで使用されます。

Rogue Wave C++ ライブラリ

Rogue Wave ライブラリは、標準 C++ ライブラリを実装したものです。Rogue Wave ライブラリの詳細については、CD-ROM に収録されている HTML 形式のマニュアルを参照して下さい。

2.1.1 支援ソフトウェア

ツールチェーンの出力のデバッグを、ELF、DWARF 2、および DWARF 3 仕様と互換性のある任意のデバッガで行うことができます。

ツールチェーンのアップデートやパッチについては、ARM Web サイトで随時提供されます。

2.1.2 関連項目

タスク

- *「64 ビットのリンカへの変更」*（ページ 2-7）

概念

- *「ARM コンパイラツールチェーンのホストプラットフォームサポート」*（ページ 2-6）
- *「ツールチェーンのライセンスされた機能」*（ページ 2-11）
- *「ツールチェーンでの標準への準拠」*（ページ 2-12）
- *「ARM アーキテクチャ用 ABI（基本標準）への準拠」*（ページ 2-13）
- *「ツールチェーンでサポートされている ARM アーキテクチャ」*（ページ 2-18）
- *「64 ビットホストプラットフォームでのツールチェーンのサポート」*（ページ 2-20）
- *「Rogue Wave のマニュアル」*（ページ 2-32）
- *「参考資料」*（ページ 2-33）

『コンパイラの使用』:

- 第 2 章 *コンパイラの概要*

『アセンブラの使用』:

- 第 2 章 *アセンブラの概要*

『リンカの使用』:

- 第 2 章 *リンカの概要*

『ARM® C および C++ ライブラリと浮動小数点サポートの使用』:

- 第 2 章 *ARM C ライブラリと C++ ライブラリ*
- 第 3 章 *ARM C マイクロライブラリ*

『armar での静的ソフトウェアライブラリの作成』:

- 第 2 章 *ARM ライブラリアンの概要*

『fromelf イメージ変換ユーティリティの使用』:

- 第 2 章 *fromelf イメージ変換ユーティリティの概要*

その他の情報

- ARM Web サイト , http://www.arm.com

2.2 ARM コンパイラツールチェーンのホストプラットフォームサポート

ARM コンパイラツールチェーンでは、以下の OS プラットフォームをサポートしています。

- Windows 7 Enterprise Edition
- Windows 7 Professional Edition
- Windows Vista Business Edition SP2（32 ビット版および 64 ビット版）
- Windows Vista Enterprise Edition SP2（32 ビット版および 64 ビット版）
- Windows XP Professional SP3（32 ビット版のみ）
- Windows Server 2003
- Solaris 10
- GNOME Window Manager と Bash シェルを使用した Red Hat Enterprise Linux WS version 4 for x86（32 ビット版および 64 ビット版）
- GNOME Window Manager と Bash シェルを使用した Red Hat Enterprise Linux 5 Desktop + Workstation for x86（32 ビット版および 64 ビット版）

2.2.1 関連項目

概念

- *「ARM コンパイラツールチェーンについて」*（ページ 2-3）

2.3 64 ビットのリンカへの変更

インストールは、64 ビットのオペレーティングシステムを使用している場合でも、デフォルトで armlink の標準 32 ビット版を使用するように設定されます。このリリースには、armlink の 64 ビット版が含まれています。64 ビット版では、64 ビットオペレーティングシステムのプロセスがより大容量のメモリにアクセスできます。64 ビット版は、このリリースの armlink でサポートされているすべての機能をサポートします。64 ビットコンピュータに ARM コンパイラツールチェーンをインストールした場合は、64 ビット版を使用できます。

armlink の 64 ビット版を使用するように ARM コンパイラツールチェーンを設定するには、次の手順を実行します。

1. 64 ビット版の実行可能ファイルのディレクトリを指すように ARMCC41BIN 環境変数を変更します。例えば、Windows では、次のように設定します。

   ```
   SET ARMCC41BIN=install_directory\...\win_x86_64;%ARMCC41BIN%
   ```

2. 64 ビット版の実行可能ファイルのディレクトリのパスを PATH 環境変数に追加します。例えば、Windows では、次のように設定します。

   ```
   SET PATH=install_directory\...\win_x86_64;%PATH%
   ```

注

64 ビット版の実行可能ファイルのディレクトリ内の armlink は 64 ビット版ですが、他のツールはすべて 32 ビット版です。これは、ツール間の呼び出し規則により、すべてのツールが同じディレクトリに配置されている必要があるためです。ツールは、次のように特定の条件下で互いを呼び出します。

- armcc が、実行可能ファイルを生成するために armlink を呼び出します。
- armlink が、リンク時コード生成のために armcc を呼び出します（--ltcg）。

これにより、必要に応じて armlink の 64 ビット版が使用されます。

2.3.1 関連項目

概念

『リンカの使用』:

- 第 2 章 *リンカの概要*

参照

- *「ツールチェーンの環境変数」*（ページ 2-15）

『リンカリファレンス』:

- 第 2 章 *リンカコマンドラインオプション*

2.4 ツールチェーンのマニュアルについて

ツールチェーンのマニュアルは以下のように構成されています。

ARM® コンパイラツールチェーンの概要（ARM DUI 0529）- 本書

本書では、ツールチェーンとその機能の概要について説明します。

ARM® プロセッサをターゲットとしたソフトウェア開発（ARM DUI 0471）

このマニュアルでは、ツールチェーンを使用して、ARM アーキテクチャベースのプロセッサ上で実行するソフトウェアを開発する方法について説明します。

『ARM® プロセッサをターゲットとしたソフトウェア開発』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0471b/index.html を参照して下さい。

ARM® コンパイラツールチェーンおよび GNU ライブラリでの Linux アプリケーションのビルド（ARM DUI 0483）

このマニュアルでは、ARM コンパイラツールチェーンと GNU ライブラリを組み合わせて使用して、ARM 組み込み Linux 上で実行するアプリケーションをビルドする方法について説明します。

『ARM® コンパイラツールチェーンおよび GNU ライブラリでの Linux アプリケーションのビルド』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0483b/index.html を参照して下さい。

コンパイラの使用（ARM DUI 0472）

このマニュアルでは、コンパイラ（armcc）のさまざまな機能の使用方法について説明します。

『コンパイラの使用』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0472b/index.html を参照して下さい。

コンパイラリファレンス（ARM DUI 0491）

このマニュアルでは、コンパイラ（armcc）のさまざまな機能の参照情報について説明します。コンパイラの各コマンドラインオプションについても詳細に説明します。

『コンパイラリファレンス』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0491b/index.html を参照して下さい。

ARM® C および C++ ライブラリと浮動小数点サポートの使用（ARM DUI 0475）

このマニュアルでは、ARM C および C++ ライブラリの機能と、その使用方法について説明します。また、ライブラリによる浮動小数点のサポートについても説明します。

『ARM® C および C++ ライブラリと浮動小数点サポートの使用』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0475b/index.html を参照して下さい。

ARM® C ライブラリ、C++ ライブラリ、および浮動小数点サポートリファレンス（ARM DUI 0492）

このマニュアルでは、ARM C ライブラリと C++ ライブラリのさまざまな機能の参照情報について説明します。

『ARM® C ライブラリ、C++ ライブラリ、および浮動小数点サポートリファレンス』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0492b/index.html を参照して下さい。

アセンブラの使用（ARM DUI 0473）

このマニュアルでは、アセンブラ（armasm）のさまざまな機能の使用方法について説明します。

『アセンブラの使用』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0473b/index.html を参照して下さい。

アセンブラリファレンス（ARM DUI 0489）

このマニュアルでは、アセンブラ（armasm）のさまざまな機能の参照情報について説明します。アセンブラの各コマンドラインオプションについても詳細に説明します。

『アセンブラリファレンス』,
http://infocenter.arm.com/help/topic/com.arm.doc.dui0489b/index.html を参照して下さい。

リンカの使用（ARM DUI 0474）

このマニュアルでは、リンカ（armlink）のさまざまな機能の使用方法について説明します。

『リンカの使用』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0474b/index.html を参照して下さい。

リンカリファレンス（ARM DUI 0493）

このマニュアルでは、リンカ（armlink）のさまざまな機能の参照情報について説明します。リンカの各コマンドラインオプションについても詳細に説明します。

『リンカリファレンス』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0493b/index.html を参照して下さい。

armar での静的ソフトウェアライブラリの作成（ARM DUI 0476）

このマニュアルでは、ライブラリアン（armar）のさまざまな機能の使用方法について説明します。armar の各コマンドラインオプションについても詳細に説明します。

『armar での静的ソフトウェアライブラリの作成』,
http://infocenter.arm.com/help/topic/com.arm.doc.dui0476b/index.html を参照して下さい。

fromelf イメージ変換ユーティリティの使用（ARM DUI 0477）

このマニュアルでは、ELF イメージ変換ユーティリティ（fromelf）のさまざまな機能の使用方法について説明します。fromelf の各コマンドラインオプションについても詳細に説明します。

『fromelf イメージ変換ユーティリティの使用』,
http://infocenter.arm.com/help/topic/com.arm.doc.dui0477b/index.html を参照して下さい。

エラーおよび警告リファレンス（ARM DUI 0496）

このマニュアルでは、ARM コンパイラ v4.1 の各ビルドツールで生成されるエラーと警告について説明します。

『エラーおよび警告リファレンス』,
http://infocenter.arm.com/help/topic/com.arm.doc.dui0496b/index.html を参照して下さい。

移行と互換性（ARM DUI 0530）

このマニュアルでは、ソフトウェアをツールチェーンの以前のバージョン（ARM RVCT Compilation Tools v4.0 など）から移行した場合の、ARM コンパイラ v4.1 ツールチェーンでの変更点について説明します。

『移行と互換性』, http://infocenter.arm.com/help/topic/com.arm.doc.dui0530b/index.html を参照して下さい。

2.5 ツールチェーンのライセンスされた機能

ツールチェーンでは、以下のライセンスが必要です。

- ツールチェーンライセンス。
- NEON ベクトル化コンパイラライセンス。NEON ベクトル化コンパイラライセンスを使用すると、コンパイラは Cortex-A8 や Cortex-A9 など、NEON ユニットを持つ ARM プロセッサをターゲットにした NEON 命令を必要に応じて生成できます。
- プロファイラによる最適化のライセンス。これにより、コンパイラは ARM プロファイラによって生成されたデータファイルに基づいて最適化を実行できるようになります。

ツールチェーンを他の ARM 製品と共に購入した場合、含まれているライセンスの詳細については、その製品の *『スタートガイド』* マニュアルを参照して下さい。

ARM 開発ツールのライセンスは、FLEX*net* ライセンス管理システムによって管理されています。

ライセンスを請求するには、*ARM Web のライセンスに関するページ*, http://license.arm.com にアクセスして、オンラインでの指示に従います。

2.5.1 関連項目

概念

『コンパイラの使用』:

- 第 4 章 *NEON ベクトル化コンパイラの使用*
- *「プロファイラによる最適化について」*（ページ 5-3）

参照

『コンパイラリファレンス』:

- *「--vectorize、--no_vectorize」*（ページ 3-133）

その他の情報

- *『ARM® ツール用 FLEXnet ライセンス管理ガイド』*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html
- *『Introducing NEON Development Article』*, http://infocenter.arm.com/help/topic/com.arm.doc.dht0002-/index.html

2.6 ツールチェーンでの標準への準拠

ツールチェーンでは、以下の標準に準拠しています。各標準ごとに、準拠のレベルが記載されています。

ar UNIX 方式のオブジェクトコードアーカイブは armar によって生成され、armlink によって使用されます。armar では、ar 形式のほとんどのオブジェクトコードアーカイブをリストして抽出することができます。また、armlink では、シンボルテーブルのメンバを含んでいる場合、別のアーカイブユーティリティで作成された ar 形式のアーカイブを使用することができます。

DWARF 3 DWARF 3（DWARF Debugging Standard バージョン 3）のデバッグテーブルは、ツールチェーンでサポートされています。

DWARF 2 DWARF 2 のデバッグテーブルは、ツールチェーンでサポートされています。また、ELF や DWARF 2 と互換性のある ARM 製のデバッガでもサポートされています。

ISO C コンパイラでは、ISO C 1990 および 1999 ソースがコンパイラへのインプットとして、サポートされています。

ISO C++ コンパイラでは、ISO C++ 2003 ソースがコンパイラへのインプットとして、サポートされています。

ELF ツールチェーンは、再配置可能なファイルと実行ファイルを ELF 形式で生成します。fromelf ユーティリティを使用すると、ELF ファイルを別の形式のファイルに変換することができます。

注

DWARF 2 および DWARF 3 標準には、フレームデータのデバッグなど、一部あいまいな領域があります。そのため、ARM コード生成ツールによって生成された DWARF をサードパーティ製のデバッガで使用できるかどうか、またはサードパーティ製のデバッガによって生成された DWARF を ARM デバッガで使用できるかどうかは保証しません。

2.6.1 関連項目

概念

- *「ARM アーキテクチャ用 ABI（基本標準）への準拠」*（ページ 2-13）

『コンパイラの使用』:

- *「コンパイラのソース言語モード」*（ページ 2-3）

その他の情報

- The DWARF Debugging Standard, http://dwarfstd.org/
- International Organization for Standardization, http://www.iso.org/iso/home.htm

2.7 ARM アーキテクチャ用 ABI（基本標準）への準拠

Application Binary Interface（ABI）for the ARM Architecture（Base Standard）（BSABI）は、標準の集合です。オープンな標準と ARM アーキテクチャ固有の標準があります。これらの標準により、ベアメタルから ARM Linux などの主要なオペレーティングシステムまで、ARM アーキテクチャベースの実行環境におけるバイナリコードや開発ツールの相互動作が管理されます。

この標準に準拠すると、ツールチェーンで生成されたオブジェクトと、プロデューサの異なるオブジェクトライブラリを連動させることができます。

BSABI は、以下の仕様から構成されています。

AADWARF *『DWARF for the ARM Architecture』*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0040-/index.html この ABI では、DWARF 3 標準を使用して、オブジェクトのプロデューサとデバッガ間のデバッグデータのやり取りを規定しています。

AAELF *ELF for the ARM Architecture*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044-/index.html 汎用的な ELF 標準をベースとし、プロデューサとコンシューマ間でのリンク可能ファイルと実行可能ファイルのやり取りを規定しています。

AAPCS *ARM アーキテクチャ向けプロシージャコール標準*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042-/index.html。実行時の関数間の制御とデータのやり取りを規定しています。ツールチェーンでサポートされている主要な実行環境の種類ごとに、AAPCS のバリアントが用意されています。

BPABI *ARM アーキテクチャ用ベースプラットフォーム ABI*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0037-/index.html。静的なリンカによって生成された実行可能ファイルや共有オブジェクトファイルの形式と内容を規定しています。ポストリンクを使用して、プラットフォーム固有の実行ファイルをサポートします。プラットフォーム ABI を派生させるのに使用する基本標準を提供します。

CLIBABI *ARM アーキテクチャ用 C ライブラリ ABI*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0039-/index.html。ABI と C ライブラリの関係を定義しています。

CPPABI *『C++ ABI for the ARM Architecture』*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0041-/index.htmlIA-64 向けに開発された汎用的な C++ ABI をベースとし、独立した C++ コンパイラ間のインターワークを規定しています。

DBGOVL *『Support for Debugging Overlay Programs』*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0049-/index.html オーバーレイプログラムのデバッグをサポートするための、*ABI for the ARM Architecture* への拡張を定義しています。

EHABI *『Exception Handling ABI for the ARM Architecture』*,
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0038-/index.html 例外がどのようにスローされ処理されるのかについて、言語に依存しない側面と C++ 固有の側面の両方から定義しています。

RTABI 『*Run-time ABI for the ARM Architecture*』, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0043-/index.html 個別に作成されたオブジェクトが浮動小数点とコンパイラヘルパ関数のサポートにおいて実行環境に何を期待できるかを規定しています。

以前のツールチェーンのリリースから ARM コンパイラ v4.1 にアップグレードする場合は、ARM 仕様の最新版を使用して下さい。

2.7.1 関連項目

その他の情報

- *Application Binary Interface for the ARM Architecture Introduction and downloads*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0036-/index.html
- *Addenda to, and Errata in, the ABI for the ARM Architecture*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0045-/index.html
- *Differences between v1 and v2 of the ABI for the ARM Architecture*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0047-/index.html
- *ABI for the ARM Architecture Advisory Note: SP must be 8-byte aligned on entry to AAPCS-conforming functions*, http://infocenter.arm.com/help/topic/com.arm.doc.ihi0046-/index.html

2.8 ツールチェーンの環境変数

ツールチェーンで使用される環境変数は、次のとおりです。

表 2-1 ツールチェーンで使用される環境変数

環境変数	設定
ARMROOT	インストールディレクトリルート（*install_directory*）。
ARMLMD_LICENSE_FILE	ARM ライセンスファイルの場所。この環境変数の情報については、*『ARM® ツール用 FLEXnet ライセンス管理ガイド』*を参照して下さい。
ARMCC41_ASMOPT	標準メイクファイルの外部で使用される追加のアセンブラオプションを定義する、任意に使用できる環境変数。以下に例を示します。 `--licretry` リストされるオプションは、メイクファイルの `armasm` コマンドで指定されるオプションの前に表示されます。したがって、この環境変数でリストされるオプションは、メイクファイルで指定されるオプションによってオーバーライドされます。
ARMCC41_CCOPT	標準メイクファイルの外部で使用される追加のコンパイラオプションを定義する、任意に使用できる環境変数。以下に例を示します。 `--licretry` リストされるオプションは、メイクファイルの `armcc` コマンドで指定されるオプションの前に表示されます。したがって、この環境変数でリストされるオプションは、メイクファイルで指定されるオプションによってオーバーライドされます。
ARMCC41_FROMELFOPT	標準メイクファイルの外部で使用される追加の `fromelf` イメージ変換ユーティリティオプションを定義する、任意に使用できる環境変数。以下に例を示します。 `--licretry` リストされるオプションは、メイクファイルの `fromelf` コマンドで指定されるオプションの前に表示されます。したがって、この環境変数でリストされるオプションは、メイクファイルで指定されるオプションによってオーバーライドされます。
ARMCC41_LINKOPT	標準メイクファイルの外部で使用される追加のリンカオプションを定義する、任意に使用できる環境変数。以下に例を示します。 `--licretry` リストされるオプションは、メイクファイルの `armlink` コマンドで指定されたオプションの前に表示されます。したがって、この環境変数でリストされるオプションは、メークファイルで指定されるオプションによってオーバーライドされます。

表 2-1 ツールチェーンで使用される環境変数 （続き）

環境変数	設定
ARMCC41BIN	32 ビットのツールチェーン実行可能ファイルの場所。パスは、使用しているホストプラットフォームと ARM 製品により異なります。 *install_directory*\...*platform* *platform* には以下のいずれかを指定できます。 • win_32-pentium（Windows の場合） • linux-pentium（Linux の場合） armlink の 64 ビット版を使用するには、64 ビット版のディレクトリを指すようにこの環境変数を定義します。*platform* には以下のいずれかを指定できます。 • win_x86_64（Windows の場合） • linux-x86_64（Linux の場合） 64 ビット版のディレクトリには、他の ARM ツールの 32 ビット版の実行可能ファイルも含まれています。
ARMCC41INC	コンパイラのインクルードファイルの場所。 *install_directory*\...\include
ARMCC41LIB	ARM C および C++ ライブラリファイルの場所。 *install_directory*\...\lib **注** パスの末尾にパスの区切り文字を含めた場合、リンカはそのディレクトリとサブディレクトリを検索します。つまり、C:\...\lib\ の場合、リンカは以下の場所を検索します。 lib lib\armlib lib\cpplib

注

ツールチェーンを他の ARM 製品のコンポーネントとして入手した場合、ARMCC41BIN、ARMCC41INC、および ARMCC41LIB 環境変数の設定は任意の使用になります。詳細については、ARM 製品の *『スタートガイド』* を参照して下さい。

2.8.1 関連項目

概念

『アセンブラの使用』:
- 第 2 章 *アセンブラの概要*

『コンパイラの使用』:
- 第 2 章 *コンパイラの概要*

『リンカの使用』:
- 第 2 章 *リンカの概要*

『ARM® C および C++ ライブラリと浮動小数点サポートの使用』:
- 第 2 章 *ARM C ライブラリと C++ ライブラリ*

『fromelf イメージ変換ユーティリティの使用』:
- 第 2 章 *fromelf イメージ変換ユーティリティの概要*

『armar での静的ソフトウェアライブラリの作成』:

- 第 2 章 *ARM ライブラリアンの概要*

参照

『アセンブラリファレンス』:

- 第 2 章 *アセンブラコマンドラインオプション*

『コンパイラリファレンス』:

- 第 3 章 *コンパイラのコマンドラインオプション*

『リンカリファレンス』:

- 第 2 章 *リンカコマンドラインオプション*

『fromelf イメージ変換ユーティリティの使用』:

- 第 4 章 *fromelf コマンドリファレンス*

その他の情報

- *『ARM® ツール用 FLEXnet ライセンス管理ガイド』*,
  http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html

2.9 ツールチェーンでサポートされている ARM アーキテクチャ

ツールチェーンでは、現在 ARM でサポートされている、ARMv4™ 以降のすべての ARM アーキテクチャがサポートされています。これには ARM NEON テクノロジも含まれます。ARMv4 より前のすべてのアーキテクチャは使用されなくなり、サポートされていません。

ターゲットプロセッサまたはアーキテクチャを指定すると、選択されたプロセッサまたはアーキテクチャ固有の追加機能を利用できます。そのためには、次のコマンドラインオプションを使用します。

- `--cpu=`*name*
- `--device=`*name*
- `--fpu=`*name*

`--arm` または `--thumb` コマンドラインオプションを使用して、起動命令セット（ARM または Thumb）を指定できます。また、`--arm_only` オプションを使用して、ARM 専用命令セットを指定することもできます。

コンパイルツールは、ARM コードと Thumb コードの混用をサポートしています。これはインターワークと呼ばれ、ARM コードと Thumb コード間の分岐を可能にします。

2.9.1 関連項目

タスク

『コンパイラの使用』:
- *「コンパイル時のターゲット CPU の選択」*（ページ 6-9）

概念

『コンパイラの使用』:
- *「NEON テクノロジ」*（ページ 4-3）

『ARM® プロセッサをターゲットとしたソフトウェア開発』:
- 第 2 章 *ARM アーキテクチャバージョンの主な機能*
- 第 5 章 *ARM と Thumb のインターワーク*

参照

『コンパイラリファレンス』:
- *「--arm」*（ページ 3-13）
- *「--arm_only」*（ページ 3-19）
- *「--cpu=name」*（ページ 3-36）
- *「--device=name」*（ページ 3-49）
- *「--fpu=name」*（ページ 3-65）
- *「--thumb」*（ページ 3-124）

『アセンブラリファレンス』:
- *「--arm」*（ページ 2-6）
- *「--arm_only」*（ページ 2-7）
- *「--cpu=name」*（ページ 2-9）
- *「--device=name」*（ページ 2-10）
- *「--fpu=name」*（ページ 2-16）

- 「--thumb」（ページ 2-26）

『リンカリファレンス』:

- 「--arm_only」（ページ 2-14）
- 「--cpu=name」（ページ 2-35）
- 「--device=name」（ページ 2-40）
- 「--fpu=name」（ページ 2-73）

その他の情報

- 『Introducing NEON Development Article』,
  http://infocenter.arm.com/help/topic/com.arm.doc.dht0002-/index.html
- *NEON Support in Compilation Tools Development Article*,
  http://infocenter.arm.com/help/topic/com.arm.doc.dht0004-/index.html

2.10 64 ビットホストプラットフォームでのツールチェーンのサポート

ツールチェーンは特定の 64 ビットプラットフォームでサポートされていますが、ツール自体は 32 ビットアプリケーションです。そのため、ツールで使用可能な仮想アドレス空間とファイルサイズが制限されます。これらの制限を超えると、メモリが不足していることを示すエラーメッセージが armlink によって報告されます。その結果、十分な物理メモリがあるにもかかわらず、アプリケーションがメモリにアクセスできないために混乱を引き起こすことがあります。

注

64 ビットコンピュータ上でより大容量のメモリを使用可能な armlink の 64 ビット版が用意されています。64 ビット版は、このリリースの armlink の 32 ビット版でサポートされているすべての機能をサポートします。

2.10.1 関連項目

タスク

- *「64 ビットのリンカへの変更」*（ページ 2-7）

概念

- *「ARM コンパイラツールチェーンについて」*（ページ 2-3）

参照

- *「ツールチェーンの環境変数」*（ページ 2-15）

2.11 コンパイルツールのコマンドラインでの特殊文字の使用

一部のコンパイルツールコマンド引数で複数のシンボル名を選択するには、以下のような特殊文字を使用できます。

- ワイルドカード文字 * を使用すると、任意の名前と一致させることができます。
- ワイルドカード文字 ? を使用すると、任意の 1 文字と一致させることができます。

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。

例えば、「`*,~*.*`」ではなく「`'*,~*.*'`」と入力します。

注

`armar` コマンドラインオプションは、前に - を付ける必要があります。一部の以前のバージョンの `armar` および一部のサードパーティのアーカイバとは、この点が異なります。

2.11.1 関連項目

参照

『アセンブラの使用』:

- *「アセンブラのコマンドライン構文」*（ページ 7-2）
- *「アセンブラコマンドの一覧（グループ別）」*（ページ 7-3）

『コンパイラの使用』:

- *「コンパイラのコマンドライン構文」*（ページ 3-3）
- *「コンパイラコマンドラインオプションのグループ別一覧」*（ページ 3-4）

『リンカの使用』:

- *「リンカのコマンドライン構文」*（ページ 2-4）
- *「リンカコマンドラインオプションの目的別一覧」*（ページ 2-5）

『armar での静的ソフトウェアライブラリの作成』:

- *「armar のコマンドラインのシンタックス」*（ページ 2-4）
- *「armar コマンドラインオプションの目的別一覧」*（ページ 2-5）

『fromelf イメージ変換ユーティリティの使用』:

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

2.12 コンパイルツールのコマンドラインオプションの規則

コンパイルツールの多くの動作を、コマンドラインオプションを使用して制御できます。

オプションのタイプに基づいて、以下の規則が適用されます。

1 文字オプション

すべての 1 文字オプション、または引数を取る 1 文字オプションには、その前に単一ダッシュ（-）が付けられます。オプションと引数の間にはスペースを挿入することができます。引数はオプションの直後に指定することもできます。以下に例を示します。

`-J directory`

`-Jdirectory`

キーワードオプション

すべてのキーワードオプション、および引数を取るキーワードオプションには、その前に二重ダッシュ（--）が付けられます。オプションと引数の間には、スペースまたは = 文字を挿入する必要があります。以下に例を示します。

```
--depend=file.d
```

```
--depend file.d
```

コンパイルツールオプションの先頭以外で使用されている - または _ には、どちらの文字を使用することもできます。例えば、`--force_new_nothrow` は `--force-new-nothrow` と同じです。

名前がダッシュで始まるファイルをコンパイルするには、POSIX オプション -- を使用して、後続のすべての引数がコマンドスイッチではなくファイル名として処理されるように指定します。例えば、`-ifile_1` というファイルをコンパイルするには、以下のコマンドを使用します。

```
armcc -c -- -ifile_1
```

一部の Unix シェルでは、一部のコマンドラインオプションで引数を使うときに、次のように引用符を付ける必要があります。

```
--keep='s.o(vect)'
```

2.12.1 関連項目

参照

『アセンブラの使用』:
- *「アセンブラのコマンドライン構文」*（ページ 7-2）
- *「アセンブラコマンドの一覧（グループ別）」*（ページ 7-3）

『コンパイラの使用』:
- *「コンパイラのコマンドライン構文」*（ページ 3-3）
- *「コンパイラコマンドラインオプションのグループ別一覧」*（ページ 3-4）

『リンカの使用』:
- *「リンカのコマンドライン構文」*（ページ 2-4）
- *「リンカコマンドラインオプションの目的別一覧」*（ページ 2-5）

『armar での静的ソフトウェアライブラリの作成』:

- *「armar のコマンドラインのシンタックス」*（ページ 2-4）
- *「armar コマンドラインオプションの目的別一覧」*（ページ 2-5）

『fromelf イメージ変換ユーティリティの使用』:

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

2.13 コンパイルツールのコマンドラインオプションの順序

一般的に、コマンドラインオプションは任意の順序で使用できます。ただし、その結果がコマンドラインで組み合わされる他の関連オプションによって異なるオプションもあります。

オプションが同じコマンドラインにある他のオプションをオーバーライドする場合、コマンドラインの最後に近いオプションが優先されます。この規則に従っていないオプションについては、そのオプションに関する説明でその点が明記されています。

`--show_cmdline` オプションを使用すると、コマンドラインがどのように処理されるかを確認できます。コマンドは正規化されて表示されます。

2.13.1 関連項目

参照

『コンパイラリファレンス』:

- 「*--show_cmdline*」（ページ 3-118）

『アセンブラリファレンス』:

- 「*--show_cmdline*」（ページ 2-25）

『リンカリファレンス』:

- 「*--show_cmdline*」（ページ 2-146）

『fromelf イメージ変換ユーティリティの使用』:

- 「*--show_cmdline*」（ページ 4-71）

『armar での静的ソフトウェアライブラリの作成』:

- 「*--show_cmdline*」（ページ 6-28）

2.14 コンパイルツールのコマンドラインオプションの自動補完

自動補完機能を使うと、コマンドラインオプションを短い形式で指定できます。

自動補完機能を使用するには、自動補完する文字列の後にピリオド（.）を入力します。

自動補完機能には、以下の規則が適用されます。

- 引数とピリオドは、等号（=）文字またはスペース文字で区切らなければなりません。
- オプションの引数に自動補完を使用することはできません。
- 自動補完されたオプションが一意になるように十分な数の文字を含める必要があります。

例えば、コマンドラインで `--diag_suppress=223` を指定するには、`--diag_su.=223` を使用します。

`--diag.=223` という指定では、`--diag.` が単一の一意なコマンドラインオプションを特定できないので、有効ではありません。

2.14.1 関連項目

参照

『アセンブラの使用』:
- *「アセンブラコマンドの一覧（グループ別）」*（ページ 7-3）

『コンパイラの使用』:
- *「コンパイラコマンドラインオプションのグループ別一覧」*（ページ 3-4）

『リンカの使用』:
- *「リンカコマンドラインオプションの目的別一覧」*（ページ 2-5）

『armar での静的ソフトウェアライブラリの作成』:
- *「armar コマンドラインオプションの目的別一覧」*（ページ 2-5）

『fromelf イメージ変換ユーティリティの使用』:
- *「fromelf コマンドラインオプションの目的別一覧」*（ページ 2-7）

2.15 テキストファイルによるコマンドラインオプションの指定

一部のオペレーティングシステムでは、コマンドラインの長さが制限されています。以下のいずれかの方法で対処できます。

- この制限を超えるオプションを、テキストファイルに入れて指定します。
- すべてのコマンドラインオプションをテキストファイルに入れます。

テキストファイルを使ってコマンドラインオプションを指定するには、次のようにします。

1. 必要なコマンドラインオプションを含むテキストファイルを作成します。すべてのオプションを 1 行で指定する必要があります。以下に例を示します。

   ```
   --debug --cpu=ARM926EJ-S
   ```

2. 必要なオプションを含むファイルの場所を指定するには、--via コマンドラインオプションを使用します。以下に例を示します。

   ```
   armcc --via myoptions.txt
   ```

   ファイル名の拡張子は、任意のものを使うことができます。拡張子を付けなくてもかまいません。

2.15.1 テキストファイル内で使用する場合のコマンドラインオプションの優先度

コンパイラは、指定されたファイルからコマンドラインオプションを読み出して、コマンドラインで指定された追加のオプションと結合します。コマンドラインオプションの優先度は、以下のものによって決まります。

- コマンドラインオプション
- コマンドラインでの --via オプションの位置

オプションの優先度を確認するには、--show_cmdline オプションを指定します。例えば、armcc.txt にオプション --debug --cpu=ARM926EJ-S が含まれている場合は、次のようになります。

- ```
  armcc -c --show_cmdline --cpu=ARM7TDMI --via=armcc.txt hello.c
  [armcc --show_cmdline --debug -c --cpu=ARM926EJ-S hello.c]
  ```

  この場合、--cpu=ARM7TDMI は使用されません。コマンドラインで、--cpu=ARM926EJ-S が --cpu の最後のインスタンスであるためです。

- ```
  armcc --via=armcc.via -c --show_cmdline --cpu=ARM7TDMI hello.c
  [armcc --show_cmdline --debug -c hello.c]
  ```

  この場合、--cpu=ARM926EJ-S は使用されません。コマンドラインで、--cpu=ARM7TDMI が --cpu の最後のインスタンスであるためです。また、--cpu=ARM7TDMI は --cpu のデフォルトオプションなので、出力には表示されません。

2.15.2 関連項目

参照

『アセンブラリファレンス』:

- *「--show_cmdline」*（ページ 2-25）
- *「--via=file」*（ページ 2-27）

『コンパイラリファレンス』:

- *「--show_cmdline」*（ページ 3-118）

- 「--via=filename」（ページ 3-134）
- 付録 B *via* ファイルの構文

『リンカリファレンス』:

- 「--show_cmdline」（ページ 2-146）
- 「--via=file」（ページ 2-187）

『armar での静的ソフトウェアライブラリの作成』:

- 「--show_cmdline」（ページ 6-28）
- 「--via=file」（ページ 6-35）

『fromelf イメージ変換ユーティリティの使用』:

- 「--show_cmdline」（ページ 4-71）
- 「--via=file」（ページ 4-80）

2.16 ホスト間でのソースファイルの移植性

ホスト間でのソースファイルの移植性を確保するには、以下の規則に従って下さい。

- ファイル名にはスペースを含めないで下さい。スペースを含むパス名やファイル名を使用する必要がある場合は、パス名またはファイル名を二重引用符（"）または単一引用符（'）で囲んで下さい。
- パス名を組み込む場合は、絶対パスではなく相対パスを指定して下さい。
- 組み込むパス名ではバックスラッシュ（\）ではなくスラッシュ（/）を使用して下さい。

2.16.1 関連項目

概念

- 第 2 章 *ARM コンパイラツールチェーンの概要*

2.17 一時ファイルディレクトリの TMP および TMPDIR 環境変数

コンパイルツールは、ファイルを処理するときに一時ディレクトリを使用します。

- Windows プラットフォームでは、環境変数 TMP で一時ファイルに使用されるディレクトリが指定されます。TMP が定義されていないか、存在しないディレクトリの名前に設定されている場合、一時ファイルは現在の作業ディレクトリに作成されます。
- Red Hat Linux プラットフォームでは、環境変数 TMPDIR で一時ファイルに使用されるディレクトリが指定されます。TMPDIR が設定されていない場合には、デフォルトの一時ディレクトリが使用されます。通常、このディレクトリは /tmp または /var/tmp です。

TMP および TMPDIR は通常、システム管理者によって設定されます。ただし、ユーザが変更できるようにすることも可能です。

2.17.1 関連項目

概念

- 第 2 章 *ARM コンパイラツールチェーンの概要*

2.18 環境変数を使用したコマンドラインオプションの指定

コマンドラインオプションは、以下に示すツール固有の環境変数の値を設定することによって指定できます。

- アセンブラでは ARMCC41_ASMOPT
- コンパイラでは ARMCC41_CCOPT
- fromelf イメージ変換ユーティリティでは ARMCC41_FROMELFOPT
- リンカでは ARMCC41_LINKOPT

構文は、コマンドライン構文と同じです。コンパイルツールは、環境変数の値を読み出し、コマンド文字列の前に挿入します。つまり、環境変数で指定されたオプションをコマンドラインの引数でオーバーライドできます。

2.18.1 関連項目

リファレンス

- *「ツールチェーンの環境変数」*（ページ 2-15）

2.19 Windows のコンパイルツールでの Cygwin パスの指定

Windows のデフォルトでは、コンパイルツールはパス名が Windows DOS 形式であることが必要です（例えば C:\myfiles）。Cygwin パス名を使用する場合は、CYGPATH 環境変数をシステムの cygpath.exe ファイルの場所に設定します。以下に例を示します。

```
set CYGPATH=C:/cygwin/bin/cygpath.exe
```

これで、コンパイルツールのコマンドラインオプションで、Cygwin パス形式を使ってファイルの場所を指定できるようになります。例えば、ファイル /cygdrive/h/main.c をコンパイルするには、次のコマンドを入力します。

```
armcc -c --debug /cygdrive/h/main.c
```

2.19.1 関連項目

参照

『アセンブラリファレンス』:

- 第 2 章 *アセンブラコマンドラインオプション*

『コンパイラリファレンス』:

- 第 3 章 *コンパイラのコマンドラインオプション*

『リンカリファレンス』:

- 第 2 章 *リンカコマンドラインオプション*

『armar での静的ソフトウェアライブラリの作成』:

- 第 6 章 *armar コマンドリファレンス*

『fromelf イメージ変換ユーティリティの使用』:

- 第 4 章 *fromelf コマンドリファレンス*

2.20 Rogue Wave のマニュアル

Rogue Wave Standard C++ ライブラリをコンパイルツールと共に使用するためのマニュアルは、ARM 製品のドキュメントディレクトリにインストールされています。以下のマニュアルを表示するには、標準の Web ブラウザを使用します。

- *『Standard C++ ライブラリクラスリファレンス』*:

  `install_directory\document_folder\...\stdref\index.htm`

- *『Standard C++ ライブラリユーザガイド - OEM エディション』*:

  `install_directory\document_folder\...\stdug\index.htm`

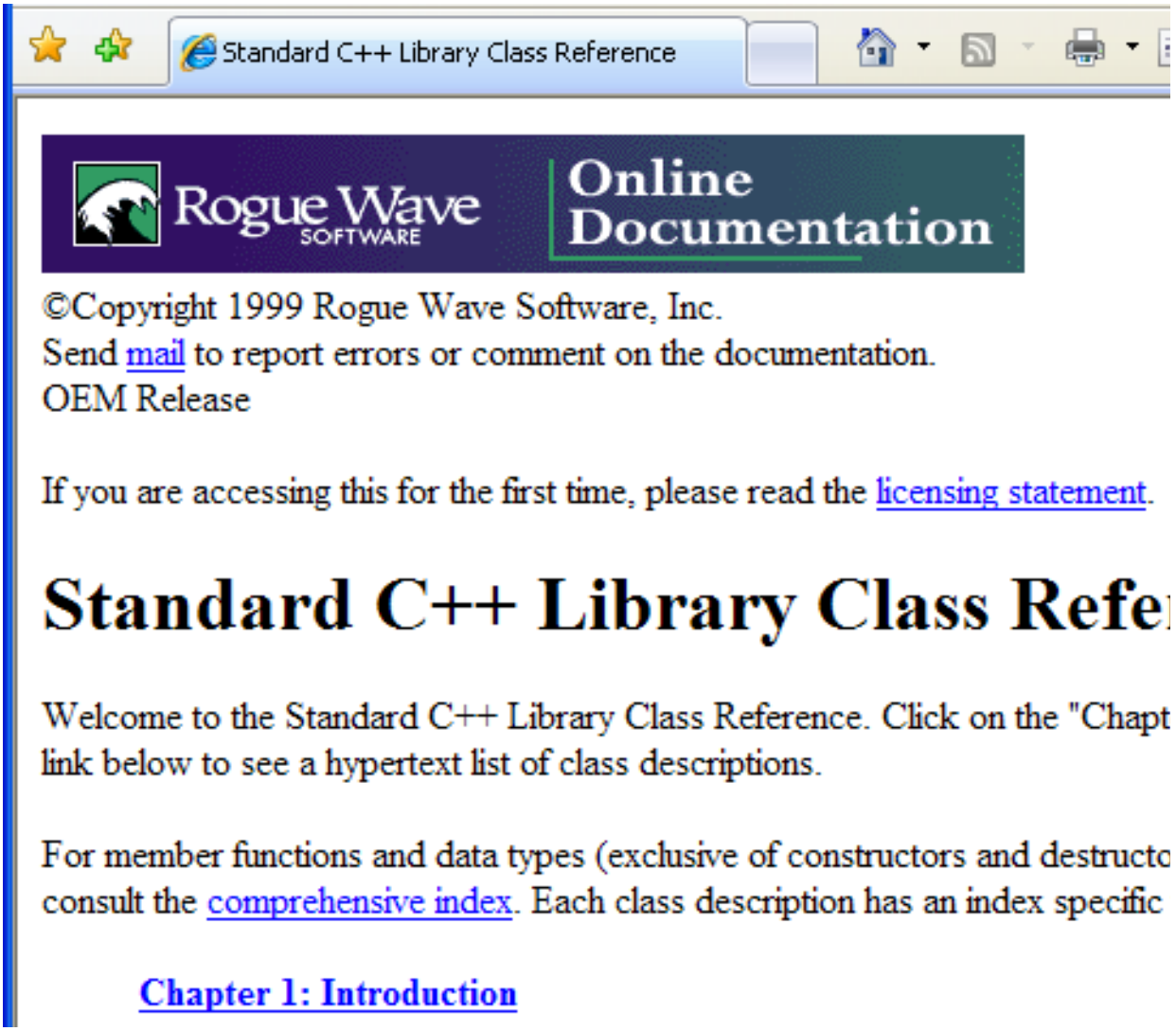

図 2-1 Rogue Wave の HTML マニュアル

2.20.1 関連項目

その他の情報

- Rogue Wave Software, http://www.roguewave.com

2.21 参考資料

ARM プロセッサファミリのコード開発に関する補足情報は、ARM からもサードパーティからも入手できます。

2.21.1 ARM の出版物

ARM は自社出版物の定期的な更新・修正を行っています。最新の正誤表、追補表、および ARM に関する FAQ については、ARM Infocenter, http://infocenter.arm.com/help/index.jsp を参照して下さい。

準拠する基本標準、ソフトウェアインタフェース、および ARM がサポートしている規格の詳細については、*『Application Binary Interface (ABI) for the ARM Architecture』*, http://infocenter.arm.com/help/topic/com.arm.doc.subset.swdev.abi/index.html を参照して下さい。

ARM 製品に関する特定の情報については、以下のマニュアルを参照して下さい。

- 『ARM アーキテクチャリファレンスマニュアル』, http://infocenter.arm.com/help/topic/com.arm.doc.subset.arch.reference/index.html
- Cortex-A シリーズプロセッサ , http://infocenter.arm.com/help/topic/com.arm.doc.set.cortexa/index.html
- Cortex-R シリーズプロセッサ , http://infocenter.arm.com/help/topic/com.arm.doc.set.cortexr/index.html
- Cortex-M シリーズプロセッサ , http://infocenter.arm.com/help/topic/com.arm.doc.set.cortexm/index.html
- ARM11 プロセッサ , http://infocenter.arm.com/help/topic/com.arm.doc.set.arm11/index.html
- ARM9 プロセッサ , http://infocenter.arm.com/help/topic/com.arm.doc.set.arm9/index.html
- ARM7 プロセッサ , http://infocenter.arm.com/help/topic/com.arm.doc.set.arm7/index.html
- Vector floating-point coprocessors, http://infocenter.arm.com/help/topic/com.arm.doc.set.vfp/index.html

2.21.2 他の出版物

ARM コンパイラツールのマニュアルは、C や C++ プログラミング言語の概要を説明することを意図したものではありません。そのため、C や C++ プログラミングを解説する C や C++ 標準のリファレンスマニュアルではありません。プログラミングに関する一般情報については、他の出版物を参照して下さい。

C++ 言語については以下の出版物を参照して下さい。

- *ISO/IEC 14882:2003, C++ Standard*.
- Stroustrup, B., *The C++ Programming Language* (3rd edition, 1997).Addison-Wesley Publishing Company, Reading, Massachusetts.ISBN 0-201-88954-4.

C++ プログラミングに関する一般情報については、以下の出版物を参照して下さい。

- Stroustrup, B., *The Design and Evolution of C++* (1994).Addison-Wesley Publishing Company, Reading, Massachusetts.ISBN 0-201-54330-3.

  初期の設計から現在使用されている言語へ C++ がどのように進化したかを説明しています。
- Vandevoorde, D and Josuttis, N.M. *C++ Templates: The Complete Guide* (2003).Addison-Wesley Publishing Company, Reading, Massachusetts.ISBN 0-201-73484-2.
- Meyers, S., *Effective C++* (1992). Addison-Wesley Publishing Company, Reading, Massachusetts.ISBN 0-201-56364-9.

効果的な C++ 開発を端的に分かりやすく説明したガイドラインです。

- Meyers, S., *More Effective C++* (2nd edition, 1997).Addison-Wesley Publishing Company, Reading, Massachusetts.ISBN 0-201-92488-9.

C プログラミングに関する一般情報については、以下の出版物を参照して下さい。

- ISO/IEC 9899:1999, *C Standard.*

  この標準規格は、国家規格団体（フランスの AFNOR、アメリカの ANSI など）から入手できます。

- Kernighan, B.W. and Ritchie, D.M., *The C Programming Language* (2nd edition, 1988).Prentice-Hall, Englewood Cliffs, NJ, USA.ISBN 0-13-110362-8.

  この本は C 言語の当初の設計者および実装者による共著で、改訂版では ANSI C の要点が追加されています。

- Harbison, S.P. and Steele, G.L., *A C Reference Manual* (5th edition, 2002). Prentice-Hall, Englewood Cliffs, NJ, USA.ISBN 0-13-089592-X.

  ANSI C に関する有益な情報を盛り込みながら、C を非常に詳しく解説しているガイドです。

- Plauger, P., *The Standard C Library* (1991).Prentice-Hall, Englewood Cliffs, NJ, USA.ISBN 0-13-131509-9.

  C ライブラリに関連する ANSI 標準規格と ISO 標準規格の総合的なガイドです。

- Koenig, A., *C Traps and Pitfalls,* Addison-Wesley (1989), Reading, Mass.ISBN 0-201-17928-8.

  C プログラミングで直面する最も一般的な問題点の回避方法を説明しています。C の初心者から上級者までを対象とした参考書です。

Debug With Arbitrary Record Format（DWARF）デバッグテーブルの標準と ELF 仕様の最新情報については、The DWARF Debugging Standard の Web サイト , http://www.dwarfstd.org を参照して下さい。

以下の出版物は、*European Telecommunications Standards Institute*（ETSI）の基本操作について説明しています。

- ETSI G.191 勧告：*Software tools for speech and audio coding standardization*
- *ITU-T Software Tool Library 2005 User's manual*。ETSI G.191 勧告の一部として含まれています。
- ETSI G723.1 勧告：*Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s*
- ETSI G.729 勧告：*Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP)*

これらの出版物はすべて International Telecommunication Union (ITU) の Web サイト , http://www.itu.int から入手できます。

Texas Instruments のコンパイラ組み込み機能に関する出版物は、Texas Instruments の Web サイト , http://www.ti.com から入手できます。

第 3 章
アプリケーションの作成

以下の各トピックでは、ツールチェーンを使用してアプリケーションを作成する方法について説明します。

タスク

3.1 コンパイルツールの使用

通常、アプリケーション開発には、以下のファイルや作業が必要になります。

- メインアプリケーションの C/C++ ソースコード（armcc）
- 割り込みサービスルーチンなど、ハードウェアに近いコンポーネントのアセンブリソースコード（armasm）
- すべてのオブジェクトをリンクしてイメージを生成（armlink）
- イメージをフラッシュメモリ用の形式（プレーンバイナリ形式、Intel Hex 形式、Motorola-S 形式）に変換（fromelf）

以下の図は、通常のアプリケーション開発でコンパイルツールがどのように使用されているかを示しています。

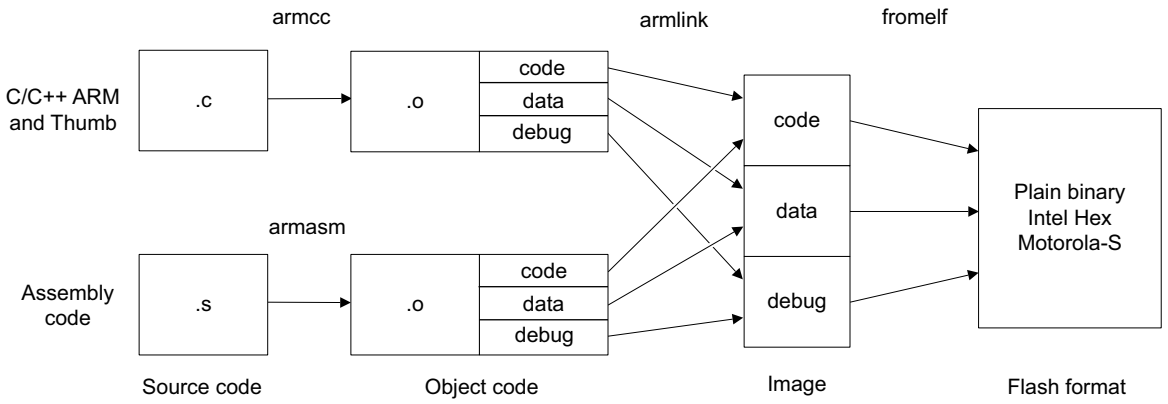

図 3-1 一般的なツール使用の流れ図

3.1.1 関連項目

タスク

- *「コンパイラの使用」*（ページ 3-3）
- *「リンカの使用」*（ページ 3-5）
- *「アセンブラの使用」*（ページ 3-6）
- *「fromelf イメージ変換ユーティリティの使用」*（ページ 3-7）

3.2 コンパイラの使用

コンパイラ（armcc）を使用すると、C および C++ ソースコードを ARM および Thumb コードにコンパイルできます。

通常、コンパイラを起動するには以下のように入力します。

```
armcc [options] ifile_1 ... ifile_n
```

複数の入力ファイルを指定できます。

3.2.1 サンプルのコンパイル

C++ サンプルソースファイル shapes.cpp をコンパイルするには、次のようにします。

1. C++ ファイル shapes.cpp を次のコマンドでコンパイルします。

   ```
   armcc --cpp --debug -c -O1 shapes.cpp -o shapes.o
   ```

   一般的に使用されるオプションは以下のとおりです。

-c	コンパイルのみ（リンクなし）を行うようにコンパイラに指示します。
-cpp	コンパイラにソースが C++ であることを指定します。
--debug	ソースレベルのデバッグを行うためのデバッグテーブルを追加するようにコンパイラに指示します。
-O1	満足できるデバッグビューと優れたコード密度およびパフォーマンスが得られる、制限された最適化を使用してコードを生成するようにコンパイラに指示します。
-o *filename*	指定した *filename* でオブジェクトファイルを作成するようにコンパイラに指示します。

   注

   --arm は、デフォルトのコンパイラオプションです。

2. 次のように、ファイルをリンクします。

   ```
   armlink shapes.o --info totals -o shapes.axf
   ```

3. ELF、DWARF 2、および DWARF 3 と互換性のあるデバッガを使用して、イメージをロードして実行します。

詳細については、サンプル付属の readme.txt ファイルを参照して下さい。

3.2.2 ARM コードのコンパイル

以下のコンパイラオプションを使用すると、ARM コードが生成されます。

--arm	Thumb コードではなく ARM コードを優先して生成するようにコンパイラに指示します。ただし、#pragma thumb を使用すると、このオプションはオーバーライドされます。 これはデフォルトのコンパイラオプションです。
--arm_only	ARM コードのみを生成するようにコンパイラに指示します。コンパイラは、Thumb コードがターゲットアーキテクチャに存在しないかのように動作します。#pragma thumb 宣言は無視されます。

3.2.3 Thumb コードのコンパイル

Thumb バージョンをビルドするには、以下のコマンドを使用します。

```
armcc --thumb ...
```

各オプションには以下の意味があります。

`--thumb` ARM コードではなく Thumb コードを優先して生成するようにコンパイラに指示します。ただし、`#pragma arm` を使用すると、このオプションはオーバーライドされます。

3.2.4 関連項目

タスク

- *「リンカの使用」*（ページ 3-5）

『コンパイラの使用』:

- 第 3 章 *コンパイラの使い方*

参照

『コンパイラリファレンス』:

- *「コマンドラインオプション」*（ページ 3-7）
- *「--arm」*（ページ 3-13）
- *「--arm_only」*（ページ 3-19）
- *「-c」*（ページ 3-26）
- *「--debug、--no_debug」*（ページ 3-42）
- *「-g」*（ページ 3-69）
- *「-o filename」*（ページ 3-98）
- *「-Onum」*（ページ 3-99）
- *「--thumb」*（ページ 3-124）
- *「#pragma arm」*（ページ 5-54）
- *「#pragma thumb」*（ページ 5-67）

3.3 リンカの使用

リンカは、複数のオブジェクトファイルの内容をオブジェクトライブラリの選択した部分と結合して、イメージファイルまたはオブジェクトファイルを生成します。

通常、リンカを起動するには以下のように入力します。

```
armlink [options] file_1 ... file_n
```

3.3.1 サンプルのリンク

オブジェクトファイル shapes.o をリンクするには、以下のコマンドを使用します。

```
armlink shapes.o --info totals -o shapes.axf
```

各オプションには以下の意味があります。

`-o` 出力ファイルとして shapes.axf を指定します。

`--info totals` 入力オブジェクトとライブラリについて、コードとデータのサイズの合計を表示するようにリンカに指示します。

3.3.2 関連項目

タスク

『リンカの使用』:

- 第 3 章 *armlink でサポートされているリンクモデル*

参照

『リンカリファレンス』:

- 第 2 章 *リンカコマンドラインオプション*

3.4 アセンブラの使用

アセンブラ（armasm）を使用するための基本的な構文を以下に示します。

```
armasm [options] inputfile
```

例えば、myfile.s というファイル内のコードをアセンブルし、生成されるオブジェクトファイル内にデバッグ情報を含めるには、以下のように入力します。

```
armasm --debug myfile.s
```

これにより、myfile.o というオブジェクトファイルが生成されます。

3.4.1 アセンブラソースからのサンプルのビルド

例えば word.s というアセンブラプログラムをビルドするには、次のようにします。

1. 以下のコマンドを使用して、ソースファイルをアセンブルします。
   ```
   armasm --debug word.s
   ```
2. 以下のコマンドを使用して、ファイルをリンクします。
   ```
   armlink word.o -o word.axf
   ```
3. ELF、DWARF 2、および DWARF 3 と互換性のあるデバッガを使用して、イメージをロードして実行します。
   プログラムをステップスルーし、レジスタがどのように変更されるかを調べます。

3.4.2 関連項目

タスク

『アセンブラの使用』:

- *「アセンブラのコマンドライン構文」*（ページ 7-2）

参照

『アセンブラリファレンス』:

- 第 2 章 *アセンブラコマンドラインオプション*

3.5 fromelf イメージ変換ユーティリティの使用

fromelf イメージ変換ユーティリティには以下の特徴があります。

- ELF 実行可能形式の実行可能イメージを他のファイル形式に変換します。
- 出力ファイル内にデバッグ情報を含めるかどうかを制御します。
- ELF イメージや ELF オブジェクトファイルを逆アセンブルします。
- 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。
- ELF イメージや ELF オブジェクトファイルに関する情報を出力します。

3.5.1 fromelf の使用例

以下に、fromelf ユーティリティの使用例を示します。

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

逆アセンブルされたコードと ELF イメージのシンボルテーブルを含むプレーンテキスト形式の出力ファイルを生成します。

```
fromelf --bin --16x2 --output=outfile.bin infile.axf
```

2 バンク、16 ビットメモリ幅のメモリコンフィギュレーションを持つターゲットシステム向けに、バイナリ形式で 2 つのファイル（outfile0.bin および outfile1.bin）を生成します。

2 番目の使用例で生成される出力ファイルは、16 ビットフラッシュメモリデバイスへの直接書き込みに適しています。

3.5.2 関連項目

タスク

『fromelf イメージ変換ユーティリティの使用』:

- 第 3 章 *fromelf ユーティリティの使用*

参照

『fromelf イメージ変換ユーティリティの使用』:

- 第 4 章 *fromelf コマンドリファレンス*

付録 A
『ARM コンパイラツールチェーンの概要』に対する改訂

『ARM コンパイラツールチェーンの概要』 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 A と発行 B の相違点

変更点	関連するトピック
ツールの一覧に armlink の 64 ビット版に関する詳細を追加しました。	*「ARM コンパイラツールチェーンについて」*（ページ 2-3）
64 ビットのリンカを使用するための変更方法に関するトピックを追加しました。	*「64 ビットのリンカへの変更」*（ページ 2-7）
64 ビットのリンカを使用するために必要な環境変数を追加しました。	*「ツールチェーンの環境変数」*（ページ 2-15）
armlink の 64 ビット版に関するメモを追加しました。	*「64 ビットホストプラットフォームでのツールチェーンのサポート」*（ページ 2-20）