ARM® コンパイラ

バージョン 5.04

移行と互換性ガイド

ARM

ARM DUI0530JJ

ARM® コンパイラ

移行と互換性ガイド

リリース情報

ドキュメント履歴

発行	日付	機密保持ステータス	変更点
A	28 5 月 2010	非機密扱い	ARM コンパイラ v4.1 リリース
B	30 9 月 2010	非機密扱い	ARM コンパイラ v4.1 のアップデート 1
C	28 1 月 2011	非機密扱い	ARM コンパイラ v4.1 パッチ 3 のアップデート 2
D	30 4 月 2011	非機密扱い	ARM コンパイラ v5.0 リリース
E	29 7 月 2011	非機密扱い	ARM コンパイラ v5.0 のアップデート 1
F	30 9 月 2011	非機密扱い	ARM コンパイラ v5.01 リリース
G	29 2 月 2012	非機密扱い	ARM コンパイラ v5.01 リリースマニュアルの更新 1
H	27 7 月 2012	非機密扱い	ARM コンパイラ v5.02 リリース
I	31 1 月 2013	非機密扱い	ARM コンパイラ v5.03 リリース
J	16 12 月 2013	非機密扱い	ARM コンパイラ v5.04 リリース

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。

無制限アクセスは、ARM 社内による分類です。

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

www.arm.com

目次
ARM® コンパイラ 移行と互換性ガイド

表の一覧
ARM® コンパイラ 移行と互換性ガイド

序章

この前書きでは、次について紹介します。*ARM® コンパイラ 移行と互換性ガイド*.

このドキュメントは、次で構成されています。

本書について

ARM コンパイラ移行と互換性ガイド。このマニュアルは、最新のリリースバージョンと以前のバージョン間の互換性情報を提供します。PDF で提供されています。

本書の構成

本書は以下の章から構成されています。

第 章 1 各バージョンの ARM Compilation Tools のコンフィギュレーション情報

各バージョンの ARM Compilation Tools でサポートされている FlexNet、GCC、および Cygwin バージョンについて説明します。

第 章 2 ARM コンパイラ v5.03 から v5.04 への移行

ARM コンパイラ v5.03 と v5.04 の間での移行と互換性に影響する変更点について説明します。

第 章 3 ARM コンパイラ v5.0.2 から v5.03 への移行

ARM コンパイラ v5.02 と v5.03 の間での移行と互換性に影響する変更点について説明します。

第 章 4 ARM コンパイラ v5.0 から v5.01 以降への移行

ARM コンパイラ v5.0 と v5.01 以降の間での移行と互換性に影響する変更点について説明します。

第 章 5 ARM コンパイラ v4.1 パッチ 3 以降から v5.0 への移行

ARM コンパイラ v4.1 パッチ 3 と v5.0 の間での移行と互換性に影響する変更点について説明します。

第 章 6 ARM コンパイラ v4.1 ビルド 561 から v4.1 パッチ 3 以降への移行

ARM コンパイラ v4.1 ビルド 561 と v4.1 パッチ 3 以降の間での移行と互換性に影響する変更点について説明します。

第 章 7 ARM コンパイラ v4.1 から v4.1 ビルド 561 への移行

ARM コンパイラ v4.1 と v4.1 ビルド 561 の間での移行と互換性に影響する変更点について説明します。

第 章 8 RVCT v4.0 から ARM コンパイラ v4.1 への移行

RVCT v4.0 と ARM コンパイラ v4.1 の間での移行と互換性に影響する変更点について説明します。

第 章 9 RVCT v3.1 から RVCT v4.0 への移行

RVCT v3.1 と RVCT v4.0 の間での移行と互換性に影響する変更点について説明します。

第 章 10 RVCT v3.0 から RVCT v3.1 への移行

RVCT v3.0 と RVCT v3.1 の間での移行と互換性に影響する変更点について説明します。

第 章 11 RVCT v2.2 から RVCT v3.0 への移行

RVCT v2.2 と RVCT v3.0 の間での移行と互換性に影響する変更点について説明します。

付録 A『移行と互換性』マニュアルに対する改訂

『移行と互換性ガイド』に対して加えられた技術的変更について 説明しています。

表記規則

italic
: 重要用語、相互参照、引用箇所を示します。

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、必要に応じて記述リスト内の重要箇所、ARM プロセッサの信号名、重要用語、および専門用語にも太字を使用しています。

`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。

`mono`space
: コマンドまたはオプションに使用可能な略語を示しています。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`
: 引数が特定の値で置き換えられる場合のモノスペーステキストの引数を示しています。

`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。

<and>
: コードまたはコードの一部のアセンブラ構文で置換可能な項が使用されている場合に、その項を囲みます。例えば、

```
MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
```

スモールキャピタル
: 「*ARM 用語集*」で定義されている専門的な意味を持つ用語について、本文中で使用されます。例えば、IMPLEMENTATION DEFINED、IMPLEMENTATION SPECIFIC、UNKNOWN、UNPREDICTABLE などです。

ご意見、ご感想

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。

- 製品名
- 製品のリビジョンまたはバージョン
- 説明にはできるだけ多くの情報を含めて下さい。適宜、症状と診断手順も含めて下さい。

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 DUI0530JJ）
- 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。

第 1 章
各バージョンの ARM Compilation Tools のコンフィギュレーション情報

各バージョンの ARM Compilation Tools でサポートされている FlexNet、GCC、および Cygwin バージョンについて説明します。

このドキュメントは、次で構成されています。

1.1 サポートされている FlexNet のバージョン

各バージョンの ARM® コンパイラ は、異なるバージョンの FLEXnet をサポートしています。

コンパイルツールでの FlexNet のバージョンは、次のとおりです。

表 1-1 FlexNet バージョン

コンパイルツールのバージョン	Windows	Linux
ARM コンパイラツールチェーン 5.04	11.10.1.0	11.10.1.0
ARM コンパイラツールチェーン 5.03	11.10.1.0	11.10.1.0
ARM コンパイラツールチェーン 5.02	10.8.10.0	10.8.10.0
ARM コンパイラツールチェーン 5.01	10.8.10.0	10.8.10.0
ARM コンパイラツールチェーン 5.0	10.8.7.0	10.8.7.0
ARM コンパイラツールチェーン 4.1	10.8.7.0	10.8.7.0
RVCT 4.0 ビルド 471	10.8.7.0	10.8.7.0
RVCT 4.0	10.8.5.0	9.2
RVCT 3.1 ビルド 836	10.8.7.0	10.8.7.0
RVCT 3.1 ビルド 739	10.8.5.0	10.8.5.0
RVCT 3.1	10.8.5.0	9.2
RVCT 3.0	10.8.5.0	10.8.0
RVCT 2.2	9.0.0	9.0.0
RVCT 2.1	9.0.0	9.0.0
RVCT 2.0	8.1b	8.1b
ADS 1.2	7.2i	7.2i

関連情報

ARM DS-5 ライセンス管理ガイド.

1.2 エミュレートされる GCC のバージョン

各バージョンの ARM コンパイラ は、異なるバージョンの GCC をサポートしています。

コンパイルツールでエミュレートされる GCC のバージョンは、次のとおりです。

表 1-2 GCC のバージョン

コンパイルツールのバージョン	GCC のバージョン
ARM コンパイラツールチェーン 5.04	4.2.0
ARM コンパイラツールチェーン 5.03	4.2.0
ARM コンパイラツールチェーン 5.02	4.2.0
ARM コンパイラツールチェーン 5.01	4.2.0
ARM コンパイラツールチェーン 5.0	4.2.0
ARM コンパイラツールチェーン 4.1	4.2.0
RVCT 4.0	4.2.0
RVCT 3.1	3.3.0

関連情報

--gnu_version=version (armcc).

1.3 サポートされている Cygwin のバージョン

各バージョンの ARM コンパイラ は、異なるバージョンの Cygwin をサポートしています。

コンパイルツールでサポートされる Cygwin のバージョンは、次のとおりです。

表 1-3 サポートされている Cygwin のバージョン

コンパイルツールのバージョン	Cygwin のバージョン
ARM コンパイラツールチェーン 5.04	1.7.25
ARM コンパイラツールチェーン 5.03	1.7.15
ARM コンパイラツールチェーン 5.02	1.7.15
ARM コンパイラツールチェーン 5.01	1.7.7.1
ARM コンパイラツールチェーン 5.0	1.7.7.1

注

ARM コンパイラ は、サポートされている Windows プラットフォーム上の Cygwin でも使用できます。ただし、CYGPATH によって有効になる Cygwin パス変換は 32 ビットの Windows プラットフォームでのみサポートされており、Windows 8 ではサポートされていません。

関連情報

Windows のコンパイルツールでの Cygwin パスの指定について.

第 2 章
ARM コンパイラ v5.03 から v5.04 への移行

ARM コンパイラ v5.03 と v5.04 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

2.1 ARM コンパイラ v5.03 と v5.04 の間でのマニュアルの変更点

ARM コンパイラ v5.04 ではさまざまな変更がマニュアルに加えられました。

マニュアルの構成とタイトルの変更

コンパイラ、アセンブラ、リンカ、ARM C および C++ の各ライブラリに対するユーザマニュアルとリファレンスマニュアルが結合され、すべてのマニュアルのタイトルが変更されています。変更は次の表にまとめられています。

5.03 マニュアル	5.04 マニュアル
ARM コンパイラツールチェーンの概要	*スタートガイド*
ARM プロセッサをターゲットとしたソフトウェア開発	*ソフトウェア開発ガイド*
コンパイラの使用	*armcc ユーザガイド*
コンパイラリファレンス	*『armcc ユーザガイド』*に結合
アセンブラの使用	*armasm ユーザガイド*
アセンブラリファレンス	*『armasm ユーザガイド』*に結合
リンカの使用	*armlink ユーザガイド*
リンカリファレンス	*『armlink ユーザガイド』*に結合
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 C および C++ ライブラリと浮動小数点サポートユーザガイド*
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 C および C++ ライブラリと浮動小数点サポートユーザガイド』*に結合
armar での静的ソフトウェアライブラリの作成	*armar ユーザガイド*
fromelf イメージ変換ユーティリティの使用	*fromelf ユーザガイド*
エラーおよび警告リファレンス	*エラーおよび警告リファレンスガイド*

マニュアルの技術的な変更

ARM コンパイラマニュアルに加えられた技術的な変更については、以下の改訂の要約を参照して下さい。

- *『移行と互換性ガイド』に対する改訂* (本書)
- *『スタートガイド』に対する改訂*
- *『ソフトウェア開発ガイド』に対する改訂*
- *『armcc ユーザガイド』に対する改訂*
- *『armasm ユーザガイド』に対する改訂*
- *『armlink ユーザガイド』に対する改訂*
- *『ARM C および C++ ライブラリと浮動小数点サポートユーザガイド』に対する改訂*
- *『armar ユーザガイド』に対する改訂*
- *『fromelf ユーザガイド』に対する改訂*
- *『エラーおよび警告リファレンスガイド』に対する改訂*

第 3 章
ARM コンパイラ v5.0.2 から v5.03 への移行

ARM コンパイラ v5.02 と v5.03 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

3.1 ARM コンパイラ v5.02 と v5.03 の間でのコンパイラの変更点

ARM コンパイラツールチェーン v5.03 ではさまざまな変更が **armcc** に加えられました。

ARM コンパイラツールチェーン v5.03 ではコンパイラに以下の変更が加えられました。

- **armcc** メッセージ番号 3001 ～ 4001 が修正されました。したがって、診断メッセージを非表示にする場合、メッセージ番号の変更が必要となることがあります。

関連情報

ツールチェーンの環境変数.
--version_number アセンブラオプション.
--version_number コンパイラオプション.
--version_number リンカオプション.
--version_number fromelf オプション.
--version_number armar オプション.

3.2 ARM コンパイラ v5.02 と v5.03 の間でのマニュアルの変更点

ARM コンパイラツールチェーン v5.03 ではさまざまな変更がマニュアルに加えられました。

ARM コンパイラツールチェーンのマニュアルに加えられた技術的な変更については、以下の改訂の要約を参照して下さい。

- *『移行と互換性』に対する改訂* (本書)
- *『ARM コンパイラツールチェーンの概要』に対する改訂*
- *『ARM プロセッサをターゲットとしたソフトウェア開発』に対する改訂*
- *『コンパイラの使用』に対する改訂*
- *『アセンブラの使用』に対する改訂*
- *『リンカの使用』*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『armar での静的ソフトウェアライブラリの作成』に対する改訂*
- *『fromelf イメージ変換ユーティリティの使用』に対する改訂*
- *『エラーおよび警告リファレンス』に対する改訂*
- *『アセンブラリファレンス』に対する改訂*
- *『コンパイラリファレンス』に対する改訂*
- *『リンカリファレンス』に対する改訂*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*

——— 注 ———

『*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*』は、ARM コンパイラのマニュアルの一部としては今後提供されません。

———————————

第 4 章
ARM コンパイラ v5.0 から v5.01 以降への移行

ARM コンパイラ v5.0 と v5.01 以降の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

4.1 ARM コンパイラ v5.0 と v5.01 以降の間での全般的な変更点

ARM コンパイラツールチェーン v5.01 ではさまざまな一般的な変更が加えられました。

ARM コンパイラツールチェーン v5.01 ではコンパイラに以下の変更が加えられました。

- ARM コンパイラ 5.01 以降では個別の NEON コンパイラライセンスは必要ありません。
- バージョン固有の環境変数は、`ARMCC5INC` などのように、1 桁のバージョンを使用するように変更されました。
- `--version_number` を使用してツールによって報告されたバージョン番号が変更されました。

  — バージョン 5.0 では、形式は `VVbbbb` です。

  — バージョン 5.01 以降では、形式は `VVVbbbb` です。

  例えば、バージョン 5.01 ビルド 2345 は `5012345` として報告されます。

関連情報

ツールチェーンの環境変数.

--version_number アセンブラオプション.

--version_number コンパイラオプション.

--version_number リンカオプション.

--version_number fromelf オプション.

--version_number armar オプション.

4.2 ARM コンパイラ v5.0 と v5.01 以降の間でのマニュアルの変更点

ARM コンパイラツールチェーン v5.01 ではさまざまな変更がマニュアルに加えられました。

ARM コンパイラツールチェーンのマニュアルに加えられた技術的な変更については、以下の改訂の要約を参照して下さい。

- *『移行と互換性』に対する改訂* (本書)
- *『ARM コンパイラツールチェーンの概要』に対する改訂*
- *『ARM プロセッサをターゲットとしたソフトウェア開発』に対する改訂*
- *『コンパイラの使用』に対する改訂*
- *『アセンブラの使用』に対する改訂*
- *『リンカの使用』*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『armar での静的ソフトウェアライブラリの作成』に対する改訂*
- *『fromelf イメージ変換ユーティリティの使用』に対する改訂*
- *『エラーおよび警告リファレンス』に対する改訂*
- *『アセンブラリファレンス』に対する改訂*
- *『コンパイラリファレンス』に対する改訂*
- *『リンカリファレンス』に対する改訂*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『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』に対する改訂*

第 5 章
ARM コンパイラ v4.1 パッチ 3 以降から v5.0 への移行

ARM コンパイラ v4.1 パッチ 3 と v5.0 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

5.1 4.1 ARM コンパイラ v4.1 パッチ 3 以降と v5.0 の間での全般的な変更点

ARM コンパイラツールチェーン v5.0 ではさまざまな一般的な変更が加えられました。

ARM コンパイラツールチェーン v5.0 ではコンパイラに以下の変更が加えられました。

- このツールでは、環境変数の設定はまったく必要でなくなりました。
- デフォルトヘッダやライブラリディレクトリを検出するための追加の命名規則が v5.0 ツールに追加されました。環境変数または関連するコマンドラインオプションがない場合は、以下のようになります。

  — コンパイラは、`../include` 内を検索します。

  — リンカは、`../lib` 内を検索します。

  これらの場所は、DS-5 `bin` ディレクトリからインクルードディレクトリおよびライブラリディレクトリへの相対パスと一致します。

関連情報

ツールチェーンの環境変数.

5.2 ARM コンパイラ v4.1 パッチ 3 以降と v5.0 の間でのコンパイラの変更点

ARM コンパイラツールチェーン v5.0 ではさまざまな変更が armcc に加えられました。

コンパイラには以下の変更が加えられています。

- コンパイラが使用する *Edison Design Group* (EDG) フロントエンドのバージョンが、4.1 に更新されました。ただし、この更新による互換性の問題は発生しません。
-
- `ARMCC50INC` が設定されていない場合に `-J` がコマンドラインにない場合、コンパイラは、**`armcc.exe`** の場所に対して相対的な場所にある `../include` 内でデフォルトのインクルードを検索します。
- GCC 互換性が改善され、GCC フォールバックモードをサポートするようになりました。
- *リンク時コード生成*(LTCG)機能が廃止されました。これに代わる方法として、ARM では `--multifile` コンパイラオプションを使用することを推奨します。
- `--profile` を使用したプロファイラによる最適化が廃止され、現時点では、ARM Streamline との互換性がなくなりました。

関連情報

アプリケーションをビルドする場合の GCC フォールバックの使用.
コンパイラの検索規則と現在の場所.
-Jdir[,dir,...] コンパイラオプション.
--multifile, --no_multifile コンパイラオプション.
-Warmcc,--gcc_fallback コンパイラオプション.
定義済みマクロ.
ツールチェーンの環境変数.

5.3 ARM コンパイラ v4.1 パッチ 3 以降と v5.0 の間でのリンカの変更点

ARM コンパイラツールチェーン v5.0 ではさまざまな変更が armlink に加えられました。

リンカには、以下の変更が加えられました。

- `ARMCC50LIB` が設定されておらず、`--libpath` がコマンドラインにない場合、リンカは `../lib` 内でデフォルトのライブラリを検索します。このパスは、`armlink.exe` の場所に相対しています。
- *リンク時コード生成*(LTCG)機能が廃止されました。これに代わる方法として、ARM では `--multifile` コンパイラオプションを使用することを推奨します。
- `--profile` を使用したプロファイラによる最適化が廃止され、現時点では、ARM Streamline との互換性がなくなりました。

関連情報

--libpath=pathlist リンカオプション.
--multifile, --no_multifile コンパイラオプション.
ツールチェーンの環境変数.

5.4 ARM コンパイラ v4.1 パッチ 3 言おうと v5.0 の間でのマニュアルの変更点

ARM コンパイラ v5.0 ではさまざまな変更がマニュアルに加えられました。

ARM コンパイラツールチェーンのマニュアルに加えられた技術的な変更については、以下の改訂の要約を参照して下さい。

- *『移行と互換性』に対する改訂* (本書)
- *『ARM コンパイラツールチェーンの概要』に対する改訂*
- *『ARM プロセッサをターゲットとしたソフトウェア開発』に対する改訂*
- *『コンパイラの使用』に対する改訂*
- *『アセンブラの使用』に対する改訂*
- *『リンカの使用』*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『armar での静的ソフトウェアライブラリの作成』に対する改訂*
- *『fromelf イメージ変換ユーティリティの使用』に対する改訂*
- *『エラーおよび警告リファレンス』に対する改訂*
- *『アセンブラリファレンス』に対する改訂*
- *『コンパイラリファレンス』に対する改訂*
- *『リンカリファレンス』に対する改訂*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『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』に対する改訂*

第 6 章
ARM コンパイラ v4.1 ビルド 561 から v4.1 パッチ 3 以降への移行

ARM コンパイラ v4.1 ビルド 561 と v4.1 パッチ 3 以降の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

6.1 ARM コンパイラ v4.1 ビルド 561 と v4.1 パッチ 3 以降の間での C および C++ ライブラリの変更点

ARM コンパイラツールチェーン v4.1 パッチ 3 ではさまざまな変更が ARM C および C++ ライブラリに加えられました。

`alloca()` の新しい実装によって、メモリがヒープ上ではなくスタック上に割り当てられるようになりました。これによって、古いヒープベースの `alloca()` を実装したソフトウェアとの互換性の問題が発生することはありません。ただし、そのようなソフトウェアには、必要なくなった演算が含まれている可能性があります。例えば、このような演算には、使用されなくなった `Alloca` 状態に対する `__user_perthread_libspace` の実装などがあります。

関連情報

ARM C および C++ ライブラリのライブラリヒープ使用の要件.
C ライブラリによる __user_libspace 静的データ領域の使用（なし）.
C ライブラリを使用しないアプリケーションの作成.

第 7 章
ARM コンパイラ v4.1 から v4.1 ビルド 561 への移行

ARM コンパイラ v4.1 と v4.1 ビルド 561 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

7.1 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間でのコンパイラの変更点

ARM コンパイラツールチェーン v4.1 ビルド 561 ではさまざまな変更が armcc に加えられました。

不完全な非配列型を含む宣言で `at` 属性を使用すると、エラーになります。例えば、`foo` が宣言されていない場合、以下のコードではエラーが発生します。

```
struct foo a __attribute__((at(0x16000)));
```

関連情報

__attribute__((at(address))) 変数属性.

7.2 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間でのリンカの変更点

ARM コンパイラ v4.1 と v4.1 ビルド 561 の間の移行に影響を与えるような技術的な変更は **armlink** に加えられていません。

7.3 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間でのアセンブラの変更点

ARM コンパイラ v4.1 と v4.1 ビルド 561 の間の移行に影響を与えるような技術的な変更は **armasm** に加えられていません。

7.4 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間での C および C++ ライブラリの変更点

ARM コンパイラツールチェーン v4.1 ビルド 561 ではさまざまな変更が ARM C および C++ ライブラリに加えられました。

下位互換性のためにシンボル `__use_accurate_range_reduction` が維持されていますが、このシンボルは効果を持ちません。

ライブラリの以前のハードウェア浮動小数点バージョンの C99 複素数関数は、*hardfp* リンケージ関数のみで、*softfp* リンケージ関数はありませんでした。新しいライブラリには、*hardfp* リンケージ関数と *softfp* リンケージ関数の両方が含まれています。これは、ライブラリから複素関数を呼び出すときに、ハードウェア浮動小数点を使用するようにビルドされた既存のオブジェクトコードが適切に動作しない可能性があることを示します。この場合、リンカは警告を発行します。影響を受ける関数を使用する可能性があり、ハードウェア浮動小数点を使用するようにビルドされているコードをすべて再コンパイルする必要があります。さらに、それらを新しいライブラリにリンクし直す必要があります。

関連情報

--apcs=qualifier...qualifier コンパイラオプション.
--fpu=name コンパイラオプション.

7.5 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間での fromelf の変更点

fromelf で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るようになりました。

関連情報

input_file fromelf オプション.

7.6 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間でのマニュアルの変更点

ARM コンパイラツールチェーン v4.1 ビルド 561 ではさまざまな変更がマニュアルに加えられました。

ARM コンパイラツールチェーンのマニュアルに加えられた技術的な変更については、以下の改訂の要約を参照して下さい。

- *『移行と互換性』に対する改訂* (本書)
- *『ARM コンパイラツールチェーンの概要』に対する改訂*
- *『ARM プロセッサをターゲットとしたソフトウェア開発』に対する改訂*
- *『コンパイラの使用』に対する改訂*
- *『アセンブラの使用』に対する改訂*
- *『リンカの使用』*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『armar での静的ソフトウェアライブラリの作成』に対する改訂*
- *『fromelf イメージ変換ユーティリティの使用』に対する改訂*
- *『エラーおよび警告リファレンス』に対する改訂*
- *『アセンブラリファレンス』に対する改訂*
- *『コンパイラリファレンス』に対する改訂*
- *『リンカリファレンス』に対する改訂*
- *『ARM C および C++ ライブラリと浮動小数点サポートの使用』に対する改訂*
- *『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』に対する改訂*

第 8 章
RVCT v4.0 から ARM コンパイラ v4.1 への移行

RVCT v4.0 と ARM コンパイラ v4.1 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

8.1 RVCT v4.0 と ARM コンパイラ v4.1 の間での全般的な変更点

ARM コンパイラツールチェーン v4.1 ではさまざまな一般的な変更が加えられました。

デフォルトヘッダやライブラリディレクトリなどを設定する環境変数の命名規則が変更されました。接頭辞として、`RVCT` ではなく、`ARMCC` が使われるようになりました。例えば、`RVCT40INC` ではなく、`ARMCC41INC` になります。

ARM コンパイラ v4.1 と従来のオブジェクトおよびライブラリとの互換性

`--apcs /adsabi` を使用してコードをビルドしていない場合、RVCT v2.x、v3.x、および v4.0 のオブジェクトおよびライブラリとの下位互換性がサポートされます。

以上のような制限事項があるため、ユーザやサードパーティによって提供されるライブラリを含むプロジェクト全体を ARM コンパイラ v4.1 を使用して再ビルドすることをお勧めします。これは、潜在的な非互換性の問題を回避し、ARM コンパイラ v4.1 が提供する向上した最適化機能、拡張機能、および新機能を十分に活用するためです。

関連情報

ツールチェーンの環境変数.

8.2 RVCT v4.0 と ARM コンパイラ v4.1 の間でのコンパイラの変更点

ARM コンパイラツールチェーン v4.1 ではさまざまな変更が armcc に加えられました。

列挙子の符号の規則が慣例に合わせて変更されました。列挙子コンテナは、負の定数が定義されていない限り、符号なしになりました。RVCT v4.0 10Q1 パッチは、この変更を GCC モードのみにします。

`-O3` は、`--multifile` を意味しなくなりました。オプション、`--multifile` オプションは、これまでも個別のオプションとして使用可能でした。これをビルドに含めることを推奨します。

関連参照

8.1 RVCT v4.0 と ARM コンパイラ v4.1 の間での全般的な変更点 (8-37 ページ).

関連情報

--multifile, --no_multifile コンパイラオプション.
-Onum コンパイラオプション.

8.3 RVCT v4.0 と ARM コンパイラ v4.1 の間でのリンカの変更点

ARM コンパイラツールチェーン v4.1 ではさまざまな変更が armlink に加えられました。

リンカについては、以下の点が変更されています。

GNU ld スクリプトのサポート

armlink v4.1 では、GNU リンカ制御スクリプトのサブセットがサポートされています。さらに GNU ld の動作と一致させるために、`--sysv` コマンドラインオプションが指定されると、**armlink** は内部リンカ制御スクリプトを使用します。**armlink** の以前のバージョンでは、内部スキャッタファイルが使用されていました。

制御スクリプトを使用することにより、RVCT v4.0 と論理的には同等であるものの、物理的には異なるレイアウトが生成されます。デフォルトのスキャッタファイルレイアウトに戻すには、コマンドラインオプション `--no_use_sysv_default_script` を使用します。

内部制御スクリプトをユーザ定義制御スクリプトに置き換えるには、`-T` オプションを使用します。

ARM/Thumb 同義語の廃止

4.1 では、使用が制限された ARM/Thumb 同義語機能のサポートが廃止されました。ARM/Thumb 同義語機能は、シンボル `S` の ARM グローバルシンボル定義と `S` の Thumb グローバルシンボル定義の共存を可能にします。ARM ステートからの分岐はすべて ARM 定義で送られ、Thumb ステートからの分岐はすべて Thumb 定義で送られます。

4.0 の **armlink** は、ARM/Thumb 同義語を検出したときに、廃止予定の機能に関する警告 `L6455E` を出力します。

4.1 の **armlink** は、ARM/Thumb 同義語を検出したときに、エラーメッセージ `L6822E` を出力します。

同義語の動作を再作成するには、ARM 定義と Thumb 定義の両方の名前を変更し、リンクし直します。未定義のシンボルごとに、ARM 同義語または Thumb 同義語で指し示す必要があります。

関連参照

8.1 RVCT v4.0 と ARM コンパイラ v4.1 の間での全般的な変更点 (8-37 ページ).

関連情報

--use_sysv_default_script、--no_use_sysv_default_script リンカオプション.
--sysv リンカオプション.

8.4 RVCT v4.0 と ARM コンパイラ v4.1 の間でのアセンブラの変更点

ARM コンパイラツールチェーン v4.1 ではさまざまな変更が armasm に加えられました。

アセンブラについては、以下の点が変更されています。

アセンブラによるファイルの読み出し方法と処理方法の変更点

以前のアセンブラでは、ソースファイルのアセンブルがアセンブラの 2 つのパス間でコードの新しいラインを導入することがありました。以下の例では、シンボル num は第 2 パスで定義されます。その理由は、シンボル foo が :DEF: foo の評価時に第 1 パスで定義されていないためです。

```
        AREA x,CODE         [ :DEF:foo num     EQU 42         ] foo     DCD
num         END
```

アセンブラによるファイルの読み出し方法と処理方法が変更され、より厳密になりました。このようなコードは、どちらのパスでもファイル内の経路が同じになるように書き直す必要があります。アセンブラの第 2 パスへの新しいラインの導入はエラーと見なされ、その結果以下のエラーが生じます。

```
エラー:A1903E :最初のパスに行がありません。アセンブルできません
```

以下のコードに、アセンブラのエラーがどこにあるかを示すの別のサンプルを示します。

```
       AREA FOO, CODE        IF :DEF:VAR        ELSE VAR    EQU 0
ENDIF        END
```

このエラーを回避するには、このコードを以下のように記述し直す必要があります。

```
       AREA FOO, CODE        IF :LNOT::DEF:VAR VAR    EQU 0
ENDIF        END
```

新しいラインがアセンブラの第 2 パスに見られないので、これは正しく機能します。その代わりに、第 1 パスにあったラインが第 2 パスで無視されます。

アセンブラによって出力されるメッセージの変更点

一般に、ソース行での位置に言及するメッセージでは、以下のように、ソース行内の問題個所を示すキャレット文字が表示されるようになりました。

```
"foo.s"、行 3(列 19):警告:A1865W:'#' が定数式の前にありません      3
00000000         ADD r0,r1,1                                        ^
```

診断メッセージの変更点

ARM 命令セットでは、Thumb-2 テクノロジーの導入に伴い、SP (r13) を使用するさまざまな命令が廃止予定になりました。それらの命令は、Thumb-2 テクノロジーをサポートする CPU 用にアセンブルしない限り、廃止予定として診断されなくなりました。以前の CPU に関する警告を有効にするには、オプション `--diag_warning=1745,1786,1788,1789,1892` を使用します。この変更は、RVCT v4.0 の 5 (ビルド 697) パッチで導入されました。

廃止されたコマンドラインオプション

`-O` コマンドラインオプションは廃止されました。代わりに `-o` を使用して下さい。

関連参照

8.1 RVCT v4.0 とARM コンパイラv4.1 の間での全般的な変更点 (8-37 ページ).

関連情報

--diag_warning=tag{, tag} アセンブラオプション.
-o filename アセンブラオプション.
アセンブラの操作方法.
第 2 パスアセンブラ診断.

8.5 RVCT v4.0 と ARM コンパイラ v4.1 の間での C および C++ ライブラリの変更点

ARM コンパイラツールチェーン v4.1 ではさまざまな変更が ARM C および C++ ライブラリに加えられました。

ライブラリは、Thumb-2 テクノロジーをサポートしているターゲットでは、より多くの 32 ビット エンコード Thumb コードを使用するようになりました。これによって、パフォーマンスは維持しつつ、コードサイズは減少することが期待されます。必要であれば、リンカオプション `--no_thumb2_library` を使って、旧式のライブラリに戻すことができます。

特殊なケースでの数学関数の戻り値が、POSIX/C99 要件に準拠するようになりました。以前の動作を有効にするには、次のようにします。

```
#pragma import __use_rvct_matherr
```

RVCT v4.0 09Q4 パッチ以降では、新しい動作を次のようにして有効にできます。

```
#pragma import __use_c99_matherr
```

関連参照

8.1 RVCT v4.0 と ARM コンパイラ v4.1 の間での全般的な変更点 (8-37 ページ).

関連情報

ARM C ライブラリが ISO C 仕様の要件を満たす方法.
--thumb2_library, --no_thumb2_library リンカオプション.

第 9 章
RVCT v3.1 から RVCT v4.0 への移行

RVCT v3.1 と RVCT v4.0 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

9.1 --gnu_version のデフォルトの 303000（GCC 3.3）から 402000（GCC 4.2）への変更

RVCT v4.0 ではデフォルトの GNU バージョンが変更されています。

これは、使用できる GNU 拡張機能に影響します。例えば、`__attribute__((visibility(…)))` や左辺値キャストなどの機能が、非 GNU モードの場合でも影響を受けます。

関連情報

--gnu_version=version コンパイラオプション.

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点

RVCT v4.0 ではさまざまな一般的な変更が加えられています。

以下の変更点は、複数のツールに影響します。

--fpu の制限

CPU アーキテクチャが ARMv5TE 以降の場合は、`--fpu=VFPv2` または `--fpu=VFPv3` のみを使用できます。この制限は、`--fpu` を使用するすべてのツールに適用されます。

——— 注 ———

アセンブラは、`--unsafe` オプションが指定された場合に VFP 命令をアセンブルします。そのため、`--unsafe` を指定している場合は `--fpu` を使用しないで下さい。`--fpu` と `--unsafe` を同時に使用した場合、アセンブラは報告されたアーキテクチャエラーを警告に降格します。

———

v5TExP とその派生アーキテクチャ、および T が含まれないすべての ARMv5 アーキテクチャのサポートの廃止

以下の `--cpu` の選択肢は廃止されました。

- 5.
- 5E
- 5ExP
- 5EJ
- 5EWMMX2
- 5EWMMX
- 5TEx
- ARM9E-S-rev0
- ARM946E-S-rev0
- ARM966E-S-rev0

RVCT v4.0 と従来のオブジェクトおよびライブラリとの互換性

`--apcs /adsabi` を使用してコードをビルドせず、RVCT v4.0 リンカおよび C/C++ ライブラリを使用している場合、RVCT v2.x、v3.x、および v4.0 のオブジェクトとライブラリコードの下位互換性がサポートされます。ただし、上位互換性は保証されていません。

以上のような制限事項があるため、ユーザやサードパーティによって提供されるライブラリを含むプロジェクト全体を RVCT v4.0 以降を使用して再ビルドすることをお勧めします。これは、潜在的な非互換性の問題を回避し、RVCT v4.0 以降のバージョンが提供する向上した最適化機能、拡張機能、および新機能を十分に活用するためです。

関連情報

--apcs=qualifier...qualifier コンパイラオプション.
--cpu=name コンパイラオプション.
--fpu=name コンパイラオプション.
--cpu=name リンカオプション.
--fpu=name リンカオプション.
--apcs=qualifier...qualifier アセンブラオプション.
--cpu=name アセンブラオプション.
--fpu=name アセンブラオプション.
--unsafe アセンブラオプション.

9.3 RVCT v3.1 と RVCT v4.0 の間でのシンボルの可視性の変更点

RVCT v4.0 ではシンボル可視性が変更されています。

シンボルの可視性については、以下の点が変更されています。

__declspec(dllexport) を表すために使用される ELF 可視性の変更点

デフォルトの --hide_all コンパイラコマンドラインオプションを使う場合、RVCT v3.1 以前では、__declspec(dllexport) を表す ELF 可視性は STV_DEFAULT でした。RVCT v4.0 では、STV_PROTECTED です。STV_PROTECTED であるシンボルは、他の DLL から参照できますが、ロード時にそれらをプリエンプティブにすることはできません。

--no_hide_all コマンドラインオプションを使っている場合、インポートまたはエクスポートされたシンボルの可視性は、RVCT v3.1 の場合と同様に、STV_DEFAULT のままです。

__attribute(visibility(…))

GNU 形式の __attribute(visibility(…)) が追加され、--gnu コンパイラコマンドラインオプションが指定されていない場合でも使用できます。これを使用すると、暗黙的な可視性はすべてオーバーライドされます。例えば、次の場合、可視性は STV_HIDDEN ではなく STV_DEFAULT になります。

```
__declspec(visibilty("default")) int x = 42;
```

RVCT v3.1 でのシンボルの可視性の一覧

次の表は、RVCT v3.1 での可視性の規則の一覧です。

表 9-1 RVCT v3.1 でのシンボルの可視性の一覧

コード	--hide_all(デフォルト)	--no_hide_all	--dllexport_all
extern int x; extern int g(void);	STV_HIDDEN	STV_DEFAULT	STV_HIDDEN
extern int y = 42; extern int f() { return g() + x; }	STV_HIDDEN	STV_DEFAULT	STV_DEFAULT
__declspec(dllimport) extern int imx; __declspec(dllimport) extern int img(void);	STV_DEFAULT	STV_DEFAULT	STV_DEFAULT

表 9-1 RVCT v3.1 でのシンボルの可視性の一覧（続き）

コード	--hide_all（デフォルト）	--no_hide_all	--dllexport_all
`__declspec(dllexport) extern int exy = 42;` `__declspec(dllexport) extern int exf() { return img() + imx; }`	STV_DEFAULT	STV_DEFAULT	STV_DEFAULT

`/* 未定義のエクスポート(異常?)*/`
`__declspec(dllexport) extern int exz;`
`__declspec(dllexport) extern int exh(void);`
	STV_HIDDEN	STV_HIDDEN	STV_HIDDEN

表 9-2 RVCT v3.1 での実行時関数への参照に関するシンボルの可視性の一覧

コード	--no_dllimport_runtime --hide_all（デフォルト）	--no_hide_all	--dllexport_all

`/* 実行時関数への参照`
`例:__aeabi_fmul */`
`float fn(float a, float b) { return a*b; }`
	STV_HIDDEN	STV_DEFAULT	STV_DEFAULT

RVCT v4.0 でのシンボルの可視性の一覧

次の表は、RVCT v4.0 での可視性の規則の一覧です。

表 9-3 RVCT v4.0 でのシンボルの可視性の一覧

コード	--hide_all（デフォルト）	--no_hide_all	--dllexport_all
`extern int x;` `extern int g(void);`	STV_HIDDEN	STV_DEFAULT	STV_HIDDEN
`extern int y = 42;` `extern int f() { return g() + x; }`	STV_HIDDEN	STV_DEFAULT	STV_PROTECTED
`__declspec(dllimport) extern int imx;` `__declspec(dllimport) extern int img(void);`	STV_DEFAULT	STV_DEFAULT	STV_DEFAULT

表 9-3 RVCT v4.0 でのシンボルの可視性の一覧（続き）

コード	--hide_all（デフォルト）	--no_hide_all	--dllexport_all
`__declspec(dllexport) extern int exy = 42;` `__declspec(dllexport) extern int exf() { return img() + imx; }`	STV_PROTECTED	STV_PROTECTED	STV_PROTECTED

`/* 未定義のエクスポート（異常?）*/`
`__declspec(dllexport) extern int exz;`
`__declspec(dllexport) extern int exh(void);`
	STV_PROTECTED	STV_PROTECTED	STV_PROTECTED

表 9-4 RVCT v4.0 での実行時関数への参照に関するシンボルの可視性の一覧

コード	--no_dllimport_runtime --hide_all（デフォルト）	--no_hide_all	--dllexport_all

`/* 実行時関数への参照`
例：`__aeabi_fmul */`
`float fn(float a, float b) { return a*b; }`
	STV_HIDDEN	STV_DEFAULT	STV_DEFAULT

関連参照

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ).

関連情報

--default_definition_visibility=visibility コンパイラオプション.
--dllexport_all, --no_dllexport_all コンパイラオプション.
--dllimport_runtime, --no_dllimport_runtime コンパイラオプション.
--gnu コンパイラオプション.
--hide_all, --no_hide_all コンパイラオプション.

9.4 RVCT v3.1 と RVCT v4.0 の間でのコンパイラの変更点

RVCT v4.0 ではさまざまな変更が **armcc** に加えられました。

コンパイラについては、以下の点が変更されました。

コンパイラ実行可能ファイルの単一化

実行可能ファイル `tcc`、`armcpp`、および `tcpp` は、提供されなくなりました。

Thumb 用にコンパイルするには、`--thumb` コマンドラインオプションを使用します。

C++ 用にコンパイルするには、`--cpp` コマンドラインオプションを使用します。

——— 注 ———

コンパイラは、これまでどおり、ファイルの拡張子が `.cpp` の場合は、自動的に C++ を選択します。

ベクトル化コンパイラ

NEON ベクトル化コンパイラが標準機能として提供されます。個別のアドオンとしては提供されなくなりました。NEON ベクトル化コンパイラを使用するためのライセンスは、ARM 開発ツールの Professional Edition に付属しています。

VAST の変更点

VAST は、2 つのバージョン（VAST 11 for 4.0 Alpha と 4.0 Alpha2 以降）を通じてアップグレードされました。以下の問題を除いて、v3.1 ビルドを変更しなくても、新しい VAST を使用できます。

RVCT v3.1 は、サチュレートする ALU 演算を再結合していました。つまり、次のようなプログラムでは、`--vectorize` と `--no_vectorize` では異なる結果が生成される場合がありました。

```
int g_448464(short *a, short *b, int n) {     int i; short s = 0;     for (i = 0; i
&lt; n; i++)  s = L_mac(s, a[i], b[i]);     return s; }
```

RVCT v4.0 では、この問題のために、パフォーマンスの低下が起きることがあります。

再結合を許可するかどうかを指定するために、`--reassociate_saturation` および `--no_reassociate_saturation` コマンドラインオプションが追加されました。

関連参照

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ).

関連情報

-cpp コンパイラオプション.
--reassociate_saturation, --no_reassociate_saturation コンパイラオプション.
--thumb コンパイラオプション.
--vectorize, --no_vectorize コンパイラオプション.

9.5 RVCT v3.1 と RVCT v4.0 の間でのリンカの変更点

RVCT v4.0 ではさまざまな変更が armlink に加えられました。

リンカについては、以下の点が変更されました。

ヘルパ関数

RVCT v4.0 以前のバージョンでは、ヘルパ関数は、ARM コンパイラに付属する `h_tf.l` などのヘルパライブラリファイルに実装されていました。すべてのヘルパライブラリのファイル名は、`h_` で始まります。RVCT v4.0 では、ヘルパライブラリの必要がなくなりました。その代わりに、ヘルパ関数はコンパイラによって生成され、オブジェクトファイルの一部となります。

スキャッタファイルを使用した ARM ライブラリヘルパ関数の配置

RVCT v3.1 以前では、ヘルパ関数は、ARM コンパイラに付属するライブラリに含まれています。したがって、スキャッタファイル内の `armlib` および `cpplib` を使用して、これらのヘルパ関数のメモリ内での配置をリンカに報告することが可能でした。

RVCT v4.0 以降では、コンパイラによってオブジェクトファイルにヘルパ関数が生成されます。これらの関数は標準 C ライブラリに属さなくなりました。つまり、スキャッタファイル内の `armlib` および `cpplib` は使用できなくなりました。その代わり、ヘルパ関数は、スキャッタファイル内の `*.* (i.__ARM_*)` を使用して配置されます。

ヘルパライブラリを使用したリンク時の警告 L6932W

RVCT v4.0 以降では、以下のリンカの警告が表示される場合があります。

```
警告:L6932W:ライブラリから警告がレポートされています:ヘルパライブラリ h_xx.l の使用は廃止されます
```

RVCT v4.0 以前のバージョンでは、ヘルパライブラリを使用してリンクを作成する理由に、オブジェクトファイルがヘルパ関数 `__ARM_switch8` を参照しているかどうかが挙げられました。これは、例えば、以下のように、`--verbose` オプションを使用してリンカからの詳細な出力を検証することでわかります。

```
Loading member object1.o from lib1.a. reference :strncmp
reference :__ARM_switch8
```

RVCT v4.0 以降のバージョンでは、ヘルパライブラリが必要ではなくなったため、リンカの詳細な出力は以下のようになる場合があります。

```
Loading member object2.o from lib2.a.
… definition:__ARM_common_switch8_thumb
```

この場合、ヘルパ関数 `__ARM_common_switch8_thumb` は、ヘルパライブラリ内ではなく、オブジェクトファイル `object2.o` 内にあります。

RVCT v4.0 を使用している場合にリンカによる警告メッセージ L6932W が表示された場合、RVCT v4.0 ではなく、RVCT v3.1 を使用してビルドされたオブジェクトまたはライブラリを使用してリンクしている可能性があります。

リンカのステアリングファイルとシンボルの可視性

RVCT v3.1 では、シンボルの可視性は、ステアリングファイルまたは `.directive` コマンド `IMPORT` および `EXPORT` によってオーバーライドされました。その場合、リンカによって次のような警告メッセージが生成されました。

警告:L6780W:STV_HIDDEN の可視性は、EXPORT を介してシンボル 'hidden_symbol' から削除されました。

RVCT v4.0 では、ステアリングファイルのメカニズムがシンボルの可視性を尊重するため、STV_HIDDEN シンボルの IMPORT または EXPORT は無視されます。v3.1 の動作を復元するには、--override_visibility コマンドラインオプションを使用します。

リンカ定義のシンボル

多くの場合、領域関連シンボルの動作は v3.1 と同じです。

セクション相対シンボル

実行領域のベースシンボルとリミットシンボルは、セクション相対になりました。$$Length シンボルには該当するセクションがないので、絶対のままです。

これは、リンカ定義シンボルが実行領域内の最も適切なセクションに割り当てられることを意味しています。以下に、この例を示します。

```
ExecRegion ER RO Section 1 ; Image$$ER$$Base and Image$$ER$$RO$$Base, val 0 RO Section 2 ; Image$$ER$$RO$
$Limit, val Limit(RO Section 2) RW Section 1 ; Image$$ER$$RW$$Base, val 0 RW Section 2 ; Image$$ER$$Limit and
Image$$ER$$RW$$Limit,              ; val Limit(RW Section 2) ZI Section 1 ; Image$$ER$$ZI$$Base, val 0 ZI
Section 2 ; Image$$ER$$ZI$$Limit, val Limit(ZI Section 2)
```

いずれの場合も、 …$$Length シンボルの値は、 …$$Limit symbol から …$$Base シンボルを引いた値になります。

実行領域内に適切なセクションがない場合は、シンボルを保持するために、適切な種類でサイズがゼロのセクションをリンカが定義します。

変更の影響

セクション相対シンボルへの変更によって、リンカの実装におけるいくつかの特殊なケースがなくなったため、信頼性が向上しました。また、SysV と BPABI のリンクでのダイナミックな再配置も、容易に実行できるようになりました。

境界整列

…$$Limit シンボルは、4 バイト境界での整列を保証しなくなりました。シンボルが定義されているセクションのリミットが、4 バイト境界で整列されているとは限らないためです。

これは、既存のコードが、整列されているシンボル値に依存している場合に、問題になることがあります。$$Limit または $$Length が整列されている必要がある場合は、シンボル値をユーザが整列させる必要があります。

例えば、次の従来の初期化コードは、Image$$<Region_Name>$$Length がワード境界で整列されていないと失敗します。

```
    LDR R1,|Load$$region_name$$Base|     LDR R0,|Image$$region_name$
$Base|     LDR R4,|Image$$region_name$$Length|     ADD R4, R4, R0
copy_rw_data     LDRNE  R3,[R1],#4     STRNE  R3,[R0],#4     CMP
R0,R4     BNE  copy_rw_data
```

以前のツールチェーンのリリースよりもシステムの初期化が複雑になっているため、独自に初期化コードを記述することはお勧めできません。ARM コンパイラツールチェーンで提供されている __main コードを使用することを推奨します。

遅延再配置

リンカに、RW 圧縮後の追加のアドレス割り当てと再配置パスが導入されました。これにより、ロードアドレスに関する、より多くの情報をリンカ定義シンボルで使用できるようになりました。

次の点に注意して下さい。

- `Load$$`*`region_name`*`$$Base` は、C ライブラリの初期化前の *`region_name`* のアドレスです。
- `Load$$`*`region_name`*`$$Limit` は、C ライブラリの初期化前の *`region_name`* のアドレスです。
- `Image$$`*`region_name`*`$$Base` は、C ライブラリの初期化後の *`region_name`* のアドレスです。
- `Image$$`*`region_name`*`$$Limit` は、C ライブラリの初期化後の *`region_name`* のリミットです。

ロード領域シンボルには、以下の特性があります。

- セクション相対シンボルは実行アドレスのみを持つことができるため、これらは絶対シンボルです。
- RW 圧縮が考慮されます。
- ZI は C ライブラリの初期化前には存在しないため、ZI は含まれません。

リンカでは、`Load$$`*`region_name`*`$$Base` の他に、以下のリンカ定義シンボルがサポートされるようになりました。

```
Load$$region_name$$Limit Load$$region_name$$Length Load$$region_name$
$RO$$Base Load$$region_name$$RO$$Limit Load$$region_name$$RO$$Length
Load$$region_name$$RW$$Base Load$$region_name$$RW$$Limit Load$
$region_name$$RW$$Length
```

遅延再配置の制限

RW 圧縮実行領域からのすべての再配置は圧縮前に実行する必要があります。これは、リンカが圧縮データでの遅延再配置を解決できないためです。

リンカが、RW 圧縮領域 `REGION` から、RW 圧縮に依存するリンカ定義シンボルへの再配置を検出した場合、`REGION` の圧縮は無効になります。

ロード領域シンボル

RVCT v4.0 では、ロード領域でのリンカ定義シンボルを使用できるようになりました。それらは実行領域での `Load$$` シンボルと同じ原則に従います。ロード領域は多くの実行領域を含む場合があるので、常に `$$RO` および `$$RW` コンポーネントを定義できるとは限りません。そのため、ロード領域シンボルは領域を全体として示すだけです。

```
; <Load Region Name> Load$$LR$$Load_Region_Name$$Base のベースアドレス
; ロード領域の内容の最後のバイトのロードアドレス。Load$$LR$$Load_Region_Name$$Limit
; リミット - ベース Load$$LR$$Load_Region_Name$$Length
```

イメージ関連シンボル

RVCT v4.0 のリンカは、これらを実行領域関連シンボルと同じように実装しています。

これらは、スキャッタファイルが使用されていない場合にのみ定義されます。そのため、これらは `--sysv` および `--bpabi` リンクモデルで使用できます。

```
Image$$RO$$Base  ; Image$$ER_RO$$Base Image$$RO$$Limit と等価; Image$$ER_RO$
$Limit Image$$RW$$Base と等価; Image$$ER_RW$$Base Image$$RW$$Limit と等価; Image$
$ER_RW$$Limit Image$$ZI$$Base と等価; Image$$ER_ZI$$Base Image$$ZI$$Limit と等
価; Image$$ER_ZI$$Limit と等価
```

ZEROPAD とのインタラクション

`ZEROPAD` キーワード付きの実行領域は、すべての ZI データを次のようにファイルに書き出します。

- `Image$$` シンボルは初期化後に実行アドレスを定義します。

  この場合は、ゼロバイトがファイルにあっても生成されても問題ありません。そのため、`Image$$` シンボルでは、`ZEROPAD` はリンカ定義シンボルの値に影響しません。
- `Load$$` シンボルは初期化前にロードアドレスを定義します。

  この場合は、ファイルに書き込まれるゼロバイトがすべて認識されます。そのため、`Limit` および `Length` は、ファイルに書き込まれるゼロバイトを考慮します。

ビルド属性

RVCT v4.0 リンカでは、ABI ビルド属性セクションの読み出しと書き込みが完全にサポートされています。リンカは `wchar_t` や `enum` サイズなどのプロパティを確認できるようになりました。そのため、ビルド属性に矛盾がある古いオブジェクトのエラーをリンカが診断する場合があります。ビルド属性関連の多くのメッセージは、**`armlink`** を続行できるようにするために降格できます。

`--cpu` オプションを使用すると、FPU 属性を調べて、選択された CPU に組み込み FPU があるかどうかを確認するようになりました。例えば、`--cpu=cortex-a8` は `--fpu=vfpv3` を意味します。RVCT v3.1 では、`--cpu` オプションが使用された場合、選択された CPU のビルド属性だけが確認されました。

エラーメッセージ `L6463U`: 入力オブジェクトに `<archtype>` 命令が含まれていますが、オブジェクト属性に基づく `<archtype>` アーキテクチャの有効なターゲットが見つかりません。`--cpu` オプションを使用して特定の `CPU` を選択することを推奨します。は、次のいずれかの場合に生成されます。

- ELF ファイルには、アーキテクチャ *`archtype`* の命令が含まれているが、ビルド属性は *`archtype`* がサポートされていないことを示している。
- リンカが既存の CPU にマップできない程度の矛盾がビルド属性にある。

`--cpu` オプションを設定しても失敗する場合は、オプション `--force_explicit_attr` がビルド属性を使ってリンカの CPU マッピングを再試行します。このビルド属性は、`--cpu=`*`archtype`* から構成されます。エラーの原因がビルド属性の矛盾だけの場合は、この方法が有効です。

C ライブラリの初期化

C ライブラリ初期化コードの処理に関するリンカへの変更によって、リンカのマップファイル内に、特別な名前付きセクションが生成されるようになりました。このファイルは、`--map` コマンドラインオプションによって作成されます。これらの特別な名前付きセクションは、無視してかまいません。

ARM Linux

共有オブジェクトをビルドするときに、リンカは可視性が STV_DEFAULT の未定義の参照を自動的にインポートします。これは、GCC の動作と一致します。これによって、現在は成功しているリンクが失敗する場合があります。

事前リンクのサポートによって、余分なスペースが予約されるため、イメージや共有オブジェクトのサイズが多少大きくなります。事前リンクのサポートは、`--no_prelink_support` を使ってオフにできます。

シンボル可視性に関して、多数の小規模な変更が行われました。それらについては、シンボル可視性の変更点で説明しています。

RW 圧縮

一部のエラー処理コードは、RW 圧縮の情報を使用できるようにするために、後から実行されます。このため、ほとんどすべての場合に、より多くのカスタマプログラムをリンクできます。RVCT v4.0 で、より多くの RW 圧縮エラーを診断できるようにするために特例がなくなったケースが 1 つあります。

RW 圧縮された複数のインプレース実行領域は、特例ではなくなりました。次のように記述することができました。

```
    LR1 0x0     {         ER1 +0 { file1.o(+RW) }         ER2 +0 { file2.o(+RW) }     }
```

v4.0 ではこのような記述はできなくなり、ER1 が ER2 上に伸張されるというエラーメッセージがリンカによって生成されます。この変更は、リンカが次のような場合を診断できるようにするために行われました。

```
    LR1 0x0     {         ER1 +0 { file1.o(+RW) }         ER2 +0 { file2.o(+RO) }   ;
NOTE RO not RW     }
```

これは、RVCT v3.1 では実行時に失敗します。

関連参照

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ).

関連情報

RW データ圧縮を使用した最適化.
リンカ定義シンボルへのアクセス.
領域関連シンボル.
Image$$ 実行領域シンボル.
Load$$ 実行領域シンボル.
C および C++ でのリンカ定義シンボルのインポート.
セクション関連シンボル.
ARM ライブラリヘルパ関数の配置例.
実行環境の初期化とアプリケーションの実行.
--bpabi リンカオプション.
--cpu=name リンカオプション.
--force_explicit_attr リンカオプション.
--fpu=name リンカオプション.
--map, --no_map リンカオプション.
--override_visibility リンカオプション.

--prelink_support, --no_prelink_support リンカオプション.
--sysv リンカオプション.
--verbose リンカオプション.
EXPORT.
IMPORT.
実行領域の属性.

9.6 RVCT v3.1 と RVCT v4.0 の間でのアセンブラの変更点

RVCT v4.0 ではさまざまな変更が armasm に加えられました。

アセンブラについては、以下の点が変更されています。

- `-O` コマンドラインオプションが廃止される予定です。`-O` は、名前付きファイルに出力する `-o` と同じ意味です。これは、ユーザが同じ名前の **armcc** オプションと混同することを避けるために、廃止されることになりました。
- `-D` コマンドラインオプションが廃止されました。代わりに `--depend` を使用して下さい。
- `LDM r0!,{r0-r4}` がライトバックを無視しなくなりました。以前の Thumb では、`LDM r0!, {r0-r4}` はエンコードされた 16 ビット `LDM` 命令にアセンブルされて警告が生成され、ライトバックは行われませんでした。構文によってライトバックが要求されますが、このエンコーディングは 32 ビット エンコード Thumb 命令がサポートされている場合でしか使用できないので、エラーが生成されます。16 ビット エンコード Thumb 命令が作成されるようにするには、ライトバックを削除する必要があります。
- 論理演算子 `|` は、ソース内の変数の置き換えにおいて問題を発生させる可能性があるため、廃止されました。アセンブラは、論理演算子として使用されている 1 つ目の `|` を検出したときに警告を出力します。代わりに、`:OR:` 演算子を使用して下さい。

関連参照

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ).

関連情報

加算、減算、および論理演算子 (armasm).
--depend=dependfile アセンブラオプション.
-o filename アセンブラオプション.

9.7 RVCT v3.1 と RVCT v4.0 の間での fromelf の変更点

RVCT v4.0 ではさまざまな変更が fromelf に加えられました。

`--text` オプションのパラメータとして単一文字を使用する方法は、区切り文字を `/` にする場合も `=` にする場合も廃止されました。構文 `--text/cd` または `--text=cd` は、使用できなくなりました。`-cd` を指定する必要があります。

関連参照

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ).

関連情報

--text fromelf オプション.

9.8 RVCT v3.1 と RVCT v4.0 の間での C および C++ ライブラリの変更点

RVCT 4.0 ではさまざまな変更が ARM C および C++ ライブラリに加えられました。

ライブラリについては、以下の点が変更されました。

非標準 C ライブラリ数学関数のサポート

非標準 C ライブラリ数学関数は `math.h` で提供されなくなりました。ライブラリ自体では、まだ提供されています。必要であれば、ARM にヘッダファイルを要求できます。購入元にお問い合わせ下さい。

`__ENABLE_LEGACY_MATHLIB` の削除

RVCT v2.2 では、C99 に準拠するために、一部の mathlib 関数の動作を変更しました。古い非 C99 の動作に依存している場合は、コンパイル時に以下の定義を行って、動作を元に戻すことができます。

```
#define __ENABLE_LEGACY_MATHLIB
```

RVCT v4.0 では、この対処ができなくなりました。

関連参照

9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ).

第 10 章
RVCT v3.0 から RVCT v3.1 への移行

RVCT v3.0 と RVCT v3.1 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

10.1 RVCT v3.0 と RVCT v3.1 の間での全般的な変更点

RVCT v3.1 ではさまざまな一般的な変更が加えられています。

以下の変更点は、複数のツールに影響します。

- 古い ABI (`--apcs=/adsabi`) はサポートされなくなりました。
- `-O3` は、`--fpmode=fast` を含みません。

RVCT v3.1 と従来のオブジェクトおよびライブラリとの互換性

RVCT v3.1 では、オプション `--apcs /adsabi` が使用できなくなりました。ADS 互換オブジェクトのコンパイルおよび古い ADS オブジェクトとライブラリのリンクも実行できなくなりました。

RVCT v3.1 リンカおよび C/C++ ライブラリを使用する場合、RVCT 2.x オブジェクトおよびライブラリコードの下位互換性がサポートされます。ただし、上位互換性は保証されていません。

以上のような制限事項があるため、ユーザやサードパーティによって提供されるライブラリを含むプロジェクト全体を RVCT v3.1 を使用して再ビルドすることをお勧めします。これは、潜在的な非互換性の問題を回避し、RVCT v3.1 が提供する向上した最適化機能、拡張機能、および新機能を十分に活用するためです。

関連情報

--apcs=qualifier...qualifier コンパイラオプション.
--dwarf2 コンパイラオプション.
--dwarf3 コンパイラオプション.
--fpmode=model コンパイラオプション.
-g コンパイラオプション.
-Onum コンパイラオプション.

10.2 RVCT v3.0 と RVCT v3.1 の間でのアセンブラの変更点

RVCT v3.1 ではさまざまな変更が armasm に加えられました。

逆アセンブリの出力が、新しい*統一アセンブリ言語*(UAL)形式に準拠するようになりました。VFP ニーモニックが変更され、`FMULS` が `VMUL.F32` になりました。

関連参照

10.1 RVCT v3.0 とRVCT v3.1 の間での全般的な変更点 (10-60 ページ).

関連情報

統一アセンブラ言語.
RVCTv2.1 以降のアセンブリ言語に関する変更点.
VFP ディレクティブとベクタ表記.

10.3 RVCT v3.0 と RVCT v3.1 の間でのリンカの変更点

RVCT v3.1 ではさまざまな変更が armlink に加えられました。

スキャッタファイルで、ルート ZI のロードアドレスの特例処理がなくなりました。例えば、以下のような場合です。

```
LR1 0x8000 {     ER_RO +0     {         *(+RO)     }     ER_RW +0
{         *(+RW)     }     ER_ZI +0     {         *(+ZI)     } } LR2 +0 {
    … }
```

RVCT v3.0 までのバージョンでは、リンカが `LR2` のベースアドレスを計算するときに、`ER_ZI` のサイズを含めていました。イメージの初期化で `ER_ZI` は `LR2` の先頭に上書きされるため、問題ないとされていました。

v3.1 以降では、次のように `ImageLimit()` 組み込み関数を使って、同等のスキャッタファイルを記述できるため、この特例がなくなりました。

```
LR1 0x8000 {     ER_RO +0     {          *(+RO)     }     ER_RW +0
{         *(+RW)     }     ER_ZI +0     {         *(+ZI)     } } LR2 ImageLimit(LR1) {
    … }
```

関連参照

10.1 RVCT v3.0 と RVCT v3.1 の間での全般的な変更点 (10-60 ページ).

関連情報

スキャッタファイルで使用する実行アドレスの組み込み関数.

第 11 章
RVCT v2.2 から RVCT v3.0 への移行

RVCT v2.2 と RVCT v3.0 の間での移行と互換性に影響する変更点について説明します。

このドキュメントは、次で構成されています。

11.1 RVCT v2.2 と RVCT v3.0 の間での全般的な変更点

RVCT v3.0 ではさまざまな一般的な変更が加えられています。

以下の変更点は、複数のツールに影響します。

- DWARF3 がデフォルトです。
- RVCT v2.1 以降で、`-g` は `-O0` を意味しなくなりました。最適化レベルを特定せずに `-g` を指定した場合、以下の警告が生成されます。

  警告:C2083W:最適化レベルが指定されない場合、`-g` はデフォルトで `-O2` になります

従来の RVCT v2.x オブジェクトやライブラリとの互換性

RVCT v3.0 リンカおよび C/C++ ライブラリを使用する場合、RVCT v2.x オブジェクトおよびライブラリコードの下位互換性がサポートされます。ただし、上位互換性は保証されていません。

リンクには、古い ARM ツールのリンカではなく、RVCT v3.0 リンカを使用する必要があります。これは、古いリンカは RVCT v3.0 コンパイラによって生成されたオブジェクトを処理できないためです。

これらの制限を踏まえて、ARM では、ユーザ指定のライブラリを含むプロジェクト全体を RVCT v3.0 で再ビルドすることを強くお勧めします。これは、潜在的な互換性の問題を回避し、RVCT v3.0 によって提供される向上した最適化機能、拡張機能、および新機能を十分に活用することを目的としています。

関連情報

--apcs=qualifier...qualifier コンパイラオプション.
--dwarf2 コンパイラオプション.
--dwarf3 コンパイラオプション.
--fpmode=model コンパイラオプション.
-g コンパイラオプション.
-Onum コンパイラオプション.

11.2 RVCT v2.2 と RVCT v3.0 の間でのコンパイラの変更点

RVCT v3.0 ではさまざまな変更が armcc に加えられました。

C++ コアの問題 #446 を解決するために、条件付きクラス rvalue 式の一部のケースでテンポラリが作成されるようになりました（以前は作成されませんでした）。

RVCT v4.0 10Q1（ビルド 771）以降では、この状況が発生した場合に `--diag_warning=2817` コマンドラインオプションを使用して警告を発行できるようになりました。

関連参照

11.1 RVCT v2.2 と RVCT v3.0 の間での全般的な変更点 (11-64 ページ).

関連情報

--diag_warning=optimizations コンパイラオプション.

11.3 RVCT v2.2 と RVCT v3.0 の間でのリンカの変更点

RVCT v3.0 ではさまざまな変更が armlink に加えられました。

リンカには、以下の変更が加えられました。

- コンパイラオプション `--fpu=softvfp` と暗黙的な VFP ハードウェアを備えた CPU を指定した場合、リンカは VFP 命令を使用するソフトウェア浮動小数点呼び出しを実装したライブラリを選択できません。この以前の動作が必要な場合は、コンパイラオプション `--fpu=softvfp+vfp` を指定します。
- **armlink** は、ロード領域の内容を出力 ELF ファイルに、ロード領域がスキャッタファイルで記述されている順序で書き込みます。各ロード領域は、1 つの ELF プログラムセグメントで表されます。RVCT v2.2 では、プログラムセグメントを記述するプログラムヘッダテーブルエントリが、ELF ファイル内のプログラムセグメントと同じ順序になります。ELF 仕様への準拠を厳密にするために、RVCT v3.0 以降ではプログラムヘッダテーブルエントリが仮想アドレスでの昇順でソートされます。
- プログラムヘッダテーブルエントリのソートをオフに切り替えるために、`--no_strict_ph` コマンドラインオプションが追加されました。

関連参照

11.1 RVCT v2.2 とRVCT v3.0 の間での全般的な変更点 (11-64 ページ).

関連情報

--strict_ph, --no_strict_ph リンカオプション.
--cpu=name コンパイラオプション.
--fpu=name コンパイラオプション.

11.4 RVCT v2.2 と RVCT v3.0 の間での C および C++ ライブラリの変更点

RVCT 3.0 ではさまざまな変更が ARM C および C++ ライブラリに加えられました。

関数 `__user_initial_stackheap()` は、初期スタックおよびヒープの位置を設定し、返します。RVCT v3.x 以降では、`__user_setup_stackheap()` を代わりに使用するようにソースコードを変更することをお勧めします。ARM C ライブラリおよび C++ ライブラリの古いバージョンとの下位互換性のために `__user_initial_stackheap()` は今でもサポートされています。

`__user_initial_stackheap()` の使用を継続する場合、RVCT v3.0 に加えられた以下の変更点に注意する必要があります。

- RVCT v2.x 以前のバージョンでは、`__user_initial_stackheap()` のデフォルトの実装で、シンボル `Image$$ZI$$Limit` の値が使用されました。`--scatter` リンカコマンドラインオプション付きでスキャッタファイルを指定した場合、このシンボルは定義されません。このため、スキャッタファイルを使用する場合は、`__user_initial_stackheap()` を再実装してヒープとスタックの境界を設定する必要があります。再実装しないと、リンク手順が正常に実行されません。

  RVCT v3.x 以降のバージョンでは、`__user_initial_stackheap()` の複数実装が ARM C ライブラリによって提供されています。RVCT は、スキャッタファイルにある情報を使用して適切な実装を自動的に選択します。つまり、スキャッタファイルを使用する場合、この関数を再実装する必要はありません。
- アプリケーションを RVCT v2.2 から RVCT v3.0 に移行する場合、以下のリンカのエラーが表示される場合があります。

  ```
  Error L6218E:Undefined symbol main (referred from kernel.o).
  ```

  このエラーによって、リンカは、アプリケーションに `main()` 関数が含まれていないことをレポートします。このエラーは、RVCT v3.0 による `__user_initial_stackheap()` の複数実装からの選択方法が原因となって生成されます。これらの実装は、`__rt_exit()` 関数を参照します。これは、`kernel.o` に含まれていますが、それは `__rt_lib_init()` 関数も含んでいます。`__rt_lib_init()` 関数が `main()` を呼び出すため、`main()` が存在しない場合に未定義のシンボルエラーが発生します。

  C コードのエントリポイントとして `main()` 関数が指定されていない場合、最も簡単な解決策は、以下のいずれかの実装です。

  — 空のダミー `main()` 関数
  — `__rt_exit()` のダミー実装

  これらのスタブのいずれかが別のソースファイルに含まれている場合は、通常、リンカの未使用セクションの削除処理によってスタブが削除されます。したがって、最終のリンク先イメージにはオーバーヘッドはありません。

関連参照

11.1 RVCT v2.2 と RVCT v3.0 の間での全般的な変更点 (11-64 ページ).

関連情報

未使用セクションの削除.
イメージシンボル.
__user_setup_stackheap() (なし).
古い関数 __user_initial_stackheap() (なし).
--scatter=filename リンカオプション.

付録 A
『移行と互換性』マニュアルに対する改訂

『移行と互換性ガイド』に対して加えられた技術的変更について 説明しています。

このドキュメントは、次で構成されています。

A.1 『移行と互換性ガイド』に対する改訂

『移行と互換性ガイド』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 I と発行 J の相違点

変更点	関連するトピック
v5.03 と v5.04 間での変更点に関する新しい章を追加しました。	*2 ARM コンパイラ v5.03 から v5.04 への移行 (2-14 ページ)*
サポートされる FlexNet、GCC、および Cygwin のバージョンを説明するトピックが拡張されました	• *1.1 サポートされている FlexNet のバージョン (1-11 ページ)* • *1.2 エミュレートされる GCC のバージョン (1-12 ページ)* • *1.3 サポートされている Cygwin のバージョン (1-13 ページ)*
NEON コンパイラライセンスに関する情報が追加されました	*4.1 ARM コンパイラ v5.0 と v5.01 以降の間での全般的な変更点 (4-20 ページ)*

表 A-2 発行 H と発行 I の相違点

変更点	関連するトピック
v5.02 と v5.03 間での変更点に関する新しい章を追加しました。	*3 ARM コンパイラ v5.0.2 から v5.03 への移行 (3-16 ページ)*
必要に応じて、16 ビット Thumb および 32 ビット Thumb が別の命令セットであることを暗示する用語を変更しました。 ——— 注 ——— ただ 1 つの Thumb 命令セットだけが存在します。Thumb-2 はテクノロジであって、別個の命令セットではありません。 ———	各トピック
サポートされる Cygwin のバージョンを説明するトピックが拡張されました	*1.3 サポートされている Cygwin のバージョン (1-13 ページ)*

表 A-3 発行 G と発行 H の相違点

変更点	関連するトピック
ARM コンパイラツールチェーンで使用されている FLEXnet のバージョンが更新されました。	*1.1 サポートされている FlexNet のバージョン (1-11 ページ)*
ARM コンパイラ v5.01 にエミュレートされる GCC バージョンが修正されました。	*1.2 エミュレートされる GCC のバージョン (1-12 ページ)*
移行と互換性に影響しない変更が削除されました。	*7.2 ARM コンパイラ v4.1 と v4.1 ビルド 561 の間でのリンカの変更点 (7-31 ページ)*

表 A-4 発行 F と発行 G の相違点

変更点	関連するトピック
FLEX*net* のバージョン表が更新されました。	*1.1 サポートされている FlexNet のバージョン (1-11 ページ)*

表 A-5 発行 D と発行 F の相違点

変更点	関連するトピック
v5.0.1 と v5.01 間での変更点に関する新しい章を追加しました。	*4 ARM コンパイラv5.0 から v5.01 以降への移行 (4-19 ページ)*
FLEX*net* のサポートされるバージョンが更新されました。	*1.1 サポートされている FlexNet のバージョン (1-11 ページ)*
GCC の更新されたバージョンがサポートされるようになりました。	*1.2 エミュレートされる GCC のバージョン (1-12 ページ)*
従来のオブジェクトおよびライブラリコードとの下位互換性に関する詳細を追加しました。	• *8.1 RVCT v4.0 と ARM コンパイラv4.1 の間での全般的な変更点 (8-37 ページ)* • *9.2 RVCT v3.1 と RVCT v4.0 の間での全般的な変更点 (9-45 ページ)* • *10.1 RVCT v3.0 と RVCT v3.1 の間での全般的な変更点 (10-60 ページ)* • *11.1 RVCT v2.2 と RVCT v3.0 の間での全般的な変更点 (11-64 ページ).*
適切な場合、 • Thumb-2 を 32 ビット Thumb に変更しました。	• *8.5 RVCT v4.0 と ARM コンパイラv4.1 の間での C および C++ ライブラリの変更点 (8-42 ページ)* • *9.6 RVCT v3.1 と RVCT v4.0 の間でのアセンブラの変更点 (9-56 ページ)*
C および C++ ライブラリの RVCT v2.2 と RVCT v3.0 での相違に関する説明を変更しました。	*11.4 RVCT v2.2 と RVCT v3.0 の間での C および C++ ライブラリの変更点 (11-67 ページ)*
ヘルパライブラリを使用したリンク作成時に発生するリンカの警告 L6932W の説明を追加しました。	*9.5 RVCT v3.1 と RVCT v4.0 の間でのリンカの変更点 (9-50 ページ)*

表 A-6 発行 C と発行 D の相違点

変更点	関連するトピック
サポートされている Cygwin のバージョンに関するトピックを追加しました。	*1.3 サポートされている Cygwin のバージョン (1-13 ページ)*
v4.1 パッチ 3 から v5.0 への移行に関する章を追加しました。	*5 ARM コンパイラv4.1 パッチ 3 以降から v5.0 への移行 (5-22 ページ)*
コンパイラオプション `--fpu=softvfp` と暗黙的な VFP ハードウェアを備えた CPU を指定した場合の **armlink** によるライブラリの選択方法の詳細を追加しました。	*11.3 RVCT v2.2 と RVCT v3.0 の間でのリンカの変更点 (11-66 ページ)*

表 A-7 発行 B と発行 C の相違点

変更点	関連するトピック
v4.1 ビルド 561 から v4.1 パッチ 3 への移行に関するトピックを追加しました。	*6.1 ARM コンパイラv4.1 ビルド 561 とv4.1 パッチ 3 以降の間での C および C++ ライブラリの変更点 (6-28 ページ)*
スキャッタファイルを使用して ARM ライブラリヘルパ関数を配置する方法についての詳細を追加しました。	*9.5 RVCT v3.1 とRVCT v4.0 の間でのリンカの変更点 (9-50 ページ)*
第 2 パスアセンブラに加えられた変更内容を示すサンプルをさらに追加しました。	*8.4 RVCT v4.0 とARM コンパイラv4.1 の間でのアセンブラの変更点 (8-40 ページ)*

表 A-8 発行 A と発行 B の相違点

変更点	関連するトピック
ARM コンパイラ v4.1 から v4.1 ビルド 561 への移行に関する詳細説明を追加しました。	*7 ARM コンパイラv4.1 から v4.1 ビルド 561 への移行 (7-29 ページ)*
v2.2 から v3.0 以降への移行に関連して `__user_initial_stackheap()` および `__user_setup_stackheap()` に関する詳細説明を追加しました。	*11.4 RVCT v2.2 とRVCT v3.0 の間での C および C++ ライブラリの変更点 (11-67 ページ)*
ライブラリのハードウェア浮動小数点バージョンでの *softfp* リンケージ関数の追加に関する詳細説明を追加しました。	*7.4 ARM コンパイラv4.1 とv4.1 ビルド 561 の間での C および C++ ライブラリの変更点 (7-33 ページ)*
| 論理演算子の廃止に関する詳細説明を追加しました。	*9.6 RVCT v3.1 とRVCT v4.0 の間でのアセンブラの変更点 (9-56 ページ)*