

ARM® IAR Embedded Workbench
マイグレーションガイド
(V4.xx V5.xx)

バージョン 4.x からバージョン 5.x への移行

本ガイドは、ARM IAR Embedded Workbench®バージョン4.xとARM IAR Embedded Workbenchバージョン5.xの主な違いと、移行についての注意点について、主に以下の内容を説明します。

- 既存アプリケーションソースコードのコンパイルとリンクの方法
- ランタイムの動作の変更点

さらに、アプリケーションソースコードおよびその他のプロジェクトファイルを新バージョン5.xに移植する方法について説明します。

ARM IAR Embedded Workbenchバージョン3.xから移行する場合は、まず英語版「ARM® IAR Embedded Workbench Migration Guide」の「Migrating to ARM® IAR Embedded Workbench version 4.x」を参照してください。

移行手順

バージョン4.xからバージョン5.xへの主な変更点は、IARビルドツールが使用する内部オブジェクトフォーマットが変更されたという点です。バージョン4.xではIARシステムズフォーマットのUBROFが使用されていますが、バージョン5.xでは業界標準フォーマットのELF(Executable and Linking Format)がデバッグ情報のDWARFと一緒に使用されています(ELF/DWARF)。これによりリンカが、IAR XLINKリンカから全く新しいリンカ(IAR ILINKリンカ)に変更されました。

オブジェクトフォーマットも、ELF/DWARFをサポートする他のベンダのツールと互換性を持つように変更されました。

この変更により、既存のアプリケーションソースコードおよびその他の関連プロジェクトファイルの変更が必要になります。つまり、バージョン4.xから5.xに移行するには、以下の項目の変更点について考慮する必要があります。

- コンパイラとCソースコード
- アセンブラとアセンブラソースコード
- リンカとリンカ設定
- ランタイムライブラリとオブジェクトファイル
- プロジェクトファイルとIAR Embedded Workbench IDEのプロジェクト設定
- デバッグ

古いプロジェクトを移行するには、説明されている移行手順に従います。移行手順のステップのすべてがプロジェクトの移行に必要なとは限らないため、実際の移行に必要な処理は何かを、注意深く検討する必要があります。

注:バージョン5.xでは、以下の点に注意してください。

- ライブラリ管理用IARシステムズアプリケーションのXARとXLIBは、IAR Archive Builder(iarchive)および一連のGNUユーティリティに置き換えられ

ました。これらのユーティリティは、IAR製品のインストール時に提供されます。

- ユーザドキュメントの構造が変更されました。以前は、コンパイラおよびリンカはそれぞれ別のガイドに記述されていましたが、現在は、両方とも『ARM® IAR C/C++ Development Guide』に記述されています。

コンパイラとCソースコード

ARM IAR C/C++コンパイラバージョン4.x用に書かれたCまたはC++のソースコードは、新しいARM IAR C/C++コンパイラバージョン5.xでも使用できます。しかし、新しいコンパイラを使用して既存のソースコードをコンパイルする前に、以下の点について確認する必要があります。

1. 以下の変更点について、使用するC/C++ソースコードファイルを確認してください。

- 絶対アドレスに配置されたconst変数の初期化ができなくなりました。したがって、以下のような構文は使用できません。

```
int const a @ 10 = 20;
```

- #pragma vector ディレクティブが無効になりました。また、割込みベクタは、あらかじめ名前が定義されています。たとえば、IRQ_Handler はcstartup.oに定義されています。

- #pragma swi_number ディレクティブおよび__swi キーワードは、関数宣言でのみ使用可能です。関数定義には使用できません。

- 組み込み関数__enable_interrupt および__disable_interrupt は、組み込み関数としては使用できなくなりましたが、ソースコードレベルでの下位互換性を維持するため、ライブラリ関数としては使用できます。

- __monitor キーワードは無効になりました。このキーワードは、以下のコードシーケンスで置き換えることができます。

```
void function_that_was_declared_monitor_in_ewarm_4xx()
{
    __istate_t istate = __get_interrupt_state();
    __disable_interrupt();
    ...
    __set_interrupt_state(istate);
}
```

- #pragma optimizeディレクティブは、動作が変更されました。バージョン5.xでは、パラメータのs(速度)、z(サイズ)、2(なし)、3(低)、6(中)、9(高)は認識はされますが、無視されます。バージョン5.xでは、これらのパラメータはspeed、size、balancedに置き換えられました。パラメータbalancedは速度とサイズのバランスを取ります。これらのパラメータは、パラメータhighと組み合わせることができます。

- セグメント演算子__segment_size は無効になりました。この演算子は、以下のような構文で置き換えることができます。

```
size = __section_end("xxx") - __section_begin("xxx");
```

- ビットフィールドのデフォルトのレイアウトは、disjoint_types からjoint_typesに変更されました。しかし、移行の際は、外部インタフェースの一部になっているビットフィールドにのみ注意すれば大丈夫です。注

意が必要なビットフィールドの情報については『ARM® IAR C/C++ Development Guide』を参照してください。

リトルエンディアンモードの場合、全てのビットフィールドが同じ基本型をしている構造体では、レイアウトが以前と変わらない点に注意してください。

2. コードおよびデータは、セグメントではなくセクションに配置されます。ソースコード内で定義済セグメント名を明示的に使用していない限り、この内部変更のためにC/C++ソースコードを変更する必要はありません。定義済セグメント名を使用している場合は、新しいセクション名を使用する必要があります。新しいセクション名については、14ページの「セグメントとセクション」を参照してください。

また、初期化セグメントの取扱いも変更されています。15ページの「初期化用セグメント」を参照してください。

3. コンパイラオプションについてもいくつか変更点があります。オプションの削除、変更、新規追加が行われました。変更点のリストについては、11ページの「ツールオプション」を参照してください。
4. ファイル名の拡張子に関連する変更については、16ページの「ファイル名拡張子」を参照してください。
5. I/O定義ヘッダファイルのディレクトリ構成が変更されました。バージョン5.xでは、これらのファイルはデバイス毎のサブフォルダに配置されます。この変更は、検索パスに影響します。したがって、ヘッダファイルを適宜変更する必要があります。以下に例を示します。

```
#include <ioml671000.h>
```

上の例は以下のように変更します。

```
#include <oki/ioml671000.h>
```

バージョン5.xの機能の詳細については、『ARM® IAR C/C++ Development Guide』を参照してください。

アセンブラとアセンブラソースコード

アセンブラ実行ファイル名がaarm からiasmarmに変更されました。

以下の項目について、使用するアセンブラソースコードを確認してください。

1. モジュール

バージョン5.xでは、アセンブラもコンパイラも、プログラムモジュールとライブラリモジュールを区別できません。モジュールをライブラリモジュールとして扱う場合は、ライブラリにモジュールを配置するために条件付きでリンクする必要があります。

したがって、既存のアセンブラソースコードでLIBRARY またはMODULE ディレクティブが使用されている場合、これらのディレクティブは機能しなくなります。

バージョン4.xでは、ファイルごとに1つまたは複数のアセンブラモジュールを定義することができました。バージョン5.xでは、定義できるのはファイル

ごとに1つのモジュールのみです。したがって、ファイルを適宜再構成する必要があります。

モジュール化プログラミングの詳細およびモジュールディレクティブの新しい構文については、『ARM® IAR Assembler Reference Guide』を参照してください。

2. セグメントとセクション

セグメントの概念は、セクションの概念に置き換えられました。これにより、以下の点が変わっています。

- セグメントで機能するアセンブラディレクティブは、削除されたか、セクションで機能する新しいディレクティブに置き換えられています。したがって、アセンブラソースコードを適宜変更する必要があります。セクション制御ディレクティブの詳細については、『ARM® IAR Assembler Reference Guide』を参照してください。
- アセンブラソースコードでバージョン4.x固有の定義済セグメントを使用している場合は、古いセグメント名をすべて新しいセクション名に置き換える必要があります。詳細については、14ページの「セグメントとセクション」を参照してください。

3. 式

バージョン5.xでは、1つの式の中での2つのシンボルを使用するなど、アセンブル時に解決できない複雑な式は使用できません。このような式をすべて書き直さない場合、アセンブルエラーになります。以下にその例を示します。

```
public glob_var
extern ext_var
var1    DC32    glob_var + ext_var    ; will fail
var2    DC32    glob_var * 5 + 3      ; will fail
var3    DC32    glob_var + 3          ; OK
```

4. アセンブラディレクティブ

アセンブラディレクティブの中には、削除されたものや構文が新しくなったもの、およびその他の変更があったものがあります。バージョン4.xと5.xの間で変更されたアセンブラディレクティブのリストについては、15ページの「アセンブラディレクティブ」を参照してください。

アセンブラソースコードでこれらのディレクティブが使用されている場合は、該当する構文を書き直す必要があります。

これらのディレクティブの詳細については、『ARM® IAR Assembler Reference Guide』を参照してください。

5. 定義済シンボル

定義済シンボル `__ASMARM__` は、シンボル `__IASMARM__` に変更されました。

6. 呼出し規約

コンパイラが使用する呼出し規約が変更されました。バージョン4.xでは、スタックのアラインメントのデフォルト値は4ですが、Advanced RISC Machines社のATPCS(Arm/Thumb Procedure Call Standard)に従うと、アラインメントを8に設定する必要があります。バージョン5.xでは、コンパイラはAAPCS(ARM

Architecture Procedure Call Standard)に準拠しているため、スタックは8バイト境界でアラインメントされます。

つまり、Cの関数でアセンブラ関数を呼び出している、またはその逆の場合は、新しい呼出し規約に従ってアセンブラルーチンを書き直す必要があります。

これらのプロシージャ呼出し規約とその違いの詳細については、『ARM@ IAR C/C++ Development Guide』を参照してください。また、Webサイト www.arm.com でも参照できます。

7. C-SPYの[Call Stack]ウィンドウ用バックトレース情報

C-SPYの[Call Stack]ウィンドウ用バックトレース情報のためのコンパイラリソース名が標準化されました。これらの名前は、`CfiCommon.h`に定義されています。したがって、独自のリソース名を定義することはできなくなりました。CFI アセンブラディレクティブを使用して独自のオブジェクト名を定義している場合は、標準化されたリソース名のサブセットを使用する必要があります。標準化されたリソース名のリストについては、『ARM@ IAR C/C++ Development Guide』を参照してください。

8. ファイル名の拡張子に関連する変更については、16ページの「ファイル名拡張子」を参照してください。

9. 環境変数ASMARM およびAARM_INC は、それぞれIASMARM および IASMARM_INCに変更されています。

リンカとリンカ設定

IAR XLINKリンカは、IAR ILINKリンカに変更されました。

XLINKとILINK

XLINKとILINKは両方とも、1つ以上の再配置可能オブジェクトファイルと1つ以上のオブジェクトライブラリの指定部分を結合し、実行可能イメージを生成します。XLINKでは、IARシステムズのツールによって生成されるUBROFフォーマットのオブジェクトファイルしか入力に使用できません。出力は、UBROFまたはXLINKがサポートしているその他の出力フォーマットで行われます。ILINKでは、ELFフォーマットのオブジェクトファイルを入力に使用し、ELFフォーマットの実行可能イメージを生成します。

バージョン4.xでは、コンパイラはコードおよびデータをUBROFのセグメントに配置します。このセグメントは、XLINKにより、リンカコマンドファイルで指定されているディレクティブに従って、メモリに配置されます。リンカコマンドファイルは、コマンドラインを拡張したもので、あらゆるXLINKのコマンドラインオプションを指定することができます。バージョン5.xでは、コンパイラはコードおよびデータをELFのセクションに配置します。ILINKは、ILINK設定ファイルで指定されている設定に従って、セクションを配置します。また、ILINK設定ファイルは、アプリケーションの初期化フェーズの自動処理をサポートしています。つまり、グローバル変数エリアやコードエリアも、初期化データのコピー(必要あれば解凍)によって初期化されます。しかし、このファイルにはコマンドラインオプションは一切指定できません。コマンドラインオプションは、コマンドライン上で指定する必要があります。

XLINKからILINKへの移行

1. 新しいIAR ILINKリンカはターゲット固有のリンカで、IAR XLINKリンカから変更されています。そのため、実行ファイル名もxlink からilinkarm に変わっています。
2. リンカコマンドファイルを新しいILINK設定ファイルに移行するには、7ページの「XLINK.XCLからILINK.ICFへの変換」の例を参照してください。
3. セグメントからセクションにマッピングを変更する方法については、14ページの「セグメントとセクション」を参照してください。
4. ファイル名の拡張子に関連する変更については、16ページの「ファイル名拡張子」を参照してください。
5. ELF出力フォーマットをIntel-hexまたはMotorola S-recordsに変換する必要がある場合は、提供されているGNUユーティリティを使用してください。

ILINKの優れた機能の詳細については、『ARM® IAR C/C++ Development Guide』のリンクに関する章を参照してください。

XLINK.XCLからILINK.ICFへの変換

XLINKのリンカコマンドファイルおよびILINKのリンカ設定ファイルは、全く異なる記述方式であるため、リンカコマンドファイルの内容は自動的に変換されません。手動でリンカ設定を変換する必要があります。

IAR Embedded Workbench IDEを使用している場合は、リンカ設定ファイルエディタを使用してリンカ設定をセットアップすることができます。

1. [Project]>[Options]を選択し、[Linker]カテゴリを選択して、[Config]タブをクリックします。
2. リンカ設定ファイルエディタを開くには、[Override default]オプションを選択して、[Edit]ボタンをクリックします。
3. 表示されるダイアログボックスに、以下の項目を定義します。
 - 割込みベクタテーブルの先頭アドレス
 - RAMおよびROMメモリエリアの先頭および最終アドレス
 - スタックおよびヒープのサイズ
4. 入力が終わったら、[Save]ボタンをクリックします。この作業を初めて行う場合は、[Save As]ダイアログボックスが表示されます。

注:すべてのビルド構成に対し、個別にリンカ設定ファイルを指定する必要があります。

コマンドラインからプロジェクトをビルドする場合は、arm¥config ディレクトリにあるリンカ設定ファイルgeneric.icfを使用できます。また、examples ディレクトリにあるサンプルプロジェクト用の設定ファイルも使用できます。これらの設定ファイルをテンプレートとして使用して、ターゲットハードウェアやアプリケーションの要件に合った設定ファイルを作成することができます。

以下は、XLINKリンカコマンドファイル(xcl)とそれに対応するILINKリンカ設定ファイル(icf)の例です。

```
-!=====
-!xlink xcl file
-!=====
-carm

-DROMSTART=08000
-DROMEND=FFFFF

-Z (CODE) INTVEC=00-3F
-Z (CODE) ICODE, DIFUNCT=ROMSTART-ROMEND
-Z (CODE) SWITAB=ROMSTART-ROMEND
-Z (CODE) CODE=ROMSTART-ROMEND
-Z (CONST) CODE_ID=ROMSTART-ROMEND
-Z (CONST) INITTAB, DATA_ID, DATA_C=ROMSTART-ROMEND
-Z (CONST) CHECKSUM=ROMSTART-ROMEND

-DRAMSTART=100000
-DRAMEND=7FFFFFFF

-Z (DATA) DATA_I, DATA_Z, DATA_N=RAMSTART-RAMEND
-Z (DATA) CODE_I=RAMSTART-RAMEND
-QCODE_I=CODE_ID

-D_CSTACK_SIZE=2000
-D_IRQ_STACK_SIZE=100
-D_HEAP_SIZE=8000

-Z (DATA) CSTACK+_CSTACK_SIZE=RAMSTART-RAMEND
-Z (DATA) IRQ_STACK+_IRQ_STACK_SIZE, HEAP+_HEAP_SIZE=RAMSTART-RAMEND
```

以下は対応するicfファイルです。

```
//=====
//ilink icf file
//-carm is not relevant to migrate because ilinkarm is
//ARM-specific.
//=====
define memory mem with size = 4G;

define region ROM_region = mem:[from 0x8000 to 0xFFFFF];
define region RAM_region = mem:[from 0x100000 to 0x7FFFFFF];

initialize by copy { rw };
do not initialize { section .noinit };

define block CSTACK with alignment = 8, size = 0x2000 { };
define block IRQ_STACK with alignment = 8, size = 0x100 { };
define block HEAP with alignment = 8, size = 0x8000 { };

place at address mem:0x0 { ro section .intvec };
place in ROM_region { ro };
place in RAM_region { rw, block CSTACK, block IRQ_STACK,
                    block HEAP };
```

プロジェクトファイルとIAR Embedded Workbench IDEのプロジェクト設定

IAR Embedded Workbench IDEの新しいバージョンにアップグレードするには、手動による調整が必要な場合があります。

プロジェクトファイルの変換

IAR Embedded Workbench IDEを使用している場合、ARM IAR Embedded Workbench IDEの新しいバージョンを起動して古いワークスペースを開きます。バージョン4.xで作成された古いプロジェクトが含まれているワークスペースを開くと、プロジェクトファイルをバージョン5.x用に変換するかどうかを確認するダイアログボックスが表示されます。[OK]をクリックすると、まず古いプロジェクトのバックアップが作成され、その後でプロジェクトが変換されます。

プロジェクトオプションの移行

バージョン4.xとバージョン5.xの間では使用可能なツールオプションが異なるため、古いプロジェクトを変換した後にオプションの設定を確認する必要があります。

コマンドラインインタフェースを使用している場合は、11ページの「ツールオプション」にある対応表とmakefileを比較して、makefileを適宜変更してください。

IAR Embedded Workbench IDEを使用している場合、2つのバージョン間で変更されていないオプションは、プロジェクトの変換時に自動的に変換されます。変更されているオプションは、デフォルト値に設定されます。

手動でオプションを確認するには、以下の説明に従ってください。

- [Compiler]カテゴリ

[Code]ページが新しくなりましたが、使用されているオプションは [General]カテゴリで使用されていたものです。これらのオプションの設定は引き継がれます。

コンパイラオプションの `-s` および `-z` は `-O` オプションに置き換えられたため、[Optimizations]ページも変更されています。詳しくは 12 ページの「`-s`、`-z` と `-O`」を参照してください。

[Output]ページが変更されました。独自のセグメント名を定義している場合、そのセグメント名は自動的にセクション名に変換されません。デフォルトのコードセクション名は `.text` です。セグメント名とセクション名の詳細については、14 ページの「セグメントとセクション」を参照してください。

- [Linker]カテゴリ

自動的に変換されるリンカオプションはありません。プロジェクトの変換時に、リンカオプションはすべてデフォルト値に設定されます。XLINK オプションと ILINK オプションの詳細については、12 ページの「リンカオプションに関連する変更点」、6 ページの「リンカとリンカ設定」を参照してください。

- [Output Converter]カテゴリ

バージョン 4.x の XLINK はさまざまなフォーマットで出力することができますため、リンカの出力に使用するフォーマットを [Output]ページで指定します。一方、バージョン 5.x の ILINK は、ELF/DWARF フォーマットで出力します。ELF 出力を Intel-hex または Motorola S-records に変換するには、[Output Converter]オプションを使用します。

- [Library Builder]カテゴリ

バージョン 5.x では、新しいライブラリビルダが使用されます。したがって、オプションは自動的に変換されません。プロジェクトの変換時に、ライブラリビルダオプションはすべてデフォルト値に設定されます。

新しいオプションを設定することを忘れないようにしてください。

IAR Embedded Workbench IDEのどこで、対応するオプションを設定するかについては、『ARM® IAR Embedded Workbench® IDE User Guide』を参照してください。

ランタイムライブラリとオブジェクトファイル

相互運用性

バージョン 5.x で作成したコードをビルドするには、バージョン 5.x で提供されているランタイムライブラリコンポーネントを使用する必要があります。バージョン 5.x で作成されたオブジェクトコードと、バージョン 4.x で提供されているコンポーネントをリンクすることはできません。したがって、バージョン 4.x のオブジェクトコードはリビルドする必要があります。場合によっては、ソースコードの変更が必要になる可能性があります。

作成するアプリケーションを AEABI (Embedded Application Binary Interface for the ARM architecture) 準拠にすると、サードパーティ製のリンカを使用した他のベンダのオブジェクトファイルを含むアプリケーションをビルドできます。

そのためには、ツールのAEABI 準拠を有効にする必要があります。この方法については、『ARM® IAR C/C++ Development Guide』を参照してください。

ランタイムライブラリファイルの選択

バージョン4.xでは、追加サポートルーチンを含むビルド済ランタイムライブラリファイルとして、C/C++の標準ライブラリが提供されています。これらのランタイムライブラリファイルは、ファイルごとに個別のコンパイラオプションセットを使用してビルドされています。

プロジェクトで使用するコンパイラオプションに従って、プロジェクトオプションに対応するランタイムライブラリファイルをコマンドラインで指定する必要があります。IAR Embedded Workbench IDEバージョン4.xでは、プロジェクト設定に基づいて、適切なライブラリファイルが自動的に使用されます。

バージョン5.xでは、別のグループのランタイムライブラリファイルが提供されます。これらのランタイムライブラリファイルは、グループごとに個別のコンパイラオプションセットを使用してビルドされています。

プロジェクトで使用するコンパイラオプションに従って、適切な標準ライブラリファイルがILINKによって自動的に使用されます。必要であれば、手動で直接コマンドラインやIAR Embedded Workbench IDEバージョン5.xにライブラリファイルを指定することもできます。

バージョン5.xで使用可能なライブラリファイルの詳細については、『ARM® IAR C/C++ Development Guide』を参照してください。

デバッグ

デバイス記述ファイル(ddf ファイル)のディレクトリ構成が変更されました。バージョン5.xでは、これらのファイルはデバイス毎のサブフォルダに配置されます。デバイス記述ファイルを明示的に指定している場合は、新しいディレクトリ構成に注意する必要があります。

フラッシュローダ

アプリケーションのダウンロードにフラッシュローダを使用するには、simple-code フォーマットの追加の出力ファイルが必要です。バージョン4.xでは、sim ファイルを追加生成するようにXLINKを手動で設定する必要があります。バージョン5.xでは、C-SPYが自動的にダウンロードのための情報を生成するため、この追加ファイルは必要ありません。

ツールオプション

ここでは、バージョン4.xとバージョン5.xの間での、コンパイラ、アセンブラ、リンカのコンマンドラインオプションの変更点について説明します。

コンパイラオプションに関連する変更点

以下の表は、変更されたバージョン4.xのコンパイラコマンドラインオプションを示したものです。

古いコンパイラオプション	説明	バージョン 5.x
--library_module	ライブラリモジュールを生成	削除
--module_name	オブジェクトモジュール名を	削除

古いコンパイラオプション	説明	バージョン 5.x
	設定	
--omit_types	型情報を除外	削除
--segment	セグメント名/セクション名を 変更	--section
--separate_cluster_for_initialized_variables	初期化変数と非初期化変数を 分離	削除
-s および -z	最適化レベルを設定	-O

表1: コンパイラオプションの変更点

-s、-z と -O

バージョン5.xでは、最適化レベルを設定するコンパイラオプションはバージョン4.xとは違った動作をします。バージョン4.xでは、最適化の速度レベルを設定するオプションと最適化のサイズレベルを設定するオプションがあり、その中から1つを選択します。バージョン5.xでは、最適化のレベルを設定するオプションがあり、それぞれのレベルでコンパイラはサイズと速度のバランスを取ります。最高レベルを指定すると、サイズまたは速度を明示的に最適化することができます。

以下の表は、-s、-zの、-Oとの対応関係を示したものです。

バージョン 4.x	バージョン 5.x
-s2, -z2	-O0 (最適化しない)
-s3, -z3	-O1 (低)
-s6, -z6	-O2 (中)
-z9	-O3 (高、サイズ優先)
-s9	-O4 (高、サイズと速度のバランスを取る)
--	-O5 (高、速度優先)

表2: コンパイラオプションの変更点

注:バージョン5.xで、オプション-s または-z を使用すると、診断メッセージが出力されます。

アセンブラオプションに関連する変更点

以下の表は、変更されたバージョン4.xのアセンブラコマンドラインオプションを示したものです。

古いアセンブラオプション	説明	バージョン 5.x
-b	ライブラリモジュールを生成	削除
-X	オブジェクトファイルの外部参照を行わない	削除

表3: アセンブラオプションの変更点

リンカオプションに関連する変更点

以下の表は、XLINKのコマンドラインオプションと、対応するILINKの機能をまとめたものです。

XLINK のオプション	説明	ILINK
-!	コメント区切り文字	icf ファイルでは、/*...*/ または //
-A	プログラムとしてロードする	削除。4 ページの「アセンブラとアセンブラソースコード」を参照
-a	静的オーバーレイを無効にする	削除

XLINK のオプション	説明	ILINK
-B	出力ファイルを常に生成する	--force_output
-b	バンクセグメントを定義する	icf ファイルで指定*
-C	ライブラリとしてロードする	削除。4 ページの「アセンブラとアセンブラソースコード」を参照
-c	プロセッサタイプを指定する	削除
-D	シンボルを定義する	--define_symbol
-d	コード生成を無効にする	削除
-E	オブジェクトコードを生成しない	削除
-e	外部シンボルの名前を変更する	--redirect
-F	出力フォーマットを指定する	削除
-f	XCL ファイル名を指定する	-f. ILINK では、オプション--config を使用して設定ファイルを指定
-G	グローバル型のチェックを無効にする	削除
-g	グローバルエントリを要求する	--keep
-H	未使用コードメモリをフィルする	--fill
-h	指定範囲をフィルする	--fill
-I	インクルードパスを指定する	削除
-J	チェックサムを生成する	--checksum
-K	コードを複製する	icf ファイルで指定*
-L	リストファイルのディレクトリを指定する	--log_file
-l	リストファイル名を指定する	--log_file
-M	論理アドレスを物理アドレスにマッピングする	icf ファイルで指定*
-n	ローカルシンボルを無視する	--no_locals
-O	複数の出力ファイル	削除
-o	出力ファイル名	変更なし。ただし、--output をエイリアスとして使用可能
-P	バックされたセグメントを定義する	icf ファイルで指定*
-p	1 ページあたりの行数を指定する	削除
-Q	分散ローディング	icf ファイルで指定*
-q	リレー関数の最適化を無効にする	削除
-R	アドレス範囲のチェックを無効にする	--diag_suppress
-r	デバッグ情報	削除。ILINK では、デバッグ情報はデフォルトで出力される。出力しない場合は、--no_debug を使用
-rt	デバッグ情報(ターミナル I/O サポート)	--semihosting
-S	メッセージ表示を無効にする	--silent

XLINK のオプション	説明	ILINK
-s	新しいアプリケーションエントリポイントを指定する	--entry
-U	アドレス空間を共有する	icf ファイルで指定*
-V	コードとデータ用に再配置可能エリアを宣言する	icf ファイルで指定*
-w	診断制御を設定する	--diag_error, --diag_remark, --diag_suppress, --diag_warning, --diagnostics_tables, --error_list, --no_warnings, --remarks, --warnings_are_errors, --warnings_affect_exit_code
-x	クロスリファレンスリストを生成する	--map
-Y	出力フォーマットのバリエーション	削除
-y	出力フォーマットのバリエーション	削除
-Z	セグメントを定義する	icf ファイルで指定*
-z	セグメントオーバーラップのワーニング	削除

表4: XLINK オプションとILINK の対応関係

*ILINKでは、この機能はリンカオプションとしてコマンドラインやIAR Embedded Workbench IDEで指定できません。代わりに、リンカ設定ファイルの設定の一部として指定します。

以下のオプションは全く変更されていません。

--image_input, --misrac, --misrac_verbose.

セグメントとセクション

ここでは、バージョン4.xのセグメントとバージョン5.xのセクションの違いについて説明します。

セクション、セクション名、およびその使用方法の詳細については、『ARM® IAR C/C++ Development Guide』を参照してください。

命名規約

セグメントの命名規約は、セクションの命名規約とは多少異なります。

4.xでは、セグメント名はすべて大文字で記述します。コードセグメントおよびデータセグメントでは、セグメントのベース名はメモリアイプを表します。また、静的データセグメントでは、サフィックスが内容のタイプを表します。

5.xでは、大文字で記述されているセクション名もありますが、ピリオドの後に小文字が使用されているセクション名もあります。これらのセクションでは、セクション名が内容のタイプを表し、サフィックスがメモリアイプを表します。

初期化用セグメント

4.xでは、初期化データ用のセグメントと初期化変数用のセグメントがそれぞれ1つずつコンパイラによって作成されます。5.xでは、初期化データを格納するデータセクションが1つ、コンパイラによって作成されます。その後、ILINKは、このセクションを適切に変換して初期化処理を行います。初期化の詳細については、『ARM® IAR C/C++ Development Guide』を参照してください。

古いセグメントと新しいセクションの対応関係

完全に削除された古いセグメントがある一方、古いセグメントと全く対応しない新しいセクションもあります。

以下の表は、古いセグメント名と、バージョン5.xでそれに対応するもの、および追加されたセクションを示しています。

古いセグメント	新しいセクション	コメント
CODE	.text	
CODE_I	.textrw	
CODE_ID	.textrw	初期化データ用セクションなし
CSTACK	CSTACK	
DATA_AC	--	定数の絶対アドレスへの配置はサポートされないため、そのためのセグメント/セクションは不要
DATA_AN	--	__no_initとして宣言されている絶対変数はエリアの予約が不要になったため、そのためのセグメント/セクションは不要
DATA_C	.rodata	
DATA_I	.data	
DATA_ID	.data	初期化データ用セクションなし
DATA_N	.noinit	
DATA_Z	.bss	
DIFUNCT	.difunct, PREDIFUNCT	
HEAP	HEAP	
ICODE	.text	
INITTAB	--	ILINKは、別の方法を使用してこの変換を行うので、そのためのセグメント/セクションは不要
INTVEC	.intvec	
IRQ_STACK	IRQ_STACK	
SWITAB	--	この機能は削除されたため、そのためのセグメント/セクションは不要

表5: セグメントとセクションの対応関係

アセンブラディレクティブ

アセンブラディレクトリの中には、削除されたものや動作が変更されたものがあります。以下の表は、バージョン4.xとバージョン5.xの間で変更されたアセンブラディレクトリを示したものです。

バージョン 4.x のアセン ブラディレクトリ	バージョン 5.x
ARGFRAME	削除
ASEG	削除

バージョン 4.x のアセンブラディレクトリ	バージョン 5.x
ASEGN	削除
BLOCK	削除
CFI	リソース名が標準化された。CFI の名前ブロックに、これらのリソース名のサブセットを入れる必要あり
COMMON	削除
DEFFN	削除
END	プログラムの先頭アドレスを引数として受け取らない
ENDMOD	認識はされるが機能はしない。ワーニングが生成される
FUNCALL	削除
FUNCTION	削除
LIBRARY	ライブラリモジュールではなく、ELF モジュールを起動。新規構文
LIMIT	削除
LOCFRAME	削除
MODULE	ライブラリモジュールではなく、ELF モジュールを起動。新規構文
MULTWEAK	削除
NAME	ELF プログラムモジュールを起動。新規構文
ORG	削除
OVERLOAD	削除
PROGRAM	ELF プログラムモジュールを起動。新規構文
RSEG	使用する RSEG ディレクティブの最初のインスタンスは、DC や DS などの、すべてのコード生成ディレクティブおよびアセンブラ命令より前に指定する必要がある。このディレクティブは、新しいディレクティブ SECTION のエイリアスである。新規構文
STACK	削除
SYMBOL	削除

表6: アセンブラディレクトリの変更点

バージョン 5.x のアセンブラディレクティブについては、『ARM® IAR Assembler Reference Guide』を参照してください。

ファイル名拡張子

以下の表は、デフォルトのファイル名拡張子に関連する変更点を示したものです。

古いファイル名拡張子	新しいファイル名拡張子	説明/コメント
s79	s	アセンブラソースファイル
r79	o	オブジェクトモジュール
r79	a	ライブラリオブジェクトモジュール
a79	out	ターゲットプログラム
d79	out	デバッグ用ターゲットプログラム

表7: ファイル名拡張子の変更点