



MPLAB[®] C30
C コンパイラ
ユーザーズガイド

マイクロチップ・テクノロジー社 (以下、マイクロチップ社) デバイスのプログラム保護機能に関して、以下の点にご注意ください。

- マイクロチップ社製品は、該当する「マイクロチップ社データシート」に記載の仕様を満たしています。
- マイクロチップ社では、通常の条件ならびに仕様どおりの方法で使用した場合、マイクロチップ社製品は現在市場に流通している同種製品としては最もセキュリティの高い部類に入る製品であると考えております。
- プログラム保護機能を解除するための不正かつ違法な方法が存在します。マイクロチップ社の確認している範囲では、このような方法のいずれにおいても、マイクロチップ社製品を「マイクロチップ社データシート」の動作仕様外の方法で使用する必要があります。このような行為は、知的所有権の侵害に該当する可能性が非常に高いと言えます。
- マイクロチップ社は、コードの保全について懸念を抱いているお客様と連携し、対応策に取り組んでいきます。
- マイクロチップ社を含むすべての半導体メーカーの中で、自社のコードのセキュリティを完全に保証できる企業はありません。プログラム保護機能とは、マイクロチップ社が製品を「解読不能」として保証しているものではありません。

プログラム保護機能は常に進歩しています。マイクロチップ社では、製品のプログラム保護機能の改善に継続的に取り組んでいます。マイクロチップ社のプログラム保護機能を解除しようとする行為は、デジタルミレニアム著作権法に抵触する可能性があります。そのような行為によってソフトウェアまたはその他の著作物に不正なアクセスを受けた場合は、デジタルミレニアム著作権法の定めるところにより損害賠償訴訟を起こす権利があります。

本書に記載されているデバイスアプリケーションなどに関する情報は、ユーザーの便宜のためにのみ提供されているものであり、更新によって無効とされることがあります。アプリケーションと仕様の整合性を保証することは、お客様の責任において行ってください。マイクロチップ社は、明示的、暗黙的、書面、口頭、法定のいずれであるかを問わず、本書に記載されている情報に関して、状態、品質、性能、商品性、特定目的への適合性をはじめとする、いかなる類の表明も保証も行いません。マイクロチップ社は、本書の情報およびその使用に起因する一切の責任を否認します。マイクロチップ社デバイスを生命維持および/または保安のアプリケーションに使用することはデバイス購入者の全責任において行うものとし、デバイス購入者は、デバイスの使用に起因するすべての損害、請求、訴訟、および出費に関してマイクロチップ社を弁護、免責し、同社に不利益が及ばないようにすることに同意するものとし、暗黙的あるいは明示的を問わず、マイクロチップ社が知的財産権を保有しているライセンスは一切譲渡されません。

商標

Microchip の名前付きロゴ、Microchip ロゴ、Accuron、dsPIC、KEELOQ、KEELOQ ロゴ、microID、MPLAB、PIC、PICmicro、PICSTART、PRO MATE、PowerSmart、rfPIC、SmartShunt は、米国およびその他の国における Microchip Technology Incorporated の登録商標です。


AmpLab、FilterLab、Linear Active Thermistor、Migratable Memory、MXDEV、MXLAB、PS ロゴ、SEEVAL、SmartSensor、The Embedded Control Solutions Company は、米国における Microchip Technology Incorporated の登録商標です。

Analog-for-the-Digital Age、Application Maestro、CodeGuard、dsPICDEM、dsPICDEM.net、dsPICworks、ECAN、ECONOMONITOR、FanSense、FlexROM、fuzzyLAB、In-Circuit Serial Programming、ICSP、ICEPIC、Mindi、MiWi、MPASM、MPLAB Certified ロゴ、MPLIB、MPLINK、PICKit、PICDEM、PICDEM.net、PICLAB、PICKit、PICtail、PowerCal、PowerInfo、PowerMate、PowerTool、Real ICE、rfLAB、rfPICDEM、Select Mode、Smart Serial、SmartTel、Total Endurance、UNI/O、WiperLock、ZENA、は米国およびその他の国における Microchip Technology Incorporated の商標です。

SQTP は米国における Microchip Technology Incorporated のサービスマークです。

その他、本書に記載されている商標は、各社に帰属します。

© 2007, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 再生紙を使用しています。

マイクロチップ社では、Chandler および Tempe (アリゾナ州)、Gresham (オレゴン州)、Mountain View (カリフォルニア州) の本部、設計部およびウエハ製造工場が ISO/TS-16949:2002 認証を取得しています。マイクロチップ社の品質システムプロセスおよび手順は、PIC® MCU および dsPIC® DSC、KEELOQ® コードホッピングデバイス、シリアル EEPROM、マイクロベリフェラル、不揮発性メモリ、アナログ製品に採用されています。また、マイクロチップ社の開発システムの設計および製造に関する品質システムは、ISO 9001:2000 の認証を受けています。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

目次

はじめに	1
第 1 章. コンパイラ概要	
1.1 序章	7
1.2 ハイライト	7
1.3 MPLAB C30 の説明	7
1.4 MPLAB C30 とその他の開発ツール	7
1.5 MPLAB C30 の特徴	9
第 2 章. MPLAB C30 と ANSI C の違い	
2.1 序章	11
2.2 ハイライト	11
2.3 キーワードの違い	11
2.4 文の差異	27
2.5 表現の差異	28
第 3 章. MPLAB C30 C コンパイラを使用する	
3.1 序章	29
3.2 ハイライト	29
3.3 概要	29
3.4 ファイル名規則	30
3.5 オプション	30
3.6 環境変数	55
3.7 事前定義制約	56
3.8 コマンドライン上の一つのファイルをコンパイルする	57
3.9 コマンドライン上の複数のファイルをコンパイルする	58
第 4 章. MPLAB C30 C コンパイラ実行時環境	
4.1 序章	59
4.2 ハイライト	59
4.3 アドレス空間	59
4.4 コードとデータセクション	61
4.5 スタートアップと初期化	63
4.6 メモリ空間	64
4.7 メモリモデル	65
4.8 コードとデータの配置	67
4.9 ソフトウェアスタック	68
4.10 C スタック使用方法	69
4.11 C ヒープの使用方法	71
4.12 関数コール規則	72
4.13 レジスタの規則	74

	4.14	ビット反転とモジュロアドレッシング	75
	4.15	Program Space Visibility (PSV) の使用方法	75
第 5 章 .	データタイプ		
	5.1	序章	77
	5.2	ハイライト	77
	5.3	データの表現	77
	5.4	整数	77
	5.5	浮動小数点	78
	5.6	ポインタ	78
第 6 章 .	デバイスサポートファイル		
	6.1	序章	79
	6.2	ハイライト	79
	6.3	プロセッサヘッダファイル	79
	6.4	レジスタ定義ファイル	80
	6.5	SFR の使用	81
	6.6	マクロの使用	83
	6.7	C コードからの EEDATA へのアクセス - dsPIC30F DSC のみ	84
第 7 章 .	割り込み		
	7.1	序章	87
	7.2	ハイライト	87
	7.3	割り込みサービスルーチンを記述する	88
	7.4	割り込みベクトルを記述する	90
	7.5	割り込みサービスルーチンのコンテキスト保存	100
	7.6	レイテンシ	100
	7.7	割り込みのネスティング	100
	7.8	割り込みの有効化 / 無効化	101
	7.9	割り込みサービスルーチンとメインラインコード間のメモリの共有	102
第 8 章 .	アセンブリ言語と C モジュールの混用		
	8.1	序章	107
	8.2	ハイライト	107
	8.3	アセンブリ言語と C 変数と関数の混用	107
	8.4	インラインアセンブリ言語を使用する	109
付録 A .	実装時定義動作		
	A.1	はじめに	115
	A.2	変換	116
	A.3	環境	116
	A.4	識別子	117
	A.5	文字	117
	A.6	整数	118
	A.7	浮動小数点	118
	A.8	配列およびポインタ	119
	A.9	レジスタ	119
	A.10	構造体、共用体、列挙およびビットフィールド	120

A.11	修飾子	120
A.12	宣言子	120
A.13	ステートメント	120
A.14	プリプロセッサ	121
A.15	ライブラリ関数	122
A.16	信号	123
A.17	ストリームおよびファイル	123
A.18	tmpfile	124
A.19	errno	124
A.20	メモリー	124
A.21	ABORT	124
A.22	EXIT	124
A.23	getenv	125
A.24	システム	125
A.25	strerror	125
付録 B.	MPLAB C30 C コンパイラ診断	
B.1	はじめに	127
B.2	エラー	127
B.3	警告	150
付録 C.	MPLAB C18 と MPLAB C30 C コンパイラ	
C.1	はじめに	175
C.2	データフォーマット	176
C.3	ポインタ	176
C.4	ストレージクラス	176
C.5	スタックの使い方	176
C.6	ストレージ修飾子	177
C.7	定義されたマクロ名	177
C.8	整数拡張	177
C.9	文字列定数	177
C.10	匿名構造体	178
C.11	アクセスメモリー	178
C.12	インラインアセンブリー	178
C.13	Pragma	178
C.14	メモリーモデル	179
C.15	コール規則	180
C.16	スタートアップコード	180
C.17	コンパイラで管理されるリソース	180
C.18	最適化	180
C.19	オブジェクトモジュールフォーマット	180
C.20	インプリメンテーション定義された動作	180
C.21	ビットフィールド	181

付録 D.	使用を廃止した機能	
D.1	はじめに	183
D.2	ハイライト	183
D.3	事前定義定数	183
付録 E.	ASCII 文字セット	
付録 F.	GNU 無料ドキュメントライセンス	
用語集		193
索引		201
世界各国での販売およびサービス		210

はじめに

顧客の皆様への注意

すべての文書には日付が記載されており、このマニュアルも例外ではありません。マイクロチップ社のツールと文書は顧客の皆様へのニーズに応えるため日々進化を続けており、このマニュアル内のダイアログおよび/またはツールの説明も変更となる可能性があります。最新のマニュアルは当社のウェブサイト (www.microchip.com) で入手してください。

マニュアルは "DS" 番号で識別されます。この番号は各ページ下のページ数の下欄に記載されています。DS 番号の番号付けは "DSXXXXXA" となっており、"XXXXX" は文書番号、"A" は改訂番号となっています。

開発ツールの最新の情報は、**MPLAB IDE** のオンラインヘルプを参照してください。「ヘルプ」メニューを選択し、「トピックス」をクリックすると、入手可能なオンラインヘルプファイルのリストが開きます。

序章

このドキュメントは、ユーザーのアプリケーションを開発するために、dsPIC® デジタルシグナルコントローラ (DSC) デバイス用のマイクロチップ MPLAB C30 C コンパイラを使用する際の手助けをすることを目的としています。MPLAB C30 は GCC (GNU コンパイラ群) ベースの言語ツールであり、フリーソフトウェアファウンデーション (FSF) からのソースコードをベースにしています。FSF の詳細につきましては、www.fsf.org をご参照ください。

マイクロチップから供給可能なその他の GNU 言語ツールは以下のとおりです。

- MPLAB ASM30 アセンブラ
- MPLAB LINK30 リンカ
- MPLAB LIB30 ライブラリアン/アーカイバ

本章で説明する内容は以下の通りです。

- 本ガイドについて
- 推奨文献
- トラブルシューティング
- マイクロチップウェブサイト
- 開発システム変更に関する顧客通知サービス

本ガイドについて

本ガイドのレイアウト



本ガイドは、ファームウェアを開発する際の MPLAB C30 の使用方法を説明します。本ガイドのレイアウトは以下の通りです。

- **第 1 章 : コンパイラ概要** – MPLAB C30、開発ツールおよび特徴について説明します。
- **第 2 章 : MPLAB C30 と ANSI C の違い** – MPLAB C30 構文でサポートされる C 言語と標準 ANSI-89 C との違いについて説明します。
- **第 3 章 : MPLAB C30 C コンパイラを使用する** – コマンドラインからの MPLAB C30 コンパイラの使い方について説明します。
- **第 4 章 : MPLAB C30 C コンパイラ実行時環境** – MPLAB C30 の実行時モデルおよびセクション情報、初期化、メモリモデル、ソフトウェアスタック等々について説明します。
- **第 5 章 : データタイプ** – MPLAB C30 で用いられる整数、浮動小数、ポインタのデータ形型を説明します。
- **第 6 章 : デバイスサポートファイル** – MPLAB C30 のヘッダ、レジスタ定義ファイルおよび SFR と一緒を使用する場合の使用法を説明します。
- **第 7 章 : 割り込み** – 割り込みの使用法を説明します。
- **第 8 章 : アセンブリ言語と C モジュールの混用** – MPLAB C30 と MPLAB ASM30 アセンブル言語モジュールを一緒に用いる場合のガイドラインを説明します。
- **付録 A: 実装時定義動作** – ANSI 標準で実装時定義と記載された MPLAB C30 の特殊パラメータについて説明します。
- **付録 B: MPLAB C30 C コンパイラ診断** – MPLAB C30 が生成するエラーや警告メッセージの一覧を示します。
- **付録 C: MPLAB C18 と MPLAB C30 C コンパイラ** – PIC18XXXXXX コンパイラ (MPLAB C18) と dsPIC コンパイラ (MPLAB C30) の違いについて特記します。
- **付録 D: 使用を廃止した機能** – 廃止された機能の詳細を説明します。
- **付録 E: ASCII 文字セット** – ASCII 文字の一覧を示します。
- **付録 F: GNU 無料ドキュメントライセンス** – 無料ソフト基金の使用許諾。

本ガイドで用いられる凡例

本マニュアルでは、以下のドキュメント凡例を用いています。

ドキュメント凡例

文字種類	意味	使用例
Arial フォント:		
イタリック文字	参照文献	<i>MPLAB IDE ユーザーズガイド</i>
	強調する文字	... は 唯一 のコンパイラ...
最初が大文字	ウィンドウ	the Output window
	ダイアログ	the Settings dialog
	メニュー選択	select Enable Programmer
引用	ウィンドウもしくはダイアログ内のフィールド名	“Save project before build”
右矢印を持ったアンダーライン付きイタリック	メニュー選択パス	<i>File>Save</i>
太字	ダイアログボタン	OK をクリック
	タブ	Power タブをクリック
<code>'bnnnn</code>	バイナリ数値。n は桁数だけ並べる。	'b00100, 'b10
角括弧 <>, 内の文字	キーボードのキー	<Enter>, <F1> を押します
クーリエニューフォント:		
通常クーリエニュー	サンプルソースコード	#define START
	ファイル名	autoexec.bat
	ファイルパス	c:\mcc18\h
	キーワード	_asm, _endasm, static
	コマンドラインオプション	-Opa+, -Opa-
	ビット値	0, 1
イタリッククーリエニュー	変数の引数	<i>file.o</i> では、 <i>file</i> は有効なファイル名を表す。
<code>0xn</code>	16 進法番号。n は 16 の各桁。	0xFFFF, 0x007A
角括弧 []	オプション引数	mcc18 [options] file [options]
中括弧 { } とパイプ文字	互いに排他的な配列もしくは OR 選択	errorlevel {0 1}
省略 ...	テキストの繰り返し	var_name [, var_name...]
	ユーザーが入力するコード	void main (void) { ... }
アイコン		
	この機能は、完全版のソフトウェアでのみサポートされます。	
	デバイスによっては、この機能をサポートしていません。サポートしている場合は、タイトルまたは本文中に記載があります。	

推奨文献

このユーザーズガイドは dsPIC デバイス用の MPLAB C30 コンパイラの使い方について説明しています。MPLAB C30 とその他のツールに関するより詳細な情報については、以下の文書をご覧ください。

README ファイル

マイクロチップツールの最新の情報については、ソフトウェアに含まれる添付 README ファイル (ASCII テキストファイル) をお読みください。

dsPIC® 言語ツールの入門 (DS70094)

dsPIC デジタルシグナルコントローラ (DSC) 用のマイクロチップ言語ツール (MPLAB ASM30, MPLAB LINK30, MPLAB C30) のインストールと使い方について説明されています。dsPIC シミュレータや MPLAB SIM30 の使用例も説明されています。

MPLAB® ASM30, MPLAB LINK30 とユーティリティユーザーズガイド (DS51317)

dsPIC DSC アセンブラ、MPLAB ASM30, dsPIC DSC リンカ、MPLAB LINK30 および種々の dsPIC DSC ユーティリティ、MPLAB LIB30 アーカイバ/ライブラリアンも含めて、使用方法が説明されています。

汎用とセンサーファミリの dsPIC30F のデータシート (DS70083)

dsPIC30F デジタルシグナルコントローラ (DSC) 用のデータシートです。デバイスとそのアーキテクチャの概要について説明しています。メモリ構成、DSP 命令と周辺機能および電気的特性の詳細について説明しています。

dsPIC30F ファミリリファレンスマニュアル (DS70046)

本ファミリリファレンスマニュアルは dsPIC30F ファミリのアーキテクチャと周辺モジュールの動作について説明しています。

dsPIC30F プログラマリファレンスマニュアル (DS70157)

dsPIC30F デバイスのプログラマガイドで、プログラマモデルと命令セットを含みます。

C 標準情報

情報システムにおける米国標準 – *Programming Language - C*. American National Standards Institute (ANSI), 11 West 42nd. Street, New York, New York, 10036.

この標準は、形式および C プログラム言語で表現されたプログラムの解釈について規定しています。種々のコンピューティングシステムにおける C 言語プログラムに関して、移植性、信頼性、保守性および効率的な実行を促進することを目的としています。

C リファレンスマニュアル

Harbison, Samuel P., and Steele, Guy L., *C A Reference Manual*, Fourth Edition, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Kernighan, Brian W., and Ritchie, Dennis M., *The C Programming Language*, Second Edition. Prentice Hall, Englewood Cliffs, N.J. 07632.

Kochan, Steven G., *Programming In ANSI C*, Revised Edition. Hayden Books, Indianapolis, Indiana 46268.

Plauger, P.J., *The Standard C Library*, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Van Sickle, Ted., *Programming Microcontrollers in C*, First Edition. LLH Technology Publishing, Eagle Rock, Virginia 24085.

トラブルシューティング

本ドキュメントに記載のない問題に対する情報につきましては、README ファイルをご覧ください。

マイクロチップウェブサイト

マイクロチップでは、マイクロチップワールドワイドウェブ (WWW) サイト www.microchip.com で、オンラインサポートを提供しています。このウェブサイトは、お客様がファイルや情報を最も簡単に入手できる手段としてご利用いただけます。サイトをご覧になるには、各自、お好みのブラウザでご覧ください。マイクロチップウェブサイトは、以下の情報を提供しています。

- **製品サポート**—データシート、エラッタ、アプリケーションノート、サンプルプログラム、デザインリソース、ユーザーズガイド、サポート資料、最新ソフトウェアリリースとアーカイブされたソフトウェア
- **一般的なテクニカルサポート**—よくある質問 (FAQ)、テクニカルサポート要求、オンラインディスカッショングループ、マイクロチップコンサルタントプログラムメンバーリスト
- **マイクロチップのビジネス**—製品選択、オーダーガイド、最新マイクロチッププレスリリース、セミナーおよびイベントのリスト、マイクロチップセールスオフィス、ディストリビュータ、工場の責任者

開発システムに関する顧客への通知サービス

マイクロチップは、お客様が容易にマイクロチップ製品の最新情報を入手できることをお手伝いできるように、顧客通知サービスを継続して行っています。一度ご登録いただければ、あなたのご指定した製品ファミリーもしくはご興味のある開発ツールに関して、変更、更新、改定もしくは正誤表が発行される毎にメールでの通知を受けることができます。

登録するには、マイクロチップのウェブサイトにアクセスし、Customer Change Notification をクリックし、指示に従ってご登録ください。

開発システム製品は下記の通り分類されます。

- **コンパイラ**—マイクロチップ C コンパイラとその他の言語ツールに関する最新情報で、以下を含みます。MPLAB C17, MPLAB C18 もしくは MPLAB C30 C コンパイラ; MPASM™, MPLAB ASM30 アセンブラ; MPLINK™, MPLAB LINK30 オブジェクトリンカ; MPLIB™, MPLAB LIB30 オブジェクトライブラリアン。
- **エミュレータ**—マイクロチップインサーキットエミュレータ、MPLAB ICE 2000 と MPLAB ICE 4000 に関する最新情報を含みます。
- **インサーキットデバッガ**—マイクロチップインサーキットデバッガ、MPLAB ICD 2 に関する最新情報を含みます。
- **MPLAB IDE**—マイクロチップ MPLAB IDE Windows® 用の統合開発環境システムツールに関する最新情報です。この項では MPLAB IDE, MPLAB SIM と MPLAB SIM30 シミュレータ、MPLAB IDE プロジェクトマネージャと全般的な編集、デバッグの特徴に注力しています。
- **プログラマ**—マイクロチップデバイスプログラマに関する最新情報で、MPLAB PM3, PRO MATE® II デバイスプログラマと PICSTART® Plus 開発用プログラマを含みます。

カスタマサポート

マイクロチップ製品のユーザーは次を経由して、サポートが受けられます。

- 販売代理店もしくは販売特約店
- 各国セールスオフィス
- 現地アプリケーションエンジニア (FAE)
- テクニカルサポート

サポートが必要な場合は、販売代理店、販売特約店もしくは現地アプリケーションエンジニア (FAE) にお電話ください。各国セールスオフィスでもお客様のサポートを行っています。販売店とその所在地については本ドキュメントの最後のページを参照してください。

テクニカルサポートはウェブサイト <http://support.microchip.com> にてご利用いただけます。

第 1 章 . コンパイラ概要

1.1 序章

デジタルシグナルコントローラ (DSC) である dsPIC® ファミリは DSP 応用システムで要求される高性能と、組込み応用システムに必要な標準マイコンの特徴を合わせ持っています。また、DSP 機能を持たない高性能のマイクロコントローラ (MCU) を使用して、他のアプリケーションにも対応できます。

これらのデバイスはすべて、最適化 C コンパイラ、アセンブラ、リンカ、およびアーカイバ/ライブラリアンを含んだ完全なソフトウェア開発ツールセットでサポートされています。

本章では、これらのツールの概要紹介や、最適化 C コンパイラの特徴を紹介し、MPLAB ASM30 アセンブラや MPLAB LINK30 リンカと一緒にどのように動作するかも説明しています。アセンブラとリンカは *MPLAB® ASM30, MPLAB LINK30 and Utilities User's Guide, (DS51317)* で詳細に説明されています。

1.2 ハイライト

本章では、次の内容を説明します。

- MPLAB C30 の説明
- MPLAB C30 とその他の開発ツール
- MPLAB C30 の特徴

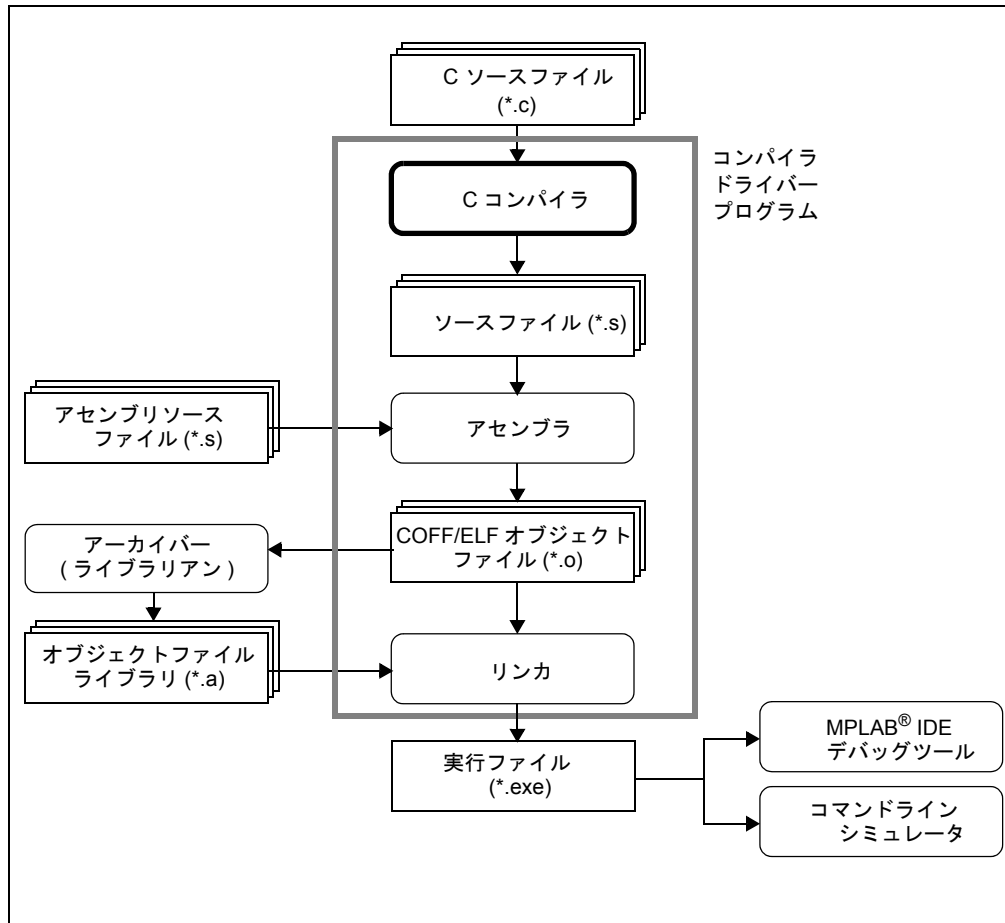
1.3 MPLAB C30 の説明

MPLAB C30 は ANSI x3.159-1989- 準拠の最適化 C コンパイラで、dsPIC DSC 組込みコントロールアプリケーション向けの言語拡張が含まれます。本コンパイラは、C コード開発用プラットフォームを提供する Windows® コンソールアプリケーションです。本コンパイラは、Free Software Foundation から供給される GCC コンパイラから移植しています。

1.4 MPLAB C30 とその他の開発ツール

MPLAB C30 は C ソースファイルをコンパイルし、アセンブル言語ファイルを生成します。これらコンパイラで生成されたファイルは、他のオブジェクトファイルやライブラリと一緒にアセンブル、リンクされて、実行可能な COFF または ELF ファイルフォーマットで最終アプリケーションプログラムが生成されます。COFF または ELF ファイルは MPLAB IDE にロードされ、テスト、デバッグされ、もしくは変換ユーティリティを用いて COFF または ELF ファイルから Intel® hex フォーマットに変換され、コマンドラインシミュレータやデバイスプログラマへのロードに適した形になります。ソフトウェア開発のデータフロー概要については図 1-1 をご参照ください。

図 1-1: ソフトウェア開発ツールのデータフロー



1.5 MPLAB C30 の特徴

MPLAB C30 の C コンパイラは豊富な特徴を持つ最適化コンパイラで、標準 ANSI C プログラムを dsPIC アセンブル言語ソースに変換します。このコンパイラはまた多くのコマンドラインオプションと言語拡張をサポートし、dsPIC DSC デバイスハード機能をすべてアクセス可能であり、コンパイラコードジェネレータを細かく制御できます。この章ではこのコンパイラの主要な特徴について説明します。

1.5.1 ANSI C 標準

MPLAB C30 コンパイラは、十分検証されたコンパイラで、ANSI 規格で定められ、Kernighan and Ritchie's *The C Programming Language* (second edition) で説明されている ANSI C 標準に準拠しています。ANSI 標準は、現在は言語の標準機能のオリジナル C 定義への拡張を含みます。この拡張により移植性が強化され、機能が增強されています。

1.5.2 最適化

本コンパイラは、C ソースから効率的でコンパクトなコードを生成する最新技術に対応した最適化パスを使用しています。最適化パスには、あらゆる C コードに対応するハイレベルの最適化および、dsPIC DSC デバイスアーキテクチャの特殊機能を利用した dsPIC DSC デバイス専用の最適化が含まれます。

1.5.3 ANSI 標準ライブラリサポート

MPLAB C30 は ANSI C 標準ライブラリと一緒に供給されます。すべてのライブラリ機能は検証されており、ANSI C ライブラリ標準に適合しています。ライブラリには、ストリング操作、ダイナミックメモリ配置、データ変換、時間管理、数学関数（三角関数、指数関数、双曲線関数）が含まれます。ファイル操作の標準 I/O 関数も含まれ、コマンドラインシミュレータを用いてホストファイルシステムへの完全アクセスをサポートします。低レベルのファイル I/O 関数に対応するソースコードがコンパイラ配布時に提供され、本機能が必要なアプリケーションのスタートポイントとして使用できます。

1.5.4 フレキシブルなメモリモデル

本コンパイラはラージ、スモール両方のコードとデータモデルをサポートします。スモールコードモデルはコール関数のより効率的な形式を利用でき、スモールデータモデルでは、SFR 空間内でのデータアクセス用にコンパクトな命令を使用できます。

本コンパイラは定数データアクセスのために 2 つのモデルを提供します。「constants in data」モデルは、ランタイムライブラリにより初期化されるデータメモリを使用します。「constants in code」モデルは、Program Space Visibility (PSV) ウィンドウからアクセスするプログラムメモリを使用します。

1.5.5 コンパイラドライバ

MPLAB C30 には、強力なコマンドラインドライバプログラムが含まれています。ドライバプログラムを用いることで、アプリケーションプログラムのコンパイル、アセンブル、リンクを 1 ステップでコンパイルできます。(図 1-1 参照)

メモ:

第 2 章 . MPLAB C30 と ANSI C の違い

2.1 序章

本章では、MPLAB C30 構文でサポートされる C 言語と、1989 年版標準 ANSI C でサポートする C 言語の相違点を説明します。

2.2 ハイライト

本章では下記について説明します。

- キーワードの違い
- 文の差異
- 表現の差異

2.3 キーワードの違い

本章では、通常の ANSI C と、MPLAB C30 で使用できる C 言語との間でのキーワードの違いを説明します。新しいキーワードは基本 GCC 搭載内容の一部であり、GCC の MPLAB C30 ポートの特定の特殊構文や意味を中心に標準 GCC ドキュメントに基づいた説明を行います。

- 変数属性の指定
- 関数の属性を指定する
- インライン関数
- レジスタ変数
- 複素数
- 倍長整数
- typedef による型参照

2.3.1 変数属性の指定

MPLAB C30 のキーワード `__attribute__` により、特殊な変数属性もしくは構文フィールドを指定することができます。このキーワードに続き、二重括弧内の属性仕様を指定します。現在、以下の属性が変数用としてサポートされています。

- `address (addr)`
- `aligned (alignment)`
- `deprecated`
- `far`
- `mode (mode)`
- `near`
- `noload`
- `packed`
- `persistent`
- `reverse (alignment)`
- `section ("section-name")`
- `sfr (address)`
- `space (space)`
- `transparent_union`
- `unordered`
- `unused`
- `weak`

属性指定方法として `__` (2つのアンダースコア) をそれぞれのキーワードの前後に配置する (例えば `'aligned'` ではなく `'__aligned__'`) ことでも指定できます。この方法を用いて、同じ名前を持つ可能性のあるマクロを考慮しなくともヘッダファイルで属性を使用できます。

複数の属性を指定するには、二重括弧内でコンマを使用して区切ります。

例えば以下のように記述します。

```
__attribute__ ((aligned (16), packed)).
```

<p>注: プロジェクト全体で、同じ変数属性を使用することが重要です。例えば、ファイル A で <code>far</code> 属性を付けて変数を定義し、ファイル B で <code>far</code> を付けずに <code>extern</code> を宣言した場合、リンクエラーが発生することがあります。</p>
--

address (addr)

`address` 属性は変数の絶対アドレスを指定します。この属性は `section` 属性とは同時に使用できません。`address` 属性が優先されます。`address` 属性を持つ変数は `auto_psv` 領域には配置できません。(`space()` 属性または `-mconst-in-code` オプションの項を参照) `auto_psv` 領域に配置しようとする警告が発生しコンパイラはその変数を `psv` 領域に配置します。

変数が `PSV` セクションに配置されると、アドレスはプログラマメモリアドレスになります。

```
int var __attribute__ ((address(0x800)));
```

aligned (alignment)

この属性は、バイト単位で、変数の最小配列を指定します。alignment は 2 のべき乗値で指定します。例えば、次のように宣言すると

```
int x __attribute__ ((aligned (16))) = 0;
```

コンパイラは 16 バイト領域でグローバル変数 x を割り当てます。dsPIC DSC デバイスでは、配列オペランドを必要とする DSP 命令やアドレッシングモードをアクセスするために、アセンブラ表記と一緒に使用することができます。

上記例で示されるように、与えられた変数に対してコンパイラで用いる配列を、(バイト単位で) 明確に指定することができます。別の方法として、配列要素数は省略して、dsPIC DSC デバイスにとって最も有効な配列になるように変数の配列指定をすることもできます。例えば、次のように記述できます。

```
short array[3] __attribute__ ((aligned));
```

配列属性の指定時に配列要素数を省略すると、コンパイラは宣言された変数の配列を、ターゲットマシン上で用いられるデータ形式の最大配列 (dsPIC DSC デバイスの場合は 2 バイト (1 ワード)) に、自動的に設定します。

aligned 属性は配列を増加させるのみですが、packed を指定すると減少させることができます。(以下を参照ください)。aligned された属性は、reverse の属性と競合します。これは両方の属性に対するエラー条件となります。

deprecated

deprecated 属性は、コンパイラで特別に認識するよう設定した宣言を生成します。deprecated 関数もしくは変数が使用されると、コンパイラはワーニングを発生します。

deprecated 定義は、いったん定義されるとすべてのオブジェクトファイル内で有効となります。例えば、以下のファイルをコンパイルした場合、

```
int __attribute__ ((__deprecated__)) i;
int main() {
    return i;
}
```

次のような警告が発生します。

```
deprecated.c:4: warning: 'i' is deprecated (declared
    at deprecated.c:1)
```

この場合でも i は通常どおり定義されオブジェクトファイルに生成されます。

far

far 属性はコンパイラに変数が near データ領域 (最初の 8 KB) に配置される必要がないことを伝えます (つまり、その変数はデータメモリ内のいずれの場所に配置されてもかまわないこととなります)。

mode (*mode*)

この属性は、データ型の宣言の際に、*mode* に対応させてどの型を使用するかを指定します。これにより、データ幅に従って整数型もしくは浮動小数点型が指定されます。*mode* で使用できる有効値は以下の通りです。

モード	データ幅	MPLAB® C30 Type
QI	8 ビット	char
HI	16 ビット	int
SI	32 ビット	long
DI	64 ビット	long long
SF	32 ビット	float
DF	64 ビット	long double

この属性は、MPLAB C30 でサポートされるターゲット全般において移植可能なコードを記述する場合に役立ちます。例えば、以下の関数は、2つの32ビット符号付き整数を加算し、結果を1つの32ビット符号付き整数で返します。

```
typedef int __attribute__((__mode__(SI))) int32;
int32
add32(int32 a, int32 b)
{
    return(a+b);
}
```

byte もしくは `__byte__` のモードを指定することで、1 バイト整数に相当するモードを指示でき、word もしくは `__word__` のモードを指定することで、1 ワード整数モードを指示でき、pointer もしくは `__pointer__` のモードを指定することで、ポインタを表すモードを指示できます。

near

near 属性はコンパイラに変数が near データ領域（データメモリの最初の 8 KB）に配置されていることを伝えます。この変数は near データ領域に配置されていない（もしくはどこに配置されているか不明な）変数よりも効率的にアクセスできる場合があります。

```
int num __attribute__((near));
```

noload

noload 属性は変数のための領域を割り当てる必要がありますが、初期値はロードする必要がないことを示します。この属性は、シリアル EEPROM 読み込みのようにアプリケーションが実行時にメモリに変数をロードするよう設計されている場合に有用です。

```
int table1[50] __attribute__((noload)) = { 0 };
```

packed

packed 属性は、aligned 属性で指定されない限り設定した値が小さい、変数もしくは構造体フィールドが取ることのできる最小サイズの配列（1 変数につき 1 バイト、1 フィールドにつき 1 ビット）に圧縮されます。

下記構造体では、フィールド x が圧縮されるので、x は a のすぐ次に続いて配置されます。

```
struct foo
{
    char a;
    int x[2] __attribute__((packed));
};
```

注： デバイスアーキテクチャではワードは偶数バイト領域で配列されていますので、packed 属性を用いる際には、ランタイムアドレッシングエラーを回避してください。

persistent

persistent 属性は起動時に初期化したりクリアにすべきではない変数を指定します。persistent 属性を持つ変数は、デバイスのリセット後の状態を保ちます。

```
int last_mode __attribute__((persistent));
```

reverse (alignment)

reverse 属性は変数の最終アドレスプラス 1 の最小の配列を指定します。配列はバイト単位で指定され、2 の累乗にする必要があります。reverse 配列の変数は dsPIC DSC アセンブリ言語で減分 modulo バッファに使用できます。この属性は、アセンブリ言語でアクセス可能な C の変数をアプリケーションが定義する場合に有用です。

```
int buf1[128] __attribute__((reverse(256)));
```

reverse 属性は aligned 属性および section 属性と競合します。reverse 配列変数にセクションの指定をしようとしても無視され、警告が発生します。reverse 属性を持つ変数は auto_psv 領域 (space() 属性または -mconst-in-code オプションを参照) には配置できません。配置を試みると、警告が発生しコンパイラはその変数を PSV 領域に配置します。

section ("section-name")

デフォルト設定では、コンパイラは生成されたオブジェクトを .data や .bss のようなセクション内に配置します。section 属性は、変数（もしくは関数）を特定のセクション内に配置するように指定し、この動作を上書きします。

```
struct array {int i[32];}
struct array buf __attribute__((section("userdata"))) = {0};
```

section 属性は address 属性および reverse 属性と競合します。いずれを使用する場合でもセクション名は無視され警告が発生します。この属性は space 属性とも競合します。詳しくは space 属性の説明を参照してください。

sfr (*address*)

sfr 属性はコンパイラに変数が SFR であることを宣言し、*address* パラメータを用いて変数のランタイムアドレスを指定します。

```
extern volatile int __attribute__ ((sfr(0x200)))ulmod;
```

エラーの発生を回避するため、外部定義であることを指定します。

注: 一般的には、sfr 属性は、プロセッサ ヘッド ファイルでのみ使用されません。特定のアドレスに一般的なユーザー変数を定義するには、address 属性を near や far と合わせて使用して、正しいアドレッシングモードを指定します。

space (*space*)

通常、コンパイラは変数を一般的なデータ領域に割り当てます。space 属性を使用して、コンパイラに変数を特定のメモリ領域に割り当てるよう指示します。メモリ領域については、**セクション 4.6 「メモリ空間」** で説明します。space 属性では、以下の引数が使用できます。

data

一般的なデータ領域に変数を割り当てます。一般的なデータ領域内の変数には通常の C ステートメントを使用してアクセスできます。これはデフォルトの割り当てです。

xmemory - dsPIC30F/33F DSC のみ

変数を X データ領域に割り当てます。X データ領域内の変数には通常の C ステートメントを使用してアクセスできます。xmemory 領域の割り当て例は以下の通りです。

```
int x[32] __attribute__ ((space(xmemory)));
```

ymemory - dsPIC30F/33F DSC のみ

変数を Y データ領域に割り当てます。Y データ領域内の変数には通常の C ステートメントを使用してアクセスできます。ymemory 領域の割り当て例は以下の通りです。

```
int y[32] __attribute__ ((space(ymemory)));
```

prog

変数をプログラム領域の実行可能コードに指定されているセクションに割り当てます。プログラム領域内の変数は通常の C ステートメントを使用してアクセスできません。プログラマーにより明示的にアクセスする必要があり、通常、テーブルアクセスインラインアセンブリ命令またはプログラム空間可視化ウィンドウを使用してアクセスします。

auto_psv

の管理下で、自動 Program Space Visibility (PSV) ウィンドウアクセスに指定されたセクションのプログラムメモリ領域に変数を割り当てます。auto_psv 内の変数は通常の C ステートメントを使用して読み取り、かつ（書き込みは不可）、合計で最大 32K の割り当て空間があります。space(auto_psv) を指定する際は、section 属性を用いてセクション名を付けることはできません。セクション名は無視され、警告が発生します。auto_psv 空間内の変数は特定のアドレスまたは逆配列には配置できません。

注: auto_psv セクション内の変数は起動時にデータメモリにアップロードされません。この属性は RAM の使用率に役立ちます。



dma - PIC24H MCU、dsPIC33F DSC のみ

変数を DMA メモリに割り当てます。DMA メモリの変数は、通常の C ステートメントを使用して、DMA の周辺機器からアクセスできます。

`__builtin_dmaoffset()` (『16-Bit Language Tools Libraries』、DS51456 を参照) を使用すると、DMA 周辺機器を設定するための正しいオフセットが見つかります。

psv

変数をプログラム空間の Program Space Visibility (PSV) ウィンドウアクセスのセクションに割り当てます。リンカは PSVPAG レジスタを設定するだけですべての変数がアクセスできるようにセクションを割り当てます。PSV 空間内の変数はコンパイラによって管理されず、通常の C ステートメントではアクセスできません。この空間にはプログラマが明示的にアクセスする必要があり、通常、テーブルアクセス インラインアセンブリ命令または PSV ウィンドウを使用します。



eedata - dsPIC30F DSC のみ

変数を EEData 空間に割り当てます。EEData 空間内の変数には通常の C ステートメントではアクセスできません。この空間にはプログラマが明示的にアクセスする必要があり、テーブルアクセス インラインアセンブリ命令または PSV ウィンドウを使用します。

transparent_union

この属性は、union という関数パラメータに付属して動作し、対応する引数がいずれかの union メンバのタイプを持ちますが、そのタイプが最初の union メンバのタイプであるかのように引渡しされます。その引数は、トランスペアレントな union の最初のメンバの呼び出し方法として関数に引き渡され、union そのものの呼び出し方法は用いられません。この引数が引き渡されて正常動作するように、Union のすべてのメンバは同じ機械語表現にする必要があります。

unordered

unordered 属性は、この変数の配置が現在の C ソースファイル内の他の変数に連携して相対的に移動してもよいことを示しています。

```
const int __attribute__((unordered)) i;
```

unused

この属性は、変数に付属して動作し、変数が用いられない可能性があることを意味します。MPLAB C30 はこのような変数に対しては、未使用変数の警告をしません。

weak

weak 属性は、weak シンボルとして生成するように宣言を行います。weak シンボルはグローバル定義で取って代わられます。weak が、外部シンボルへの参照に適用される時には、リンクは不要です。例えば、以下のようになります。

```
extern int __attribute__((__weak__)) s;
int foo() {
    if (&s) return s;
    return 0; /* possibly some other value */
}
```

上記プログラムで、s が他のモジュールで定義されていない場合には、プログラムはリンクしますが、s はアドレスが付与されません。条件文は s が定義されていることをチェックし、もしあればその値を返し、そうでなければ、'0' を返します。この特徴には多くの使用方法がありますが、ほとんどの場合はオプションライブラリと一緒にリンクされるジェネリックコードの生成に使用されます。

weak 属性は変数と同様に関数へも適用できます。

```
extern int __attribute__((__weak__))
compress_data(void *buf);
int process(void *buf) {
    if (compress_data) {
        if (compress_data(buf) == -1) /* error */
        }
    }
    /* process buf */
}
```

上記のコードでは、関数 `compress_data` がリンクされていれば、他のモジュールから使用可能です。この特徴の使用可否は、コンパイル時ではなく、リンク時に決定します。

定義時に weak 属性の与える影響はさらに複雑で、複数のファイルを記述する必要があります。

```
/* weak1.c */
int __attribute__((__weak__)) i;

void foo() {
    i = 1;
}

/* weak2.c */
int i;

extern void foo(void);

void bar() {
    i = 2;
}

main() {
    foo();
    bar();
}
```

ここで、`i` の `weak2.c` での定義はシンボルにとって強い定義になります。リンクエラーは発行されず、両方の `i` は同じ記憶位置を参照します。記憶位置は `weak1.c` 内の `i` に割り当てられますが、この領域はアクセスできません。

両方の `i` が同じデータ型か否かチェックしません。`weak2.c` の `i` を変更して `float` タイプにしてもリンクはできますが、関数 `foo` は期待どおりに動作せず、`foo` は 32 ビット浮動小数点の値の最下位部に値を書き込みます。逆に、`weak1.c` の weak 定義の `i` のタイプを変更して `float` にすると、悲劇的な結果になります。32 ビット浮動小数点の値を 16 ビット整数型領域に書き込み、`i` の直後にある変数を上書きします。

weak 定義のみが存在する場合は、リンカは最初のその定義の記憶域を選択し、その他の定義はアクセスできません。

シンボルの型にかかわらず、動作は一致し、関数と変数は同様に動作します。

2.3.2 関数の属性を指定する

MPLAB C30 では、プログラムの中で使用する関数の一部を宣言します。したがって、コンパイラが関数コールを最適化し、コードをより注意深くチェックできます。

キーワード `__attribute__` により、宣言をする際に特殊な属性を規定できます。このキーワードに続き、二重括弧内に属性を指定します。以下の属性が、関数用として現状サポートされています。

- `address (addr)`
- `alias ("target")`
- `const`
- `deprecated`
- `far`
- `format (archetype, string-index, first-to-check)`
- `format_arg (string-index)`
- `interrupt [([save(list)] [, irq(irqid)] [, altirq(altirqid)] [, preprologue(asm)])]`
- `near`
- `no_instrument_function`
- `noload`
- `noreturn`
- `section ("section-name")`
- `shadow`
- `unused`
- `weak`

属性指定方法として `__` (2つのアンダースコア) をそれぞれのキーワードの前後に配置する (例えば `'shadow'` ではなく `'__shadow__'`) ことでも指定できます。この方法を用いて、同じ名前を持つ可能性のあるマクロを考慮しなくともヘッダファイルで属性を使用できます。

二重括弧内でコンマにより区切るか、一つの属性宣言に引続き他の属性宣言をすることで、宣言内で複数の属性を指定することができます。

address (addr)

`address` 属性は、関数の絶対アドレスを指定します。この属性を `section` 属性と同時に使用できません。`address` 属性が優先します。

```
void foo() __attribute__((address(0x100))) {  
    ...  
}
```

alias ("target")

`alias` 属性は、特定のシンボルの生成を宣言します。

この属性を使用すると、外部のリファレンスを参照するため、リンクフェーズで使用してください。

const

多くの関数は引数以外の値を検証せず、戻り値以外には影響を与えません。その場合、共通する副表記の省略や算術オペレータのようにループの最適化ができます。これらの関数は属性 `const` で宣言します。例えば、

```
int square (int) __attribute__ ((const int));
```

は、上記の仮定関数 `square` がプログラムが示すよりも少ない回数だけコールするほうが安全であることを示しています。

注意点として、ポインタ引数を持ち、ポイントされたデータを処理する関数は `const` 宣言できません。同様に、`non-const` 関数は通例 `const` 宣言できません。`void` の戻りタイプを持つ関数に `const` を適用する必要はありません。

deprecated

`deprecated` 属性についての情報は、[セクション 2.3.1 「変数属性の指定」](#) をご参照ください。

far

`far` 属性はコンパイラに対して、より効率のよい形式のコール命令を用いて関数がコールしないように通知します。

format (archetype, string-index, first-to-check)

`format` 属性は、関数が、フォーマット文字列に反していないかのタイプチェックを受ける `printf`、`scanf` もしくは `strftime` スタイル変数を取ることを規定します。例えば、以下の宣言を考えてみましょう。

```
extern int  
my_printf (void *my_object, const char *my_format, ...)  
    __attribute__ ((format (printf, 2, 3)));
```

コンパイラは、`my_printf` をコールするときの引数が、`printf` スタイルのフォーマット文字列関数 `my_format` に合っているかをチェックします。

パラメータ `archetype` はフォーマット文字列がどのように解釈されるかを決定し、`printf`、`scanf` もしくは `strftime` のいずれかを決定します。パラメータ `string-index` はどの引数がフォーマット文字列引数かを指定します (引数は 1 から始まり、左側より数えられます)。一方、`first-to-check` はフォーマット文字列と比較しチェックしようとする最初の引数の番号です。引数がチェックに使用できない場合 (`vprintf` などの場合)、3 番目のパラメータを 0 に指定します。この場合、コンパイラはフォーマット文字列の整合性のみをチェックします。

上記例では、フォーマット文字列 (`my_format`) は関数 `my_printf` の第二変数であり、チェックすべき変数は第三の変数から開始します。したがって、`format` 属性の正しいパラメータは 2 と 3 になります。

`format` 属性は、フォーマット文字列を引数とする独自関数を使用する場合、MPLAB C30 がこれらの関数へのコールをエラーとしてチェックできるようにします。コンパイラは、(`-Wformat` を用いて) ワーニングが必要な時は常に ANSI ライブラリ関数 `printf`、`fprintf`、`sprintf`、`scanf`、`fscanf`、`sscanf`、`strftime`、`vprintf`、`vfprintf`、`vsprintf` 用のフォーマットチェックを行います。従ってヘッダファイル `stdio.h` を修正する必要はありません。

format_arg (string-index)

format_arg 属性は、関数が printf もしくは scanf スタイルの引数を取り、修正をかけ (例えば、他の言語に変換する等)、printf もしくは scanf スタイルの関数に渡すことを規定します。例えば以下の宣言を考えてみましょう。

```
extern char *  
my_dgettext (char *my_domain, const char *my_format)  
    __attribute__ ((format_arg (2)));
```

これは結果を printf、scanf、strftime タイプ関数に渡す my_dgettext をコールする引数が、printf スタイルの文字列引数である my_format にあっているかをコンパイラにチェックさせます。

パラメータ string-index は、どの引数がフォーマット文字列引数であるかを規定します。(1 から開始)

format-arg 属性は、フォーマット文字列を修正する独自関数を作成したとき、MPLAB C30 が、独自関数をコールするオペランドを持つ printf、scanf もしくは strftime 関数コールをチェックできるようにします。

interrupt [([save(list)] [, irq(irqid)] [, altirq(altirqid)] [, preprologue(asm)])]

このオプションを使用して、指定した関数が割り込みハンドラであることを指示します。コンパイラは、この属性が存在する場合に割り込みハンドラでの使用に適した、関数の prologue および epilogue シーケンスを生成します。オプションパラメータ save は、関数のプロローグおよびエピローグでそれぞれ保存および復元される変数の一覧を指定します。オプションパラメータ irq および altirq は、使用する割り込みベクタテーブルの ID を指定します。オプションパラメータ preprologue は、コンパイラが生成するプロローグコードの前に挿入するアセンブリコードを指定します。例も含めた詳しい説明については、**第7章.「割り込み」**を参照してください。

near

near 属性はコンパイラに対して、より効率のよい形式のコール命令を用いて関数がコールされるよう通知します。

no_instrument_function

通常コマンドラインオプション -finstrument-function が付与されると、プロファイリング関数コールが、ユーザーによりコンパイルされた関数の出入口で生成されます。本属性を持つ関数はこの動作をしないようにします。

noload

noload 属性は空間を関数に割り当てますが、実際のコードはメモリにロードされません。この属性は、実行時にアプリケーションがシリアル EEPROM などから関数をメモリにロードするよう設計されている場合に有用です。

```
void bar() __attribute__ ((noload)) {  
    ...  
}
```

noreturn

abort や exit 等のいくつかの標準ライブラリ関数はリターンしません。MPLAB C30 はこれを自動的に認知します。プログラムではリターンしない関数を定義することがあります。noreturn を宣言することで、このことをコンパイラに通知することができます。例えば、以下のように記述します。

```
void fatal (int i) __attribute__ ((noreturn));

void
fatal (int i)
{
    /* Print error message. */
    exit (1);
}
```

noreturn キーワードは fatal が戻りのないことを前提としてコンパイラに通知します。そして、fatal の過去のリターンの有無に関わらず最適化し、コードを改良できます。また、初期化されていない変数による誤った警告を回避できます。

noreturn 関数には、void 以外の戻りタイプを使用しません。

section ("section-name")

通常コンパイラは、生成したコードを .text セクション内に配置します。しかし、追加のセクションや特殊なセクションに関数が必要となる場合があります。section 属性は関数が特定のセクションを指定します。例えば、次の宣言を参照してください。

```
extern void foobar (void)
__attribute__ ((section (".libtext")));
```

これは関数 foobar を .libtext セクションに配置します。

section 属性は address 属性と競合します。セクション名は無視され警告が発生します。

shadow

shadow 属性はコンパイラに対して、レジスタ待避を、ソフトスタックではなくシャドウレジスタを使用するよう指示します。この属性は通常 interrupt 属性と共に用いられます。

```
void __attribute__ ((interrupt, shadow)) _T1Interrupt (void);
```

unused

この属性は関数に付随し、関数を使用しない可能性があることを意味します。MPLAB C30 はこの関数に対して unused 関数警告を生成しません。

weak

weak 属性の詳細については、[セクション 2.3.1 「変数属性の指定」](#)をご参照ください。

2.3.3 インライン関数

関数 `inline` を宣言すると、MPLAB C30 が、関数コーラー用のコードにその関数コードを入れ込むように指示できます。これは通常、関数コールのオーバーヘッドを無くすことができプログラムの実行速度を上げることができます。さらに、実際の引数の値が定数である時は、既知の値をコンパイル時に単純化することにより、インライン関数のコードの一部を省略することができます。従ってコードサイズの影響は予測できません。機械語のサイズは、状況により、インライン関数とともに増減します。

注： 関数インライン化は関数の定義が明確 (プロトタイプのみではなく) な場合にのみ発生します。一つ以上のソースファイルに関数をインライン化するには、それぞれのソースファイルにインクルードするヘッダファイルに関数定義を置くこととなります。

関数 `inline` を宣言するには、以下のように、宣言文の中で `inline` キーワードを使用します。

```
inline int
inc (int *a)
{
    (*a)++;
}
```

(`-traditional` もしくは `-ansi` オプションの使用中は、`inline` ではなく `__inline__` を使用してください。) コマンドラインオプション `-finline-functions` を用いると、「十分簡潔」な関数インライン化ができます。コンパイラは、関数サイズの推定結果を基に、この方法で組込むにあたりどの関数が十分簡潔であるかを、自ら決定します。

注： `inline` キーワードは、`-finline` がある場合、または最適化が有効な場合のみ認識されます。

関数定義の中で、インライン置き換えには不適当な場合があります。これには、`varargs`、`alloca`、可変長データ、計算結果の `goto`、非ローカル `goto` の使用が含まれます。コマンドラインオプション `-winline` は、インラインとマークされた関数が展開されない場合にはワーニングを発生し、その理由も出力します。

MPLAB C30 構文では、`inline` キーワードは関数のリンクには影響を与えません。

関数が `inline` でかつ `static` である場合、すべての関数コールがコーラーに組み込まれ、関数のアドレスが使用されない場合、関数自身のアセンブラコードは参照されません。この場合、MPLAB C30 は、コマンドラインオプション `-fkeep-inline-functions` を指定しない限り、実際に関数のアセンブラコードを生成しません。また、コールによっては種々の理由により組み込まれません。特に、関数の定義より前に発生するコールや、定義の中再帰コールは組み込まれません。組み込まれないコールが存在する場合、関数は通常通りアセンブラコードにコンパイルされます。プログラムがその (関数) アドレスを参照する場合には通常通りに関数もコンパイルする必要があります。アドレスは `inline` 化されないため、関数が `static` であると宣言され、関数が関数の使用以前に定義されている場合のみ、コンパイラはインライン関数を省略します。

`inline` 関数が `static` でない場合、コンパイラは、別のソースコードからのコールが発生すると仮定します。グローバルシンボルはプログラム内で一度しか定義できないので、関数は他のソースファイル内では定義できず、したがって、そのコールは組み込まれません。つまり、非 `static` 関数は常に通常通りコンパイルされます。

関数定義で `inline` と `extern` を同時に指定した場合は、定義はインラインのみで使用されます。関数のアドレスを明確に参照しても、この場合は、関数はコンパイルされません。その場合のアドレスは、関数を宣言し定義しなかった場合と同様に、外部参照となります。

このように `inline` と `extern` の組合せは、マクロにも同様な影響を与えます。これらのキーワードと一緒にヘッダファイル内に関数定義を置き、ライブラリファイルには、別の定義のコピー (`inline` と `extern` の無いもの) を置きます。ヘッダファイル内の定義により、関数へのほとんどのコールはインライン化されます。残りは、ライブラリ内の 1 コピーを参照します。

2.3.4 レジスタ変数

MPLAB C30 はいくつかのグローバル変数に指定されたハードレジスタを使用できます。

注: レジスタ、とくに W0 レジスタの多用は、MPLAB C30 のコンパイル機能に影響する場合があります。

普通のレジスタ変数を割り当てるレジスタを指定することもできます。

- グローバルレジスタ変数はプログラム全体のレジスタを予約し、頻繁にアクセスする複数のグローバル変数を持つプログラム言語翻訳器のようなプログラムに最適です。
- 特定のレジスタ内のローカルレジスタ変数はレジスタを予約しません。コンパイラのデータフロー解析により、指定されたレジスタが使用中か、その他の用途に使用できるかを決定できます。ローカルレジスタ変数が未使用の場合そのレジスタ変数の内容は消去されます。ローカルレジスタ変数へのリファレンスは消去されるか、移動されるか、簡略化されます。

アセンブラ命令の出力を直接特定のレジスタに書き込む場合には、これらのローカル変数を拡張インラインアセンブルと共に使用すると便利です。(第 8 章、「アセンブリ言語と C モジュールの混用」を参照ください)。これは、インラインアセンブル記述内のオペランドに指定された制約に、指定レジスタが適合した場合に有効です。

2.3.4.1 グローバルレジスタ変数の定義

MPLAB C30 でグローバルレジスタ変数を以下のように定義できます。

```
register int *foo asm ("w8");
```

ここで、`w8` は使用するレジスタの名前です。ライブラリルーチンが影響を与えないよう関数コールで通常保存回復されるレジスタ (W8-W13) を選択してください。

あるレジスタでグローバルレジスタ変数を指定すると、現在のコンパイル内で、完全にそのレジスタを予約できます。レジスタは、現在のコンパイルでは関数のその他の目的には割り当てません。無効になってもこのレジスタの内容は消去されませんが、リファレンスは消去、移動もしくは簡略化されます。

割り込みハンドラもしくは一つ以上のスレッドコントロールからグローバルレジスタ変数をアクセスすることは推奨しません。特別にタスク用に再コンパイルしない場合には、システムライブラリルーチンがレジスタを他の目的で一時的に使用する場合があります。

グローバルレジスタ変数を使用する関数が、この変数の認知なし、すなわち、変数が宣言されていないソースファイル内で、コンパイルされた第三の関数 `lose` を経由してグローバル変数を使用する関数 `foo` を呼ばないでください。これは `lose` がレジスタを保存し、他の値を置く場合があるためです。例えば、グローバルレジスタ変数が `qsort` へ渡す比較関数の中で使用できることは期待できません、なぜなら、`qsort` はそのレジスタに別の値を収める可能性があるためです。この問題は `qsort` を同じグローバルレジスタ変数定義と一緒に再コンパイルすることで回避できます。

qsort もしくは、グローバルレジスタ変数を実際に使用しないその他のソースファイルを再コンパイルして、その他の目的で当該レジスタを使用しないよう認定するには、コンパイラにコマンドラインオプション `-ffixed-reg` を指定します。実際にグローバルレジスタ宣言をソースコードに追加する必要はありません。

グローバルレジスタ変数の値を変更する関数は、この変数無しにコンパイルされた関数からは、確実にコールできません、なぜなら、コーラーが戻り値として期待する値が破壊されている可能性があるためです。したがって、グローバルレジスタ変数を使用するプログラム部分への入り口となる関数は、コーラーに従属する値を明確に保存、回復する必要があります。

ライブラリ関数 `longjmp` は、`setjmp` で保持した値を、それぞれのグローバルレジスタ変数に回復します。

すべてのグローバルレジスタ変数宣言はすべての関数定義よりも先に実行する必要があります。もしそのような宣言が関数定義の後に現れると、レジスタは、先に定義された関数の中で、別の用途で使用される場合があります。

グローバルレジスタ変数は初期値を持ちません。実行ファイルはレジスタに初期値を与える手段を持たないためです。

2.3.4.2 ローカル変数用のレジスタ指定

指定されたレジスタでローカルレジスタ変数を以下のように定義できます。

```
register int *foo asm ("w8");
```

ここで、`w8` は使用されるレジスタの名前です。これはグローバルレジスタ変数を定義する構文と同じことに注意してください。但し、ローカル変数の場合は、関数の中に現れます。

レジスタ変数を定義しても予約できず、フロー制御により変数の値が有効でないと判定した場合には、別の用途に利用できます。この機能を使用すると、ある関数のコンパイルに必要なレジスタの数が少数に限定される場合があります。

このオプションは、指定レジスタ内に変数を割り当てるコードを、MPLAB C30 が常に生成することを保証するものではありません。このレジスタへの明示的な参照を `asm` ステートメントにコード化すべきではなく、またこの変数は常に参照されると期待してはいけません。

ローカルレジスタ変数に対する割り当ては、不使用と判断された時点で消去されます。ローカルレジスタ変数へのリファレンスは消去、移動もしくは簡略化されます。

2.3.5 複素数

MPLAB C30 は複素数データを扱うことができます。キーワード `__complex__` 用いることで、複素整数型、複素浮動小数点型を宣言できます。

例えば、`__complex__ float x` により、`x` の実部、虚部がともに `float` 型の変数であることを宣言できます。`__complex__ short int y` は、`y` の実部、虚部がともに `short int` 型の変数であることを宣言しています。

複素数データ型の定数を記述するには、添え字 `'i'` もしくは `'j'` (どちらか一つ; どちらも同等です。) を使用します。例えば、`2.5fi` は `__complex__ float` 型であり、`3i` は `__complex__ int` 型です。このような定数は純粋に虚数値ですが、実数定数を付加することであらゆる複素数値を指定できます。

複素数値の表記 *exp* の実数部を抽出するには、`__real__ exp` と記述します。同様に虚数部を抽出するには、`__imag__` を使用します。例えば、次のように記述します。

```
__complex__ float z;
float r;
float i;

r = __real__ z;
i = __imag__ z;
```

演算子 ‘~’ は、複素数型の値を扱う場合には、複素共役を生成します。

MPLAB C30 は複素自動変数を、非連続的に割り当てます。実数部分をレジスタに、虚数部分をスタックに（逆も可能）設定することも可能です。デバッグ情報フォーマットでは、このような非連続的な割り当てを表現することはできません。したがって、MPLAB C30 では非連続な複素変数を、非複素数型の別々な二つの変数として記述します。もし変数の実際の名前が *foo* ならば、二つの仮想変数は *foo\$real* と *foo\$imag* と定義されます。

2.3.6 倍長整数

MPLAB C30 は `long int` の 2 倍の長さを持つ整数型をサポートしています。符号付き整数は `long long int` と記述し、符号なし整数は、`unsigned long long int` と記述します。`long long int` 型の整数定数を作るには、整数に添え字 `LL` を付加します。`unsigned long long int`, 型の整数定数を作るには、整数に添え字 `ULL` を付加します。

これらのデータ型は他の整数型のように算術的に使用できます。また、加算、減算、ビットのブール演算はオープンコードですが、割り算、シフト演算はオープンコードではありません。オープンコードでない演算には MPLAB C30 で提供する特別ライブラリを使用します。

2.3.7 typedef による型参照

表現型を参照する別の方法としては、`typedef` キーワードがあります。このキーワードを使用する構文は `sizeof` に似ていますが、構成は、`typedef` のように動作します。

`typedef` に対する引数を記述するには二つの方法があります。表現で記述する方法と型で記述する方法です。表現で記述する例は以下の通りです。

```
typedef (x[0] (1))
```

これは、*x* が関数の配列であり、型は関数そのものの型になります。

変数の型名での記述例は以下の通りです。

```
typedef (int *)
```

ここでは、記述された型は `int` に対するポインタです。

もし ANSI C プログラムに含めるヘッダファイルを記述する場合、`typedef` ではなく、`__typedef__` と記述します。

`typedef` は宣言内、キャスト内、`sizeof`、`typedef` 内など、`typedef` 名が使用できるところであればいずれの場所でも使用できます。

- *x* が指すタイプで *y* を宣言します。
`typedef (*x) y;`
- 上記の値の配列として *y* を宣言します。
`typedef (*x) y[4];`
- 文字列へのポインタの配列として *y* を宣言します。
`typedef (typedef (char *) [4]) y;`
これは、従来の C 宣言と同等義です。
`char *y[4];`

typedef を用いた宣言の意味を確認し、なぜそれが記述するのに便利な方法なのか、マクロを書き直してみます。

```
#define pointer(T) typedef(T *)
#define array(T, N) typedef(T [N])
```

この宣言は以下のように書き直すことができます。

```
array (pointer (char), 4) y;
```

ここで、array (pointer (char), 4) は char に対する 4 つのポインタの配列です。

2.4 文の差異

本セクションでは、通常の ANSI C と MPLAB C30 で受け付けられる C との文の差異について述べます。文の差異は基本 GCC の部分であり、本章での説明は標準 GCC ドキュメントに基づいて、GCC の MPLAB C30 移植の特定の構文と記号に準じます。

- 値としてのラベル
- 省略されたオペランドを持った条件文
- Case 範囲

2.4.1 値としてのラベル

現在の関数（もしくは含まれる関数）内でオペレータ ‘&&’ とともに定義されるラベルのアドレスを入手できます。その値は void * のタイプを持っています。この値は定数でありその型の定数が有効であればいずれの場所でも使用できます。例えば、以下のように設定します。

```
void *ptr;
...
ptr = &&foo;
```

これらの値を使って直接ジャンプできます。これは計算 goto 文、goto *exp で実現できます。例えば、以下のように設定します。

```
goto *ptr;
```

タイプ void * をしたいかなる表現も可能です。

これらの定数を使用する方法の一つは、ジャンプテーブルとして提供されるスタティック配列を初期化することにあります。

```
static void *array[] = { &&foo, &&bar, &&hack };
```

すると、インデックス同様にラベルを選択できます。

```
goto *array[i];
```

注： 記述文が範囲内かはチェックしません。(C 言語での配列インデックスでもチェックしない)

ラベル値の配列は switch 文の目的に合わせて動作します。switch 文は簡潔になり配列には最適です。

ラベル値は、スレッドコード用の翻訳としても使用できます。その場合、翻訳関数内のラベルは、スレッドコードに保存され高速な分岐処理ができるようにします。

このメカニズムは別の関数内のコードにジャンプするという誤用する可能性があります。コンパイラでは回避できないため、ターゲットアドレスが現在の関数内で有効であることを十分注意して確認してください。

2.4.2 省略されたオペランドを持った条件文

条件記述の中の間オペランドは省略できる場合があります。第1オペランドがゼロでない場合、その値は条件記述の値になります。

したがって、例えば

```
x ? : y
```

は、 x がゼロでない場合には、 x の値を、そうでなければ y の値をとります。

この例は、以下の式と同様です。

```
x ? x : y
```

この例の場合、中間のオペランドを省略できるか否かは、特に問題ではありません。これが有益になるのは、第1オペランドが副作用を持つか、もしくは（マクロ変数の場合）持つ可能性のある場合です。オペランドを繰り返し中間の値とするのは二倍の作用をもたらすこととなります。中間オペランドを省略するのは、再計算の手間を省き、すでに計算された値を使用することです。

2.4.3 CASE 範囲

単一 CASE ラベル内で連続する値の範囲を、以下のように指定できます。

```
case low ... high:
```

これは、個別 CASE ラベルの適切な個数、すなわち、*low* から *high* までの一つ一つの整数値、と同じ効果を持ちます。

この特徴は ASCII 文字コードの範囲に対して特に有効です。

```
case 'A' ... 'Z':
```

注意: ..., の前後にスペースを入れてください。そうでないと、整数変数をともに使用されたときに誤って構文解析されます。例えば、以下のように記述します。

```
case 1 ... 5:
```

以下のように記述しないでください。

```
case 1...5:
```

2.5 表現の差異

このセクションでは、通常の ANSI C と、MPLAB C30 で受け入れられる C 言語の表現の差異を説明します。

2.5.1 バイナリ定数

0b または 0B（'b' または 'B' の前に数字 '0' が付きます）が付いたバイナリ数のシーケンスはバイナリ整数として解釈されます。バイナリ数は数字 '0' と '1' から成ります。たとえば、10 進数 255 は 0b11111111 と記述されます。

他の整数と同様に、バイナリ定数の末尾には、サイン無しを示す 'u' または 'U' が付きます。または、long であることを示す 'l' または 'L' が末尾に付くこともあります。接尾辞 'll' または 'LL' が付くと、long long バイナリ定数であることを示します。

第 3 章 . MPLAB C30 C コンパイラを使用する

3.1 序章

本章ではコマンドライン上での MPLAB C30 C コンパイラの使用方法について説明します。MPLAB® IDE と一緒に MPLAB C30 を使用する方法については、*dsPIC® DSC Language Tools Getting Started (DS70094)* をご参照ください。

3.2 ハイライト

本章で説明する内容は以下の通りです。

- 概要
- ファイル名規則
- オプション
- 環境変数
- 事前定義制約
- コマンドライン上の一つのファイルをコンパイルする
- コマンドライン上の複数のファイルをコンパイルする

3.3 概要

コンパイルドライバプログラム (`pic30-gcc`) は、C 言語とアセンブル言語とライブラリアーカイブをコンパイル、アセンブル、リンクします。ほとんどのコンパイラのコマンドラインオプションは、GCC ツールセットの実装方法と共通です。MPLAB C30 コンパイラ独自のオプションがいくつかあります。

コンパイラコマンドラインの基本形は以下の通りです。

```
pic30-gcc [options] files
```

<p>注: コマンドラインオプションとファイル名拡張子は、大文字と小文字を区別します。</p>
--

使用可能なオプションは **セクション 3.5 「オプション」** に記載されています。

下記例は、C ソースファイル `hello.c` をコンパイル、アセンブル、リンクして、絶対実行ファイル `hello.exe` を生成します。

```
pic30-gcc -o hello.exe hello.c
```

3.4 ファイル名規則

コンパイラドライバは以下のようなファイル拡張子を認識し、この拡張子は大文字と小文字を区別します。

表 3-1: ファイル名

拡張子	定義
<i>file.c</i>	プリプロセスする C ソースファイル
<i>file.h</i>	ヘッダーファイル (コンパイルもリンクも実行されない)
<i>file.i</i>	プリプロセスが不要な C ソースファイル
<i>file.o</i>	オブジェクトファイル
<i>file.p</i>	前処理後のアセンブリ言語ファイル
<i>file.s</i>	アセンブラコード
<i>file.S</i>	プリプロセスされるべきアセンブラコード
other	リンクに渡されるファイル

3.5 オプション

MPLAB C30 はコンパイルを制御するために多くのオプションを持っており、それらはすべて大文字と小文字を区別します。

- dsPIC DSC デバイスに特有のオプション
- 出力類の制御オプション
- C の方言を制御するオプション
- ワーニングとエラーの制御オプション
- デバッグ用オプション
- 最適化を制御するオプション
- プリプロセッサを制御するオプション
- アセンブラのオプション
- リンク用オプション
- ディレクトリ検索用オプション
- コード生成規則オプション

3.5.1 dsPIC DSC デバイスに特有のオプション

メモリモデルの詳細は、セクション 4.7「メモリモデル」を参照してください。

表 3-2: dsPIC® DSC デバイスに特有のオプション

オプション	定義
-mconst-in-code	定数を auto_psv 領域に置きます。コンパイラは PSV ウィンドウを用いてこれらの定数をアクセスできます。(これはデフォルトです)
-mconst-in-data	定数をデータメモリ領域に置きます。
-merrata=id[,id]*	このオプションは id で認識される dsPIC 特有のエラッタ対策を有効にします。時折生じる id 変更の有効な値で、変更の種類によっては不要です。「id of list」は現在サポートされているエラッタ識別子をそのエラッタの簡単な説明とともに示します。「id of all」は現在サポートされているエラッタ対策すべてを有効にします。
-mlarge-code	ラージコードモデルを使用してコンパイルします。呼び出された関数の局所性については問いません。 このオプションを選択すると、32k 以上のサイズのシングル関数はサポートされずアセンブリタイムエラーが発生します。関数内のすべての分岐がショート型であるためです。
-mlarge-data	ラージデータモデルを使用してコンパイルします。静的変数と外部変数の場所については問いません。
-mcpu=<it>target	このオプションは、特定のターゲットを選択します(また、この処理を実行する場合は、ターゲットをアセンブラおよびリンカに通信します)。このオプションは、一部の定義済み定数の設定方法に影響します。詳細については、セクション 3.7「事前定義制約」を参照してください。許可されるすべてのターゲットの一覧は、リリースに付属する README.TXT にあります。
-mpa ⁽¹⁾	プロシージャ抽象化の最適化を有効にします。ネストのレベルに制限はありません。
-mpa=n ⁽¹⁾	レベル n までのプロシージャ抽象化の最適化を有効にします。n がゼロの場合、最適化は無効化されます。n が 1 の場合、抽象化は書記第 1 レベルで許可され、ソースコード内の命令シーケンスはサブルーチンに抽象化されます。n が 2 の場合、第 2 レベルの抽象化が許可され、最初のレベルでサブルーチンに抽象化された命令は 1 レベル分深く抽象化されます。このパターンは n の値が 3 以上でも同様に続きます。正味の効果としては、サブルーチン呼び出しネストの深さが n の値を上限に制限されます。
-mno-pa ⁽¹⁾	プロシージャ抽象化の最適化を有効にしません。(これはデフォルトです)



注 1: プロシージャ抽象化はインラインの逆の動作をします。動作はコンパイル単位中の複数のサイトから共通コードシーケンスを抽出するようにデザインされており、抽出したシーケンスをコードの共通領域に配置します。このオプションは一般的には、生成されたコードのランタイムパフォーマンスには影響を与えませんが、コードのサイズを確実に節約できます。-mpa でコンパイルされたプログラムはデバッグが困難ですので、このオプションは COFF オブジェクトフォーマットを使用したデバッグの際に使用を推奨しません。
プロシージャ抽象化は、アセンブリファイル生成後にコンパイルの個別のフレーズとして呼び出されます。このフレーズは複数のコンパイル単位にわたる最適化は行いません。プロシージャ最適化フレーズが有効化されると、インラインアセンブリコードは有効なマシン命令のみに制限されます。無効なマシン命令または命令シーケンス、またはアセンブラ擬似命令 (セクションングディレクティブ、マクロ、インクルードファイルなど) は使用できませんし、これらを使用すると、プロシージャ抽象化フレーズはうまく動作せず、出力ファイルの生成が禁止されます。

表 3-2: dsPIC® DSC デバイスに特有のオプション (つづき)

オプション	定義
-mno-isr-warn	デフォルトでは、コンパイラは、認識された割り込みベクタ名に <code>__interrupt__</code> が付加されていない場合に警告を生成します。このオプションは、その機能をオフにします。
-momf= <i>omf</i>	コンパイラで使用する OMF (Object Module Format) を選択します。 <i>omf</i> は、次のいずれかになります: <code>coff</code> COFF オブジェクトファイルを生成します。 (これはデフォルトです) <code>elf</code> ELF オブジェクトファイルを生成します。 ELF オブジェクトに使用するデバッグフォーマットは DWARF 2.0 になります。
-msmall-code	スモールコードモデルを使用してコンパイルします。呼び出される関数は近接 (コーラの 32k ワード以内) と想定されます。 (これはデフォルトです)
-msmall-data	スモールデータモデルを使用してコンパイルします。すべての静的変数および外部変数はデータメモリ領域の 8 KB より下に配置される可能性があります。 (これはデフォルトです)
-msmall-scalar	-msmall-data のように、静的スケーラと外部スケーラはデータメモリ領域の 8 KB より下に配置されると想定されます。 (これはデフォルトです)
-mtext= <i>name</i>	-mtext= <i>name</i> を指定するとテキスト (プログラムコード) がデフォルトの <code>.text</code> セクションではなく、 <i>name</i> という名前のセクションに配置されます。= の前後に空白スペースは入れません。
-msmart-io [= <i>0</i> <i>1</i> <i>2</i>]	このオプションは <code>printf</code> 、 <code>scanf</code> 、に渡される <code>format</code> 文字列、およびそれらの <code>f</code> と <code>v</code> のバリエーションについて分析をします。非浮動小数点引数を使用すると、整数のみのライブラリ関数バリエーションに変換されます。 -msmart-io=0 はこのオプションを無効化し、-msmart-io=2 はコンパイラの検査を緩めにし関数呼び出しを変数または未知のフォーマットの引数に変換します。-msmart-io=1 はデフォルトで、明確な定数字のみを変換します。

注 1: プロシージャ抽象化はインラインの逆の動作をします。動作はコンパイル単位中の複数のサイトから共通コードシーケンスを抽出するようにデザインされており、抽出したシーケンスをコードの共通領域に配置します。このオプションは一般的には、生成されたコードのランタイムパフォーマンスには影響を与えませんが、コードのサイズを確実に節約できます。-mpa でコンパイルされたプログラムはデバッグが困難ですので、このオプションは COFF オブジェクトフォーマットを使用したデバッグの際に使用を推奨しません。

プロシージャ抽象化は、アセンブリファイル生成後にコンパイルの個別のフレーズとして呼び出されます。このフレーズは複数のコンパイル単位にわたる最適化は行いません。プロシージャ最適化フレーズが有効化されると、インラインアセンブリコードは有効なマシン命令のみに制限されます。無効なマシン命令または命令シーケンス、またはアセンブラ擬似命令 (セクションングディレクティブ、マクロ、インクルードファイルなど) は使用できませんし、これらを使用すると、プロシージャ抽象化フレーズはうまく動作せず、出力ファイルの生成が禁止されます。

3.5.2 出力類の制御オプション

表 3-3: 出力類の制御オプション

オプション	定義
-c	ソースコードをコンパイルもしくはアセンブルしますが、リンクはしません。デフォルトの出力ファイルの拡張子は .o です。
-E	プリプロセスの後、つまりコンパイルが実行される前に停止します。デフォルトの出力ファイルは stdout です。
-o <i>file</i>	<i>file</i> に出力をします。
-S	適切なコンパイルの後、つまりアセンブラ呼び出しの前に停止します。デフォルトの出力ファイル拡張子は .s です。
-v	コンパイルの各ステージで実行されたコマンドを印刷します。
-x	<p>-x オプションを使用して明示的に入力言語を指定できます。</p> <p><u>-x language</u></p> <p>後続の入力ファイルの言語を明示的に指定します（コンパイラにファイル名接尾語によって決まるデフォルト値を選択させるのではなく）。このオプションは次の -x オプションまで後続の入力ファイルすべてに適用されます。以下の値が MPLAB C30 でサポートされています。</p> <p>c c-header cpp-output assembler assembler-with-cpp</p> <p><u>-x none</u></p> <p>言語仕様をオフにし、後続のファイルがそのファイル名接尾辞に従って処理されるようにします。これはデフォルトの動作ですが、他の -x オプションがすでに使用されている場合に必要です。</p> <p>例： pic30-gcc -x assembler foo.asm bar.asm -x none main.c mabonga.s</p> <p>上記例では -x none オプションがない場合、コンパイラはすべての入力ファイルはアセンブラが処理すべきものと解釈します。</p>
--help	コマンドラインオプションの説明を印刷します。

3.5.3 C の方言を制御するオプション

表 3-4: C の方言制御オプション

オプション	定義
-ansi	ANSI 規格 C プログラムすべて (のみ) をサポート。
-aux-info filename	ヘッダーファイルも含め、コンパイル単位内の宣言および定義されたまたはそのいずれかを満たすすべての関数について、プロトタイプ宣言しているファイル名を出力します。このオプションは C 以外の言語では無視されます。宣言に加え、ファイルはコメントによって各宣言 (ソースファイルと行) が暗示的なものか、プロトタイプか、プロトタイプ以外かを示し (I と N は新規、O は既存で、行番号とコロンの後の最初にこれらの文字が付きます)、それが宣言によるものか、定義によるものかを示します (2 番目の文字が C または F になります)。関数定義の場合、宣言に続く引数の K&R スタイルリストがコメント内の宣言の後に付きます。
-ffreestanding	独立した環境でコンパイルが行なわれるよう指定します。これは -fno-builtin を意味します。独立した環境には、標準のライブラリが存在するとは限らず、プログラム起動が必ずしもメインにはなりません。最も顕著な例は OS カーネルです。これは -fno-hosted と同等です。
-fno-asm	asm, inline もしくは typeof をキーワードと認識しないため、コードの中でこれらの単語を識別子として使用できます。代わりに <code>__asm__</code> , <code>__inline__</code> および <code>__typeof__</code> をキーワードとして使用します。-ansi は暗黙の内に -fno-asm を指定します。
-fno-builtin -fno-builtin-function	<code>__builtin__</code> を接頭辞としない組み込み関数を認識しません。
-fsigned-char	型 char に符号を付け、signed char とします。(これはデフォルトです)
-fsigned-bitfields -funsigned-bitfields -fno-signed-bitfields -fno-unsigned-bitfields	このオプションは、宣言が符号付き、符号なしのどちらも指定しない場合、ビットフィールドを符号付き、符号なしのいずれにするかを制御します。デフォルトでは、-traditional が使用されない限り、このようなビットフィールドは符号付きになります。-traditional を使用する場合は常に符号なしになります。
-funsigned-char	型 char の符号をはずし、unsigned char とします。
-fwritable-strings	書き込み可能データセグメント内に文字列を保存し、その文字列が一意にならないようにします。

3.5.4 ワーニングとエラーの制御オプション

ワーニングは、本来はエラーではないが、エラーとなる可能性の高いことを示す、診断メッセージです。

-w で始まるオプションを使用して、多くの特別なワーニングをリクエストできます。例えば、-Wimplicit は、暗黙の宣言についてのワーニングをリクエストすると、特別なワーニングオプションも -Wno- で始まる否定形を持っており、ワーニングの出力を停止します。例えば、-Wno-implicit 等です。このマニュアルでは、二つの形の内のデフォルトではない一つのみを記述しています。

以下のオプションは、MPLAB C30 C コンパイラで生成されるワーニングの量と種類を制御します。

表 3-5: -Wall に含まれるワーニング/エラーオプション

オプション	定義
-fsyntax-only	コードの構文をチェックしますが、それ以上のことはしません。
-pedantic	厳密な ANSI C によって要求されるすべてのワーニングを発行し、禁止された拡張子を使用したプログラムを拒絶します。
-pedantic-errors	ワーニングよりもエラーを発行し、それ以外は、-pedantic と同様です。
-w	すべてのワーニングメッセージを禁止します。
-Wall	この表でリストされたすべての -w オプションを組み合わせたものです。これにより、マクロの結合内でも、ユーザによっては疑わしく見える構文で、容易に回避（もしくはワーニング出力が出ないように修正する）できる構文についてもワーニングが出るようになります。
-Wchar-subscripts	配列添え字に char 型を使っている場合にワーニングを出します。
-Wcomment -Wcomments	コメント開始シーケンスである /* が /* スタイルのコメントに現れた時、もしくは Backslash-Newline が // スタイルのコメント内に現れた時は必ずワーニングを発生します。
-Wdiv-by-zero	コンパイル時に分子がゼロでの整数除算が発生した場合にワーニングを発生します。ワーニングメッセージを禁止するには -Wno-div-by-zero を使用します。ゼロでの浮動小数点の除算はワーニングを発生しません。無限値や NaNs を得るのに理論的な方法であるためです。(これはデフォルトです。)
-Werror-implicit-function-declaration	関数が宣言以前に使用された場合にエラーを発生します。
-Wformat	printf や scanf に対するコールをチェックし、与えられた引数が指定されたフォーマット文に対して適切な型であることを確認します。
-Wimplicit	-Wimplicit-int と -Wimplicit-function-declaration の両方を指定することと等価です。
-Wimplicit-function-declaration	関数が宣言以前に使用された場合にワーニングを発生します。
-Wimplicit-int	宣言において型が指定されていない場合にワーニングを発生します。
-Wmain	main の型が疑わしい場合にワーニングを発生します。main は int 型の値を返し、ゼロ、2つもしくは3つの適切な型の引数をもつ外部結合の関数です。
-Wmissing-braces	集合体もしくはユニオンの初期化が完全に括弧でくくられていない時にワーニングを発生します。以下の例では、a についてのイニシャライザは完全に括弧でくくられていませんが、b については完全に括弧でくくられています。 int a[2][2] = { 0, 1, 2, 3 }; int b[2][2] = { { 0, 1 }, { 2, 3 } };

表 3-5: -Wall に含まれるワーニング/エラーオプション (つづき)

オプション	定義
-Wmultichar -Wno-multichar	複数文字の <i>character</i> 定数が使用時にワーニングを発生します。通常、そのような定数はタイプミスですが、このような定数は実装依存の値を持つため、移植性のあるコード内では使用しないでください。以下の例では複数文字の <i>character</i> 定数の使用例を示しています。 <pre>char xx(void) { return('xx'); }</pre>
-Wparentheses	ある文脈の中で括弧が省略されているとワーニングを発生します。たとえば、真値が期待されている文脈の中で代入がある場合や、演算子が入れ子になっていて、後から見るとその優先度についてしばしば混乱するような場合です。
-Wreturn-type	関数が、 <i>int</i> をデフォルトの戻り値の型として定義されている場合にワーニングを発生します。戻り値の型が <i>void</i> でない関数で、戻り値を持たない <i>return</i> 文がひとつでもあればワーニングを発生します。
-Wsequence-point	C 標準の中でシーケンスポイントに違反するような未定義の意味を持つコードに関してワーニングを発生します。 C 標準は、C プログラム内での表現をシーケンスポイントの観点から評価して順序を規定します。そのシーケンスポイントはプログラムの部分ごとに、シーケンスポイント以前に実行されるものと、以後に実行されるものとに分けて順序を表示しています。これらは、以下の場合に発生します。 full expression (larger expression の一部ではないもの) の評価の後、第一オペランド &&, , ? : もしくは、 (カンマ) オペレータ、の評価の後、関数がコールされる前 (但しその引数とコールされる関数を意味する表現の評価の後) などの場合です。シーケンスポイントルールで表現されたもの以外に、表現の sub expressions の評価の順番は規定できません。これらすべてのルールは全体的な順序というよりも、部分的な順序のみを記述しています。たとえば、もし二つの関数がそれらの間のシーケンスポイントのない一つの表現でコールされると、関数がコールされる順番は規定できません。 オブジェクトの値に対する修正が、シーケンスポイント間で、有効になった場合、これは規定できません。動作がこれに依存するようなプログラムでは、動作は不定となります。C 標準は、「前段と次のシーケンスポイント間で、オブジェクトは、表現の評価により一度だけストアされた値を修正できます。さらに、前値は、ストアされるべき値を決定するためにのみ読み出される」ということを規定しています。プログラムがこのルールを無視すると、このような実装の結果は予想できなくなります。 未定義の動作を持ったコードの例は、 <code>a = a++; a[n] = b[n++]</code> と <code>a[i++] = i;</code> です。より複雑な場合はこのオプションでは診断できません。また、間違えて正しいという結果を出す場合がありますが、一般的には、プログラム中で検出することが有効とされています。
-Wswitch	<i>switch</i> 文が列挙型のインデックスを持つ場合、その列挙型により名前付けされた定数に対応する <i>case</i> が不足している場合に常に警告を發します。(default のラベルが存在する場合、この警告は発生しません) 同様に、このオプションの使用中に列挙型の範囲外の <i>case</i> ラベルがあると、警告が発生します。

MPLAB C30 C コンパイラを使用する

表 3-5: -Wall に含まれるワーニング/エラーオプション(つづき)

オプション	定義
-Wsystem-headers	システムヘッダーファイル内に構文が見つかった場合、警告メッセージが発生します。システムヘッダーファイルからの警告は通常、実際的な問題を意味していないと想定され、コンパイラ出力が難解になるだけなので、出力しないようにします。本コマンドラインオプションは、MPLAB C30 にシステムヘッダーからの警告をユーザーコード内で発生したものと同じように除外するよう指示します。ただし、このオプションと同時に -Wall を使用してもシステムヘッダー内の未知の pragma に対して警告は発生しません。そのため、-Wunknown-pragma も同時に使用する必要があります。
-Wtrigraphs	三重字表記があると警告が発生します (有効化されていると想定)。
-Wuninitialized	初期化なしに自動変数が使用されると、警告が発生します。この警告は、最適化の際にのみ評価されるデータフロー情報を必要とし、最適化が有効な場合にのみ生成可能で、レジスタ割り当て候補の変数に対してのみ警告が発生します。したがって、volatile として宣言された変数やアドレスが取得されている変数、サイズが 1、2、4、8 バイト以外の変数に対しては警告は発生しません。また、構造体、共用体、配列に対しては、レジスタ化されているものであっても警告は発生しません。値を計算するためだけに使用される変数に対しては、その計算結果自体が使用されなければ警告が発生しない場合があります。そのような計算は警告が印刷される前にデータフロー分析により削除される場合があるためです。
-Wunknown-pragma	MPLAB C30 が理解不能な #pragma ディレクティブがある場合に警告が発生します。このコマンドラインオプションを使用すると、システムヘッダーファイル内の未知の pragma に対して警告が発生します。ただし、-Wall コマンドラインオプションのみ有効化されている場合は警告は発生しません。
-Wunused	宣言以外に変数が使用されない場合、関数が静的に宣言されながら定義されていない場合、ラベルが宣言されて使用されない場合、ステートメントが明示的に使用されない結果を計算する場合、必ず警告が発生します。使用されていない関数パラメータに関する警告を得るには、-W と -Wunused の両方を指定する必要があります。void のための表現で式をキャストすると、その表現式に対する警告は抑制されます。同様に、unused 属性は、使用されていない変数、パラメータ、ラベルに対する警告を抑制します。
-Wunused-function	静的関数が宣言されながら、定義されていない場合、または非インライン静的関数が使用されていない場合に必ず警告が発生します。
-Wunused-label	ラベルが宣言されながら使用されていない場合に警告が発生します。この警告を抑制するには、unused 属性を使用します (セクション 2.3.1 「変数属性の指定」参照)。
-Wunused-parameter	関数パラメータが宣言以外に使用されていない場合、必ず警告が発生します。この警告を抑制するには、unused 属性を使用します (セクション 2.3.1 「変数属性の指定」参照)。
-Wunused-variable	ローカル変数または非定数の静的変数が宣言以外に使用されていない場合に必ず警告が発生します。この警告を抑制するには、unused 属性を使用します (セクション 2.3.1 「変数属性の指定」参照)。
-Wunused-value	ステートメントが明示的に使用されていない結果を計算した場合に必ず警告が発生します。この警告を抑制するには、void の表現をキャストします。

以下の `-w` オプションは、`-Wall` には含まれません。中には、ユーザが通常疑問を抱かない、しかし場合によっては確認したい構文について警告を発生します。構文に関する警告には、回避が必要もしくは困難な場合があり、警告を制限するようコードを容易に修正することはできません。

表 3-6: `-Wall` に含まれないワーニング / エラーオプション

オプション	定義
<code>-w</code>	<p>以下のイベントに対する警告メッセージを印刷します。</p> <ul style="list-style-type: none"> <code>volatile</code> 宣言されていない自動変数が、<code>longjmp</code> をコールすることで変更される場合があります。これらの警告は最適化コンパイル時のみ可能です。コンパイラは <code>setjmp</code> へのコールのみ観察しています。どこで <code>longjmp</code> がコールされるかはわかりません。実際、シグナルハンドラはコード中どのポイントでもコールできます。その結果、<code>longjmp</code> は問題が発生する場所でコールされることは実際には発生せず、事実上問題ない場合にも警告が発生する場合があります。 関数は、戻り値を持って復帰することも、戻り値を持たずに復帰することもできます。関数本体が終端に達することは、戻り値を持たずに復帰するとみなされます。 式文もしくはコンマ式の左側の部分が全く副作用を持たず、警告を制限するには使われない式を <code>void</code> でキャストします。例えば <code>x[i, j]</code> のような式は警告を発生させますが、<code>x[(void) i, j]</code> では発生しません。 符号無し値がゼロと < もしくは <code><=</code> で比較されている場合です。 <code>x<=y<=z</code> のような比較が行われている場合です。これは <code>(x<=y ? 1 : 0) <= z</code> と同等であり、通常の算術記述とは異なる解釈になります。 <code>static</code> のような記憶クラス指定子は宣言の中で最初に記述されません。C 標準では、この方法は使用されていません。 <code>-Wall</code> もしくは <code>-Wunused</code> が使用されると、未使用関数に関する警告が発生します。 符号付き値が符号無し値に変換されると、符号付き、符号無しの間での比較は誤った結果を生む可能性があります。(ただし、<code>-Wno-sign-compare</code> が合わせて指定されると、警告は発生しません。) 集合体の初期化を囲む括弧が部分的に記述されている場合です。例えば、以下のコードは、<code>x.h</code> のイニシャライザの前後に大括弧がないので、その警告を発生します。 <pre>struct s { int f, g; }; struct t { struct s h; int i; }; struct t x = { 1, 2, 3 };</pre> 集合体はイニシャライザを持ち、すべてのメンバを初期化していません。例えば、以下のコードは、<code>x.h</code> が暗示的にゼロに初期化されるので、その警告を発生します。 <pre>struct s { int f, g, h; }; struct s x = { 3, 4 };</pre>
<code>-Waggregate-return</code>	構造体や共用体を返す関数が定義されるかコールされた場合に警告を発生します。
<code>-Wbad-function-cast</code>	適合しない型に対して関数コールがキャストされた場合に警告を発生します。例えば、 <code>int foof() *</code> にキャストされた時に警告を発生します。

MPLAB C30 C コンパイラを使用する

表 3-6: -Wall に含まれないワーニング / エラーオプション (つづき)

オプション	定義
-Wcast-align	ターゲットに必要な配列が増加するように、ポインタがキャストされた場合に警告が発生します。例えば、char * に対して、int * にキャストされた場合に警告が発生します。
-Wcast-qual	ポインタの指す型から型修飾子を削除するようにポインタがキャストされた場合に必ず警告が発生します。例えば、const char * に対して、通常の char * にキャストされた場合に警告が発生します。
-Wconversion	プロトタイプが無い状態で同じ引数に行ったものとは別のタイプ変換を、プロトタイプが行った場合に警告が発生します。これには固定小数点から浮動小数点への変換、もしくはその逆の変換を含みます。また、デフォルトの拡張と同じ場合以外の固定小数点引数の幅もしくは符号変更も含みます。さらに、負の整数定数式が暗示的に符号なしタイプに変換されると、警告が発生します。例えば、x が符号なしの場合、代入 x = -1 に関して警告が発生します。(unsigned) -1 のような明確にキャストされている場合は警告が発生しません。
-Werror	すべての警告をエラーにします。
-Winline	関数が inline 宣言されるか、もしくはそれが -finline-functions オプションとされていて、その関数が inline 化できなかった場合、警告が発生します。
-Wlarger-than-len	len バイト以上のオブジェクトが定義された場合に必ず警告が発生します。
-Wlong-long -Wno-long-long	long long 型が使用されたときに警告が発生します。これはデフォルトです。警告を禁止するには -Wno-long-long を使用します。-Wlong-long と -Wno-long-long フラグは、-pedantic フラグが使用された場合に考慮されます。
-Wmissing-declarations	グローバル関数が事前の宣言なしに定義された時に警告が発生します。定義自体がプロトタイプを生成しても警告が発生します。
-Wmissing-format-attribute	-Wformat が有効になると、フォーマット属性用の候補になる可能性の高い関数について警告が発生します。これらは可能性のある候補のみであって、絶対候補ではありません。このオプションは -Wformat が有効にならない限り有効になりません。
-Wmissing-noreturn	noreturn 属性の候補になる可能性の高い関数についての警告が発生します。これらは可能性のある候補であって、絶対的な候補ではありません。noreturn 属性を追加する前に、十分に注意して関数が実際に復帰しないことをマニュアルで検証する必要があります。確認しない場合、コード生成上の微妙なバグが入る場合があります。
-Wmissing-prototypes	グローバル関数が事前のプロトタイプ宣言無しに定義された時に警告が発生します。この警告は、定義自体がプロトタイプを生成しても発生します。(このオプションは、ヘッダファイル内で宣言されなかったグローバル関数を検出するために使用されます。)
-Wnested-externs	extern 宣言が関数内で見つかった場合に警告が発生します。
-Wno-deprecated-declarations	deprecated マークされた関数、変数、型の使用については警告が発生しません。
-Wpadded	パディングが、構造体の要素か構造体全体を揃えるためにパディングが行われた場合に警告が発生します。

表 3-6: -Wall に含まれないワーニング / エラーオプション (つづき)

オプション	定義
-Wpointer-arith	関数型のサイズもしくは void のサイズに依存するものに関して警告を発生します。MPLAB C30 は、計算するときの便宜上、void * ポインタと関数に対するポインタ型に 1 のサイズを割り当てます。
-Wredundant-decls	複数宣言が有効で、何も変化させない場合でも、同じスコープ内で一度以上宣言されたら警告を発生します。
-Wshadow	ローカル変数が別のローカル変数を隠す時に警告を発生します。
-Wsign-compare -Wno-sign-compare	符号付値が符号無しに変換された時に、符号付 / 符号無し値間の比較結果が、正しくない結果を生じた時に警告を発生します。この警告は -W; でも有効になります。この警告なしに、他の -W の警告が必要なときは、-W -Wno-sign-compare を使用します。
-Wstrict-prototypes	引数の型を指定しない状態で、関数が宣言もしくは定義された場合に警告を発生します。(旧式の関数定義では、引数の型を指定する宣言が前にあれば、警告を発生することなく、許可されます。)
-Wtraditional	伝統的な C と ANSI C の中での動作が異なる構文について警告を発生します。 <ul style="list-style-type: none"> マクロ本体の中のストリング定数内に現れるマクロ引数。これらは、伝統的な C 内では引数に置き換わりませんが、ANSI C 内では定数の一部となります。 あるブロックで外部宣言された関数でブロックの終端より後ろに使用されるものです。 long 型のオペランドを持つスイッチステートメントです。 非静的な関数宣言が静的な関数宣言に続いているものです。この構成はいくつかの伝統的な C コンパイラでは受け入れられません。
-Wundef	未定義の識別子が #if ディレクティブ内で評価された場合に警告を発生します。
-Wunreachable-code	実行されない可能性のあるコードを、コンパイラが検出した場合に警告を発生します。これはこのオプションが、警告を受けたラインの一部が実行される状況になっても警告を発生する場合があります。したがって、明らかに到達しないコードを削除するには注意が必要です。例えば、関数がインライン化された時に、一つだけのインライン化された関数のコピー内で到達しないことを、警告が意味する場合もあります。
-Wwrite-strings	non-const char * ポインタへのアドレスのコピーにより警告が発生するように、ストリング定数に const char[length] 型を与えます。これらの警告から、ストリング定数へ書き込みしようとするコードを、コンパイル時に見つけられますが、宣言やプロトタイプ内の定数を使用する場合に十分注意して使用し、そうでなければ不要です。-Wall がこれらの警告の要求を催促しないのはこのためです。

3.5.5 デバッグ用オプション

表 3-7: デバッグ用オプション

オプション	定義
-g	<p>デバッグ情報を生成します。</p> <p>MPLAB[®] C30 は -g オプションを -O オプションと一緒にサポートし、最適化されたコードをデバッグするよう設定できます。コードの最適化によって、予期しない結果がでる場合があります。</p> <ul style="list-style-type: none">- 宣言した変数の中には全く存在しない可能性もあります。- 制御フローは予期しない場所へ移動される場合があります。- 構文によっては、定数の結果を計算するか、もしくはそれらの値はすでに入手済みで、実行されない場合があります。- 構文によっては、ループ外に移動し、別の場所で実行される場合があります。 <p>以上の条件を満たしていても、最適化された出力のデバッグが可能です。このため、バグの可能性の高いプログラムを最適化適用することも可能です。</p>
-Q	<p>コンパイル時にそれぞれの関数名をコンパイラが印刷し、コンパイルが終了する時にそれぞれのパスについての統計値を印刷します。</p>
-save-temps	<p>中間ファイルを保存します。それらを、現ディレクトリに置き、ソースファイルに基づいて名前をつけます。したがって、'foo.c' を '-c -save-temps' でコンパイルすると、以下のようなファイルを生成します。</p> <ul style="list-style-type: none">'foo.i' (事前処理されたファイル)'foo.p' (事前手続き上で抽象化されたアセンブリ言語ファイル)'foo.s' (アセンブリ言語ファイル)'foo.o' (オブジェクトファイル)

3.5.6 最適化を制御するオプション

表 3-8: 一般的な最適化オプション

オプション	定義
-O0	最適化しません。(これがデフォルトです。) 最適化しない場合コンパイラのゴールは、コンパイルのコストを低減し、デバッグが期待された結果を生成させることとなります。ステートメントは独立しています。ステートメント間のブレークポイントでプログラムを停止させると、どんな変数に対しても新しい値を代入したりもしくは関数内の他のステートメント移るようプログラムカウンタを変更したりできます。そしてソースコードから期待する結果そのものが得られます。コンパイラはレジスタ内に register 宣言された変数を割り当てのみです。
-O -O1	最適化します。最適化コンパイルは少し時間がかかり、大きい関数に対してはより多くのホストメモリを使用します。-O をつけると、コンパイラはコードサイズと実行時間を減らそうとします。 -O が設定されるとコンパイラは -fthread-jumps と -fdefer-pop をオンします。また -fomit-frame-pointer をオンします。
-O2	より最適化をかけます。MPLAB® C30 はスピードとサイズのトレードオフを伴わないほとんどすべてのサポートされた最適化を実行します。-O2 は、ループ展開や (-funroll-loops) と関数インライン化 (-finline-functions) と厳密なエイリアス最適化 (-fstrict-aliasing) を除くすべての最適化オプションをオンにします。また、メモリオペランドの強制コピー (-fforce-mem) と、フレームポインタ削除 (-fomit-frame-pointer) をオンにします。-O に比べて、このオプションはコンパイルに要する時間を増加させ、生成されたコードのパフォーマンスを向上させます。
-O3	さらに最適化をかけます。-O3 は、-O2 で規定されるすべての最適化を有効にし、さらに inline-functions オプションもオンします。
-Os	サイズの最適化をします。-Os は、コードサイズを増加させない、すべての -O2 最適化を有効にします。これ以外にコードサイズを低減させるように設計された最適化も実行します。



MPLAB C30 C コンパイラを使用する

以下のオプションは特定の最適化を制御します。-O2 オプションは -funroll-loops, -funroll-all-loops と -fstrict-aliasing を除いて、これらのすべての最適化をオンします。

以下のフラグは、最適化の微調整「fine-tuning」が必要となる場合に使用できます。

表 3-9: 特別な最適化オプション








オプション	定義
 -falign-functions -falign-functions= <i>n</i>	<p><i>n</i> より大きい次の 2 のべき乗境界に、関数の開始を揃え、<i>n</i> バイトまでスキップアップします。例えば、-falign-functions=32 は次の 32 バイト領域に関数を揃えます。ただし、-falign-functions=24 は、23 バイト以下をスキップすることにより可能な場合のみ、次の 32 バイトに揃えます。</p> <p>-fno-align-functions と -falign-functions=1 は同じで、関数が整列されないことを意味します。アセンブラは、<i>n</i> が 2 のべき乗の時にのみこのフラグをサポートしますので、<i>n</i> は切り上げられます。<i>n</i> を指定しない場合は、マシンに依存するデフォルト値を使用します。</p>
 -falign-labels -falign-labels= <i>n</i>	<p>すべての分岐ターゲットを、-falign-functions のように、<i>n</i> バイトまでスキップして、2 のべき乗に向けて境界に揃えます。このオプションを使用すると、コード実行は遅くなる場合があります。分岐ターゲットが通常のコードフロー内に到達するまでにダミーオペレーションを挿入するためです。</p> <p>-falign-loops もしくは -falign-jumps が適用されこの値より大きい場合、それらの値が、代わりに使用されます。<i>n</i> を指定しない場合は、マシンに依存するデフォルト値 (1 になる場合がほとんど) を使用しますが、これは整列をしないことを意味します。</p>
 -falign-loops -falign-loops= <i>n</i>	<p>ループを、-falign-functions のように、<i>n</i> バイトまでスキップして、2 のべき乗に対して境界に揃えます。ループが複数実行され、ダミーオペレーション実行の代わりとなります。<i>n</i> を指定しない場合は、マシンに依存するデフォルト値を使用します。</p>
 -fcaller-saves	<p>関数コールで上書きされるレジスタがある場合、コードに関して、レジスタを保存、回復するための命令を追加します。その場合の割り当ては、それ以外で生成されるコードに比べ、良いコードとなる場合のみ実行されます。</p>
 -fcse-follow-jumps	<p>共通部分式を除去する際に、ジャンプ命令のジャンプ先が、そのジャンプ命令以外のパスからは使用されない場合、そのジャンプ命令のジャンプ先を調べます。例えば、CSE が else 文を伴った if ステートメントを見つけた場合、CSE は条件が偽である場合のジャンプを追跡します。</p>
 -fcse-skip-blocks	<p>これは -fcse-follow-jumps と似ていますが、CSE をブロック条件付きでスキップするジャンプを CSE に追跡させません。CSE が単純 if 文で else ステートメントがない場合、-fcse-skip-blocks は CSE を、if の本体を飛び越すジャンプを追跡させます。</p>
 -fexpensive-optimizations	<p>比較的成本のかかる一部最適化を行います。</p>
 -ffunction-sections -fdata-sections	<p>それぞれの関数もしくはデータ項目を出力ファイル自身のセクションに置きます。関数の名前もしくはデータ項目の名前は出力ファイルのセクションの名前を決定します。大きな効果がある場合にのみこれらのオプションを使用します。これらのオプションを指定する時にはアセンブラとリンカは大きなオブジェクトと実行ファイルを生成し、速度は遅くなります。</p>

表 3-9: 特別な最適化オプション (つづき)



オプション	定義
-fgcse	グローバル共通部分式除去のパスを実行します。このパスはグローバル定数とコピー伝播も実行します。
-fgcse-lm	-fgcse-lm が有効になると、グローバル共通部分式除去が、自分自身へのストアでのみ消されるロードを移動させようとします。これにより、ロードストアシーケンスを含むループがループの外のロードに変更され、ループ内はコピーストアに変更されます。
-fgcse-sm	-fgcse-sm が有効になると、ストアモーションパスが、グローバル共通部分式除去の後に実行されます。このパスは、ストアをループ外に移動させようとします。 -fgcse-lm と一緒に使用されると、ロードストアシーケンスを含むループは、ロードはループの前に、ストアはループの後に変更されます。
-fmove-all-movables	ループ内のすべての不変の計算はループ外に移動されます。
-fno-defer-pop	関数が戻った後、直ちに関数コールに対して引数を POP します。コンパイラは通常、引数をいくつかの関数コール用のスタックに蓄積し、一度にすべて POP します。
-fno-peephole -fno-peephole2	マシン特有のピープホール最適化を無効にします。ピープホール最適化はコンパイル中の様々な場面で発生します。 -fno-peephole はマシン命令に関するピープホール最適化を無効にし、-fno-peephole2 は高度なピープホール最適化を無効にします。ピープホールを完全に無効化するには、これら両方を使用します。
-foptimize-register-move -fregmove	レジスタ結合を最大化するために move 命令内および、単純命令のオペランドのレジスタ番号を再割当します。 -fregmove と -foptimize-register-moves は同じ最適化です。
-freduce-all-givs	ループ内のすべての汎用帰納変数の能力を低減させます。これらのオプションによりコードが良くなる場合、悪化する場合があります。結果はソースコード内のループの構造に高く依存します。
-frename-registers	レジスタ割り当て後に過剰なレジスタを利用することで、予定されたコード内でできなかったところを再度割り当てようとします。この最適化は多くのレジスタを持つプロセッサに最適ですが、デバッグができなくなる場合があります。変数がもとのレジスタ「home register」にとどまっていなかったためです。
-frerun-cse-after-loop	ループ最適化後、共通部分式除去を再実行します。
-frerun-loop-opt	ループ最適化を 2 度実行します。
-fschedule-insns	dsPIC® DSC の Read-After-Write ストール (詳しくは <i>dsPIC30F Family Reference Manual</i> を参照してください) を無くすために命令を並べ直します。特に、コードサイズに影響を与えずにパフォーマンスを向上させます。
-fschedule-insns2	-fschedule-insns と同様ですが、レジスタ割り当てが完了した後に命令スケジュールの追加実行を要求します。
-fstrength-reduce	ループの能力低減と繰り返し変数の除去の最適化を実行します。

表 3-9: 特別な最適化オプション (つづき)



オプション	定義
-fstrict-aliasing	<p>コンパイルされる言語に適用できる最も厳密なエイリアシング規則を、コンパイラに適用します。C 言語については、これが、型表現を基にした最適化を活性化します。特に、ある型のオブジェクトが、ほとんど同じ型でないならば、別の型のオブジェクトとして同じアドレスのところには無いものと仮定されます。例えば、unsigned int は int の別名になり得ますが、void* もしくは double にはなりません。文字型は別の型になりえます。</p> <p>以下のようなコードには特に注意してください。</p> <pre>union a_union { int i; double d; }; int f() { union a_union t; t.d = 3.0; return t.i; }</pre> <p>最も最近書き込まれた (「type-punning」と呼ばれる) ものとは異なったユニオンメンバから読み出すことはよくあることです。メモリがユニオン型でアクセスされる場合、-fstrict-aliasing を持った type-punning は許可されません。したがって、上記コードは期待通りに動作し、次のようになります。</p> <pre>int f() { a_union t; int* ip; t.d = 3.0; ip = &t.i; return *ip; }</pre>
-fthread-jumps	<p>比較によるジャンプの分岐先において、最初の比較に含まれるような別の比較があるかどうかをチェックするように最適化が実行されます。もしそうなると、条件の真か偽が判った時に、最初の分岐は 2 番目の分岐もしくは直後の分岐先のうち、どちらかの分岐先に再設定されます。</p>
-funroll-loops	<p>ループ解除の最適化を実行します。これは、繰り返し回数がコンパイル時もしくは実行時に決定されるループのためにのみ使用され、-funroll-loops は -fstrength-reduce と -frerun-cse-after-loop の両方を含みます。</p>
-funroll-all-loops	<p>ループ解除の最適化を実行します。すべてのループに適用され、通常はプログラムの実行速度が遅くなります。-funroll-all-loops は -fstrength-reduce と -frerun-cse-after-loop を含みます。</p>

形式 `-fflag` のオプションはマシンに依存しないフラグを指定します。ほとんどのフラグは正および負の形式を持ち、`-ffoo` の負形式は `-fno-foo` です。以下の表では、一つの形式のみリストアップしています。(デフォルトではない形式)

表 3-10: マシンに依存しない最適化オプション

オプション	定義
<code>-fforce-mem</code>	メモリオペランドを、算術計算する前にレジスタにコピーします。この手順は、すべてのメモリアドレスを共通部分式にし、より良いコードを生成します。それらが、共通部分式では無い場合には、命令の組合せで、個別にレジスタに値をロードしないでください。 <code>-O2</code> オプションはこのオプションをオンします。
<code>-finline-functions</code>	すべての単純関数をそのコーラーにインライン展開します。コンパイラは、どの関数がこのような方法で組込むのに十分単純であるかを決定します。与えられた関数すべてのコールに展開され、関数が <code>static</code> であると宣言された場合、関数は通常は本来姿のアセンブラコードとして出力されません。
<code>-finline-limit=n</code>	デフォルトでは、MPLAB C30 はインラインされる関数のサイズを制限します。このフラグはインラインと (<code>inline</code> キーワードで) 明確にマークされた関数に対する制限をコントロールします。 <code>n</code> はインライン化できる関数のサイズを、仮想命令数を単位として示します。(パラメータ処理の部分は含まれません)。 <code>n</code> のデフォルト値は <code>10000</code> です。この値を増加すると、コンパイル時間とメモリ使用量は増えますが、インライン化されるコードを増やすことができます。 減少させるとコンパイル時間を短縮し、インライン化されたコードを少なくしますが、プログラムの速度は遅くなる可能性があります。このオプションは特にインライン化を多く使用するプログラムに有効です。 注: 仮想命令は、この特別なコンテキストでは、関数のサイズの抽象的な尺度を意味します。アセンブリ命令のカウント値を表すものではなく、その正確な意味はコンパイラのリリースごとに変更される可能性があります。
<code>-fkeep-inline-functions</code>	与えられた関数に対するすべてのコールがインライン展開されて、関数が <code>static</code> であると宣言されても、別々の関数のコール可能な実行バージョンを出力します。このスイッチは、 <code>extern</code> のインライン関数には影響ありません。
<code>-fkeep-static-consts</code>	変数が参照されていなくても、最適化がオンされていない場合に <code>static-consts</code> と宣言された変数を出力します。MPLAB C30 はこのオプションをデフォルトで有効にしています。最適化がオン/オフにかかわらず、変数が参照されたかどうかをコンパイラでチェックするには、 <code>-fno-keep-static-consts</code> オプションを使用します。
<code>-fno-function-cse</code>	関数アドレスをレジスタに置かず、ある決まった関数をコールするそれぞれの命令に関数のアドレスを明示的に含ませます。このオプションを使用するとコード効率は下がりますが、アセンブラ出力を変更する方法の中には、このオプションを使用しないと、最適化の際に問題が発生する場合があります。



MPLAB C30 C コンパイラを使用する

表 3-10: マシンに依存しない最適化オプション (つづき)

オプション	定義
-fno-inline	inline キーワードを無視します。通常このオプションは、コンパイラが inline の関数を拡大するのを防ぐのに使用されます。最適化が有効でない場合、関数は全くインライン展開されません。
-fomit-frame-pointer	フレームポインタが不要な関数のレジスタ内のフレームポインタを保持しません。これにより、命令がフレームポインタを保存、セットアップ、回復することを回避できます。また、複数の関数で余剰レジスタを利用できるようにします。
-foptimize-sibling-calls	シブリングとテイル再帰コールを最適化します。

3.5.7 プリプロセッサを制御するオプション

表 3-11: プリプロセッサオプション

オプション	定義
-Aquestion (answer)	#if #question(answer) のようなプリプロセッシング条件でテストした場合、質問 question に対する回答 answer を診断します。-A- は通常ターゲットマシンを記述する標準の診断を無効にします。 例えば、メインプログラムの関数プロトタイプは以下のように宣言されます。 #if #environ(freestanding) int main(void); #else int main(int argc, char *argv[]); #endif -A コマンドラインは二つのプロトタイプから選択するために使用されます。例えば、二つのうちの最初の方を選択するには、以下のようなコマンドラインオプションが使用されます。 -Aenviron(freestanding)
-A -predicate =answer	述部 predicate と回答 answer の診断をキャンセルします。
-A predicate =answer	述部 predicate と回答 answer の診断を発行します。この形式は、依然としてサポートされている旧式 -A predicate (answer) よりも使用されている場合が多く、シェル特別文字を使用しないことが理由です。
-C	コメントを廃棄しないようプリプロセッサに指示します。-E オプションと共に使用します。
-dD	プリプロセッサに、マクロの正しい順序ですべてのマクロ定義を出力に渡すようプリプロセッサに指示します。
-Dmacro	string l を持ったマクロ macro を定義します。
-Dmacro=defn	マクロ macro を defn と定義します。コマンドラインの -D のすべてのインスタンスを、-U オプションの以前に処理します。
-dM	プリプロセッシングの終了時に有効になるマクロ定義のリストのみを出力するように、プリプロセッサに指示します。-E オプションと一緒に使用されます。
-dN	マクロ引数とコンテンツが省略されることを除いて、-dD と同様です。#define name のみが出力に含まれます。
-fno-show-column	診断時に、コラム番号を印刷しません。もし診断結果が、dejagnu のようにコラム番号を理解できないプログラムによりスキャンされる場合には必要な場合があります。

表 3-11: プリプロセッサオプション (つづき)

オプション	定義
-H	通常動作に加えて、使用されるヘッダファイルの名前を印刷します。
-I-	-I- オプションの前に -I オプションを指定する、どんなディレクトリも、 <code>#include "file"</code> の場合にのみ検索され、 <code>#include <file></code> の場合には検索されません。追加ディレクトリが、-I- の後に -I オプション付きで指定されると、これらのディレクトリはすべての <code>#include</code> 命令で検索されます (通常すべての -I ディレクトリはこのように使用される)。さらに、-I- オプションは (現入力ファイルがある) 実行ディレクトリを、 <code>#include "file"</code> で最初に検索するディレクトリとして禁止します。-I- のこの効果を上書きすることはできません。-I を用いると、コンパイラ起動時のディレクトリの検索を指示できます。それは、プリプロセッサがデフォルトで実行するものとは異なりますが、適切な結果が得られる場合もあります。-I- はヘッダファイルの標準システムディレクトリの使用を禁止しません。したがって、-I- と -nostdinc は独立です。
-Idir	ヘッダファイルを検索するために、ディレクトリのリストの頭に、ディレクトリ <i>dir</i> を付加します。これによりシステムヘッダファイルを上書きできます。これらのディレクトリはシステムヘッダファイルディレクトリ以前に検索されるため、1 個以上の -I オプションを使用すると、ディレクトリは左から右へスキャンされ、その後標準システムディレクトリがスキャンされます。
-idirafter <i>dir</i>	ディレクトリ <i>dir</i> を第二のインクルードパスに追加します。第二のインクルードパスは、メインのインクルードパス (-I を追加されたパス) のディレクトリ内でヘッダファイルが見つからない場合に検索されます。
-imacros <i>file</i>	通常の入力ファイルを処理する前に、ファイルを入力として処理し、出力は廃棄します。ファイルから生成された出力は廃棄されるので、-imacros <i>file</i> の唯一の効果は、メインの入力で使用できるファイルの中でマクロを定義することにあります。コマンドライン上の -D や -U オプションは、それらが出現を問わず、常時 -imacros <i>file</i> 以前に処理されます。すべての -include と -imacros オプションは、出現順番に処理されます。
-include <i>file</i>	通常の入力ファイルを処理する前に、ファイルを入力として処理します。ファイルの内容が先にコンパイルされます。コマンドライン上の -D や -U オプションは、順番を問わず、常時 -include <i>file</i> 以前に処理されます。すべての -include と -imacros オプションは、出現順番に処理されます。
-iprefix <i>prefix</i>	<i>prefix</i> を次の -iwithprefix オプションとして指定します。
-isystem <i>dir</i>	第二のインクルードパスの始めにディレクトリを追加し、標準システムディレクトリに適用されるものと同様にシステムディレクトリとします。
-iwithprefix <i>dir</i>	第二のインクルードパスにディレクトリを追加します。ディレクトリの名前はプリーフィックスと <i>dir</i> を結合することで作成します。ここで、プリーフィックスとは事前に -iprefix で指定されます。プリーフィックスが指定されていない場合は、コンパイラのインストールパスを含むディレクトリがデフォルトとして使用されます。

MPLAB C30 C コンパイラを使用する

表 3-11: プリプロセッサオプション (つづき)

オプション	定義
<code>-iwithprefixbefore dir</code>	メインのインクルードパスにディレクトリを追加します。ディレクトリの名前は、 <code>-iwithprefix</code> の場合と同様に、 <code>prefix</code> と <code>dir</code> を結合するして作成されます。
<code>-M</code>	それぞれのオブジェクトファイルの依存性の記述に適した規則を出力するよう、プリプロセッサに指示します。それぞれのソースファイル用に、ターゲットがソースファイルのオブジェクトファイル名であり、依存性が、使用するすべての <code>#include</code> ヘッダファイルとなるよう指定してプリプロセッサが出力します。このルールは単一行であるか、もし長ければ <code>\-newline</code> で続けます。ルールへのリストは、事前処理された C プログラムではなく、標準出力で印刷されます。 <code>-M</code> は <code>-E</code> を含みます。(セクション 3.5.2 「出力類の制御オプション」参照)。
<code>-MD</code>	<code>-M</code> と同様ですが依存性情報はファイルに書き込まれ、コンパイルは継続します。依存性情報を含んだファイルは <code>.d</code> 拡張子を持ったソースファイルと同じ名前が付与されます。
<code>-MF file</code>	<code>-M</code> もしくは <code>-MM</code> と同時に使用し依存性が書き込まれるファイル指定します。 <code>-MF</code> スイッチが与えられない場合、事前処理された出力と同じ場所に、プリプロセッサがルールを配置します。ドライバオプション <code>-MD</code> または <code>-MMD</code> 、 <code>-MF</code> を使用すると、デフォルトの依存性出力ファイルに上書きします。
<code>-MG</code>	存在しないヘッダファイルが生成されたファイルのように扱い、ソースファイルと同じディレクトリ内に存在すると仮定します。 <code>-MG</code> が指定されると、 <code>-M</code> もしくは <code>-MM</code> も指定されねばなりません。 <code>-MG</code> は <code>-MD</code> もしくは <code>-MMD</code> と一緒にサポートされません。
<code>-MM</code>	<code>-M</code> と同様ですが、出力は <code>#include "file"</code> に含まれたユーザヘッダファイルのみ言及します。 <code>#include <file></code> に含まれたシステムヘッダファイルは省略されます。
<code>-MMD</code>	<code>-MD</code> と同様ですが、システムヘッダファイルではなく、ユーザヘッダファイルのみ言及します。
<code>-MP</code>	このオプションは、 CPP に、メインファイルとは別の依存性に、偽のターゲットを追加するように指示し、いずれにも依存しないよう設定します。これらのダミールールは、メイクファイルを更新せずにヘッダファイルを削除すると、メイクファイルが起因となるエラーを回避します。一般的な出力例は次の通りです。 test.o: test.c test.h test.h:
<code>-MQ</code>	<code>-MT</code> と同様、特別な make 用の文字を引用します。 <code>-MQ '\$(objpfx)foo.o'</code> は次の通りです。 \$\$\$(objpfx)foo.o: foo.c デフォルトのターゲットは、 <code>-MQ</code> で与えられたように自動で引用されます。

表 3-11: プリプロセッサオプション (つづき)

オプション	定義
-MT <i>target</i>	依存性生成により作成されたルールのターゲットを変更します。デフォルトでは、CPP は、いかなるパスも含むメインの入力ファイルの名前を取り、.c のようなファイル添え字を削除し、プラットフォームの通常のオブジェクト添え字を添付します。結果がターゲットです。-MT オプションはターゲットを、指定したストリングと同じものに設定します。複数のターゲットが必要なら、それを -MT に対する単一引数として指定するか、もしくは複数の -MT オプションを使用します。例えば、 -MT '\$(objpfx)foo.o' は \$(objpfx)foo.o: foo.c を生成します。
-nostdinc	ヘッダーファイル用の標準システムディレクトリを検索しません。-I オプションで指定したディレクトリ (および、もし適切であれば現ディレクトリ) のみが検索されます。-I については、(セクション 3.5.10 「ディレクトリ検索用オプション」参照) -nostdinc と -I- の両方を使用することで、インクルードファイル検索パスは明示的に指定されたディレクトリのみ限定されます。
-P	#line ディレクティブを生成しないようにプリプロセッサに指示します。-E オプションと一緒に使用されます。(セクション 3.5.2 「出力類の制御オプション」参照)。
-trigraphs	ANSI C の三重文字をサポートします。-ansi オプションは同様の効果を有します。
-U <i>macro</i>	マクロ <i>macro</i> を定義解除し、-U オプションはすべての -D オプションの後に、-include オプションや -imacros オプションより先に評価されます。
-undef	アーキテクチャフラグを含む、非標準のマクロの事前定義をしません。

3.5.8 アセンブラのオプション

表 3-12: アセンブラオプション

オプション	定義
-Wa, <i>option</i>	<i>option</i> をオプションとして、アセンブラに渡します。 <i>option</i> がコンマを含んでいる場合、コンマの部分で複数のオプションに分割されます。

3.5.9 リンク用オプション

オプション `-c`, `-s` もしくは `-E` が使用されると、リンカは実行されず、オブジェクトファイル名は引数として使用されません。

表 3-13: リンク用オプション

オプション	定義
<code>-Ldir</code>	<code>dir</code> をコマンドラインオプション <code>-l</code> で指定されるディレクトリをライブラリ検索対象ディレクトリのリストに追加します。
<code>-llibrary</code>	<p>リンク時に <code>library</code> と名づけられたライブラリを検索します。</p> <p>リンカは標準のライブラリのディレクトリのリストを検索します。ライブラリのファイル名は、<code>liblibrary.a</code> です。リンカはこのファイルを正確な名前指定された時に使用します。このオプションをコマンドのどの場所に記述するか、および、リンカプロセスライブラリ、オブジェクトファイルの指定の順番によって相違があります。<code>foo.o -lz bar.o</code> はファイル <code>foo.o</code> の後にライブラリ <code>z</code> を検索しますが、それは <code>bar.o</code> の検索の前になります。<code>bar.o</code> が <code>libz.a</code> 内の関数を参照した場合、これら関数はロードされません。</p> <p>検索されたディレクトリはいくつかの標準システムディレクトリと <code>-I</code> で指定されるものを持っています。</p> <p>通常このようにして見つかったファイルはライブラリファイル（アーカイブファイルで、そのメンバがオブジェクトファイルです。）です。リンカはアーカイブファイルをスキャンし、参照されるが定義されていないシンボルを定義するメンバを探します。ただし、ここで検索されたのが通常のオブジェクトファイルならば、通常の方法でリンクされます。<code>-l</code> オプション（すなわち <code>-lmylib</code>）を使用する場合と、ファイル名（すなわち <code>libmylib.a</code>）を指定する場合で唯一異なるのは、<code>-l</code> が、指定された通りにいくつかのディレクトリを検索することです。</p> <p>デフォルトでは、リンカは、<code>-l</code> オプションで指定されたライブラリを求めて <code><install-path>\lib</code> を検索するように指示されます。デフォルト位置にインストールされたコンパイラには、<code>c:\Program Files\Microchip\MPLAB C30\lib</code> となり、セクション 3.6「環境変数」 で定義される環境変数を使用すると上書きされます。</p>
<code>-ndefaultlibs</code>	リンク時に標準システムライブラリを使用しません。指定されたライブラリのみがリンカに渡されます。コンパイラは <code>memcpy</code> , <code>memset</code> と <code>memcpy</code> に対してコールを生成します。通常、これらのエントリは標準コンパイラライブラリ内のエントリとして解決されます。これらのエントリポイントは、このオプションを指定する場合に他のメカニズムから付与されます。
<code>-nostdlib</code>	リンク時に標準システムスタートアップファイルもしくはライブラリを使用しません。スタートアップファイルは渡さず、指定されたライブラリのみがリンカに渡されます。コンパイラは <code>memcpy</code> , <code>memset</code> と <code>memcpy</code> に対してコールを生成します。通常、これらのエントリは標準コンパイラライブラリ内のエントリとして解決されます。これらのエントリポイントは、このオプションを指定する場合に他のメカニズムから付与されます。
<code>-s</code>	実行可能物ファイルからシンボル表や再配置情報を削除します。
<code>-u symbol</code>	<code>symbol</code> を未定義として、ライブラリモジュールを強制的にリンクさせこの記号を定義します。 <code>-u</code> を異なる記号で複数回使用し追加のライブラリモジュールを強制的にロードしても不正ではありません。
<code>-Wl,option</code>	<code>option</code> をオプションとしてリンカに渡します。 <code>option</code> がコンマを含む場合には、コンマの位置で複数のオプションに分割されます。
<code>-Xlinker option</code>	<code>option</code> をオプションとしてリンカに渡します。これを使用して、システムに特有なリンカオプションで MPLAB C30 が認識できないオプションを付与できます。

3.5.10 ディレクトリ検索用オプション

表 3-14: ディレクトリ検索用オプション

オプション	定義
<code>-Bprefix</code>	このオプションは、実行可能ファイル、ライブラリ、インクルードファイル、コンパイラ自身のデータファイルを検索する場所を指定します。コンパイラドライバプログラムは、1つ以上のサブプログラム、 <code>pic30-cpp</code> 、 <code>pic30-cc1</code> 、 <code>pic30-as</code> および <code>pic30-ld</code> を実行します。また、 <code>prefix</code> をそれぞれのプログラムの接頭辞としてとらえ、実行させます。サブプログラムを実行するために、コンパイラドライバは最初に、 <code>-B prefix</code> を試みます。サブプログラムが見つからない場合、もしくは、 <code>-B</code> が指定されていない場合は、ドライバは <code>PIC30_EXEC_PREFIX</code> 環境変数内にある値を使用します。詳細は セクション 3.6「環境変数」 を参照ください。また、ドライバは、サブプログラム用の現状 <code>PATH</code> の環境変数を探します。効果的にディレクトリの名前を指定する <code>-B</code> 接頭辞はリンカのライブラリにも適用されます。コンパイラがこれらのオプションをリンカ用の <code>-L</code> オプションに変換するため、プリプロセッサ内のインクルードファイルにも適用します。コンパイラがこれらのオプションを、プリプロセッサ用の <code>-isystem</code> オプションに変換するため、コンパイラはインクルードを接頭辞に付与します。 <code>-B</code> 接頭辞と同様に接頭辞を指定するには、環境変数 <code>PIC30_EXEC_PREFIX</code> を使用方法もあります。
<code>-specs=file</code>	コンパイラが標準仕様ファイルを読み込んだ後にファイル进行处理します。それはどのスイッチが <code>pic30-cc1</code> 、 <code>pic30-as</code> 、 <code>pic30-ld</code> 等に渡されるべきかを決定する際に、 <code>pic30-gcc</code> ドライバプログラムが使用するデフォルトを上書きするためです。一つ以上の <code>-specs=file</code> がコマンドライン上で指定でき、左から右へと順番に処理されます。

3.5.11 コード生成規則オプション

形式 *-fflag* のオプションはマシンに依存しないフラグを指定します。フラグの多くは正負両方の形式を持ち、*-ffoo* の負形式は *-fno-foo* です。以下の表には、形式のうち一つ（デフォルトでない形式）のみリストアップします。

表 3-15: コード生成規則オプション

オプション	定義
-fargument-alias -fargument-noalias -fargument-noalias-global	<p>パラメータ間およびパラメータとグローバルデータ間の可能性のある関係を指定します。</p> <p>-fargument-alias は引数（パラメータ）が別名であり、グローバルストレージとも別名であることを指定します。</p> <p>-fargument-noalias は引数が別名ではないが、グローバルストレージとは別名でを指定します。</p> <p>-fargument-noalias-global は非引数が別名でなく、グローバルストレージとも別名ではないことを指定します。それぞれの言語は、言語標準により要求されるオプションをすべて自動的に使用します。これらのオプションをマニュアルで使用する必要はありません。</p>
-fcall-saved-reg	<p><i>reg</i> と名づけられたレジスタを関数で保存される割り当て可能なレジスタとして扱います。コール全般にわたって有効な一時的なものもしくは変数として割り当てられます。このようにコンパイルされた関数は、レジスタを使用している場合、レジスタ <i>reg</i> を保存、復帰します。</p> <p>このフラグをフレームポインタやスタックポインタと一緒に使用するのはエラーになります。マシンの実行モデル内で、固定の役割を持つ他のレジスタ用としてこのフラグを使用すると重大な障害を引き起こします。また、このフラグを、関数戻り値レジスタ用として使用しても重大な障害が起こります。このフラグはすべてのモジュールで統一し、使用してください。</p>
-fcall-used-reg	<p><i>reg</i> と名づけられたレジスタを関数コールで上書きされる割り当て可能なレジスタとして扱います。コール全般にわたって有効ではない一時的なものもしくは変数として割り当てられます。コンパイルされた関数は、レジスタ <i>reg</i> の保存、復帰はしません。このフラグをフレームポインタやスタックポインタと一緒に使用するのはエラーになります。</p> <p>マシンの実行モデルに関わる固定の役割を持つ他のレジスタ用にこのフラグを使用すると重大な障害を引き起こします。このフラグはすべてのモジュールで統一し、使用してください。</p>
-ffixed-reg	<p><i>reg</i> と名づけられたレジスタを固定されたレジスタとして扱い、生成されたコードは（スタックポインタ、フレームポインタ、もしくはその他の固定された役割以外）参照しません。</p> <p><i>reg</i> はレジスタの名前（例えば、<i>-ffixed-w3</i>）を使用してください。</p>
-finstrument-functions	<p>関数の出入り口に実装コールを生成します。関数の入り口直後と出口直前で以下のプロファイリング関数が現在の関数とそのコールサイトのアドレスとともにコールされます。</p> <pre>void __cyg_profile_func_enter (void *this_fn, void *call_site); void __cyg_profile_func_exit (void *this_fn, void *call_site);</pre> <p>最初の引数は現関数のスタートのアドレスで、シンボル表の中から検索できます。プロファイリング関数はユーザで用意しなければなりません。</p> <p>関数実装はフレームポインタを使用するよう指示しますが、最適化レベルによりフレームポインタの使用を不可にするため、<i>-fno-omit-frame-pointer</i> を使用すれば回避できます。</p>

表 3-15: コード生成規則オプション (つづき)

オプション	定義
	<p>この実装は別の関数内で、関数に拡張されたインラインにも実行されます。プロファイリングコールは、概念的には、インライン関数が入り出る場所を指示します。つまり関数のアドレス可能なバージョンを使用することです。使用する関数すべてが拡張されたインラインである場合、これはコードサイズの増加を意味します。C コードの中で <code>extern inline</code> が使用されている場合、関数のアドレス可能なバージョンが必要です。</p> <p>関数は、属性 <code>no_instrument_function</code> を与えられる場合、この 00 実装は実行されません。</p>
<code>-fno-ident</code>	<code>#ident</code> ディレクティブを無視します。
<code>-fpack-struct</code>	<p>構造体メンバすべてをまとめます。通常、このオプションは使用しません。コードを部分的に最適化し、構成メンバのオフセットがシステムライブラリとは一致しなくなるためです。</p> <p>dsPIC® DSC デバイスは偶数バイト境界上にワード整列を要求するため、<code>packed</code> 属性を使用する際は実行時にアドレスエラーが起こらないよう注意してください。</p>
<code>-fpcc-struct-return</code>	<p>レジスタではなく、メモリ内に短い <code>struct</code> や <code>union</code> を返します。この規則は効率良くありませんが、MPLAB® C30 でコンパイルされたファイルとその他のコンパイラでコンパイルされたファイル間の互換性に関して利点があります。</p> <p>短い構造体やユニオンはそのサイズと配列が <code>int</code> 型のサイズと配列に適合します。</p>
<code>-fno-short-double</code>	<p>デフォルトでは、コンパイラは <code>float</code> 小数点型に <code>double</code> 型を使用します。このオプションは <code>double</code> 型を <code>long double</code> 型相当にします。このオプションをモジュール間で混用すると、モジュールが、引数を通して直接か、もしくは共有されている領域を通して間接的に倍精度データを共有している場合には、予期せぬ結果をもたらすことがあります。ライブラリは、両方のスイッチ設定をもったプロダクト関数とともに提供されます。</p>
<code>-fshort-enums</code>	<p>ある値の宣言された範囲に必要なとされるだけのバイト数を、<code>enum</code> タイプのみに割り当てます。特に、その <code>enum</code> タイプは、十分な余裕を持つ最小の整数型に相当します。</p>
<code>-fverbose-asm</code> <code>-fno-verbose-asm</code>	<p>読みやすくするために、生成されたアセンブリコード内に追加のコメント情報を入れます。デフォルトは <code>-fno-verbose-asm</code> であり、追加情報は省略され、二つのアセンブラファイルを比較する場合に役立ちます。</p>
<code>-fvolatile</code>	ポインタ経由のすべてのメモリ参照を <code>volatile</code> と見なします。
<code>-fvolatile-global</code>	外部もしくはグローバルデータアイテムに対するすべてのメモリ参照を <code>volatile</code> とみなします。このスイッチの使用は静的データに影響しません。
<code>-fvolatile-static</code>	静的データに対するすべてのメモリ参照を <code>volatile</code> と見なします。

3.6 環境変数

この章での変数はオプションですが、定義するとコンパイラで使用されます。コンパイラドライバ、もしくは他のサブプログラムは、もし環境変数の値が設定されていないならば、以下の環境変数の一部に対する適切な値を決定するよう選択します。ドライバもしくは他のサブプログラムは、MPLAB C30 の組込みに関しての内部知識を利用します。組込み構造が完全であり、すべてのサブディレクトリと実行ファイルが同じ相対位置にある限り、ドライバもしくはサブプログラムで値を決定できません。

表 3-16: コンパイラに関係する環境変数

オプション	定義
PIC30_C_INCLUDE_PATH	この変数の値は PATH のように、セミコロンで区切られたディレクトリリストです。MPLAB [®] C30 がヘッダファイルを検索する際に、-I で指定されるディレクトリの後で標準ヘッダファイルディレクトリの前に、変数内のディレクトリを検索します。もし環境変数が定義されないと、プリプロセッサは、標準インストールに基づいて適切な値を選択します。デフォルトでは、インクルードファイル用として、以下のディレクトリが検索されます。 <install-path>\include と <install-path>\support\h
PIC30_COMPILER_PATH	PIC30_COMPILER_PATH の値は、PATH と同様に、セミコロンで区切られたディレクトリリストです。PIC30_EXEC_PREFIX を使用したサブプログラムを検索できない場合、MPLAB C30 は、サブプログラムを検索する際には指定されたディレクトリをサーチします。
PIC30_EXEC_PREFIX	もし PIC30_EXEC_PREFIX が設定されると、それは、接頭辞を指定し、コンパイラで実行されるサブプログラムの名前の中で使用します。この接頭辞がサブプログラムの名前と接合されるとき、ディレクトリパスは使用されませんが、スラッシュで終わる接頭辞を指定することもできます。MPLAB C30 が、指定された接頭辞を使用しているサブプログラムを検索できない場合、PATH の中で環境変数を探索そうとします。 PIC30_EXEC_PREFIX 環境変数が設定されないか、もしくは空の値を設定すると、コンパイラドライバは標準インストールに基づいて適切な値を選択します。そのインストールが変更されていない場合、ドライバが、必要とされるサブプログラムを割り当てることができます。 -B コマンドラインオプションで指定されるその他の接頭辞は、ユーザもしくはドライバで定義される値 PIC30_EXEC_PREFIX により先行します。 通常、この値は定義せず、ドライバ自身にサブプログラムを位置付けるようにさせます。
PIC30_LIBRARY_PATH	この変数の値は、PATH と同様に、セミコロンで区切られたディレクトリリストです。この変数はディレクトリのリストを指定し、リンクに渡します。この変数のデフォルトは以下の通りです。 <install-path>\lib; <install-path>\support\gld.
PIC30_OMF	OMF (オブジェクトモジュールフォーマット) を指定し MPLAB C30 で使用できるようにし、デフォルトで、ツールは COFF オブジェクトファイルを作成します。環境変数 PIC30_OMF が値 elf であれば、ツールは ELF オブジェクトファイルを作成します。
TMPDIR	TMPDIR が設定された場合、一時ファイル生成用として使用するディレクトリを指定します。MPLAB C30 は次のステージの入力として使用するコンパイルの出力を保持するためにテンポラリファイルを使用します。例えば、プリプロセッサの出力が、コンパイラ本体の入力になります。

3.7 事前定義制約

コンパイラの出力のカスタマイズに使用する定数です。

3.7.1 定数

次のプリプロセッシング記号は、使用中のコンパイラによって定義されます。

コンパイラ	記号	-ansi コマンドライン オプションで定義されているか
MPLAB® C30	C30	いいえ
	__C30	はい
	__C30__	はい
ELF 特有	C30ELF	いいえ
	__C30ELF	はい
	__C30ELF__	はい
COFF 特有	C30COFF	いいえ
	__C30COFF	はい
	__C30COFF__	はい

次の記号はターゲット ファミリを定義します。

記号	-ansi コマンドライン オプションで定義されているか
__dsPIC30F__	はい
__dsPIC33F__	はい
__PIC24F__	はい
__PIC24H__	はい

また、コンパイラは、-mcpu= に設定されたターゲット デバイスに基づいて記号を定義します。例えば、-mcpu=30F6014 は、記号 __dsPIC30F6014__ を定義します。

コンパイラは、定数 __C30_VERSION__ を定義し、バージョン識別子に数値を与えます。これは、古いバージョンとの下位互換性を維持しながら、新しいコンパイラの機能を利用するのに使用できます。

値は、現行リリースの主バージョン番号および副バージョン番号に基づきます。例えば、リリースバージョン 2.00 の __C30_VERSION__ 定義は 200 になります。このマクロは、通常のプリプロセッサ比較ステートメントと組み合わせて、さまざまなコード構造体を条件付きでインクルードまたはエクスクルードするのに使用できます。

現在の __C30_VERSION__ の定義は、コマンドラインに --version を追加すると表示される他、リリースに付属する README.TXT ファイルにも記載されています。

3.7.2 使用できない定数

推奨されなくなった定数は、付録 D. 「使用を廃止した機能」に示されています。

3.8 コマンドライン上の一つのファイルをコンパイルする

この章では、一つのファイルをコンパイルして、リンクする方法を示します。ここでは、コンパイラは c: ドライブ上の、pic30-tools と呼ばれるディレクトリ内にインストールされているものとします。したがって、以下の内容が適用されます。

表 3-17: コンパイラに関連したディレクトリ

ディレクトリ	定義
c:\Program Files\ Microchip\MPLAB C30\ include	ANSI C ヘッダファイル用のインクルードディレクトリです。このディレクトリは、標準 C ライブラリシステムヘッダファイルをコンパイラがストアする場所です。PIC30_C_INCLUDE_PATH 環境変数はそのディレクトリを指します (DOS コマンドプロンプトから set をタイプすればチェックします)。
c:\Program Files\ Microchip\MPLAB C30\ support\h	dsPIC [®] DSC デバイス特定のヘッダファイル用のインクルードディレクトリです。このディレクトリはコンパイラが dsPIC デバイス特定ヘッダファイルを保存する場所です。PIC30_C_INCLUDE_PATH 環境変数はそのディレクトリを指します (DOS コマンドプロンプトから set をタイプすればこれをチェックします)。
c:\Program Files\ Microchip\MPLAB C30\ lib	ライブラリディレクトリです。このディレクトリは、ライブラリと事前コンパイルされたオブジェクトファイルが置かれるところです。
c:\Program Files\ Microchip\MPLAB C30\ support\gld	リンカスクリプトディレクトリです。このディレクトリはデバイス特有のリンカスクリプトのある場所です。
c:\Program Files\ Microchip\MPLAB C30\ bin	実行可能なディレクトリです。このディレクトリは、コンパイルプログラムが置かれる場所です。PATH 環境変数にはこのディレクトリを含む必要があります。

以下は、二つの数を足し算する簡単な C プログラムです。

以下のプログラムをテキストエディタで作成し、ex1.c として保存します。

```
#include <p30f2010.h>
int main(void);
unsigned int Add(unsigned int a, unsigned int b);
unsigned int x, y, z;
int
main(void)
{
    x = 2;
    y = 5;
    z = Add(x,y);
    return 0;
}
unsigned int
Add(unsigned int a, unsigned int b)
{
    return(a+b);
}
```

プログラムの第一行目は、ヘッダファイル p30f2010.h を含み、ヘッダファイルはすべての特殊関数レジスタ用の定義を与えます。ヘッダファイルの詳細については、**第 6 章**、「デバイスサポートファイル」を参照ください。

DOS プロンプトで以下をタイプし、プログラムをコンパイルします。

```
C:\> pic30-gcc -o ex1.o ex1.c
```

コマンドラインオプション `-o ex1.o` は、出力 COFF 実行可能なファイルの名前を付けます（もし `-o` オプションが指定されていないと、出力ファイル名は `a.exe` と名づけられます）。COFF 実行可能なファイルは、MPLAB IDE にロードします。

hex ファイルが必要な場合、例えばデバイスプログラマにロードするには、以下のコマンドを使用します。

```
C:\> pic30-bin2hex ex1.o
```

`ex1.hex` と名づけられたインテル hex ファイルが生成されます。

3.9 コマンドライン上の複数のファイルをコンパイルする

アプリケーション内の複数ファイルの使い方をデモするために、`Add()` 関数を `add.c` と呼ばれるファイルへ移動します。それは下記のようにします。

ファイル 1

```
/* ex1.c */
#include <p30f2010.h>
int main(void);
unsigned int Add(unsigned int a, unsigned int b);
unsigned int x, y, z;
int main(void)
{
    x = 2;
    y = 5;
    z = Add(x,y);
    return 0;
}
```

ファイル 2

```
/* add.c */
#include <p30f2010.h>
unsigned int
Add(unsigned int a, unsigned int b)
{
    return(a+b);
}
```

DOS プロンプトで次のように入力し、両ファイルをコンパイルします。

```
C:\> pic30-gcc -o ex1.o ex1.c add.c
```

このコマンドは、モジュール `ex1.c` と `add.c` をコンパイルします。コンパイルされたモジュールはコンパイラライブラリと一緒にリンクされ、実行可能ファイル `ex1.o` が生成されます。

第 4 章 . MPLAB C30 C コンパイラ実行時環境

4.1 序章

本章では MPLAB C30 C コンパイラ実行時環境について説明します。

4.2 ハイライト

本章で説明する項目は以下の通りです。

- アドレス空間
- コードとデータセクション
- スタートアップと初期化
- メモリ空間
- メモリモデル
- コードとデータの配置
- ソフトウェアスタック
- C スタック使用方法
- C ヒープの使用方法
- 関数コール規則
- レジスタの規則
- ビット反転とモジュロアドレッシング
- Program Space Visibility (PSV) の使用方法

4.3 アドレス空間

dsPIC デジタルシグナルコントローラ (DSC) デバイスは、伝統的な PICmicro マイクロコントローラ (MCU) の特徴 (周辺、ハーバードアーキテクチャ、RISC) と新しい DSP の機能を一緒にしたものです。dsPIC DSC デバイスは、二つの異なるメモリ領域を持っています。

- プログラムメモリ (図 4-1) は実行可能コードとオプションの定数データを含みます。
- データメモリ (図 4-2) は、外部変数、静的変数、システムスタック、およびファイルレジスタを含みます。データメモリは、**near** データ (メモリ空間の最初の 8KB) と、**far** データ (メモリ空間の上位にある 56KB) から構成されます。

プログラムとデータメモリ領域は明確に区別されますが、コンパイラは Program Space Visibility (PSV) ウィンドウを介してプログラムメモリ内の定数データにアクセスできます。

図 4-1: プログラム空間メモリマップ

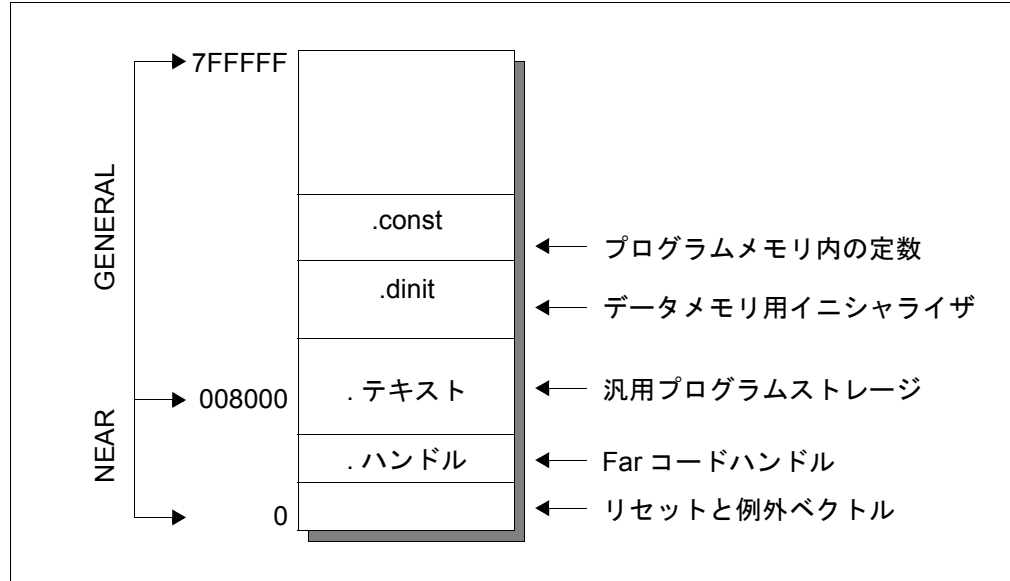
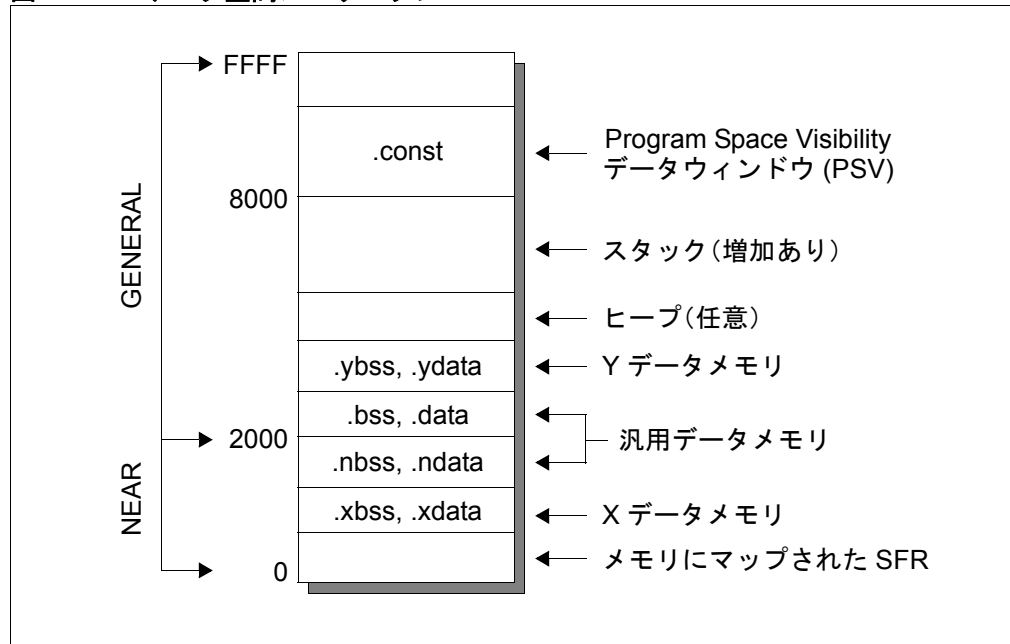


図 4-2: データ空間メモリマップ



4.4 コードとデータセクション

セクションとは、コードもしくはデータの、移動可能なブロックで、dsPIC DSC デバイスメモリの中で、連続した位置を占めます。どんなオブジェクトファイルにも、普通はいくつかのセクションがあります。例えば、ファイルには、プログラムコード用セクションと、初期化されていないデータ用のセクション等々があります。

MPLAB C30 コンパイラは、セクション属性を用いて他で指示されない限り、コードとデータをデフォルトのセクションに置きます（セクション属性の情報については、**セクション 2.3 「キーワードの違い」**をご参照ください。）。コンパイラが生成する、すべての実行可能コードは、`.text` という名前の付けられたセクションに割り当てられ、データは、表 4-1 で示されるように、データのタイプに基づき、異なるセクションに割り当てられます。

表 4-1: コンパイラが生成するデータセクション

	初期化されるもの		初期化されないもの	
	変数	ROM 内の定数	RAM 内の定数	変数
near	<code>.ndata</code>	<code>.const</code>	<code>.ndconst</code>	<code>.nbss</code>
far	<code>.data</code>	<code>.const</code>	<code>.dconst</code>	<code>.bss</code>

それぞれのデフォルトセクションと、そのセクションにストアされた情報のタイプを記述したものは、以下にリストされます。

.text

実行可能なコードは、`.text` セクションに割り当てられます。

.data

`far` 属性を持つ初期化された変数は、`.data` セクションに割り当てられます。ラージデータのメモリモデルが選択された時（すなわち、`-mlarge-data` コマンドラインオプションを用いる時）は、これが初期化された変数用のデフォルト位置になります。

.ndata

`near` 属性を持つ初期化された変数は、`.ndata` セクションに割り当てられます。スモールデータのメモリモデルが選択された時（すなわち、`-msmall-data` コマンドラインオプションを用いる時）は、これが初期化された変数用のデフォルト位置になります。

.const

ストリング定数や `const-` 修飾子付き変数のような、定数値は、デフォルトの `-mconst-in-code` コマンドライン オプションを用いた時には、`.const` セクションに割り当てられます。このセクションはプログラムメモリに配置するように割り当てられ、PSV ウィンドウを使用してアクセスされます。

変数は、`-mconst-in-code` オプションがコマンドライン上に存在するかどうかに関わらず、以下の `section` 属性を使用して `.const` セクションに配置されることがあります。

```
int i __attribute__((space(auto_psv)));
```

.dconst

ストリング定数や `const`-修飾子付き変数のような定数値は、`-mconst-in-code` コマンドライン オプションを使用せず、`-mlarge-data` コマンドライン オプションを用いた時には、`.dconst` セクションに割り当てられます。リンカオプション `--no-data-init` が指定されていないと、MPLAB C30 のスタートアップコードは、`.dinit` セクションからのデータをコピーすることでこのセクションを初期化します。`.dinit` セクションはリンカで生成されプログラムメモリに位置しています。

.ndconst

ストリング定数や `const`-修飾子付き変数のような定数値は、`-mconst-in-code` コマンドラインオプションを使用せず、`-msmall-data` コマンドラインオプションを用いた時には、`.ndconst` セクションに割り当てられます。リンカオプション `--no-data-init` が指定されていないと、MPLAB C30 のスタートアップコードは、`.dinit` セクションからのデータをコピーすることでこのセクションを初期化します。`.dinit` セクションはリンカで生成されプログラムメモリに位置しています。

.bss

`far` 属性を持った、初期化されない変数は `.bss` セクションに割り当てられます。ラージデータメモリモデルが選択された時（すなわち、`-mlarge-data` コマンドラインオプションを用いる時）は、これは初期化されない変数用のデフォルト位置になります。

.nbss

`near` 属性を持った、初期化されない変数は `.nbss` セクションに割り当てられます。スモールデータメモリモデルが選択された時（すなわち、`-msmall-data` コマンドラインオプションを用いる時）は、これは初期化されない変数用のデフォルト位置になります。

.pbss - Persistent Data

デバイスリセットに影響されない RAM 内のデータストレージを要求するアプリケーションは、この目的のために、セクション `.pbss` を使用します。セクション `.pbss` は `near` データメモリ内に割り当てられ、`libpic30.a` 内のデフォルトスタートアップモジュールによっては変更されません。

初期化されない変数は、以下のセクション属性を用いて `.pbss` セクションに置かれます。

```
int i __attribute__((persistent));
```

Persistent データストレージを利用するために、`main()` 関数はどのタイプのリセットが発生したかを決定するテストから始まります。`RCON` リセットコントローラレジスタ内の種々のビットがリセットソースを決定するためにテストされます。詳細は *dsPIC30F Family Reference Manual (DS70046)* の 8 章をご参照ください。

4.5 スタートアップと初期化

2つのCランタイムスタートアップモジュールが、libpic30.aアーカイブ/ライブラリに含まれます。両方のエントリポイントは__resetです。リンクスクリプトは、プログラムメモリ内の位置0でGOTO __reset命令を構成し、その命令はデバイスリセットにより制御が転送されます。

プライマリスタートアップモジュール(crt0.o)はデフォルトでリンクされ、以下のような動きをします。

1. スタックポインタ(W15)とスタックポインタリミットレジスタ(SPLIM)は、リンクもしくはカスタムリンクスクリプトで与えられる値を用いて、初期化されます。詳細については、**セクション 4.9「ソフトウエアスタック」**をご参照ください。
2. もし、.constセクションが定義されると、PSVPAGとCORCONレジスタの初期化によって、Program Space Visibility(PSV)ウィンドウにマッピングされます。.constセクションはMPLAB IDE内で「Constants in code space」オプションが選択された時、もしくは、デフォルトの-mconst-in-codeオプションが、MPLAB C30コマンドラインで指定されたときに定義されることに注意して下さい。
3. セクション.dinit内のデータ初期化テンプレートが読み出され、初期化されないセクションすべてがクリアされ、プログラムメモリから読み出される値で、すべての初期化されるセクションが初期化されます。データ初期化テンプレートはリンクで生成され、ユーザが定義するセクションと、**セクション 4.4「コードとデータセクション」**でリストアップされる標準セクションをサポートします。

注： Persistent データセクション .pbss はクリアも初期化もされません。

4. 関数mainはパラメータ無しでコールされます。
5. mainから戻ると、プロセッサはリセットされます。

代替のスタートアップモジュール(crt1.o)は、-W1, --no-data-initオプションが指定された時にリンクされます。ステップ(3) (これは省略されますが)を除いて、同じオペレーション動作をします。代替のスタートアップモジュールは初期モジュールよりもかなり小さく、初期化が必要ない場合は、プログラムメモリを節約するために選択されます。

両方のモジュールのソースコード(dsPIC DSCアセンブリ言語)は、c:\Program Files\Microchip\MPLAB C30\srcディレクトリ内に提供されています。スタートアップモジュールは、必要に応じて修正できます。例えば、アプリケーションがパラメータと一緒にコールされるmainを要求する場合、条件付きアセンブリ指示により、このサポートを実行します。

4.6 メモリ空間

静的または外部変数は通常、汎用データメモリ空間に割り当てられます。定数修飾された変数は、`constants-in-data` メモリモデルが選択されていると汎用データメモリに割り当てられ、`constants-in-code` メモリモデルが選択されている場合にはプログラムメモリに割り当てられます。

MPLAB C30 は複数の特殊用途メモリ空間を定義して dsPIC DSC のアーキテクチャ上の機能に一致させます。静的または外部変数は、**セクション 2.3.1 「変数属性の指定」** で述べられている `space` 属性を使用して特殊用途メモリ空間に割り当てられることがあります。

data

汎用データ空間。汎用データ空間内の変数には通常の C ステートメントを使用してアクセスできます。デフォルトではこのように割り当てられます。



xmemory - dsPIC30F/dsPIC33F デバイスのみ

X データアドレス空間。X データ空間内の変数には通常の C ステートメントを使用してアクセスできます。X データアドレス空間は DSP 指向ライブラリおよび/またはアセンブリ言語命令に対し特に適合性があります。



ymemory - dsPIC30F/dsPIC33F デバイスのみ

Y データメモリ空間。Y データ空間内の変数には通常の C ステートメントを使用してアクセスできます。Y データアドレス空間は DSP 指向ライブラリおよび/またはアセンブリ言語命令に対し特に適合性があります。

prog

汎用プログラム空間。通常は実行コードのために確保されています。プログラム空間内の変数には通常の C ステートメントを使用してもアクセスできません。これらの変数には通常、テーブルアクセスインランアセンブリ命令または PSV ウィンドウを使用してプログラマが明示的にアクセスする必要があります。

const

コンパイラにより管理されたプログラム空間内の領域で、PSV ウィンドウからのアクセスのためにデザインされています。`const` 空間内の変数は通常の C ステートメントを使用して読み取り可能で（書き込みは不可）、割り当て可能な空間は最大で合計 32K です。

psv

PSV ウィンドウからのアクセスのためにデザインされたプログラム空間です。`psv` 空間内の変数はコンパイラでは管理されず、通常の C ステートメントではアクセスできません。これらの変数には通常、テーブルアクセスインランアセンブリ命令または PSV ウィンドウを使用してプログラマが明示的にアクセスする必要があります。PSVPAG レジスタの設定をするだけで PSV エリアの変数にアクセスできるようになります。



eedata - dsPIC30F/dsPIC33F デバイスのみ

EEPROM 空間は 16 ビット幅不揮発性メモリ領域で、プログラムメモリ内の上位アドレスに配置されます。`eedata` 空間内の変数には通常の C ステートメントではアクセスできません。これらの変数には通常、テーブルアクセスインランアセンブリ命令または PSV ウィンドウを使用してプログラマが明示的にアクセスする必要があります。



dma - PIC24H MCU、dsPIC33F DSC のみ

DMA メモリ。DMA メモリの変数は、通常の C ステートメントをまたは、DMA 周辺からアクセスできます。

4.7 メモリモデル

コンパイラはいくつかのメモリモデルをサポートします。アプリケーションに最適なメモリモデルを選択するために、現在使用中の特定 dsPIC DSC デバイスと使用メモリのタイプに基づき、コマンドラインオプションが利用できます。

表 4-2: メモリモデルコマンドラインオプション

オプション	メモリー定義	定義
-msmall-data	8 KB までのデータメモリ これはデフォルトです。	データメモリにアクセスするための PIC18 のような命令の使用を許可します。
-msmall-scalar	8 KB までのデータメモリ これはデフォルトです。	データメモリ内のスカラーをアクセスするための PIC18 のような命令の使用を許可します。
-mlarge-data	8 KB を超えるデータメモリ	データリファレンス用の間接的アクセスを使用します。
-msmall-code	32K ワードまでのプログラムメモリ。これはデフォルトです。	関数ポインタはジャンプテーブルを経由しません。関数コールは RCALL 命令を使用します。
-mlarge-code	32K ワードを超えるプログラムメモリ	関数ポインタはジャンプテーブルを経由します。関数コールは CALL 命令を使用します。
-mconst-in-data	データメモリ内に配置された定数	値はスタートアップコードで、プログラムメモリからコピーされます。
-mconst-in-code	プログラムメモリ内に配置された定数。これはデフォルトです。	値は Program Space Visibility (PSV) データウインドウ経由でアクセスされます。

コマンドラインオプションはコンパイルされたモジュールにグローバルに適用されます。個別の変数と関数は、コード生成を適切に制御するために、near もしくは far として、宣言されます。個別の変数もしくは関数属性の設定については、**セクション 2.3.1 「変数属性の指定」**と**セクション 2.3.2 「関数の属性を指定する」**をご参照ください。

4.7.1 Near データと Far データ

もし変数が near データセクションに割り当てられると、コンパイラは変数が near データセクションに割り当てられない場合に比べよりコンパクトなコードを生成します。もしアプリケーションすべての変数が、near データの 8 KB 以内に収まれば、一つ一つのモジュールをコンパイルする時に、デフォルトの -msmall-data コマンドラインオプションを用いて変数を near データに置でにより使用されるデータの総量が 8 KB 未満であれば、デフォルトの -msmall-scalar が使用でき、コンパイラにアプリケーション用のスカラーを near データセクションに割り当てるよう要求します。

これらのグローバルオプションのうちいずれにも該当しない場合は、次の代替が利用できます。

1. `-mlarge-data` もしくは `-mlarge-scalar` コマンドライン オプションを使用して、アプリケーションをコンパイルできます。この場合、それらのモジュールで使用される変数のみが、`far` データセクションに割り当てられます。この代替を使用する場合、外部定義された変数の使用には注意が必要です。つまり、これらのオプションの一つを用いてコンパイルされたモジュールで使用する変数が外部で定義されていると、それが定義されているモジュールもまた同じオプションを用いてコンパイルする必要があります。もしくは、変数宣言と定義が `far` 属性を付加されている必要があります。
2. もしコマンドラインオプション `-mlarge-data` もしくは `-mlarge-scalar` が使用された場合、個々の変数は、`near` 属性を付けることで `far` データ空間から除外されます。
3. モジュールスコープを持つコマンドラインオプションを用いる代わりに、個々の変数に、`far` 属性を付けることで `far` データセクションに置くことができます。

アプリケーション用のすべての `near` 変数が 8K の `near` データ空間に収まらない場合は、リンカはエラーメッセージを出力します。

4.7.2 Near コードと Far コード

互いに半径 32K ワード以内にある `near` 関数は、そうでない場合よりもより効率的に互いにコールできます。もしアプリケーション内のすべての関数が `near` にあることがわかれば、個々のモジュールをコンパイルする際に、関数コールをより効率的な形式を使用するようコンパイラに指示し、デフォルトの `-msmall-code` コマンドライン オプションが使用できます。

このデフォルトオプションの使用が適切でない場合、以下の代替が利用できます。

1. `-msmall-code` コマンドライン オプションを用いてアプリケーションの一部のモジュールをコンパイルできます。この場合、それらのモジュール内の関数コールのみが、関数コールのより効率的な形式を使用します。
2. `-msmall-code` コマンドライン オプションが使用された場合、`far` 属性を付けることで個々の関数用の関数コールにロング形式を使用するようにコンパイラに指示します。
3. モジュールスコープを持つコマンドライン オプションを用いる代わりに、関数の宣言や定義に `near` 属性を付けることにより、関数コールにより効率的な形式を使用するよう個々の関数をコールします。

`-msmall-code` コマンドラインオプションは、`-msmall-data` コマンドライン オプションとは異なり前者の場合コンパイラは、関数が互いに近い位置に割り当てられることを特に保証しませんが、後者の場合、コンパイラは特別のセクションに変数を割り当てます。

`near` と宣言された関数が関数コールの効率的な形式を用いても、コーラの一つにたどり着かない場合には、リンカはエラーメッセージを出力します。

4.8 コードとデータの配置

セクション 4.4 「コードとデータセクション」で前述の通り、コンパイラは、.text セクションにコードを置き、使用中のメモリモデルやデータが初期化されているか否かにしたがって、データは名前を付けられた複数のセクションのうちの 1 つに置かれます。モジュールがリンク時に結合する際、リンカはその属性に基づき様々なセクションの開始アドレスを決定します。

関数もしくは変数が特定のアドレス、もしくはあるアドレス範囲内に置かれる場合には問題が発生します。この問題は、セクション 2.3 「キーワードの違い」で述べた通り、address 属性を使用して解決できます。例えば、関数 PrintString をプログラムメモリのアドレス 0x8000 に配置するには以下のように宣言します。

```
int __attribute__((address(0x8000))) PrintString (const char *s);
```

同様に、変数 Mabonga をデータメモリ内のアドレス 0x1000 に配置するには、以下のようにします。

```
int __attribute__((address(0x1000))) Mabonga = 1;
```

コードまたはデータを配置するもう 1 つの方法として、ユーザー定義のセクションに関数または変数を置き、カスタムリンクスクリプト内のセクションの開始アドレスを指定するという方法があります。具体的には、以下を実行します。

1. C ソース内のコードもしくはデータの宣言を修正し、ユーザー定義のセクションを指定します。
2. ユーザー定義のセクションをリンクスクリプトファイルに追加し、セクションの開始アドレスを指定します。

例えば、関数 PrintString をプログラムメモリ上のアドレス 0x8000 に置くためには、まず、C ソースの中で以下のように関数を宣言します。

```
int __attribute__((__section__(".myTextSection")))
PrintString(const char *s);
```

セクション属性は、デフォルトの .text セクションではなく .myTextSection と名づけられたセクションに関数を置くように指定します。ユーザー定義のセクションが配置される場所は指定しません。この指定は以下のようにカスタムリンクスクリプトで実行します。デバイス特有のリンクスクリプトをベースに使用する場合は、以下のセクションの定義を追加します。

```
.myTextSection 0x8000 :
{
    *(.myTextSection);
} >program
```

これは、出力ファイルが、.myTextSection と名づけられたセクションで、配置 0x8000 から開始するセクションを含み、.myTextSection と名づけられたすべての入力セクションを含むことを指定します。この例では、そのセクション内には一つの関数 PrintString があるので、関数はプログラムメモリの 0x8000 アドレスに配置されます。

同様に、データメモリ内のアドレス 0x1000 に変数 Mabonga を配置するには、最初に、C ソースの中で以下のように変数を宣言します。

```
int __attribute__((__section__(".myDataSection"))) Mabonga = 1;
```

セクション属性は、デフォルトの `.data` セクションではなく `.myDataSection` と名づけられたセクションに関数が置くよう指定します。ユーザ定義のセクションが配置される場所は指定しません。その指定は、以下のようにカスタムリンクスクリプトで実行します。デバイス特有のリンクスクリプトをベースに使用する場合は、以下のセクションの定義を追加します。

```
.myDataSection 0x1000 :  
{  
    * (.myDataSection);  
} >data
```

これは、出力ファイルが、`.myDataSection` と名づけられたセクションで、配置 `0x1000` から開始するセクションを含み、`.myDataSection` と名づけられたすべての入力セクションを含むことを指定します。この例では、そのセクション内には一つの変数 `Mabonga` があるので、変数はデータメモリの `0x1000` アドレスに配置されます。

4.9 ソフトウェアスタック

dsPIC DSC デバイスは、レジスタ `W15` をソフトウェアスタックポインタとして使用します。関数コール、割り込み、例外処理を含むすべてのプロセッサのスタック動作は、ソフトウェアスタックを使用します。スタックは、より上位のメモリアドレスへと上方向に積み重ねられます。

dsPIC DSC デバイスはまた、スタックオーバーフロー検出もサポートします。もしスタックポインタリミットレジスタ `SPLIM` が初期化されると、デバイスはすべてのスタックポインタ使用時についてオーバーフローのテストをします。オーバーフローが発生した場合、プロセッサはスタックエラー例外を発生します。デフォルトでは、これによりプロセッサはリセットされます。`_StackError` と名づけられた割り込み関数を定義することで、アプリケーションでもスタックエラー例外ハンドラをインストールできます。詳細は第7章、「割り込み」をご覧ください。

C runtime スタートアップモジュールは、スタートアップや初期化シーケンス内でスタックポインタ (`W15`) とスタックポインタ制限レジスタ (`SPLIM`) を初期化します。初期値は通常リンクで付与され、未使用のデータメモリから得られる最大のスタックを割り当てます。スタックの位置は、リンクマップ出力ファイルにレポートされます。アプリケーションは、少なくとも最小限のサイズのスタックが、`--stack` リンカコマンドラインオプションで利用することによって保証されます。詳細については、*MPLAB® ASM30, MPLAB LINK30* と *Utilities User's Guide (DS51317)* をご参照ください。

他の方法として、特定サイズのスタックを、カスタムリンクスクリプトのユーザ定義セクションで割り当てることができます。以下の例では、データメモリの `0x100` バイトがスタック用に確保されます。C runtime スタートアップモジュールで使用するため、`__SP_init` と `__SPLIM_init` の2つのシンボルが宣言されています。

```
.stack :  
{  
    __SP_init = .;  
    . += 0x100  
    __SPLIM_init = .;  
    . += 8  
} >data
```

`__SP_init` はスタックポインタ (`W15`) の初期値を定義し、`__SPLIM_init` はスタックポインタリミットレジスター (`SPLIM`) の初期値を定義します。

`__SPLIM_init` は、物理的スタックのリミットより少なくとも8バイト下側であり、スタックエラー例外処理用に残しておきます。この値は、スタックエラー割り

込みハンドラがインストールされていれば、割り込みハンドラ自体が使用するスタック用に低減できます。デフォルトの割り込みハンドラは、追加のスタックを使用しません。

4.10 C スタック使用方法

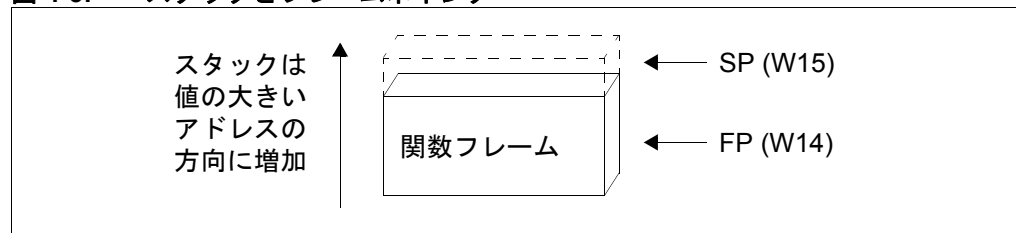
C コンパイラはソフトウェアスタックを使用して、次の内容を実行します。

- 自動変数の割り当て
- 引数の関数への引渡し
- 割り込み関数内でプロセッサ状態を保存
- 関数の戻りアドレスの保存
- 途中の一時結果のストア
- 関数コール間のレジスタの保存

実行時のスタックは低いアドレスから高位のアドレスへと上方向へ積み重ねていきます。コンパイラは二つのワーキングレジスタを用い、スタックを管理します。

- W15 - これはスタックポインタ (SP) です。スタック上の最初の未使用位置と定義されるスタックのトップを示します。
- W14 - これはフレームポインタ (FP) です。実行中の関数のフレームを指します。各関数は、要求されると、スタックの現在のトップからフレームを新規作成し、自動変数や一時変数を配置します。コンパイラオプション `-fomit-frame-pointer` は FP の使用を制限するために使用されます。

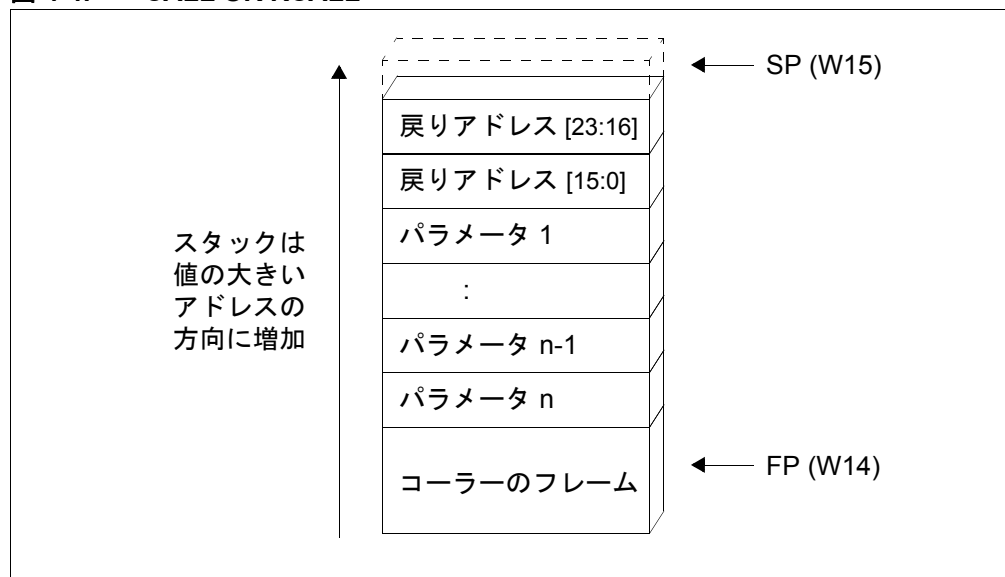
図 4-3: スタックとフレームポインタ



C 実行時スタートアップモジュール (libpic30.a 内の crt0.o と crt1.o) はスタックポインタ W15 を初期化し、スタックの底を示します。また、スタックポインタ制限レジスタを初期化してスタックのトップを示します。スタックは上に向けて積み重ねられ、スタックポインタ制限レジスタの値を越えると、スタックエラー発生します。ユーザはスタックの積み重ねをさらに制限するため、スタックポインタ制限レジスタを初期化することができます。

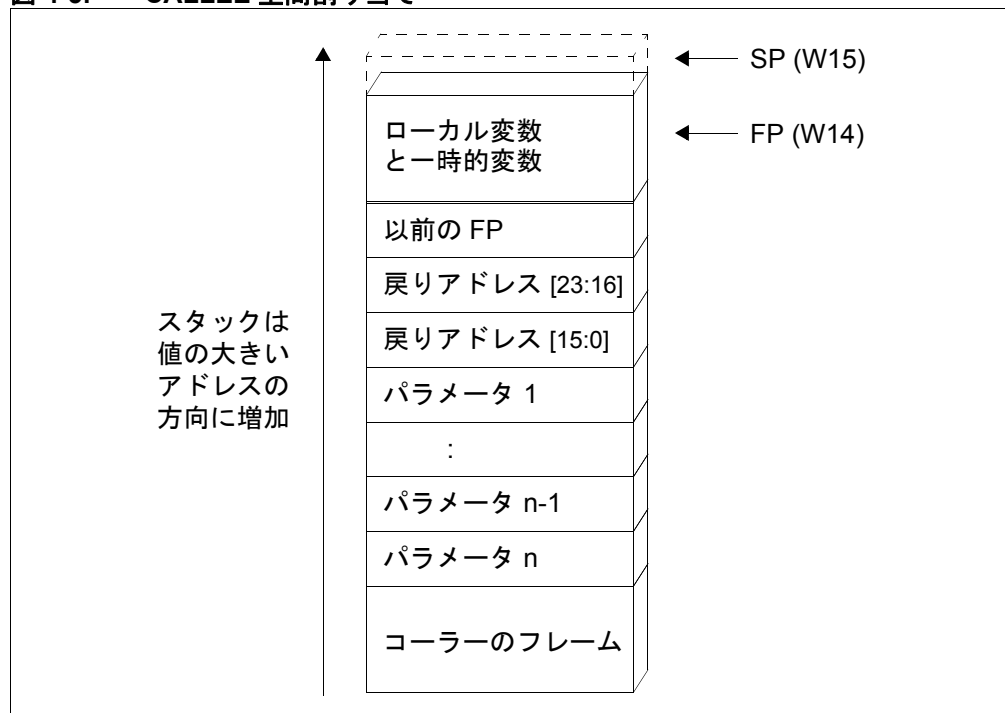
以下の図は関数コールに関する手順を図示しています。CALL もしくは RCALL 命令を実行すると、ソフトウェア上の戻りアドレスを push します。図 4-4 をご覧ください。

図 4-4: CALL OR RCALL



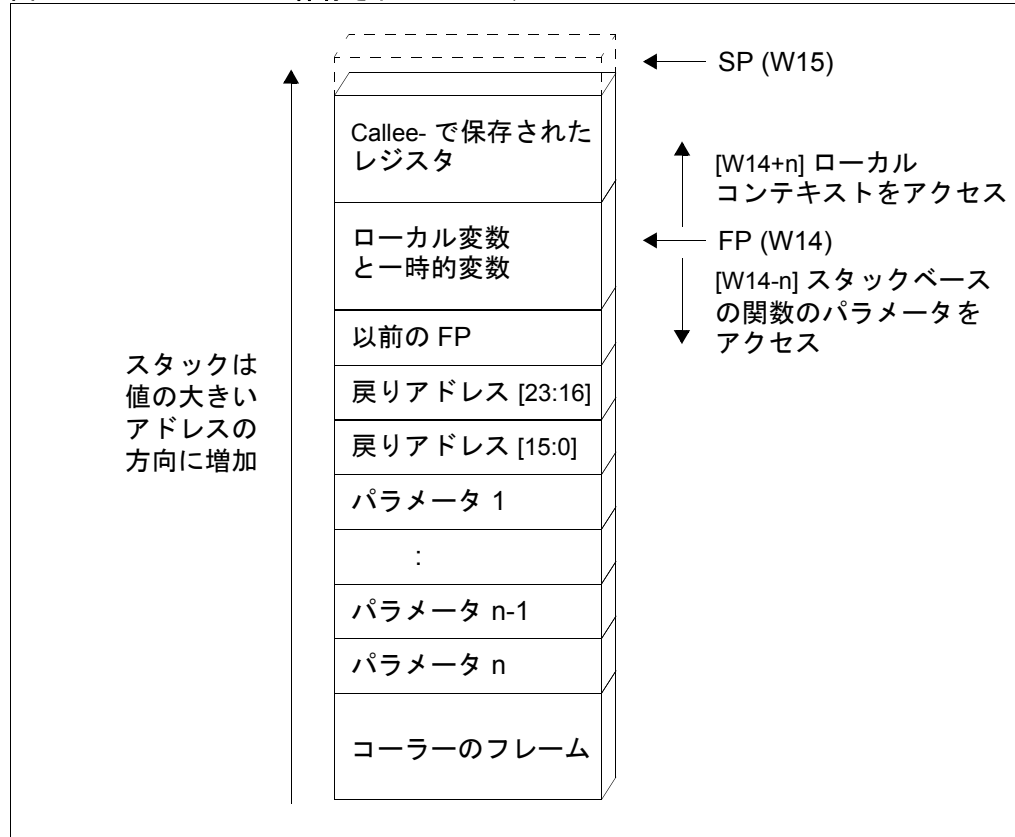
コールされた関数 (callee) はローカルコンテキスト用の空間を割り当てることができます。(図 4-5)。

図 4-5: CALLEE 空間割り当て



最後に、関数 (callee) で保存されたレジスタが push されます。(図 4-6)

図 4-6: CALLEE- で保存されたレジスタの PUSH



4.11 C ヒープの使用方法

C 実行時のヒープはデータメモリの初期化されない領域で、標準 C ライブラリのダイナミックメモリ管理関数 `calloc`、`malloc` と `realloc` を用いてダイナミックメモリ割り当てをする際に使用されます。もし、これらの関数を用いなければ、ヒープを割り当てる必要はありません。デフォルトでは、ヒープは生成されません。

ダイナミックメモリ割り当てをする場合には、直接的にメモリ割り当て関数の一つをコールするか、間接的に標準 C ライブラリの入出力関数を用いるかどちらかの方法になりますが、ヒープを生成する必要があります。ヒープは、`--heap` リンカコマンドラインオプションを用いて、リンカコマンドライン上でサイズを指定して生成されます。コマンドラインを用いて、512 バイトのヒープを割り当てる例は以下の通りです。

```
pic30-gcc foo.c -Wl,--heap=512
```

リンカはスタックのすぐ下にヒープを割り当てます (図 4-2)。

もし、標準 C ライブラリ入力、出力関数を使用する場合、ヒープを割り当てる必要があります。stdout が唯一の使用ファイルであれば、ヒープサイズはゼロ、すなわち、以下のようにコマンドラインオプションを用います。

```
-Wl,--heap=0
```

ファイルを開く場合、ヒープサイズは、同時に開くファイルの 1 つのファイルにつき 40 バイトを含みます。ヒープメモリが不十分な場合、オープン関数がエラー表示を返します。バッファモードでファイルを開く場合には、ファイル 1 つにつき 514 バイトのヒープ空間が必要です。バッファ用のヒープメモリが不十分な場合、ファイルは非バッファモードで開きます。

4.12 関数コール規則

関数をコールする場合は、

- レジスタ W0 - W7 はコールする側が保存します。コールする関数は、レジスタ値を保持するため、値をスタックの上にプッシュしなければなりません。
- レジスタ W8 - W14 はコールされる側が保存します。コールされる関数は、関数を使用するレジスタすべてを保存する必要があります。
- レジスタ W0-W4 は関数戻り値に使用されます。

表 4-3: 必要なレジスタ

データタイプ	必要なレジスタ数
char	1
int	1
short	1
pointer	1
long	2 (隣接 - 偶数レジスタに揃えます)
float	2 (隣接 - 偶数レジスタに揃えます)
double*	2 (隣接 - 偶数レジスタに揃えます)
long double	4 (隣接 - 4 倍数レジスタに揃えます)
structure	構造体内の 2 バイトあたり 1 レジスタ

* double は `-fno-short-double` が使用されている場合は long double と同等です。

パラメータは、利用できるレジスタのうち最初に揃えられる連続したレジスタ内に配置されます。コールする関数は、必要に応じてパラメータを保持する必要があります。構造体は整列させるという制限がないため、構造体パラメータは、構造体全体を保持するのに十分なレジスタがあれば、レジスタを占有します。関数の結果は W0 で始まる連続したレジスタ内に格納されます。

4.12.1 関数パラメータ

最初の 8 個のワーキングレジスタ (W0-W7) は関数パラメータ用に使用されます。パラメータは、レジスタ内で左から右へ順に割り当てられ、一つのパラメータは、適切に整列された最初に利用できるレジスタに割り当てられます。

以下の例では、すべてのパラメータはレジスタに引き渡されますが、宣言に現れる順番通りではありません。このフォーマットを使用すると、MPLAB C30 コンパイラが、利用できるパラメータレジスタを最も効率的に使用できます。

例 4-1: 関数コールモデル

```
void
params0(short p0, long p1, int p2, char p3, float p4, void *p5)
{
    /*
    ** W0          p0
    ** W1          p2
    ** W3:W2      p1
    ** W4          p3
    ** W5          p5
    ** W7:W6      p4
    */
    ...
}
```

次の例は、構造がどのように関数に引き渡されるかを示しています。構造全体が利用できるレジスタ内に適合していれば、構造はレジスタ経由で引き渡されます。そうでない場合には、構造の引数はスタックの上に置かれます。

例 4-2: 関数コールモデルと構造の引渡し

```
typedef struct bar {
    int i;
    double d;
} bar;

void
params1(int i, bar b) {
    /*
     ** W0          i
     ** W1          b.i
     ** W5:W2      b.d
     */
}
```

可変長引数リストの省略(...)に対応するパラメータはレジスタには割り当てられません。レジスタに割り当てられないパラメータは、右から左の順にスタックの上にプッシュされます。

次の例では、構造パラメータが大きすぎるため、レジスタに配置できませんが、それ以降のパラメータはレジスタスポットを使用できます。

例 4-3: 関数コールモデルとスタックをベースとした引数

```
typedef struct bar {
    double d,e;
} bar;

void
params2(int i, bar b, int j) {
    /*
     ** W0          i
     ** stack      b
     ** W1          j
     */
}
```

スタックの上に置かれる引数へのアクセスは、フレームポインタが生成されるかどうかによって依存します。一般的にコンパイラは指定のない限りフレームポインタを生成し、スタックを基にしたパラメータはフレームポインタ (W14) 経由でアクセスできます。上の例では、b は W14-22 からアクセスできます。フレームポインタのオフセット、-22 は、2 バイトを前の FP 用、4 バイトを戻りアドレス用、16 バイトを b 用にとることで計算されます (図 4-6 を参照ください)。

フレームポインタを使用しない場合は、アセンブリプログラマは、エントリからプロシジャまでにいくつのスタック空間が使用されるかを把握しておく必要があります。もしそれ以上のスペースを使用しない場合、計算は上記の例と同様で、b は W15-20 経由でアクセスでき、4 バイトが戻りアドレス用、16 バイトが b のスタートをアクセスに使用されます。

4.12.2 戻り値

関数戻り値は、8 もしくは 16 ビットのスカラーは W0 に、32 ビットのスカラーは W1:W0 に、64 ビットのスカラーは W3:W2:W1:W0 に戻ります。集合は間接的に W0 経由で戻り、それは集合値のアドレスを含むように関数コーラーにより設定されます。

4.12.3 複数の関数コールにまたがるレジスタの保存

コンパイラは W8-W15 レジスタが、複数の通常の関数コールにまたがって保存されるようアレンジします。レジスタ W0-W7 は、寄せ集めのレジスタとして利用できません。割り込み関数については、コンパイラが必要なレジスタすべて、すなわち、W0-W15 と RCOUNT が保存されるようアレンジします。

4.13 レジスタの規則

特定のレジスタは C 実行時環境中で特定の役割があります。レジスタ変数は、表 4-4 に示されるよう、1 つまたは複数のワーキングレジスタを使用します。

表 4-4: レジスタの規則

変数	ワーキングレジスタ
char, signed char, unsigned char	フレームポインタとして使用されなければ W0-W13、および W14。
short, signed short, unsigned short	フレームポインタとして使用されなければ W0-W13、および W14。
int, signed int, unsigned int	フレームポインタとして使用されなければ W0-W13、および W14。
void * (or any pointer)	フレームポインタとして使用されなければ W0-W13、および W14。
long, signed long, unsigned long	連続する二つのレジスタで、最初のが {W0, W2, W4, W6, W8, W10, W12} からのレジスタ。低い番号のレジスタが値の下位 16 ビットを含みます。
long long, signed long long, unsigned long long	四つの連続するレジスタで、最初のが {W0, W4, W8} からのレジスタ。低い番号のレジスタが値の下位 16 ビットを含みます。続く高い番号のレジスタが、続く上位のビットを含みます。
float	連続する二つのレジスタで、最初のが {W0, W2, W4, W6, W8, W10, W12} からのレジスタ。低い番号のレジスタが値の下位 16 ビットを含みます。
double*	連続する二つのレジスタで、最初のが {W0, W2, W4, W6, W8, W10, W12} からのレジスタ。低い番号のレジスタが上位 16 ビットを含みます。
long double	四つの連続するレジスタで、最初のが {W0, W4, W8} からのレジスタ。低い番号のレジスタが値の下位 16 ビットを含みます。続く高い番号のレジスタが、続く上位のビットを含みます。

* double は -fno-short-double が使用されている場合は long double と同等です。

4.14 ビット反転とモジュールアドレッシング

コンパイラはビット反転やモジュールアドレッシングの使用は直接サポートしません。このいずれかのアドレッシングがレジスタに対して有効な場合、コンパイラが、そのレジスタをポインタとして使用しないようプログラムする必要があります。これらのアドレッシングモードのうちの1つが有効の時に割り込みが発生する場合には、特に注意してください。

モジュールアドレッシングに適したメモリ配列のために、C 内の配列を定義できません。増分モジュールバッファとして使用する配列を定義するのに `aligned` 属性が使用できます。減分モジュールバッファとして使用する配列を定義するには、`reverse` 属性が使用できます。属性については、**セクション 2.3 「キーワードの違い」** を参照してください。モジュールアドレッシングについては、*dsPIC30F Family Reference Manual (DS70046)* の第 3 章を参照してください。

4.15 PROGRAM SPACE VISIBILITY (PSV) の使用方法

デフォルトでは、コンパイラは、ストリングと `const`-修飾子付き初期化された変数を `.const` セクション内に割り当てるよう自動的にアレンジします。`.const` セクションは PSV ウィンドウにマッピングされます。PSV 管理は、コンパイラ管理に託され、それを移動せず、PSV ウィンドウ自体のサイズでアクセス可能なプログラムのサイズを制限します。

また、アプリケーションが、それ自体の目的のために自ら、PSV ウィンドウの制御を行う場合もあります。アプリケーションの中で PSV を直接コントロールを行う長所として、PSV ウィンドウに永久にマッピングされる 1 つの `.const` セクションを持つよりもさらに使用しやすくなります。短所は、アプリケーションが PSV コントロールレジスタとビットを管理する必要がある点です。PSV ウィンドウを使用しないようコンパイラに指示するには、`-mconst-in-data` オプションを指定します。

`space` 属性は PSV ウィンドウ内に配置する変数を定義します。コンパイラが管理するセクション `.const` 内に特定の変数を配置するには、`space (auto_psv)` 属性を使用します。コンパイラで管理していないセクション内の PSV 領域に変数を配置するには、`space (psv)` 属性を使用します。属性については、**セクション 2.3 「キーワードの違い」** を参照してください。

PSV 使用については、*MPLAB[®] ASM30, MPLAB LINK30 and Utilities User's Guide (DS51317)* をご参照ください。

メモ:

第 5 章 . データタイプ

5.1 序章

本章では MPLAB C30 データタイプについて説明します。

5.2 ハイライト

本章で説明される項目は以下の通りです。

- データの表現
- 整数
- 浮動小数点
- ポインタ

5.3 データの表現

複数バイトの量が「little endian」フォーマットで格納され、次の意味を示します。

- 最低位のアドレスに下位 8 ビットが格納されます。
- 最下位の番号をつけられたビット位置に最下位ビットが格納されます。

例として、0x12345678 のロング値はアドレス 0x100 に以下のように格納されます。

0x100	0x78	0x56	0x101
0x102	0x34	0x12	0x103

別の例として、0x12345678 のロング値はレジスタ w4 と w5 に以下のように格納されます。

w4	w5
0x5678	0x1234

5.4 整数

表 5-1 に MPLAB C30 でサポートされる整数データタイプを示します。

表 5-1: 整数データタイプ

タイプ	ビット	最小	最大
char, signed char	8	-128	127
unsigned char	8	0	255
short, signed short	16	-32768	32767
unsigned short	16	0	65535
int, signed int	16	-32768	32767
unsigned int	16	0	65535
long, signed long	32	-2 ³¹	2 ³¹ - 1
unsigned long	32	0	2 ³² - 1
long long**, signed long long**	64	-2 ⁶³	2 ⁶³ - 1
unsigned long long**	64	0	2 ⁶⁴ - 1

** ANSI-89 extension

実装時定義された、整数の動作については **セクション A.6「整数」** をご参照ください。

5.5 浮動小数点

MPLAB C30 は IEEE-754 フォーマットを使用します。表 5-2 にサポートされる浮動小数点データタイプを示します。

表 5-2: 浮動小数点データタイプ

タイプ	ビット	E 最小	E 最大	N 最小	N 最大
float	32	-126	127	2^{-126}	2^{128}
double*	32	-126	127	2^{-126}	2^{128}
long double	64	-1022	1023	2^{-1022}	2^{1024}

E = 指数

N = 正規化された (概数)

* double は `-fno-short-double` が使用されている場合は long double と同等です。

実装時定義された浮動小数点数番号の動作については、**セクション A.7「浮動小数点」** の章をご参照ください。

5.6 ポインタ

すべての MPLAB C30 ポインタは、16 ビット幅です。これは全データ空間 (64 KB) のアクセスとスモールコードモデル (32K のコード) には十分です。ラージコードモデル (>32K ワードのコード) では、ポインタがハンドラーに委ねます。つまり、ポインタが、プログラム空間の最初の 32K ワードに位置付けられる GOTO 命令用のアドレスになります。

第 6 章 . デバイスサポートファイル

6.1 序章

本章では MPLAB C30 コンパイルのサポートで使用するデバイスファイルについて説明します。

6.2 ハイライト

本章で説明する項目は以下の通りです。

- プロセッサヘッダファイル
- レジスタ定義ファイル
- SFR の使用
- マクロの使用
- C コードからの EEDATA へのアクセス - dsPIC30F DSC のみ

6.3 プロセッサヘッダファイル

プロセッサヘッダファイルは言語ツールで配布されます。これらのヘッダファイルは、各 dsPIC DSC デバイスに使用する特別関数レジスタ (SFR) を定義します。C の中でヘッダファイルを使用するには、

```
#include <p30fxxxx.h>
```

を使用します。

ここで xxxx はデバイスパーツ番号に対応します。C ヘッダファイルは、support\h ディレクトリに配置されます。

ヘッダファイルのインクルードは、SFR 名 (例えば CORCON ビット) の使用に必要です。

例えば、以下の PIC30F2010 用にコンパイルモジュールは 2 つの関数を含みます。1 つは PSV ウィンドウを有効にするものと、もう 1 つは PSV ウィンドウを無効にするものです。

```
#include <p30f2010.h>
void
EnablePSV(void)
{
    CORCONbits.PSV = 1;
}
void
DisablePSV(void)
{
    CORCONbits.PSV = 0;
}
```

プロセッサヘッダファイルの規則は、それぞれの SFR が、そのデバイスのデータシート内に現れるものと同じ名前を使用します。例えば、コアコントロールレジスタは CORCON となります。レジスタが関連のある個別のビットを持つ場合、その SFR のために定義された構造体があり、構造体の名前は SFR の名前と同じで、「bits」が付属されます。例えば、CORCONbits はコア制御レジスタの名前です。個別のビット（もしくはビットフィールド）はデータシート内の名前を使用して、構造体の中で名づけます。例えば CORCON レジスタの PSV ビットは PSV と名づけます。以下に CORCON の全定義を示します（変更もあります）。

```
/* CORCON: CPU Mode control Register */
extern volatile unsigned int CORCON __attribute__((__near__));
typedef struct tagCORCONBITS {
    unsigned IF      :1; /* Integer/Fractional mode          */
    unsigned RND     :1; /* Rounding mode                    */
    unsigned PSV     :1; /* Program Space Visibility enable  */
    unsigned IPL3    :1;
    unsigned ACCSAT  :1; /* Acc saturation mode              */
    unsigned SATDW   :1; /* Data space write saturation enable */
    unsigned SATB    :1; /* Acc B saturation enable          */
    unsigned SATA    :1; /* Acc A saturation enable          */
    unsigned DL      :3; /* DO loop nesting level status    */
    unsigned         :4;
} CORCONBITS;
extern volatile CORCONBITS CORCONbits __attribute__((__near__));
```

注： シンボル CORCON と CORCONbits は同じ名前を参照し、リンク時に同じアドレスを決定します。

6.4 レジスタ定義ファイル

セクション 6.3 「プロセッサヘッダファイル」で記述されているプロセッサヘッダファイルはそれぞれの製品のすべての SFR を名づけますが、SFR のアドレスを定義するものではありません。デバイス特有のリンクスクリプトファイルが個別にそれぞれに用意され、support\gld ディレクトリに配置されています。これらのリンクスクリプトファイルが SFR アドレスを定義します。これらのファイルの 1 つを使用するには、以下のリンクコマンドラインオプションを指定します。

```
-T p30fxxxx.gld
```

ここで、xxxx はデバイス製品番号に相当します。

例えば、app2010.c と名づけられたファイルが存在すると仮定し、それが PIC30F2010 製品のアプリケーションを含むとすると、以下のコマンドラインを用いてコンパイル、リンクが実行されます。

```
pic30-gcc -o app2010.o -T p30f2010.gld app2010.c
```

-o コマンドラインオプションは、出力 COFF 実行可能ファイルに名前をつけ、-T オプションは PIC30F2010 という名前を付けます。p30f2010.gld が実行ディレクトリにない場合は、リンクは既知のライブラリパスの中を検索します。デフォルトのインストールでは、リンクスクリプトは PIC30_LIBRARY_PATH に含まれます。参考として、セクション 3.6 「環境変数」をご覧ください。

6.5 SFR の使用

SFR をアプリケーション内で使用するには、3つのステップがあります。

1. 適切なデバイス用のプロセッサヘッダファイルをインクルードします。そのデバイスで利用できる SFR と一緒にソースコードを提供し、例えば、以下の文は PIC30F6014 製品用のヘッダファイルをインクルードします。

```
#include <p30f6014.h>
```

2. 他の C 変数のように SFR にアクセスします。ソースコードが SFR に書き込むか、SFR から読み出します。

例えば、以下の文は Timer1 用の特殊レジスタ内のすべてのビットをゼロにクリアします。

```
TMR1 = 0;
```

この次の文は、TICON レジスタの 15 番目のビット（「timer on」ビット）を表し、これはタイマをスタートする TON と名づけられたビットに 1 をセットします。

```
T1CONbits.TON = 1;
```

3. レジスタ定義ファイルもしくは適切なデバイス用のリンカスクリプトとリンクします。リンカは SFR のアドレスを付与します（ビット構造体が、リンク時に SFR と同じアドレスを持つ）。例 6.1 では、次のファイルを使用しています。

```
p30f6014.gld
```

リンカスクリプトの使用の詳細は、*MPLAB[®] ASM30, MPLAB LINK30 と Utilities User's Guide (DS51317)* をご参照ください。

以下の例は、サンプルのリアルタイムクロックで、複数の SFR を使用します。SFR の記述は p30f6014.h ファイルにあります。このファイルは、p30f6014.gld のデバイス特定のリンカスクリプトとリンクされます。

例 6-1: サンプルリアルタイムクロック

```
/*
** Sample Real Time Clock for dsPIC
**
** Uses Timer1, TCY clock timer mode
** and interrupt on period match
*/

#include <p30f6014.h>

/* Timer1 period for 1 ms with FOSC = 20 MHz */
#define TMR1_PERIOD 0x1388

struct clockType
{
    unsigned int timer;      /* countdown timer, milliseconds */
    unsigned int ticks;     /* absolute time, milliseconds */
    unsigned int seconds;   /* absolute time, seconds */
} volatile RTclock;

void reset_clock(void)
{
    RTclock.timer = 0;      /* clear software registers */
    RTclock.ticks = 0;
    RTclock.seconds = 0;

    TMR1 = 0;              /* clear timer1 register */
    PR1 = TMR1_PERIOD;     /* set period1 register */
    T1CONbits.TCS = 0;     /* set internal clock source */
    IPC0bits.T1IP = 4;     /* set priority level */
    IFS0bits.T1IF = 0;     /* clear interrupt flag */
    IEC0bits.T1IE = 1;     /* enable interrupts */

    SRbits.IPL = 3;       /* enable CPU priority levels 4-7*/
    T1CONbits.TON = 1;    /* start the timer*/
}

void __attribute__((__interrupt__)) _T1Interrupt(void)
{
    static int sticks=0;

    if (RTclock.timer > 0) /* if countdown timer is active */
        RTclock.timer -= 1; /* decrement it */
    RTclock.ticks++;       /* increment ticks counter */
    if (sticks++ > 1000)
    {
        /* if time to rollover */
        sticks = 0;        /* clear seconds ticks */
        RTclock.seconds++; /* and increment seconds */
    }

    IFS0bits.T1IF = 0;    /* clear interrupt flag */
    return;
}
```


6.6 マクロの使用

プロセッサヘッダファイルは、特殊レジスタ (SFR) に加えて、デジタルシグナルコントローラ (DSC) の dsPIC30F ファミリーで使用されるマクロを定義しています。

6.6.1 コンフィギュレーションビット設定マクロ

マクロを使用して、コンフィギュレーションビットを設定します。FOSC ビットを設定するために C ソースコード前に、下記のコードラインを挿入します。

```
_FOSC(CSW_FSCM_ON & EC_PLL16);
```

この場合、PLL を用いた外部クロックを 16x に設定し、クロックスイッチとフェイルセーフクロックモニタリングを有効にします。

同様に FBORPOR ビットを設定するため、以下のコードラインを挿入します。

```
_FBORPOR(PBOR_ON & BORV_27 & PWRT_ON_64 & MCLR_DIS);
```

ブラウンアウトリセットを 2.7 ボルトで有効にし、パワーアップタイマーを 64 ミリ秒に初期化し、I/O 用に MCLR ピンを使用するコンフィギュレーションに設定します。

各コンフィギュレーションビットの設定リストは、プロセッサヘッダファイルをご参照ください。

6.6.2 インラインアセンブリ使用マクロ

C 内でアセンブリコードを定義するために使用されるマクロのいくつかは以下にリストアップされています。

```
#define Nop()      {__asm__ volatile ("nop");}
#define ClrWdt()  {__asm__ volatile ("clrwdt");}
#define Sleep()  {__asm__ volatile ("pwrsav #0");}
#define Idle()   {__asm__ volatile ("pwrsav #1");}
```

6.6.3 データメモリ割り当てマクロ

データメモリ内に空間を割り当てるために使用されるマクロを示します。引数を必要とするマクロと必要としないマクロの 2 つのタイプがあります。

以下のマクロは、整列を指定する引数 N を必要とします。N は、2 の乗数で最小値は 2 です。

```
#define _XBSS(N)      __attribute__((space(xmemory), aligned(N)))
#define _XDATA(N)    __attribute__((space(xmemory), aligned(N)))
#define _YBSS(N)      __attribute__((space(ymemory), aligned(N)))
#define _YDATA(N)    __attribute__((space(ymemory), aligned(N)))
#define _EEDATA(N)   __attribute__((space(eedata), aligned(N)))
```

例えば、32 バイトアドレスに整列される、X メモリ内の初期化されない配列を宣言するには、以下のように記述します。

```
int _XBSS(32) xbuf[16];
```

特に整列を使用しない、データ EEPROM 内の初期化されない配列を宣言するには、以下のように記述します。

```
int _EEDATA(2) table1[] = {0, 1, 1, 2, 3, 5, 8, 13, 21};
```

以下のマクロは、引数を必要としません。それらは、Persistent データメモリ内もしくは near データメモリ内に変数を配置するために使用されます。

```
#define _PERSISTENT __attribute__((persistent))
#define _NEAR       __attribute__((near))
```

例えば、デバイスリセットされてもその値を保持する 2 つの変数を宣言するには、以下のように記述します。

```
int _PERSISTENT var1, var2;
```

6.6.4 ISR 宣言マクロ

以下のマクロは割り込みサービスルーチン (ISR) を宣言するために使用されます。

```
#define _ISR __attribute__((interrupt))
#define _ISRFAST __attribute__((interrupt, shadow))
```

例えば、タイマ 0 割り込み用の ISR を宣言するには、以下のように記述します。

```
void _ISR _INT0Interrupt(void);
```

例えば、高速のコンテキスト保存ができる SPI1 割り込み用の ISR を宣言するには、以下のように記述します。

```
void _ISRFAST _SPI1Interrupt(void);
```

注: ISR は、もしセクション 7.4 「割り込みベクトルを記述する」にリストアップされている予約名が使用されていると、自動的に割り込みベクタテーブルにインストールされます。



6.7 C コードからの EEDATA へのアクセス - dsPIC30F DSC のみ

MPLAB C30 はデバイスの EE データ空間内へのデータ配置を可能にする便利なマクロ定義をいくつか提供します。次の定義で非常に簡単に実行できます。

```
int _EEDATA(2) user_data[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

user_data は EE データ空間内に配置され、初期値で 10 ワードを確保します。

dsPIC DSC デバイスはメモリのこの空間に二つのアクセスする手段をプログラマに提供します。最初の方法は、PSV ウィンドウを介した方法です。二つ目の方法は、特殊なマシン命令 (TBLRDx) を使用したやり方です。

6.7.1 PSV を介した EEDATA へのアクセス

コンパイラは通常、PSV ウィンドウを制御してプログラムメモリ内に格納された定数にアクセスします。また、それ以外にも、PSV ウィンドウは EEDATA メモリへのアクセスにも使用されます。

PSV ウィンドウを使用するには：

- PSVPAG レジスタをアクセスする先のプログラムメモリ内の適切なアドレスに設定する必要があります。EE データでは、0xFF がこれに当たりますが、__builtin_psvpage() 関数を使用するのが最も適切な方法です。
- PSV ウィンドウは CORCON レジスタの PSV ビットを設定して有効化する必要があります。このビットが設定されていない場合、PSV ウィンドウは常に 0x0000 を読み込みます。

例 6-2: PSV を介した EEDATA へのアクセス

```
#include <p30fxxxx.h>
int main(void) {
    PSVPAG = __builtin_psvpage(&user_data);
    CORCONbits.PSV = 1;

    /* ... */

    if (user_data[2]) ;/* do something */
}
```

この手順は1度だけ実行します。PSVPAGが変更されない限り、EEデータ空間内の変数は、例で示されるように、通常のC変数としてこれらを参照することで読み込まれます。

注: このアクセスモデルはコンパイラ管理のPSV (-mconst-in-code) モデルとの互換性はありません。競合が発生しないよう注意してください。

6.7.2 TBLRDx 命令を使用した EEDATA へのアクセス

TBLRDx 命令はコンパイラで直接サポートされていませんが、インラインアセンブリを介して使用できます。PSV アクセスのように、23-ビットアドレスはSFR値とエンコードされる命令の一部から形成されます。前述の例と同じメモリにアクセスするには、以下のコードを使用できます。

TBLRDx 命令を使用するには：

- TBLPAG レジスタをアクセスする先のプログラムメモリ内の適切なアドレスに設定する必要があります。EEデータでは、これは0x7Fに当たりますが、`__builtin_tblpage()` 関数を使用するのが最も適切な方法です。
- TBLRDx 命令は `__asm__` ステートメントのみからアクセスできます。この命令については、*dsPIC30F/dsPIC33F Programmers Reference Manual (DS70157)* を参照してください。

例 6-3: テーブル読み込みを介した EEDATA へのアクセス

```
#include <p30fxxxx.h>
#define eedata_read(src, dest) { \
    register int eedata_addr; \
    register int eedata_val; \
    \
    eedata_addr = __builtin_tbloffset(&src); \
    __asm__ ("tblrdl [%1], %0" : "=r"(eedata_val) : "r"(eedata_addr)); \
    dest = eedata_val; \
}

int main(void) {
    int value;

    TBLPAG = __builtin_tblpage(&user_data);

    eedata_read(user_data[2], value);
    if (value) ; /* do something */

}
```

6.7.3 その他の情報

dsPIC30F Family Reference Manual (DS70046) の第5章では、dsPIC DSC デバイスで提供される FLASH プログラムメモリと EE データメモリの使用について説明しています。また、プログラムメモリと EE データメモリ両方のランタイムプログラミングについても説明しています。

メモ:

第 7 章 . 割り込み

7.1 序章

割り込み処理は、マイクロコントローラアプリケーションの重要なタスクです。割り込みは、リアルタイムで発生するイベントとソフトウェア動作との同期を取るために実行されます。割り込みが発生すると、通常のソフトウェア実行フローは中断され、特殊関数が起動し、そのイベントが処理されます。割り込み処理が終了すると、それ以前のコンテキスト情報が復帰され通常の実行が継続します。

dsPIC30F デバイスは、内部と外部両方のソースからの複数割り込みをサポートします。また、実行中の低プライオリティ割り込みはいつでも高プライオリティ割り込みに上書きされます。

MPLAB C30 コンパイラは C もしくはインラインアセンブリコードの割り込み処理をフルサポートします。この章では割り込み処理の概要について説明します。

7.2 ハイライト

この章では以下の項目について説明します。

- **割り込みサービスルーチンを記述する** — つもしくはそれ以上の C 関数を割り込みサービスルーチン (ISR) として指定し、割り込みの発生により起動させます。性能を高めるためには、長時間を要する計算もしくはライブラリコールを必要とするオペレーションはメインアプリケーション内に置きます。この方針は性能を最適化し、割り込みイベントが急に発生した時に、情報を失う可能性を最小限に抑えます。
- **割り込みベクトルを記述する** — dsPIC30F デバイスでは、割り込みが発生時にアプリケーションコントロールの転送用に割り込みベクタを使用します。割り込みベクタは、ISR ごとくアドレスを指定し、プログラムメモリ内の固定位置に配置されます。アプリケーションは、割り込みを使用するために、これらの位置に有効な関数アドレスを設定します。
- **割り込みサービスルーチンのコンテキスト保存** — 割り込みから、割り込み以前と同じ条件状態で戻るよう特殊レジスタなどのコンテキスト情報を保存する必要があります。
- **レイテンシー** — 割り込みがコールされた時から、最初の ISR 命令が実行されるまでの時間が、割り込みのレイテンシです。
- **割り込みのネスティング** — MPLAB C30 はネスティングされた割り込みをサポートします。
- **割り込みの有効化 / 無効化** — 割り込みソースの許可および禁止は二つのレベル、グローバルと個別的で行います。
- **割り込みサービスルーチンとメインラインコード間のメモリの共有** — この方法を使用した場合の潜在的な影響を緩和する方法

7.3 割り込みサービスルーチンを記述する

この章のガイドラインにしたがって、割り込みサービスルーチン (ISR) を含む C 言語構文のみを使用して、すべてのアプリケーションコードを記述することができます。

7.3.1 ISR を記述するためのガイドライン

ISR を記述するためのガイドラインは以下の通りです。

- パラメータ無しで、void 戻りタイプを宣言する。(必須)
- ISR がメインのラインコードでコールされないようにする。(必須)
- ISR が、その他の関数をコールしないようにする。(推奨)

MPLAB C30 ISR はその他の C 関数同様、その中にローカル変数を持ったり、グローバル変数にアクセスしたりしますが、ISR は、パラメータ無し、戻り値無しで宣言する必要があります。つまり、ハードウェア割り込みもしくはトラップへの対応として、メインラインの C プログラムに対して非同期的に起動されるためです (すなわち、通常の方法ではコールされず、そのためパラメータや戻り値は適用されない)。

ISR は、ハードウェア割り込みもしくはトラップだけで起動させ、その他の C 関数から起動しないでください。ISR は、関数から抜け出るためには、通常の RETURN 命令ではなく割り込みからの戻り命令 (RETFIE) を使用します。割り込み処理以外で RETFIE 命令を使用すると、ステータスレジスタなどプロセッサリソースを破壊します。

最後に、ISR は他の関数をコールしないでください。これは、レイテンシの問題で推奨されています。詳細は、**セクション 7.6 「レイテンシ」**をご参照ください。

7.3.2 ISR を記述するための構文

C 関数を割り込みハンドラとして宣言するには、関数に割り込み属性でタグ付けします (§ 2.3、`__attribute__` キーワードの説明を参照)。割り込み属性の構文は以下の通りです。

```
__attribute__((interrupt [(  
    [ save(symbol-list)]  
    [, irq(irqid)]  
    [, altirq(altirqid)]  
    [, preprologue(asm)]  
    ]))  
))
```

`interrupt` 属性の名前とパラメータの名前は、名前の前後に一对のアンダースコア文字をつけて記述できます。したがって、`interrupt` と `__interrupt__` は同等であり、`save` と `__save__` は同等です。

オプションの `save` パラメータは、ISR への入り口と ISR からの出口で保存、回復されるべき 1 つあるいは複数の変数のリストを指定します。この名前リストは、括弧内に記載し、名前はコンマで区切ります。

値が外部で使用されないためには、ISR 内で変更される可能性のあるグローバル変数を保存する必要があり、ISR 内で変更されるグローバル変数は、`volatile` の属性をもつ必要があります。

オプションの `irq` パラメータは、指定した割り込みの割り込みベクタを指定し、オプションの `altirq` パラメータは、指定割り込みの代替割り込みベクタを指定します。それぞれのパラメータは、括弧でくくられた ID 番号を必要とします。(割り込み ID のリストに関しては、**セクション 7.4 「割り込みベクトルを記述する」**を参照してください。)

オプションの `preprologue` パラメータは、コンパイラが生成する関数 `preprologue` の直前に、アセンブリ言語文コードに挿入します。

7.3.3 ISR のコーディング

以下のプロトタイプは、関数 `isr0` が割り込みハンドラであることを宣言します。

```
void __attribute__((__interrupt__)) isr0(void);
```

このプロトタイプが示すように、割り込み関数は、パラメータを取らず、値を戻したりもしません。コンパイラはすべてのワーキングレジスタを保存し、必要に応じて、ステータスレジスタやリピートカウンタレジスタも保存します。その他の変数は、`interrupt` 属性のパラメータとして名前を指定して保存されます。例えば、コンパイラが自動的に変数 `var1` や `var2` を保存、回復するためには、以下のプロトタイプを使用します。

```
void __attribute__((__interrupt__(__save__(var1,var2)))) isr0(void);
```

コンパイラが (`push.s` と `pop.s` 命令を用いて) 高速コンテキスト保存を要求する場合には、関数に `shadow` 属性を付けます。**セクション 2.3.2 「関数の属性を指定する」**を参照してください。) 例えば以下のように書きます。

```
void __attribute__((__interrupt__, __shadow__)) isr0(void);
```

7.3.4 マクロを用いて単純 ISR を宣言する

割り込みハンドラが、割り込み属性のオプションパラメータを必要としない場合、簡略化した構文が使用できます。以下のマクロが、デバイス特有のヘッダファイル内で定義されています。

```
#define _ISR __attribute__((interrupt))
#define _ISRFAST __attribute__((interrupt, shadow))
```

例えば、タイマ 0 割り込み用の割り込みハンドラを宣言するには、以下のように記述します。

```
#include <p30fxxxx.h>
void _ISR _INT0Interrupt(void);
```

高速コンテキスト保存を使用する SPI1 割り込み用の割り込みハンドラを宣言するには、以下のように書きます。

```
#include <p30fxxxx.h>
void _ISRFAST _SPI1Interrupt(void);
```

7.4 割り込みベクトルを記述する

dsPIC30F/33F DSC および PIC24F/H MCU デバイスは 2 つの割り込みベクトル表、プライマリテーブルと代替テーブルを持っています。それぞれの表は割り込みベクトルを含みます。

割り込みソースは、プライマリと代替割り込みベクトルと関連づけられ、以下の表にあるようにプログラムワードを占有します。代替ベクトル名は、INTCON2 レジスタ内の ALTIVT ビットが設定される時に使用されます。

注: デバイスリセットは割り込みベクトル表で取り扱われません。その代わり、デバイスリセットにより、プログラムカウンタがクリアされます。これにより、プロセッサがアドレスゼロから実行を始めるようにします。通常リンクスクリプトは、C 実行時のスタートアップモジュールをジャンプ先とする GOTO 命令を 0 番地に配置します。

- 表 7-1 割り込みベクタ - dsPIC30F DSC (非 SMPS)
- 表 7-2 割り込みベクタ - dsPIC30F DSC (SMPS)
- 表 7-3 割り込みベクタ - PIC24F MCU
- 表 7-4 割り込みベクタ - dsPIC33F DSC/PIC24H MCU

表 7-1: 割り込みベクタ - dsPIC30F DSC (非 SMPS)

IRQ#	Primary 名	Alternate 名	ベクタ関数
該当なし	_ReservedTrap0	_AltReservedTrap0	予約
該当なし	_OscillatorFail	_AltOscillatorFail	Oscillator 異常トラップ
該当なし	_AddressError	_AltAddressError	アドレスエラー トラップ
該当なし	_StackError	_AltStackError	スタックエラー トラップ
該当なし	_MathError	_AltMathError	算術エラー トラップ
該当なし	_ReservedTrap5	_AltReservedTrap5	予約
該当なし	_ReservedTrap6	_AltReservedTrap6	予約
該当なし	_ReservedTrap7	_AltReservedTrap7	予約
0	_INT0Interrupt	_AltINT0Interrupt	INT0 外部割り込み 0
1	_IC1Interrupt	_AltIC1Interrupt	IC1 入力キャプチャ 1
2	_OC1Interrupt	_AltOC1Interrupt	OC1 出力比較 1
3	_T1Interrupt	_AltT1Interrupt	TMR1 タイマ 1
4	_IC2Interrupt	_AltIC2Interrupt	IC2 入力キャプチャ 2
5	_OC2Interrupt	_AltOC2Interrupt	OC2 出力比較 2
6	_T2Interrupt	_AltT2Interrupt	TMR2 タイマ 2
7	_T3Interrupt	_AltT3Interrupt	TMR3 タイマ 3
8	_SPI1Interrupt	_AltSPI1Interrupt	SPI1 シリアル周辺インターフェース 1
9	_U1RXInterrupt	_AltU1RXInterrupt	UART1RX Uart 1 受信
10	_U1TXInterrupt	_AltU1TXInterrupt	UART1TX Uart 1 送信
11	_ADCInterrupt	_AltADCInterrupt	ADC 変換完了
12	_NVMInterrupt	_AltNVMInterrupt	NMM NVM 書き込み完了
13	_SI2CInterrupt	_AltSI2CInterrupt	Slave I ² C™ 割り込み
14	_MI2CInterrupt	_AltMI2CInterrupt	Master I ² C™ 割り込み
15	_CNInterrupt	_AltCNInterrupt	CN 入力変化割り込み
16	_INT1Interrupt	_AltINT1Interrupt	INT1 外部割り込み 1
17	_IC7Interrupt	_AltIC7Interrupt	IC7 入力キャプチャ 7
18	_IC8Interrupt	_AltIC8Interrupt	IC8 入力キャプチャ 8

表 7-1: 割り込みベクタ - dsPIC30F DSC (非 SMPS) (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
19	_OC3Interrupt	_AltOC3Interrupt	OC3 出力比較 3
20	_OC4Interrupt	_AltOC4Interrupt	OC4 出力比較 4
21	_T4Interrupt	_AltT4Interrupt	TMR4 タイマ 4
22	_T5Interrupt	_AltT5Interrupt	TMR5 タイマ 5
23	_INT2Interrupt	_AltINT2Interrupt	INT2 外部割り込み 2
24	_U2RXInterrupt	_AltU2RXInterrupt	UART2RX Uart 2 受信
25	_U2TXInterrupt	_AltU2TXInterrupt	UART2TX Uart 2 送信
26	_SPI2Interrupt	_AltSPI2Interrupt	SPI2 シリアル周辺インターフェース 2
27	_C1Interrupt	_AltC1Interrupt	CAN1 複合 IRQ
28	_IC3Interrupt	_AltIC3Interrupt	IC3 入力キャプチャ 3
29	_IC4Interrupt	_AltIC4Interrupt	IC4 入力キャプチャ 4
30	_IC5Interrupt	_AltIC5Interrupt	IC5 入力キャプチャ 5
31	_IC6Interrupt	_AltIC6Interrupt	IC6 入力キャプチャ 6
32	_OC5Interrupt	_AltOC5Interrupt	OC5 出力比較 5
33	_OC6Interrupt	_AltOC6Interrupt	OC6 出力比較 6
34	_OC7Interrupt	_AltOC7Interrupt	OC7 出力比較 7
35	_OC8Interrupt	_AltOC8Interrupt	OC8 出力比較 8
36	_INT3Interrupt	_AltINT3Interrupt	INT3 外部割り込み 3
37	_INT4Interrupt	_AltINT4Interrupt	INT4 外部割り込み 4
38	_C2Interrupt	_AltC2Interrupt	CAN2 複合 IRQ
39	_PWMInterrupt	_AltPWMInterrupt	PWM 周期一致
40	_QEInterrupt	_AltQEInterrupt	QEI ポジション カウンタ比較
41	_DCIInterrupt	_AltDCIInterrupt	DCI CODEC 転送完了
42	_LVDInterrupt	_AltLVDInterrupt	PLVD 低電圧検出
43	_FLTAInterrupt	_AltFLTAInterrupt	FLTA MCPWM 異常信号 A
44	_FLTBInterrupt	_AltFLTBInterrupt	FLTB MCPWM 異常信号 B
45	_Interrupt45	_AltInterrupt45	予約
46	_Interrupt46	_AltInterrupt46	予約
47	_Interrupt47	_AltInterrupt47	予約
48	_Interrupt48	_AltInterrupt48	予約
49	_Interrupt49	_AltInterrupt49	予約
50	_Interrupt50	_AltInterrupt50	予約
51	_Interrupt51	_AltInterrupt51	予約
52	_Interrupt52	_AltInterrupt52	予約
53	_Interrupt53	_AltInterrupt53	予約

表 7-2: 割り込みベクタ - dsPIC30F DSC (SMPS)

IRQ#	Primary 名	Alternate 名	ベクタ関数
該当なし	_ReservedTrap0	_AltReservedTrap0	予約
該当なし	_OscillatorFail	_AltOscillatorFail	Oscillator 異常トラップ
該当なし	_AddressError	_AltAddressError	アドレスエラー トラップ
該当なし	_StackError	_AltStackError	スタックエラー トラップ
該当なし	_MathError	_AltMathError	算術エラー トラップ
該当なし	_ReservedTrap5	_AltReservedTrap5	予約

表 7-2: 割り込みベクタ - dsPIC30F DSC (SMPS) (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
該当なし	_ReservedTrap6	_AltReservedTrap6	予約
該当なし	_ReservedTrap7	_AltReservedTrap7	予約
0	_INT0Interrupt	_AltINT0Interrupt	INT0 外部割り込み 0
1	_IC1Interrupt	_AltIC1Interrupt	IC1 入力キャプチャ 1
2	_OC1Interrupt	_AltOC1Interrupt	OC1 出力比較 1
3	_T1Interrupt	_AltT1Interrupt	TMR1 タイマ 1
4	_Interrupt4	_AltInterrupt4	予約
5	_OC2Interrupt	_AltOC2Interrupt	OC2 出力比較 2
6	_T2Interrupt	_AltT2Interrupt	TMR2 タイマ 2
7	_T3Interrupt	_AltT3Interrupt	TMR3 タイマ 3
8	_SPI1Interrupt	_AltSPI1Interrupt	SPI1 シリアル周辺インターフェース 1
9	_U1RXInterrupt	_AltU1RXInterrupt	UART1RX Uart 1 受信
10	_U1TXInterrupt	_AltU1TXInterrupt	UART1TX Uart 1 送信
11	_ADCInterrupt	_AltADCInterrupt	ADC 変換完了
12	_NVMInterrupt	_AltNVMInterrupt	NVM 書き込み完了
13	_I2CInterrupt	_AltI2CInterrupt	I ² C™ 割り込み
14	_I2CErrInterrupt	_AltI2CErrInterrupt	I ² C™ エラー割り込み
15	_Interrupt15	_AltInterrupt15	予約
16	_INT1Interrupt	_AltINT1Interrupt	INT1 外部割り込み 1
17	_INT2Interrupt	_AltINT2Interrupt	INT2 外部割り込み 2
18	_PWMSpEvent MatchInterrupt	_AltPWMSpEvent MatchInterrupt	PWM 特殊イベント割り込み
19	_PWM1Interrupt	_AltPWM1Interrupt	PWM 周期一致 1
20	_PWM2Interrupt	_AltPWM2Interrupt	PWM 周期一致 2
21	_PWM3Interrupt	_AltPWM3Interrupt	PWM 周期一致 3
22	_PWM4Interrupt	_AltPWM4Interrupt	PWM 周期一致 4
23	_Interrupt23	_AltInterrupt23	予約
24	_Interrupt24	_AltInterrupt24	予約
25	_Interrupt25	_AltInterrupt25	予約
26	_Interrupt26	_AltInterrupt26	予約
27	_Interrupt27	_AltInterrupt27	予約
28	_Interrupt28	_AltInterrupt28	予約
29	_CMP1Interrupt	_AltCMP1Interrupt	アナログ コンパレータ割り込み 1
30	_CMP2Interrupt	_AltCMP2Interrupt	アナログ コンパレータ割り込み 2
31	_CMP3Interrupt	_AltCMP3Interrupt	アナログ コンパレータ割り込み 3
32	_CMP4Interrupt	_AltCMP4Interrupt	アナログ コンパレータ割り込み 4
33	_Interrupt33	_AltInterrupt33	予約
34	_Interrupt34	_AltInterrupt34	予約
35	_Interrupt35	_AltInterrupt35	予約
36	_Interrupt36	_AltInterrupt36	予約
37	_Interrupt37	_AltInterrupt37	予約
38	_Interrupt38	_AltInterrupt38	予約
39	_Interrupt39	_AltInterrupt39	予約
40	_Interrupt40	_AltInterrupt40	予約

表 7-2: 割り込みベクタ - dsPIC30F DSC (SMPS) (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
41	_Interrupt41	_AltInterrupt41	予約
42	_Interrupt42	_AltInterrupt42	予約
43	_Interrupt43	_AltInterrupt43	予約
44	_Interrupt44	_AltInterrupt44	予約
45	_Interrupt45	_AltInterrupt45	予約
46	_Interrupt46	_AltInterrupt46	予約
47	_Interrupt47	_AltInterrupt47	予約
48	_Interrupt48	_AltInterrupt48	予約
49	_Interrupt49	_AltInterrupt49	予約
50	_Interrupt50	_AltInterrupt50	予約
51	_Interrupt51	_AltInterrupt51	予約
52	_Interrupt52	_AltInterrupt52	予約
53	_Interrupt53	_AltInterrupt53	予約

表 7-3: 割り込みベクタ - PIC24F MCU

IRQ#	Primary 名	Alternate 名	ベクタ関数
該当なし	_ReservedTrap0	_AltReservedTrap0	予約
該当なし	_OscillatorFail	_AltOscillatorFail	Oscillator 異常トラップ
該当なし	_AddressError	_AltAddressError	アドレスエラー トラップ
該当なし	_StackError	_AltStackError	スタックエラー トラップ
該当なし	_MathError	_AltMathError	算術エラー トラップ
該当なし	_ReservedTrap5	_AltReservedTrap5	予約
該当なし	_ReservedTrap6	_AltReservedTrap6	予約
該当なし	_ReservedTrap7	_AltReservedTrap7	予約
0	_INT0Interrupt	_AltINT0Interrupt	INT0 外部割り込み 0
1	_IC1Interrupt	_AltIC1Interrupt	IC1 入力キャプチャ 1
2	_OC1Interrupt	_AltOC1Interrupt	OC1 出力比較 1
3	_T1Interrupt	_AltT1Interrupt	TMR1 タイマ 1
4	_Interrupt4	_AltInterrupt4	予約
5	_IC2Interrupt	_AltIC2Interrupt	IC2 入力キャプチャ 2
6	_OC2Interrupt	_AltOC2Interrupt	OC2 出力比較 2
7	_T2Interrupt	_AltT2Interrupt	TMR2 タイマ 2
8	_T3Interrupt	_AltT3Interrupt	TMR3 タイマ 3
9	_SPI1ErrInterrupt	_AltSPI1ErrInterrupt	SPI1 エラー割り込み
10	_SPI1Interrupt	_AltSPI1Interrupt	SPI1 転送完了割り込み
11	_U1RXInterrupt	_AltU1RXInterrupt	UART1RX Uart 1 受信
12	_U1TXInterrupt	_AltU1TXInterrupt	UART1TX Uart 1 送信
13	_ADC1Interrupt	_AltADC1Interrupt	ADC 1 変換完了
14	_Interrupt14	_AltInterrupt14	予約
15	_Interrupt15	_AltInterrupt15	予約
16	_SI2C1Interrupt	_AltSI2C1Interrupt	Slave I ² C™ 割り込み 1
17	_MI2C1Interrupt	_AltMI2C1Interrupt	Master I ² C™ 割り込み 1
18	_CompInterrupt	_AltCompInterrupt	コンパレータ割り込み
19	_CNInterrupt	_AltCNInterrupt	CN 入力変化割り込み

表 7-3: 割り込みベクタ - PIC24F MCU (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
20	_INT1Interrupt	_AltINT1Interrupt	INT1 外部割り込み 1
21	_Interrupt21	_AltInterrupt21	予約
22	_Interrupt22	_AltInterrupt22	予約
23	_Interrupt23	_AltInterrupt23	予約
24	_Interrupt24	_AltInterrupt24	予約
25	_OC3Interrupt	_AltOC3Interrupt	OC3 出力比較 3
26	_OC4Interrupt	_AltOC4Interrupt	OC4 出力比較 4
27	_T4Interrupt	_AltT4Interrupt	TMR4 タイマ 4
28	_T5Interrupt	_AltT5Interrupt	TMR5 タイマ 5
29	_INT2Interrupt	_AltINT2Interrupt	INT2 外部割り込み 2
30	_U2RXInterrupt	_AltU2RXInterrupt	UART2RX Uart 2 受信
31	_U2TXInterrupt	_AltU2TXInterrupt	UART2TX Uart 2 送信
32	_SPI2ErrInterrupt	_AltSPI2ErrInterrupt	SPI2 エラー割り込み
33	_SPI2Interrupt	_AltSPI2Interrupt	SPI2 転送完了割り込み
34	_Interrupt34	_AltInterrupt34	予約
35	_Interrupt35	_AltInterrupt35	予約
36	_Interrupt36	_AltInterrupt36	予約
37	_IC3Interrupt	_AltIC3Interrupt	IC3 入力キャプチャ 3
38	_IC4Interrupt	_AltIC4Interrupt	IC4 入力キャプチャ 4
39	_IC5Interrupt	_AltIC5Interrupt	IC5 入力キャプチャ 5
40	_Interrupt40	_AltInterrupt40	予約
41	_OC5Interrupt	_AltOC5Interrupt	OC5 出力比較 5
42	_Interrupt42	_AltInterrupt42	予約
43	_Interrupt43	_AltInterrupt43	予約
44	_Interrupt44	_AltInterrupt44	予約
45	_PMPInterrupt	_AltPMPInterrupt	パラレル マスター ポート割り込み
46	_Interrupt46	_AltInterrupt46	予約
47	_Interrupt47	_AltInterrupt47	予約
48	_Interrupt48	_AltInterrupt48	予約
49	_SI2C2Interrupt	_AltSI2C2Interrupt	Slave I ² C™ 割り込み 2
50	_MI2C2Interrupt	_AltMI2C2Interrupt	Master I ² C™ 割り込み 2
51	_Interrupt51	_AltInterrupt51	予約
52	_Interrupt52	_AltInterrupt52	予約
53	_INT3Interrupt	_AltINT3Interrupt	INT3 外部割り込み 3
54	_INT4Interrupt	_AltINT4Interrupt	INT4 外部割り込み 4
55	_Interrupt55	_AltInterrupt55	予約
56	_Interrupt56	_AltInterrupt56	予約
57	_Interrupt57	_AltInterrupt57	予約
58	_Interrupt58	_AltInterrupt58	予約
59	_Interrupt59	_AltInterrupt59	予約
60	_Interrupt60	_AltInterrupt60	予約
61	_Interrupt61	_AltInterrupt61	予約
62	_RTCCInterrupt	_AltRTCCInterrupt	リアルタイム クロックとカレンダー
63	_Interrupt63	_AltInterrupt63	予約

表 7-3: 割り込みベクタ - PIC24F MCU (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
64	_Interrupt64	_AltInterrupt64	予約
65	_U1EInterrupt	_AltU1EInterrupt	UART1 エラー割り込み
66	_U2EInterrupt	_AltU2EInterrupt	UART2 エラー割り込み
67	_CRCInterrupt	_AltCRCInterrupt	巡回冗長検査 (CRC)
68	_Interrupt68	_AltInterrupt68	予約
69	_Interrupt69	_AltInterrupt69	予約
70	_Interrupt70	_AltInterrupt70	予約
71	_Interrupt71	_AltInterrupt71	予約
72	_Interrupt72	_AltInterrupt72	予約
73	_Interrupt73	_AltInterrupt73	予約
74	_Interrupt74	_AltInterrupt74	予約
75	_Interrupt75	_AltInterrupt75	予約
76	_Interrupt76	_AltInterrupt76	予約
77	_Interrupt77	_AltInterrupt77	予約
78	_Interrupt78	_AltInterrupt78	予約
79	_Interrupt79	_AltInterrupt79	予約
80	_Interrupt80	_AltInterrupt80	予約
81	_Interrupt81	_AltInterrupt81	予約
82	_Interrupt82	_AltInterrupt82	予約
83	_Interrupt83	_AltInterrupt83	予約
84	_Interrupt84	_AltInterrupt84	予約
85	_Interrupt85	_AltInterrupt85	予約
86	_Interrupt86	_AltInterrupt86	予約
87	_Interrupt87	_AltInterrupt87	予約
88	_Interrupt88	_AltInterrupt88	予約
89	_Interrupt89	_AltInterrupt89	予約
90	_Interrupt90	_AltInterrupt90	予約
91	_Interrupt91	_AltInterrupt91	予約
92	_Interrupt92	_AltInterrupt92	予約
93	_Interrupt93	_AltInterrupt93	予約
94	_Interrupt94	_AltInterrupt94	予約
95	_Interrupt95	_AltInterrupt95	予約
96	_Interrupt96	_AltInterrupt96	予約
97	_Interrupt97	_AltInterrupt97	予約
98	_Interrupt98	_AltInterrupt98	予約
99	_Interrupt99	_AltInterrupt99	予約
100	_Interrupt100	_AltInterrupt100	予約
101	_Interrupt101	_AltInterrupt101	予約
102	_Interrupt102	_AltInterrupt102	予約
103	_Interrupt103	_AltInterrupt103	予約
104	_Interrupt104	_AltInterrupt104	予約
105	_Interrupt105	_AltInterrupt105	予約
106	_Interrupt106	_AltInterrupt106	予約
107	_Interrupt107	_AltInterrupt107	予約

表 7-3: 割り込みベクタ - PIC24F MCU (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
108	_Interrupt108	_AltInterrupt108	予約
109	_Interrupt109	_AltInterrupt109	予約
110	_Interrupt110	_AltInterrupt110	予約
111	_Interrupt111	_AltInterrupt111	予約
112	_Interrupt112	_AltInterrupt112	予約
113	_Interrupt113	_AltInterrupt113	予約
114	_Interrupt114	_AltInterrupt114	予約
115	_Interrupt115	_AltInterrupt115	予約
116	_Interrupt116	_AltInterrupt116	予約
117	_Interrupt117	_AltInterrupt117	予約

表 7-4: 割り込みベクタ - dsPIC33F DSC/PIC24H MCU

IRQ#	Primary 名	Alternate 名	ベクタ関数
該当なし	_ReservedTrap0	_AltReservedTrap0	予約
該当なし	_OscillatorFail	_AltOscillatorFail	Oscillator 異常トラップ
該当なし	_AddressError	_AltAddressError	アドレスエラー トラップ
該当なし	_StackError	_AltStackError	スタックエラー トラップ
該当なし	_MathError	_AltMathError	算術エラー トラップ
該当なし	_DMACError	_AltDMACError	DMA 競合エラー トラップ
該当なし	_ReservedTrap6	_AltReservedTrap6	予約
該当なし	_ReservedTrap7	_AltReservedTrap7	予約
0	_INT0Interrupt	_AltINT0Interrupt	INT0 外部割り込み 0
1	_IC1Interrupt	_AltIC1Interrupt	IC1 入力キャプチャ 1
2	_OC1Interrupt	_AltOC1Interrupt	OC1 出力比較 1
3	_T1Interrupt	_AltT1Interrupt	TMR1 タイマ 1
4	_DMA0Interrupt	_AltDMA0Interrupt	DMA 0 割り込み
5	_IC2Interrupt	_AltIC2Interrupt	IC2 入力キャプチャ 2
6	_OC2Interrupt	_AltOC2Interrupt	OC2 出力比較 2
7	_T2Interrupt	_AltT2Interrupt	TMR2 タイマ 2
8	_T3Interrupt	_AltT3Interrupt	TMR3 タイマ 3
9	_SPI1ErrInterrupt	_AltSPI1ErrInterrupt	SPI1 エラー割り込み
10	_SPI1Interrupt	_AltSPI1Interrupt	SPI1 転送完了割り込み
11	_U1RXInterrupt	_AltU1RXInterrupt	UART1RX Uart 1 受信
12	_U1TXInterrupt	_AltU1TXInterrupt	UART1TX Uart 1 送信
13	_ADC1Interrupt	_AltADC1Interrupt	ADC 1 変換完了
14	_DMA1Interrupt	_AltDMA1Interrupt	DMA 1 割り込み
15	_Interrupt15	_AltInterrupt15	予約
16	_SI2C1Interrupt	_AltSI2C1Interrupt	Slave I ² C™ 割り込み 1
17	_MI2C1Interrupt	_AltMI2C1Interrupt	Master I ² C™ 割り込み 1
18	_Interrupt18	_AltInterrupt18	予約
19	_CNInterrupt	_AltCNInterrupt	CN 入力変化割り込み
20	_INT1Interrupt	_AltINT1Interrupt	INT1 外部割り込み 1
21	_ADC2Interrupt	_AltADC2Interrupt	ADC 2 変換完了
22	_IC7Interrupt	_AltIC7Interrupt	IC7 入力キャプチャ 7

表 7-4: 割り込みベクタ - dsPIC33F DSC/PIC24H MCU (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
23	_IC8Interrupt	_AltIC8Interrupt	IC8 入力キャプチャ 8
24	_DMA2Interrupt	_AltDMA2Interrupt	DMA 2 割り込み
25	_OC3Interrupt	_AltOC3Interrupt	OC3 出力比較 3
26	_OC4Interrupt	_AltOC4Interrupt	OC4 出力比較 4
27	_T4Interrupt	_AltT4Interrupt	TMR4 タイマ 4
28	_T5Interrupt	_AltT5Interrupt	TMR5 タイマ 5
29	_INT2Interrupt	_AltINT2Interrupt	INT2 外部割り込み 2
30	_U2RXInterrupt	_AltU2RXInterrupt	UART2RX Uart 2 受信
31	_U2TXInterrupt	_AltU2TXInterrupt	UART2TX Uart 2 送信
32	_SPI2ErrInterrupt	_AltSPI2ErrInterrupt	SPI2 エラー割り込み
33	_SPI2Interrupt	_AltSPI2Interrupt	SPI2 転送完了割り込み
34	_C1RxRdyInterrupt	_AltC1RxRdyInterrupt	CAN1 データ受信可
35	_C1Interrupt	_AltC1Interrupt	CAN1 割り込み完了
36	_DMA3Interrupt	_AltDMA3Interrupt	DMA 3 割り込み
37	_IC3Interrupt	_AltIC3Interrupt	IC3 入力キャプチャ 3
38	_IC4Interrupt	_AltIC4Interrupt	IC4 入力キャプチャ 4
39	_IC5Interrupt	_AltIC5Interrupt	IC5 入力キャプチャ 5
40	_IC6Interrupt	_AltIC6Interrupt	IC6 入力キャプチャ 6
41	_OC5Interrupt	_AltOC5Interrupt	OC5 出力比較 5
42	_OC6Interrupt	_AltOC6Interrupt	OC6 出力比較 6
43	_OC7Interrupt	_AltOC7Interrupt	OC7 出力比較 7
44	_OC8Interrupt	_AltOC8Interrupt	OC8 出力比較 8
45	_Interrupt45	_AltInterrupt45	予約
46	_DMA4Interrupt	_AltDMA4Interrupt	DMA 4 割り込み
47	_T6Interrupt	_AltT6Interrupt	TMR6 タイマ 6
48	_T7Interrupt	_AltT7Interrupt	TMR7 タイマ 7
49	_SI2C2Interrupt	_AltSI2C2Interrupt	Slave I ² C™ 割り込み 2
50	_MI2C2Interrupt	_AltMI2C2Interrupt	Master I ² C™ 割り込み 2
51	_T8Interrupt	_AltT8Interrupt	TMR8 タイマ 8
52	_T9Interrupt	_AltT9Interrupt	TMR9 タイマ 9
53	_INT3Interrupt	_AltINT3Interrupt	INT3 外部割り込み 3
54	_INT4Interrupt	_AltINT4Interrupt	INT4 外部割り込み 4
55	_C2RxRdyInterrupt	_AltC2RxRdyInterrupt	CAN2 データ受信可
56	_C2Interrupt	_AltC2Interrupt	CAN2 割り込み完了
57	_PWMInterrupt	_AltPWMInterrupt	PWM 周期一致
58	_QEInterrupt	_AltQEInterrupt	QEI ポジション カウンタ比較
59	_DCIErrInterrupt	_AltDCIErrInterrupt	DCI CODEC エラー割り込み
60	_DCIInterrupt	_AltDCIInterrupt	DCI CODEC 転送実行済み
61	_DMA5Interrupt	_AltDMA5Interrupt	DMA チャンネル 5 割り込み
62	_Interrupt62	_AltInterrupt62	予約
63	_FLTAInterrupt	_AltFLTAInterrupt	FLTA MCPWM 異常信号 A
64	_FLTBInterrupt	_AltFLTBInterrupt	FLTB MCPWM 異常信号 B
65	_U1ErrInterrupt	_AltU1ErrInterrupt	UART1 エラー割り込み
66	_U2ErrInterrupt	_AltU2ErrInterrupt	UART2 エラー割り込み

表 7-4: 割り込みベクタ - dsPIC33F DSC/PIC24H MCU (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
67	_Interrupt67	_AltInterrupt67	予約
68	_DMA6Interrupt	_AltDMA6Interrupt	DMA チャンネル 6 割り込み
69	_DMA7Interrupt	_AltDMA7Interrupt	DMA チャンネル 7 割り込み
70	_C1TxReqInterrupt	_AltC1TxReqInterrupt	CAN1 データ送信要求
71	_C2TxReqInterrupt	_AltC2TxReqInterrupt	CAN2 データ送信要求
72	_Interrupt72	_AltInterrupt72	予約
73	_Interrupt73	_AltInterrupt73	予約
74	_Interrupt74	_AltInterrupt74	予約
75	_Interrupt75	_AltInterrupt75	予約
76	_Interrupt76	_AltInterrupt76	予約
77	_Interrupt77	_AltInterrupt77	予約
78	_Interrupt78	_AltInterrupt78	予約
79	_Interrupt79	_AltInterrupt79	予約
80	_Interrupt80	_AltInterrupt80	予約
81	_Interrupt81	_AltInterrupt81	予約
82	_Interrupt82	_AltInterrupt82	予約
83	_Interrupt83	_AltInterrupt83	予約
84	_Interrupt84	_AltInterrupt84	予約
85	_Interrupt85	_AltInterrupt85	予約
86	_Interrupt86	_AltInterrupt86	予約
87	_Interrupt87	_AltInterrupt87	予約
88	_Interrupt88	_AltInterrupt88	予約
89	_Interrupt89	_AltInterrupt89	予約
90	_Interrupt90	_AltInterrupt90	予約
91	_Interrupt91	_AltInterrupt91	予約
92	_Interrupt92	_AltInterrupt92	予約
93	_Interrupt93	_AltInterrupt93	予約
94	_Interrupt94	_AltInterrupt94	予約
95	_Interrupt95	_AltInterrupt95	予約
96	_Interrupt96	_AltInterrupt96	予約
97	_Interrupt97	_AltInterrupt97	予約
98	_Interrupt98	_AltInterrupt98	予約
99	_Interrupt99	_AltInterrupt99	予約
100	_Interrupt100	_AltInterrupt100	予約
101	_Interrupt101	_AltInterrupt101	予約
102	_Interrupt102	_AltInterrupt102	予約
103	_Interrupt103	_AltInterrupt103	予約
104	_Interrupt104	_AltInterrupt104	予約
105	_Interrupt105	_AltInterrupt105	予約
106	_Interrupt106	_AltInterrupt106	予約
107	_Interrupt107	_AltInterrupt107	予約
108	_Interrupt108	_AltInterrupt108	予約
109	_Interrupt109	_AltInterrupt109	予約
110	_Interrupt110	_AltInterrupt110	予約

表 7-4: 割り込みベクタ - dsPIC33F DSC/PIC24H MCU (つづき)

IRQ#	Primary 名	Alternate 名	ベクタ関数
111	_Interrupt111	_AltInterrupt111	予約
112	_Interrupt112	_AltInterrupt112	予約
113	_Interrupt113	_AltInterrupt113	予約
114	_Interrupt114	_AltInterrupt114	予約
115	_Interrupt115	_AltInterrupt115	予約
116	_Interrupt116	_AltInterrupt116	予約
117	_Interrupt117	_AltInterrupt117	予約

割り込みを処理するため、関数アドレスはベクトル表内の対応アドレスに配置する必要があります。また、関数は使用するシステムリソースを保存し、RETFIE プロセッサ命令を用いて、以前のタスクに戻る必要があります。割り込み関数は、C で記述することもできます。C 関数が割り込みハンドラとして指定されると、コンパイラは、コンパイラが使用するすべてのシステムリソースを待避し、適切な命令を使用して関数から戻るようにアレンジします。コンパイラは、割り込み関数のアドレスで埋められた割り込みベクトル表を、オプションとして作成できます。

コンパイラが、割り込み関数を指す割り込みベクタを満たすようにアレンジするため、前記の表を参照して関数に名前をつけます。例えば、スタックエラーベクタは、以下のように関数が定義されると、自動的ベクタ表にアドレスを生成します。

```
void __attribute__((__interrupt__)) _StackError(void);
```

前にアンダースコアをつけることに注意してください。同様に `alternate` スタックエラーベクタは、以下のように関数が定義されると、自動的に処理します。

```
void __attribute__((__interrupt__)) _AltStackError(void);
```

ここでも、前にアンダースコアを使用することに注意してください。

特定のハンドラを持たないすべての割り込みベクタ用として、デフォルトの割り込みハンドラがあらかじめインストールされています。デフォルト割り込みハンドラはリンクが提供し、単にデバイスをリセットします。_DefaultInterrupt という名前を持った割り込み関数を宣言することで、アプリケーション用のデフォルトの割り込みベクタを生成することもできます。

それぞれの表の最後の 9 つの割り込みベクタは事前に定義されたハードウェア関数を持ちません。これらの割り込み用ベクタは前記の表で示した名前を用いるか、もしくはアプリケーションに最適な名前を使用します。または、割り込み属性のパラメータ `irq` もしくは `altirq` を使用して適切なベクタエントリを満たすことができます。例えば、関数が 52 のプライマリ割り込みベクタを使用するように指定するには、以下のように使用します。

```
void __attribute__((__interrupt__(__irq__(52)))) MyIRQ(void);
```

同様に、関数が 52 の代替割り込みベクタを使用するように指定するには、以下のように使用します。

```
void __attribute__((__interrupt__(__altirq__(52)))) MyAltIRQ(void);
```

`irq/altirq` に指定する数は 45 から 53 の割り込み要求の一つです。割り込み属性の `irq` パラメータを使用すると、コンパイラは外部シンボル `__Interruptn` を生成し、`n` はベクトル番号になります。したがって、C 識別子 `_Interrupt45` から `_Interrupt53` はコンパイラによって確保されています。同様に、割り込み属性の `altirq` パラメータが使用されると、`__AltInterruptn` という外部シンボルを生成し、`n` はベクタ番号です。したがって、C 識別子の `_AltInterrupt45` から `_AltInterrupt53` はコンパイラによって確保されています。

7.5 割り込みサービスルーチンのコンテキスト保存

割り込みは、まさにその性質により、予想できない時に発生します。したがって、割り込まれたコードは、割り込みが発生した時に存在したのと同じマシン状態で復帰されねばなりません。

割り込みからの戻りを正確に取り扱うためには、ISR 関数用の設定 (prologue) コードは、コンパイラが管理するワーキングレジスタと特殊レジスタをスタック上に保存し、ISR 終了時にこれらを復帰します。他の追加変数や特殊レジスタが待避、復帰されるように指定するには、interrupt 属性のオプション save パラメータを使用します。

アプリケーションによっては、アセンブリ文を、コンパイラが生成する関数プロローグの直前に、割り込みサービスルーチンに挿入することが必要です。例えば、割り込みサービスルーチンへの入り口で、semaphore をすぐに増分させることが必要であるとしてします。これは以下のように記述されます。

```
void
__attribute__((__interrupt__(__preprologue__("inc _semaphore"))))
_isr0(void);
```

7.6 レイテンシ

割り込みソースが発生する時間から ISR コードの最初の命令が実行されるまでの間のサイクル数に影響を与える二つの要素を次に示します。

- **割り込みのプロセッササービス**—割り込みを認識し、最初の割り込みベクタに分岐するために、プロセッサが必要とする総時間です。この値を決定するには、そのプロセッサと使用される割り込みソースに関して、プロセッサのデータシートを参照ください。
- **ISR コード**—MPLAB C30 は ISR 内で使用するレジスタを保存します。これは、ワーキングレジスタと RCOUNT 特殊レジスタを含みます。さらに、ISR が通常関数をコールすると、コンパイラは、すべてのワーキングレジスタと RCOUNT を (ISR ですべて確実に使用されない場合を含む) 保存します。コンパイラは、一般的にどのリソースがコールされた関数で使用されるかわからないからです。

7.7 割り込みのネスティング

dsPIC30F デバイスは割り込みのネスティングをサポートします。プロセッサリソースは ISR 内でスタック上に保存されるので、ネスティングされる ISR はネスティングされないものと同様にコーディングできます。割り込みのネスティングは、INTCON1 レジスタの NSTDIS (多重された割り込み禁止) ビットをクリアすることで有効となります。dsPIC30F デバイスは多重割り込み可能な状態でリセットから復帰するため、これがデフォルトです。それぞれの割り込みソースは、割り込みプライオリティコントロールレジスタ (IPCn) でプライオリティが割り当てられます。もし、ペンディング割り込み要求 (IRQ) のプライオリティレベルが現在のプロセッサプライオリティレベル (ST レジスタの CPUPRI ビットフィールド) と同じか、より大きい場合、割り込み要求がプロセッサに受理されます。

7.8 割り込みの有効化 / 無効化

それぞれの割り込みソースは、個別に有効化もしくは無効化できます。各 IRQ に対して1つの割り込み有効ビットが、割り込み制御レジスタ (IECn) に割り当てられます。割り込み有効ビットを1に設定すると、対応する割り込みが有効になり、割り込み有効ビットをゼロにクリアすると、対応する割り込みが無効になります。デバイスがリセットから回復した起動時には、すべての割り込み有効ビットはゼロクリアされています。また、プロセッサは、指定された命令サイクル数においてすべての割り込みを無効にする割り込み無効命令 (DISI) を持っています。

注: アドレスエラートラップのようなトラップは、無効化できませんが IRQs のみ無効化できます。

DISI 命令は、インラインアセンブリを通して、C プログラム内で使用できます。例えば、以下のようなインラインアセンブリ文は、ソースプログラム内で現れる位置で指定された DISI 命令を発行します。

```
__asm__ volatile ("disi #16");
```

DISI を上記の方法で使用する場合の欠点は、C プログラムが、C コンパイラがどのように C ソースをマシン命令に変換するかについて、常に把握してはいないという点です。したがって、DISI 命令のサイクル数の決定が困難な場合があります。DISI 命令により、割り込みから保護するコードを囲むことで、この問題を回避できます。最初の命令は、サイクル数を最大値に設定し、次の命令はサイクル数をゼロに設定します。例えば、以下のように記述します。

```
__asm__ volatile("disi #0x3FFF"); /* disable interrupts */
/* ... protected C code ... */
__asm__ volatile("disi #0x0000"); /* enable interrupts */
```

別の方法として、DISICNT レジスタに直接書き込んで、割り込みを有効化することもできます。DISICNT レジスタは、DISI 命令の発行後に DISICNT レジスタの内容がゼロでない場合のみ変更できます。

```
__asm__ volatile("disi #0x3FFF"); /* disable interrupts */
/* ... protected C code ... */
DISICNT = 0x0000; /* enable interrupts */
```

一部のアプリケーションでは、レベル7の割り込みも無効にする必要があります。これは、COROCON IPL フィールドを変更することによってのみ無効にできます。付属のサポートファイルには、安全に IPL 値を変更する場合に役立つプリプロセッサマクロ関数がいくつか含まれています。マクロは次のとおりです。

```
SET_CPU_IPL(ipl)
SET_AND_SAVE_CPU_IPL(save_to, ipl)
RESTORE_CPU_IPL(saved_to)
```

例えば、コードセクションを割り込みから保護する場合、次のコードは、現在の IPL 設定を調整し、また IPL を前の値に戻します。

```
void foo(void) {
    int current_cpu_ipl;

    SET_AND_SAVE_CPU_IPL(current_cpu_ipl, 7); /* disable interrupts */
    /* protected code here */
    RESTORE_CPU_IPL(current_cpu_ipl);
}
```

7.9 割り込みサービス ルーチンとメインライン コード間のメモリの共有

メインまたは低プライオリティ割り込みサービス ルーチン (ISR) と高プライオリティ ISR 内で同じ変数を変更する場合には、注意が必要です。高プライオリティの割り込みは有効化されると、同じ変数にアクセスする複数命令から成る読み込み - 変更 - 書き込みシーケンスを低プライオリティの関数が実行していた場合に、複数命令シーケンスに割り込むことができるため、結果が予想外となる可能性があります。したがって、組み込みシステムは、**atomic** 動作を実装することによって、低プライオリティの ISR が読み込んだ後、書き込みを未完了の変数が、介入した高プライオリティの ISR によって書き込まれないようにする必要があります。

atomic 動作は、構成要素に分解できない動作で、割り込みされません。関連する特定のアーキテクチャによっては、C の式すべてが **atomic** 動作に翻訳されるとは限りません。dsPIC DSC デバイスでは、こうした式は主に、32 ビット式、浮動小数点演算、除算、およびマルチビットのビット フィールドに対する演算というカテゴリに分類できます。他にも、メモリ モデル設定、最適化レベル、リソースの使用可能度などの要因によって、**atomic** 動作が生成されるかどうかが決まります。

一般的な式を例にして説明します。

```
foo = bar op baz;
```

演算子 (op) は、デバイスのアーキテクチャによって、**atomic** の場合とそうでない場合があります。いずれにしても、コンパイラは常に **atomic** 動作を生成できるとは限らず、次に挙げる複数の要因にかなり依存します。

- 適切な **atomic** マシン命令の有無
- リソースの使用可能度 - 特殊レジスタまたはその他の制約条件
- 最適化レベルなどの、データ / コード配置に影響するオプション

アーキテクチャを除外して考慮したた場合、この一般的な式は、2 つの読み込み (各オペランドに対して 1 つ) と、結果を格納するための 1 つの書き込みを必要とすると考えられます。割り込みシーケンスが存在すると、複数の問題が発生する場合がありますが、個々のアプリケーションによって大きく異なります。

7.9.1 問題の考察

例をいくつか示します。

例 7-1: BAR は BAZ とマッチングする必要がある

bar と baz はマッチングする (つまり、相互に同期して更新される) 必要がある場合、bar または baz のいずれかが高プライオリティの割り込み式内で更新された場合、問題が発生する可能性があります。その場合のフロー シーケンスの例をいくつか示します。

1. 安全 bar を読み込む
 baz を読み込む
 演算を実行
 結果を foo に書き戻す
2. 危険 bar を読み込む
 割り込みが baz を変更
 baz を読み込む
 演算を実行
 結果を foo に書き戻す

```
3. 安全  bar を読み込む
        baz を読み込む
        割り込みが bar または baz を変更
        演算を実行
        結果を foo に書き戻す
```

最初の例は、すべての割り込みが式の境界の外で起こるため安全です。2 番目の例では、アプリケーションは bar と baz が相互に同期更新されることを要求するので安全ではありません。3 番目の例はおそらく安全です。foo は古い値を持つかもしれませんが、その値は式の開始時点で利用できたデータと一貫性があります。

例 7-2: FOO、BAR、および BAZ の型

次に、foo、bar、および baz の型によって相違が生じる例を示します。演算、「bar を読み込む」、「baz を読み込む」、「結果を foo に書き戻す」は、ターゲットプロセッサのアーキテクチャによっては atomic でない場合があります。例えば、dsPIC DSC デバイスは、1 つ (atomic) の命令で 8 ビット、16 ビット、または 32 ビットの量を読み込みまたは書き込みできます。ただし、32 ビットは、命令の選択によっては 2 つの命令を必要とする場合があります。言い換えれば最適化およびメモリ モデル設定によって異なります。型が long で、コンパイラがデータのアクセスに atomic 演算を選択できないと仮定します。その場合、アクセスは次のようになります。

```
bar の下位ワードを読み込む
bar の上位ワードを読み込む
baz の下位ワードを読み込む
baz の上位ワードを読み込む
演算を実行 ( 下位ワードおよび上位ワードに対して )
演算を実行
下位ワードの結果を foo に書き戻す
上位ワードの結果を foo に書き戻す
```

このため、bar または baz の更新結果が予想外のデータに変わる可能性が更に高まります。

例 7-3: ビットフィールド

3 番目の懸案要因はビットフィールドです。C では、メモリがビット レベルで割り当てられますが、ビット演算は定義しない場合があります。理論的には、ビットに対するすべての演算は、ビットフィールドの基底型に対する演算として扱われ、通常、bar および baz からフィールドを抽出したり、foo にフィールドを挿入するにはなんらかの演算が必要です。留意する必要がある重要な考慮事項は (ここでも、命令アーキテクチャ、最適化レベル、およびメモリの設定に左右されますが)、foo があるビットフィールドの一部に書き込む、割り込まれたルーチンが影響を受けやすいことです。これは、オペランドの 1 つが格納先でもある場合、特に顕著になります。

dsPIC DSC 命令セットは、1 ビットを atomic 演算できます。コンパイラは、こうした命令を最適化レベル、メモリの設定、およびリソースの使用可能度に基づいて選択することがあります。

例 7-4: レジスタにキャッシュされたメモリ値

最後に、コンパイラは、メモリ値をレジスタにキャッシュする場合があります。参照されるレジスタ変数は、割り込みによる書き換えが特に起こりやすくなります。変数を含む演算が割り込まない場合でも影響を受ける場合があります。必ず、ISR と割り込み可能な関数の間で共有されるメモリ リソースを `volatile` と指定してください。こうすることで、メモリ ロケーションが、連続するコードシーケンスの行外で更新される場合があることがコンパイラに伝えられます。非 `atomic` 演算の影響から保護されるわけではありませんが、重要な作業です。

7.9.2 解決方法の考察

潜在的な危険性を回避する方法をいくつか示します。

- 競合イベントが発生しないようにソフトウェア システムを設計し、ISR とその他の関数の間でメモリを共有しないようにします。ISR はできるだけ単純にし、実際の操作はメインコードに移します。
- メモリを共有する場合には注意を払い、可能であれば、マルチビットを含むビット フィールドを共有しないようにします。
- コードの重要なセクションを保護するように、共有メモリの非 `atomic` 更新を割り込みから保護します。次のマクロは、この目的に使用できます。

```
#define INTERRUPT_PROTECT(x) { \
    char saved_ipl; \
    \
    SET_AND_SAVE_CPU_IPL(saved_ipl, 7); \
    x; \
    RESTORE_CPU_IPL(saved_ipl); } (void) 0;
```

このマクロは、現在のプライオリティ レベルを 7 に上げることで割り込みを無効にし、目的のステートメントを実行した後に前のプライオリティ レベルに戻します。

7.9.3 アプリケーション例

次に、この節で説明したポイントを含む例を示します。

```
void __attribute__((interrupt))
HigherPriorityInterrupt(void) {
    /* User Code Here */
    LATGbits.LATG15 = 1; /* Set LATG bit 15 */
    IPC0bits.INT0IP = 2; /* Set Interrupt 0
                          priority (multiple
                          bits involved) to 2 */
}

int main(void) {
    /* More User Code */
    LATGbits.LATG10 ^= 1; /* Potential HAZARD -
                          First reads LATG into a W reg,
                          implements XOR operation,
                          then writes result to LATG */

    LATG = 0x1238; /* No problem, this is a write
                   only assignment operation */

    LATGbits.LATG5 = 1; /* No problem likely,
                       this is an assignment of a
                       single bit and will use a single
                       instruction bit set operation */

    LATGbits.LATG2 = 0; /* No problem likely,
                       single instruction bit clear
                       operation probably used */

    LATG += 0x0001; /* Potential HAZARD -
                   First reads LATG into a W reg,
                   implements add operation,
                   then writes result to LATG */

    IPC0bits.T1IP = 5; /* HAZARD -
                       Assigning a multiple bitfield
                       can generate a multiple
                       instruction sequence */
}
```

ステートメントは、前述の `INTERRUPT_PROTECT` マクロを使用して、割り込みから保護できます。この場合の例は、次のようになります。

```
INTERRUPT_PROTECT(LATGbits.LATG15 ^= 1); /* Not interruptible by
                                           level 1-7 interrupt
                                           requests and safe
                                           at any optimization
                                           level */
```

メモ:

第 8 章 . アセンブリ言語と C モジュールの混用

8.1 序章

本章ではアセンブリ言語と C モジュールを同時に使用方法について説明します。例として、C 変数と関数をアセンブリコード内で使用する例と、アセンブリ言語の変数と関数を C 内で使用する例について示します。

8.2 ハイライト

この章では以下の項目について説明します。

- **アセンブリ言語と C 変数と関数の混用**— 個別のアセンブリ言語モジュールがアセンブルされ、それから C モジュールとリンクされます。
- **インラインアセンブリ言語を使用する**— アセンブリ言語命令が C コードに直接組み込まれます。インラインアセンブラは、単純（パラメータ無し）アセンブリ言語文と拡張（パラメータ有り）アセンブリ言語文の両方をサポートします。ここで C 変数は、アセンブラ命令のオペランドとしてアクセスできます。

8.3 アセンブリ言語と C 変数と関数の混用

以下のガイドラインは、個別のアセンブリ言語モジュールと C モジュールとのインターフェースを取る方法を示します。

- **セクション 4.13 「レジスタの規則」** で述べられているレジスタ規則に従います。特に、レジスタ W0-W7 はパラメータの受け渡しに使用されます。アセンブリ言語関数は、これらのレジスタで、パラメータを受け取り、コールされた関数への引数の引渡しを行います。
- 割り込みハンドリング中にコールされない関数はレジスタ W8-W15 を保持しなければなりません。すなわち、レジスタ中の値は修正前に待避され、コールする関数に戻る前に復帰する必要があります。レジスタ W0-W7 はそれらの値を保持せずに使用できます。
- 割り込み関数はすべてのレジスタを保持する必要があります。通常関数コールとは異なり、割り込みは、プログラムの実行中のどの位置でも発生します。通常プログラムに戻るときは、すべてのレジスタは割り込みが発生する前の状態である必要があります。
- C ソースファイルでも参照される個別のアセンブリファイル内で宣言される変数もしくは関数は、アセンブラ指定子 `.global` でグローバル宣言する必要があります。外部シンボルは少なくとも 1 つのアンダースコアを前につけます。C 関数の `main` は、アセンブリの中では、`_main` と名づけられ、逆にアセンブリシンボル `_do_something` は、C 内では `do_something` として参照されます。アセンブリファイルの中で使用される未宣言シンボルは、外部定義されたものとして扱われます。

以下の例では、それらが元々どこで定義されたかにかかわらず、アセンブリ言語と C の両方の中で使用される変数と関数の使用方法について示します。

`ex1.c` ファイルは `foo` と `cVariable` を定義し、アセンブリ言語ファイル内で使用されます。C ファイルは、アセンブリ関数 `asmFunction` をコールする方法と、アセンブリ定義された変数 `asmVariable` のアクセス方法を示します。

例 8-1: C とアセンブリの混用

```
/*
** file: ex1.c
*/
extern unsigned int asmVariable;
extern void asmFunction(void);
unsigned int cVariable;
void foo(void)
{
    asmFunction();
    asmVariable = 0x1234;
}
```

ex2.s ファイルは、asmFunction と asmVariable を定義し、リンクされたアプリケーション内で使用されます。アセンブリファイルはまた、C 関数 foo のコール方法と、C で定義された変数 cVariable のアクセス方法を示します。

```
;
; file: ex2.s
;
.text
.global _asmFunction
_asmFunction:
    mov #0,w0
    mov w0,_cVariable
    return

.global _begin
_main:
    call _foo
    return

.bss
.global _asmVariable
.align 2
_asmVariable: .space 2
.end
```

C ファイル ex1.c では、アセンブリファイル内で宣言されたシンボルに対する外部参照は、標準 extern キーワードを用いて宣言できます。アセンブリソース内の _asmFunction、つまり asmFunction は、void の関数でありそのように宣言されています。

アセンブリファイル ex1.s 内では、シンボル _asmFunction、_main と _asmVariable は、.global アセンブラ指定子でグローバルに参照できるようになっており、他のいずれのソースファイルからもアクセスできます。シンボル _main は参照されるのみで、宣言されません。したがって、アセンブラはこれを外部リファレンスと設定します。

以下の MPLAB C30 の例では、二つのパラメータでアセンブリ関数をコールする方法を示しています。call1.c 内の C 関数 main は call2.s 内の asmFunction を、二つのパラメータを用いてコールします。

例 8-2: C 内のアセンブリ関数のコール

```
/*
** file: call1.c
*/
extern int asmFunction(int, int);
int x;
void
main(void)
{
    x = asmFunction(0x100, 0x200);
}
```

アセンブリ言語関数は二つのパラメータを加算し、結果を戻します。

```
;
; file: call2.s
;
.global _asmFunction
_asmFunction:
    add w0,w1,w0
    return
.end
```

C 内で受け渡されるパラメータはセクション 4.12.2 「戻り値」で詳細に説明します。先の例では、二つの整数引数が W0 と W1 レジスタ内で受け渡されます。整数の戻り結果はレジスタ W0 経由で渡されます。より複雑なパラメータリストは別のレジスタ群を必要とし、ガイドラインに従って注意深くアセンブリプログラムを作成する必要があります。

8.4 インラインアセンブリ言語を使用する

C 関数内で、コンパイラが生成するアセンブリ言語にアセンブリ言語コードを挿入するため、asm 文が用いられます。インラインアセンブリは二つの形式、単純と拡張、を持ちます。

単純形式では、アセンブラ命令は以下の形式を用いて記述されます。

```
asm ("instruction");
```

ここで、*instruction* は有効なアセンブリ言語構造です。もし、ANSI C プログラム形式でインラインアセンブリを記述する場合は、asm ではなく、`__asm__` と書きます。

注: インラインアセンブリの単純な形式では、1つの文字列のみが使用可能です。

asm を用いた**拡張**アセンブラ命令では、命令のオペランドは C 表現を用いて指定されます。拡張形式では以下のように表されます。

```
asm("template" [ : [ "constraint"(output-operand) [ , ... ] ]
                [ : [ "constraint"(input-operand) [ , ... ] ]
                [ "clobber" [ , ... ] ]
                ]
    );
```

アセンブラ命令 *template* と、それぞれのオペランドに *constraint* ストリングオペランドを追加することで指定しなければなりません。*template* は命令ニックを指定し、オプションにはオペランド用を指定します。*constraint* ストリングはオペランド制約、例えば、オペランドはレジスタでなければならない (通常の場合)、もしくはオペランドは直接の値でなければならない、等の制約を指定します。

以下の制約文字が MPLAB C30 でサポートされています。

表 8-1: MPLAB® C30 でサポートされる制約文字

文字	制約
a	WREG を要求します。
b	除算サポート レジスタ W1
c	乗算サポート レジスタ W2
d	汎用データ レジスタ W1 ~ W14
e	非除算サポート レジスタ W2 ~ W14
g	汎用レジスタではないレジスタを除く、どんなレジスタ、メモリもしくは直接代入整数オペランドが使用できます。
i	直接代入整数オペランド (固定値を持ったもの) が使用できます。これは、その値がアセンブル時にのみ知ることができるシンボリック定数を含みます。
r	汎用レジスタ内にあるものであれば、レジスタオペランドが使用できます。
v	AWB レジスタ W13
w	累算器レジスタ A ~ B
x	x 先取りレジスタ W8 ~ W9
y	y 先取りレジスタ W10 ~ W11
z	MAC 先取りレジスタ W4 ~ W7
0, 1, ... , 9	指定されたオペランド番号に適合するオペランドが使用できます。もし、digit が同じ alternative 内の文字と一緒に使用されると、digit が最後に来ます。
T	near もしくは far データのオペランドです。
U	near データのオペランドです。

表 8-2: MPLAB® C30 でサポートされる制約修飾子

文字	制約
=	このオペランドはこの命令用の書き込み専用であることを意味します。以前の値は捨てられ、出力データに置き換えられます。
+	このオペランドは命令により読み書きされることを意味します。
&	このオペランドは先に書き込まれる オペランドであることを意味し、命令が入力オペランドを用いることで終了する前に修正されます。したがって、このオペランドは、入力オペランドもしくはメモリアドレスの一部として使用されるレジスタには配置されません。

例 8-3: C 変数渡し

この例では、swap 命令 (コンパイラは通常では使用しない) の使用方法を説明します。

```
asm ("swap %0" : "+r"(var));
```

ここで var はオペランドの C 表現で、入力と出力両方のオペランドです。オペランドはタイプ r の制約で、レジスタオペランドを示します。+r の + はオペランドが入力と出力両方のオペランドであることを示します。

それぞれのオペランドは、括弧内のオペランド制約ストリングに続き C 表現で記述されます。最初のコロンはアセンブラのテンプレートを最初の出力オペランドと分け、次のコロンは最初の入力を (もしあれば) 最後の出力オペランドと分けます。コンマは複数ある出力オペランドと入力オペランドを分けます。

もし出力オペランドが無く、入力オペランドがある場合は、2つの連続するコロンで、出力オペランドが無いことを明確にしなければなりません。コンパイラは、出力オペランド表現が L 値でなければならないことを要求します。入力オペランドは、L 値である必要はありません。コンパイラは、実行される命令用として適切なデータタイプをオペランドが持っているかどうかについてはチェックできません。コンパイラはアセンブラ命令テンプレートを分析せず、それが何を意味するか、もしくはそれが有効なアセンブラ入力であるかどうかはわかりません。拡張 asm の特徴は、

マシン命令（コンパイラ自身が、存在することを知らない命令）用として使用される場合があります。出力表現が直接アドレスできないもの（例えば、ビットフィールド）である場合、レジスタに制約を与える必要があります。その場合、MPLAB C30 は asm の出力としてレジスタを使用し、レジスタを出力に格納します。もし出力オペランドが書き込みのみであるとすると、MPLAB C30 は、命令実行の前にこれらのオペランド内にある値が無効となってしまうため、生成する必要が無いと仮定します。

例 8-4: レジスタの上書き

特定のハードレジスタを上書きする命令もあります。このような命令を記述するには、入力オペランドの後に 3 つ目のコロンを付け上書きされるハードレジスタの名前をその後に記述します（文字列がコンマで区切られている場合）。以下は例です。

```
asm volatile ("mul.b %0"  
: /* no outputs */  
: "U" (nvar)  
: "w2");
```

この場合、オペランド nvar は、near データ空間内で宣言される文字変数で、「U」制約で指定されます。もしアセンブラ命令がフラグ（条件コード）レジスタを変更するならば、書き換えられるレジスタリストに「cc」を追加します。アセンブラ命令が、予測不可能な方法でメモリを変更する場合、書き換えられるレジスタのリストに「memory」を追加します。つまりアセンブラ命令によりレジスタに置かれる値を MPLAB C30 が保持しないこととなります。

例 8-5: 複数アセンブラ命令の使用

複数のアセンブラ命令を、単一の asm テンプレートに配置するには、改行（\n と書きます）で区切ります。入力オペランドと出力オペランドのアドレスには、上書きされるレジスタを使用しないことを保証しなければいけません。したがって、出力レジスタは何度でも読み書き可能です。以下に、テンプレート内の複数命令の例を示します。これはサブルーチン _foo がレジスタ W0 と W1 内の引数を受け付けることを仮定しています。

```
asm ("mov %0,w0\nmov %1,W1\ncall _foo"  
: /* no outputs */  
: "g" (a), "g" (b)  
: "W0", "W1");
```

この例では、制約文字列「g」が汎用オペランドであることを示しています。

例 8-6: 入力レジスタの上書きを防ぐための '&' の使用

出力オペランドが & 制約修正子をもたなければ、MPLAB C30 は、入力出力が生成される前に使用される前提で、それを入力オペランドと同じレジスタに割り当てます。アセンブラコードが実際に 1 つ以上の命令から構成される場合、この仮定は不正確になる場合があります。その場合、入力オペランドとオーバーラップしないような出力オペランドに & を使用します。例えば、以下の関数を参照してください。

```
int  
exprbad(int a, int b)  
{  
    int c;  
  
    __asm__ ("add %1,%2,%0\n s1 %0,%1,%0"  
: "=r" (c) : "r" (a), "r" (b));  
  
    return(c);  
}
```

目的は、値 $(a + b) \ll a$ を計算することですが、演算結果の値と同一とは限りません。以下例示するように、オペランド c は `asm` 命令が終了する前に変更されることをコンパイラに通知する必要があります。

```
int
exprgood(int a, int b)
{
    int c;

    __asm__ ("add %1,%2,%0\n s1 %0,%1,%0"
            : "=&r"(c) : "r"(a), "r"(b));

    return(c);
}
```

例 8-7: オペランドのマッチング

アセンブラ命令が読み書きオペランドを持っている場合や、もしくはいくつかのビットのみが変更されるようなオペランドを持っている場合は、論理的にその動作を二つの別々のオペランド、すなわち 1 つの入力オペランドと 1 つの出力結果を書き込むだけのオペランド、に分けなければなりません。二者の繋がり、命令実行時に両者が同じ位置にあるよう制約し、記述します。両方のオペランドの表現方法として同じ C 記述も可能ですし、異なる記述もできます。例えば、以下の例では、`bar` を読み出し専用ソースオペランドとし、また `foo` を読み書き用の対象オペランドとする `add` 命令が示されています。

```
asm ("add %2,%1,%0"
    : "=r" (foo)
    : "0" (foo), "r" (bar));
```

オペランド 1 用の制約「0」は、オペランド 1 がオペランド 0 と同じ位置を占める必要があることを示します。制約記号は、入力オペランド内でのみ使用が許され、出力オペランドを参照しなければなりません。この制約記号によってのみ、1 つのオペランドがもう 1 つのものと同じ位置にあることを保証できます。`foo` が両方のオペランドの値であるというだけでは、生成されたアセンブラコード内で同じアドレスになるというのを保証できません。以下の例は正常に動作できない例です。

```
asm ("add %2,%1,%0"
    : "=r" (foo)
    : "r" (foo), "r" (bar));
```

最適化もしくはリロードにより、オペランド 0 と 1 が異なるレジスタに存在する場合があります。例えば、コンパイラはあるレジスタ内で `foo` の値のコピーを見つけ、それをオペランド 1 用に使用しますが、出力オペランド 0 を別のレジスタに格納します（それを後で `foo` 自身のアドレスにコピーします）。

例 8-8: オペランドの命名

アセンブラコードテンプレート内で参照可能なシンボリック名を使用して入 / 出力オペランドを指定することもできます。シンボリック名は制約文字列前の角括弧内で指定され、オペランド番号が後に続く % 記号の代わりに `%[name]` を使用してアセンブラコードテンプレート内で参照可能です。名前付きオペランドを使用して前述の例を以下のようにコード化できます。

```
asm ("add %[foo],[bar],[foo]"
    : [foo] "=r" (foo)
    : "0" (foo), [bar] "r" (bar));
```

asm の後にキーワード `volatile` を記述することで、asm 命令が消去されたり、大きく移動されたり、もしくは結合されるのを回避できます。例えば、以下のように記述します。

```
#define disi(n) \  
asm volatile ("disi %#0" \  
: /* no outputs */ \  
: "i" (n))
```

この場合、制約文字「i」は、disi 命令で要求される直接代入オペランドを示します。

例 8-9: 制御フローの変更

インラインアセンブリ ステートメント内で制御フローを変更する場合には、特別に注意しなければならない事項があります。

インラインアセンブリ ステートメントによって制御フローの変更が発生することをコンパイラに伝える方法はありません。制御はアセンブリ ステートメントに進入した以上、常に次のステートメントに進みます。

良い制御フロー:

```
asm("call _foo" : /* outputs */  
      : /* inputs */  
      : "w0", "w1", "w2", "w3", "w4", "w5",  
        "w6", "w7");  
/* next statement */
```

この例は、foo を呼び出した後に次のステートメントが実行されるため、使用を推奨できます。このコードは、一部のレジスタがこのステートメントによって保持される必要がないことをコンパイラに伝えます。これらは、foo によって保存されないレジスタに相当します。

悪い制御フロー:

```
asm("bra OV, error");  
/* next statement */  
return 0;  
  
asm("error: ");  
return 1;
```

このフローは推奨できません。次のステートメント、`return 0` が実行されない場合でもコンパイラが実行されるとみなすためです。この場合、`asm("error: ")` 以降のステートメントは、到達できないため削除されます。asm ステートメントのラベルについては、詳細を参照してください。

許容できる制御フロー:

```
asm("cp0 _foo\n\t"  
    "bra nz, eek\n\t"  
    "; some assembly\n\t"  
    "bra eek_end\n\t"  
    "eek:\n\t"  
    "; more assembly\n\t"  
    "eek_end:" : /* outputs */  
              : /* inputs */  
              : "cc");  
/* next statement */
```

使用はできますが、予想どおりに機能しない場合があります(つまり、asm ステートメント内の分岐にかかわらず、次のステートメントは常に実行されます)。asm ステートメントのラベルについては、詳細を参照してください。コードは、cc が上書きされると識別することで、このステートメントのステータスフラグが無効になったと示すことに留意してください。

ラベルと制御フロー

アセンブリ ステートメント内のラベルは、最適化オプションによっては予想外の動作をする場合があります。インライナによって、asm ステートメント内のラベルが複数回定義される場合があります。

また、プロシージャアグリゲータ ツール (-mpa) は、ローカル ラベル シンタックスを受け付けません。次の例を参照してください。

```
inline void foo() {
    asm("do #6, loopend");
    /* some C code */
    asm("loopend: ");
    return;
}
```

このフローは、複数の理由から、推奨できません。まず、コンパイラが認識していない暗黙の制御フローを asm ステートメントが開始しています。次に、foo() がインライン化されると、ラベル loopend は複数回定義されます。

解決方法として、次の例のように『MPLAB ASM30, MPLAB LINK30 and Utilities User's Guide』(DS51317)に記載のローカル ラベル シンタックスを使用できます。

```
inline void foo() {
    asm("do #6, 0f");
    /* some C code */
    asm("0: ");
    return;
}
```

この形式では、少なくともラベルが複数回定義される問題を解決する点でわずかながら良いフローであるといえます。ただし、プロシージャアグリゲータ ツール (-mpa) は、0: という形式のラベルを受け付けません。

例 8-10: 必須レジスタの使用

dsPIC DSC 命令セット内の一部の命令では、オペランドが特定のレジスタまたはレジスタ グループである必要があります。表 8-1 には、生成する命令の制約を満たす際に使用される制約文字の一部を掲載しています。

目的にかなう制約がない場合や、特定のレジスタを asm ステートメント内で使用するよう指定するには、次のコード サンプルに示されているように、サポート (および、レジスタを上書き済みと示す必要性を減らすこと) を目的としてコンパイラによって提供されているレジスタ指定拡張を使用できます。このサンプルは、いくつかの奇数レジスタを条件とする仮想の命令を使用しています。

```
{ register int in1 asm("w7");
  register int in2 asm("w9");
  register int out1 asm("w13");
  register int out2 asm("w0");

  in1 = some_input1;
  in2 = some_input2;
  __asm__ volatile ("funky_instruction %2,%3,%0; = %1" :
                    /* outputs */ "=r"(out1), "=r"(out2) :
                    /* inputs */ "r"(in1), "r"(in2));
  /* use out1 and out2 in normal C */
}
```

この例では、funky_instruction には 1 つの明示的な出力、out1 と、1 つの暗黙の出力、out2 があります。両方とも asm テンプレートに置かれているため、(暗黙の出力はコメント ステートメント内にありますが) コンパイラはレジスタの使用状況を適切に追跡できます。表示されている入力とは通常通りです。別の方法としては、拡張レジスタ宣言子シンタックスを使用して、仮想命令 funky_instruction の制約を満たす特定のハード レジスタを指定できます。

付録 A. 実装時定義動作

A.1 はじめに

本章では、MPLAB C30 の実装時定義動作を取り扱います。ISO 規格では、C 言語に関してベンダーが言語の「実装時定義」関数の詳細を文章化することを義務付けています。

本章で取り扱う項目は以下の通りです。

- 変換
- 環境
- 識別子
- 文字
- 整数
- 浮動小数点
- 配列およびポインタ
- レジスタ
- 構造体、共用体、列挙およびビットフィールド
- 修飾子
- 宣言子
- ステートメント
- プリプロセッサ
- ライブラリ関数
- 信号
- ストリームおよびファイル
- tmpfile
- errno
- メモリー
- ABORT
- EXIT
- getenv
- システム
- strerror

A.2 変換

変換の実装時定義動作は ANSI C 規格のセクション G3.1 で取り上げられています。
連続するスペース文字列は、改行文字以外は維持されますか。それともスペース文字 1 つで置換されますか。(ISO 5.1.1.2)

空白文字 1 つで置換されます。

診断メッセージはどのように識別されますか。(ISO 5.1.1.3)

診断メッセージは、ソースファイル名とメッセージに対応する行番号をコロンの (':') で区切って診断メッセージの前に付けることによって識別されます。

クラスの異なるメッセージはありますか。(ISO 5.1.1.3)

はい。

ある場合は何ですか。(ISO 5.1.1.3)

出力ファイルの生成を抑制するエラーと、出力ファイルの生成を抑制しない警告です。

各メッセージクラスの変換戻り状態コードは何ですか。(ISO 5.1.1.3)

エラーの戻り状態コードは 1、警告の戻り状態コードは 0 です。

診断のレベルを制御することは可能ですか。(ISO 5.1.1.3)

はい。

可能な場合、どのような形式で制御されますか。(ISO 5.1.1.3)

コンパイラコマンドラインオプションを使用して警告メッセージの生成を要請または抑制することができます。

A.3 環境

環境の実装時定義動作は ANSI C 規格のセクション G3.2 で取り上げられています。

独立プログラムではどのようなライブラリ関数が利用可能ですか。(ISO 5.1.2.1)

標準 C ライブラリのすべての関数が利用可能です。ただし「ランタイムライブラリ」セクションで説明されているように、ある一定の関数が環境に合わせてカスタマイズされていることが条件となります。

独立プログラムにおけるプログラム終了処理を説明してください。(ISO 5.1.2.1)

関数 main が戻されるか関数 exit が呼び出された場合、HALT 命令が無限ループで実行されます。この動作はカスタマイズできます。

関数 main にパスされる引数(パラメータ)を説明してください。(ISO 5.1.2.2.1)

main にはパラメータはパスされません。

以下のうち、有効な相互作用デバイスはどれですか:(ISO 5.1.2.3)

非同期端末 いいえ

対応するディスプレイとキーボード いいえ

プログラム間接続 いいえ

他にある場合、それは何ですか。

ありません。

A.4 識別子

識別子の実装時定義動作は ANSIC 規格のセクション G.3.3 で取り上げられています。

外部結合なしの識別子では、31 以上何文字まで認識可能ですか。(ISO 6.1.2)

すべての文字が認識可能です。

外部結合ありの識別子では、6 以上何文字まで認識可能ですか。(ISO 6.1.2)

すべての文字が認識可能です。

外部結合ありの識別子では、大文字と小文字の違いは区別されますか。(ISO 6.1.2)

はい。

A.5 文字

文字の実装時定義動作は ANSIC 規格のセクション G.3.4 で取り上げられています。

規格で明確に指定されていないソース文字および実行文字を説明して下さい。(ISO 5.2.1)

ありません。

記載されている文字列に対応するエスケープ文字列の値は何ですか。(ISO 5.2.2)

表 A-1: エスケープ文字列および値

文字列	値
\a	7
\b	8
\f	12
\n	10
\r	13
\t	9
\v	11

実行文字セットの1つの文字は何ビットですか。(ISO 5.2.4.2)

8 ビットです。

(文字リテラルと文字列リテラルにおける) 実行文字セットのメンバに対するソース文字セットのメンバのマッピングは何ですか。(ISO 6.1.3.4)

関数として識別されます。

単純文字のタイプは何ですか。(ISO 6.2.1.1)

次のどちらでも可能です(ユーザ定義)。デフォルトは signed char です。コンパイラコマンドラインオプションを使用して unsigned char をデフォルトにすることができます。

A.6 整数

整数の実装時定義動作は ANSI C 規格のセクション G3.5 で取り上げられています。
以下の表には様々な型の整数の容量と範囲が記載されています: (ISO 6.1.2.5)

表 A-2: 整数型

記号	サイズ (ビット)	範囲
char	8	-128 ... 127
signed char	8	-128 ... 127
unsigned char	8	0 ... 255
short	16	-32768 ... 32767
signed short	16	-32768 ... 32767
unsigned short	16	0 ... 65535
int	16	-32768 ... 32767
signed int	16	-32768 ... 32767
unsigned int	16	0 ... 65535
long	32	-2147483648 ... 2147438647
signed long	32	-2147483648 ... 2147438647
unsigned long	32	0 ... 4294867295

値が表示できない場合に整数をより短い符号付き整数に変換するとどうなりますか。
また、符号なし整数を同じ長さの符号付に変換するとどうなりますか。 (ISO 6.2.1.2)
値の意味が失われます。エラー信号は出されません。

符号付き整数に対するビット単位の演算の結果はどうなりますか。 (ISO 6.3)

シフト演算子は符号を維持します。その他の演算子は、オペランドが符号なし整数であるかのように動作します。

整数除算の余りの符号は何ですか。 (ISO 6.3.5)

+ です。

負の値の符号付き整数型を右にシフトするとどうなりますか。 (ISO 6.3.7)

符号は維持されます。

A.7 浮動小数点

浮動小数点の実装時定義動作は ANSI C 規格のセクション G3.6 で取り上げられています。

その型の表示可能な値の範囲内にある浮動小数点定数の縮尺後の値は表示可能な近似値ですか。それとも表示可能なより大きな値ですか。それとも表示可能な値のより小さな値ですか。 (ISO 6.1.3.1)

表示可能な近似値です。

以下の表には様々な型の浮動小数点数の容量と範囲が記載されています: (ISO 6.1.2.5)

表 A-3: 浮動小数点型

記号	サイズ (ビット)	範囲
float	32	1.175494e-38 ... 3.40282346e+38
double*	32	1.175494e-38 ... 3.40282346e+38
long double	64	2.22507385e-308 ... 1.79769313e+308

* double は `-fno-short-double` が使用されている場合は long double と同等です。

整数を元の値を正確に表示することができない浮動小数点数に変換する場合の打ち切りの方向はどうなりますか。 (ISO 6.2.1.3)

切り捨てです。

浮動小数点数をより小さな浮動小数点数に変換する場合の切り捨てあるいは切り上げの方向はどうなりますか。 (ISO 6.2.1.4)

切り捨てです。

A.8 配列およびポインタ

配列およびポインタの実装時定義動作は ANSI C 規格のセクション G.3.7 で取り上げられています。

配列の最大容量、つまり演算子のサイズの型 `size_t` を維持するのに必要な整数の型は何ですか? (ISO 6.3.3.4, ISO 7.1.1)

unsigned int.

ポインタを整数型に変換するのに必要な整数のサイズを教えてください。 (ISO 6.3.4)

16 ビットです。

ポインタを整数にキャストする、あるいは整数をポインタにキャストするとどうなりますか。 (ISO 6.3.4)

識別関数でマッピングします。

同じ配列の2つのポインタの差 `ptrdiff_t` を維持するのに必要な整数の型は何ですか。 (ISO 6.3.6, ISO 7.1.1)

unsigned int.

A.9 レジスタ

レジスタの実装時定義動作は ANSI C 規格のセクション G.3.8 で取り上げられています。

ストレージクラス指定子レジスタはどの程度まで実際にレジスタの中のオブジェクトに影響を与えますか。 (ISO 6.5.1)

最適化が無効になっている場合 register ストレージクラスの有効化を試みますが、最適化が有効な場合は無視されます。

A.10 構造体、共用体、列挙およびビットフィールド

構造体、共用体、列挙およびビットフィールドの実装時定義動作は ANSI C 規格のセクション A.6.3.9 および G.3.9 で取り上げられています。

共用体オブジェクトのメンバに異なるタイプのメンバを使用してアクセスするとどうなりますか。 (ISO 6.3.2.3)

型変換は適用されません。

構造体のメンバのパディングと整列を説明してください。 (ISO 6.5.2.1)

Char はバイトごとに整列しています。その他のすべてのオブジェクトはワードごとに整列しています。

単純 int ビットフィールドの型は何ですか。 (ISO 6.5.2.1)

ユーザ定義です。デフォルトでは、signed int ビットフィールドです、コマンドラインオプションを使用して unsigned int ビットフィールドにすることが可能です。

int 内のビットフィールドの割り当ての順序を教えてください。 (ISO 6.5.2.1)

ビットは低位ビットから高位ビットへ割り当てられます。

ビットフィールドはストレージユニットの境界をまたぐことは可能ですか。 (ISO 6.5.2.1)

はい。

列挙型の値の表示にはどの整数型が選択されていますか。 (ISO 6.5.2.2)

int。

A.11 修飾子

修飾子の実装時定義動作は ANSI C 規格のセクション G.3.10 で取り上げられています。

どのようなアクションが、volatile 修飾子を有するオブジェクトへのアクセスとみなされますか。 (ISO 6.5.3)

オブジェクト名が表示されている場合、アクセスされています。

A.12 宣言子

宣言子の実装時定義動作は ANSI C 規格のセクション G.3.11 で取り上げられています。

演算子、構造体、あるいは共用体のタイプを変更する宣言子の最大数を教えてください。 (ISO 6.5.4)

制限はありません。

A.13 ステートメント

ステートメントの実装時定義動作は ANSI C 規格のセクション G.3.12 で取り上げられています。

switch ステートメントの case 値の最大数はいくらかですか。 (ISO 6.6.4.2)

制限はありません。

A.14 プリプロセッサ

プリプロセッサの実装時定義動作は ANSIC 規格のセクション G.3.13 で取り上げられています。

条件付き取り込みを制御する定数式における単一文字定数の値は、実行文字セットの中の同じ文字定数の値と合致しますか。(ISO 6.8.1)

はい。

そのような文字定数は負の値を持つことはできますか。(ISO 6.8.1)

はい。

インクルードソースファイルの位置を指定するにはどのような方法が使用されますか。(ISO 6.8.2)

前処理プログラムが現在ディレクトリを検索し、次にコマンドラインオプションを使用して指定したディレクトリを検索します。

ヘッダはどのように識別されますか。また、場所はどのように指定されますか。(ISO 6.8.2)

ヘッダは #include 指令で < > の区切り文字か “ ” の区切り文字の間に入れて識別されます。場所はコマンドラインオプションを使用して指定します。

インクルード可能なソースファイルでは引用名はサポートされていますか。(ISO 6.8.2)

はい。

区切り文字列と外部ソースファイル名間のマッピングは何ですか。(ISO 6.8.2)

識別関数です。

下記 #pragma 指令の動作を説明してください。(ISO 6.8.6)

表 A-4: #PRAGMA 動作

Pragma	動作
#pragma code section-name	コードセクション名をつける。
#pragma code	コードセクション名をデフォルト値にリセットする (viz., .text)。
#pragma idata section-name	初期化データセクション名をつける。
#pragma idata	初期化データセクション名をデフォルト値にリセットする (viz., .data)。
#pragma udata section-name	非初期化データセクション名をつける。
#pragma udata	非初期化データセクション名をデフォルト値にリセットする (viz., .bss)。
#pragma interrupt function-name	関数名を割り込み関数として指定する。

変換のデータと時間が利用可能でない場合の __DATE__ と __TIME__ のそれぞれの定義は何ですか。(ISO 6.8.8)

該当ありません。これらの関数が利用不可の環境ではコンパイラはサポートしていません。

A.15 ライブラリ関数

ライブラリ関数の実装時定義動作は ANSIC 規格のセクション G.3.14 で取り上げられています。

マクロ `NULL` が展開される空ポインタ定数は何ですか。 (ISO 7.1.5)

0 です。

アサート関数が表示する診断はどのように認識されますか。また、この関数の終了時の動作を教えてください。 (ISO 7.2)

アサート関数はファイル名、行番号およびテスト表現をコロンの (':') で区切って表示します。その後 `abort` 関数を呼び出します。

`isalnum`、`isalpha`、`iscntrl`、`islower`、`isprint` および `isupper` 関数でテスト済みの文字は何ですか。 (ISO 7.3.1)

表 A-5: 関数でテスト済みの文字

関数	テストされた文字
<code>isalnum</code>	いずれかの文字または数字: <code>isalpha</code> 又は <code>isdigit</code> 。
<code>isalpha</code>	いずれかの文字: <code>islower</code> 又は <code>isupper</code> 。
<code>iscntrl</code>	標準動作制御文字 5 つと、バックスペース、アラートのうちのいずれか: <code>\f</code> 、 <code>\n</code> 、 <code>\r</code> 、 <code>\t</code> 、 <code>\v</code> 、 <code>\b</code> 、 <code>\a</code> 。
<code>islower</code>	「a」から「z」までのいずれかの文字。
<code>isprint</code>	図形文字又は空白文字: <code>isalnum</code> 又は <code>ispunct</code> 又はスペース。
<code>isupper</code>	「A」から「Z」までのいずれかの文字。
<code>ispunct</code>	以下のいずれかの文字: <code>!"#\$%&'();<=>?[\]*+,-./:^</code>

ドメインエラーの発生後、返される演算結果を教えてください。 (ISO 7.5.1)

NaN です。

数学関数はアンダーフロー範囲エラーのマクロ `ERANGE` に整数式 `errno` を設定しますか。 (ISO 7.5.1)

はい。

`fmod` 関数の第 2 引数が 0 の場合ドメインエラーが表示されますか、それとも 0 が返されますか。 (ISO 7.5.6.4)

ドメインエラーが表示されます。

A.16 信号

信号関数用の信号を教えてください。(ISO 7.7.1.1)

表 A-6: 信号関数

名前	説明
SIGABRT	ABORT
SIGINT	双方向アテンション信号の受信
SIGILL	無効関数の検出
SIGFPE	算術演算の誤り
SIGSEGV	ストレージへの無効アクセス
SIGTERM	プログラムへの終了リクエストの送信

信号関数で認識される各信号のパラメータと使用方法を説明してください。(ISO 7.7.1.1)
アプリケーションで定義されます。

信号関数で認識される各信号のデフォルト時の処理とプログラム起動時の処理を説明してください。(ISO 7.7.1.1)

ありません。

信号ハンドラの呼び出し前に信号 (sig、SIG_DFL) の値が実行されない場合、信号の遮断は実施されますか。(ISO 7.7.1.1)

いいえ。

信号関数特定のハンドラが SIGILL 信号を受信するとデフォルト処理はリセットされますか。(ISO 7.7.1.1)

いいえ。

A.17 ストリームおよびファイル

テキストストリームの最終行には終端改行文字が必要ですか。(ISO 7.9.2)

いいえ。

テキストストリームの改行文字のすぐ前に記述された空白文字は、そのテキストストリームが逆読み込みされた場合に表示されますか。(ISO 7.9.2)

はい。

2進法ストリームで記述されたデータには空文字はいくつまで付加することができますか。(ISO 7.9.2)

できません。

付加モードストリームのファイル位置インジケータは最初はファイルの始めか終わりのいずれに配置されますか。(ISO 7.9.3)

始めです。

テキストストリームに記述すると関連ファイルがその地点までで切り捨てられますか。(ISO 7.9.3)

アプリケーションで定義されます。

ファイルのバッファリングの特徴は何ですか。(ISO 7.9.3)

完全にバッファされます。

長さが0のファイルは実際に存在することは可能ですか。(ISO 7.9.3)

はい。

有効なファイル名を作成するための規則はありますか。(ISO 7.9.3)

アプリケーションで定義されます。

同じ名前のファイルを同時に開くことは可能ですか。(ISO 7.9.3)

アプリケーションで定義されます。

開いているファイルに対して関数削除はどのような影響がありますか。(ISO 7.9.4.1)

アプリケーションで定義されます。

ファイル名変更関数を呼び出す前に既に新しい名前のファイルが存在している場合はどうなりますか。(ISO 7.9.4.2)

アプリケーションで定義されます。

`fprintf` 関数における `%p` 変換の出力はどのような形式になりますか。(ISO 7.9.6.1)

16 進数表示です。

`fscanf` 関数における `%p` 変換の入力はどのような形式になりますか。(ISO 7.9.6.2)

16 進数表示です。

A.18 TMPFILE

プログラムの `ABORT` 時には、使用中のテンポラリーファイルは削除されますか。(ISO 7.9.4.3)

はい。

A.19 ERRNO

不具合の場合 `fgetpos` 又は `ftell` 関数はマクロ `errno` をどの値に設定しますか。(ISO 7.9.9.1, (ISO 7.9.9.4)

アプリケーションで定義されます。

`perror` 関数で生成されるメッセージのフォーマットは何ですか。(ISO 7.9.10.4)

`perror` の引数、コロン、`errno` 値のテキスト記述です。

A.20 メモリー

要求されている容量が 0 の場合、`calloc`、`malloc` または `realloc` 関数の動作はどうなりますか。(ISO 7.10.3)

ゼロの長さのブロックが割り当てられます。

A.21 ABORT

`abort` 関数が呼び出されると使用中またはテンポラリーファイルはどうなりますか。(ISO 7.10.4.1)

影響ありません。

A.22 EXIT

引数の値が 0 または `EXIT_SUCCESS`、または `EXIT_FAILURE` 以外の場合、`EXIT` 関数によって返される値は何ですか。(ISO 7.10.4.3)

引数の値です。

A.23 GETENV

環境名にはどのような制限がありますか。(ISO 7.10.4.4)

アプリケーションで定義されます。

`getenv` 関数の呼び出しによって取得された環境リストを変更する方法を教えてください。(ISO 7.10.4.4)

アプリケーションで定義されます。

A.24 システム

システム関数にパスされる文字列のフォーマットを説明してください。(ISO 7.10.4.5)

アプリケーションで定義されます。

システム関数の実行形態を教えてください。(ISO 7.10.4.5)

アプリケーションで定義されます。

A.25 STRERROR

`strerror` 関数によるエラーメッセージ出力のフォーマットを説明してください。(ISO 7.11.6.2)

単純文字列です。

`strerror` 関数の呼び出しによって返されるエラーメッセージ文字列の内容を説明してください。(ISO 7.11.6.2)

表 A-7: エラーメッセージ文字列

Errno	メッセージ
0	エラーはありません
EDOM	ドメインエラーです
ERANGE	範囲エラーです
EFPOS	ファイル位置エラーです
EFOPEN	ファイルオープン時のエラーです
nnn	エラー #nnn

メモ:

付録 B. MPLAB C30 C コンパイラ診断

B.1 はじめに

この別紙では、MPLAB C30 コンパイラにより生成される最も一般的な診断メッセージを示します。

MPLAB C30 コンパイラは、エラーと警告、2種類の診断メッセージを生成できます。それぞれの種類で目的が異なります。

- エラーメッセージはプログラムのコンパイルを不可能にする問題を報告します。MPLAB C30 は明らかに問題のあるソースファイルの名前とその行番号を報告します。
- 警告メッセージは、コンパイルが可能、または実際にコンパイルが処理されている場合に、コード内のその他問題を示唆するような通常とは異なる状態を報告します。警告メッセージもソースファイルの名前と行番号を報告しますが、エラーメッセージとの区別のため、'warning:' という文字を含みます。

警告メッセージが出る場合に、プログラムが本当に本来の動作をしているか、古い機能や MPLAB C30 C の標準機能でない機能が使用をされていないかを確認する必要がある危険ポイントに達していることを示す場合があります。警告メッセージの多くはユーザーが `-w` オプションを使用してリクエストした時のみ発行されます（例えば、`-Wall` は様々な有効な警告をリクエストします）。

稀に、コンパイラが内部エラーメッセージレポートを発行する場合があります。これは、コンパイラ自体がマイクロチップサポートに報告すべき障害を検出したということを示します。サポートの連絡先の詳細はこの文書に記載されています。

B.2 エラー

記号

\x used with no following HEX digits – \x の後ろに 16 進数字がありません

エスケープシーケンス `\x` の後には 16 進数字を付けます。

'&' constraint used with no register class – '&' 制約にレジスタクラスが付いていません

asm ステートメントが無効です。

'%' constraint used with last operand – '%' 制約に最終オペランドが付いています

asm ステートメントが無効です。

#elif after #else – #else の後に #elif があります

プリプロセッサ条件式では、`#else` 文は必ず `#elif` 文の後に続けます。

#elif without #if – #if が #elif に付いていません

プリプロセッサ条件下では、`#elif` 文の前に必ず `#if` 文を付けます。

#else after #else – #else の後に #else があります

プリプロセッサ条件下では、`#else` 文は 1 度しか使用できません。

#else without #if – #if が #else に付いていません

プリプロセッサ条件下では、`#else` 文の前に必ず `#if` 文を付けます。

#endif without #if – #if が #endif に付いていません

プリプロセッサ条件下では、#endif 文の前に必ず #if 文を付けます。

#error 'message' – #error 'メッセージ'

このエラーは #error ディレクティブに反応して表示されます。

#if with no expression – 表現式を持たない #if です

定数演算子の値を求めるための表現式が必要です。

#include expects "FILENAME" or <FILENAME> – #include は "FILENAME" または <FILENAME> を求めています

#include のファイル名が見つからないか不完全です。引用符か括弧 < 内にファイル名を入れてください。

'#' is not followed by a macro parameter – '#' の後にマクロパラメータが付いていません

ストリングサイズ演算子では '#' のあとにマクロ引数名をつける必要があります。

'#keyword' expects "FILENAME" or <FILENAME> – '#keyword' は "FILENAME" または <FILENAME> を求めています

指定の #keyword が引数として引用符内または括弧 < 内に入れたファイル名を求めています。

'#' is not followed by a macro parameter – '#' の後にマクロパラメータが付いていません

演算子 '#' のあとにマクロ引数名をつける必要があります。

'##' cannot appear at either end of a macro expansion – '##' をマクロ展開の最初または末尾に付けないでください

連結演算子 '##' はマクロ展開の最初または末尾に付けてはいけません。

A

a parameter list with an ellipsis can't match an empty parameter name list declaration – 省略符号付きのパラメータリストが空のパラメータ名リスト宣言と一致しません

関数の宣言と定義は同一にする必要があります。

"symbol" after #line is not a positive integer – #lineの後の "symbol" が正の整数ではありません

#line はソース行番号が正の値であることを求めています。

aggregate value used where a complex was expected – 複素数値が必要なところで集合値が使用されています

複素数値が必要なところで集合値を使用しないでください。

aggregate value used where a float was expected – 浮動小数点が必要なところで集合値が使用されています

浮動小数点が必要なところで集合値を使用しないでください。

aggregate value used where an integer was expected – 整数が必要なところで集合値が使用されています

整数が必要なところで集合値を使用しないでください。

alias arg not a string – エイリアス引数が文字列ではありません

現在の識別子がエイリアスの場合のエイリアス属性の引数は文字列にする必要があります。

alignment may not be specified for 'identifier' – 'identifier' の整列を指定しないでください

整列属性は変数でしか使用できません。

'__alignof' applied to a bit-field – '__alignof' がビットフィールドに適用されていません

'__alignof' 演算子はビットフィールドに適用できません。

alternate interrupt vector is not a constant – 代替割り込みベクトルが定数ではありません

割り込みベクトルの値は整数である必要があります。

alternate interrupt vector number *n* is not valid – 代替割り込みベクトルの数値 *n* が無効です

有効な割り込みベクトルの数値が必要です。

ambiguous abbreviation argument – 曖昧な省略引数です

指定されたコマンドラインの省略語が曖昧です。

an argument type that has a default promotion can't match an empty parameter name list declaration. – 空のパラメータ名リスト宣言に一致しないデフォルトの拡張を持つ引数です。

関数の宣言と定義は同一にする必要があります。

args to be formatted is not ... – フォーマットする引数は ... ではありません

フォーマット属性の最初のチェックのためのインデックス引数が「...」の内容で指定されていないパラメータを指定しています。

argument 'identifier' doesn't match prototype – 引数 'identifier' がプロトタイプと一致しません

関数の引数のタイプは、その関数のプロトタイプと同一にする必要があります。

argument of 'asm' is not a constant string – 'asm' の引数が定数の文字列ではありません

'asm' の引数は定数の文字列にする必要があります。

argument to '-B' is missing – '-B' の引数が見つかりません

ディレクトリ名が見つかりません。

argument to '-I' is missing – '-I' の引数が見つかりません

ライブラリ名が見つかりません。

argument to '-specs' is missing – '-spec' の引数が見つかりません

specs ファイル名が見つかりません。

argument to '-specs=' is missing – '-spec=' の引数が見つかりません

specs ファイル名が見つかりません。

argument to '-x' is missing – '-x' の引数が見つかりません

言語名が見つかりません。

argument to '-Xlinker' is missing – '-Xlinker' の引数が見つかりません

リンクに渡された引数が見つかりません。

arithmetic on pointer to an incomplete type – ポインタの演算が不完全なタイプです

不完全なタイプのポインタの演算は許可されません。

array index in non-array initializer – 配列インデックスは非配列イニシャライザです

非配列イニシャライザに配列インデックスを使用しないでください。

array size missing in 'identifier' – 'identifier' 内の配列サイズが見つかりません
配列サイズが見つかりません。

array subscript is not an integer – 配列サブスクリプトが整数ではありません
配列添字は整数にする必要があります。

'asm' operand constraint incompatible with operand size – 'asm' オペランド制約がオペランドのサイズに適合しません

asm ステートメントが無効です。

'asm' operand requires impossible reload – 'asm' オペランドが不可能なりロードを要求しています

asm ステートメントが無効です。

asm template is not a string constant – asm テンプレートが文字列定数ではありません

asm テンプレートは文字列定数にする必要があります。

assertion without predicate – アサーションに述語がありません

#assert または #unassert の後に単一の述語を付ける必要があります。

'attribute' attribute applies only to functions – 'attribute' 属性は関数にのみ適用されます

'attribute' 属性は関数にのみ適用します。

B

bit-field 'identifier' has invalid type – ビットフィールド 'identifier' が無効なタイプです

ビットフィールドは列挙型か整数タイプにする必要があります。

bit-field 'identifier' width not an integer constant – ビットフィールド 'identifier' の幅が整数定数ではありません

ビットフィールドの幅は整数定数にする必要があります。

both long and short specified for 'identifier' – ロングとショートの間方が 'identifier' に指定されています

変数のタイプはロングとショートの間方にはできません。

both signed and unsigned specified for 'identifier' – 符号付きと符号なしの間方が 'identifier' に指定されています

変数は符号付きと符号なしの間方であってはなりません。

braced-group within expression allowed only inside a function – 表現式内の中括弧で区切られたグループは関数内でのみ許可されます

表現式内の中括弧で区切られたグループを関数以外で使用するの是不正です。

break statement not within loop or switch – ループまたはスイッチ内にはないブレークステートメントです

ブレークステートメントはループまたはスイッチ内でのみ使用できます。

__builtin_longjmp second argument must be 1 – __builtin_longjmp の 2 番目の引数は 1 にする必要があります

__builtin_longjmp では 2 番目の引数を 1 にする必要があります。

C

called object is not a function – コールされたオブジェクトは関数ではありません
C 言語では関数のみコールできます。

cannot convert to a pointer type – ポインタータイプに変換できません
表現をポインタータイプに変換できません。

cannot put object with volatile field into register – volatile フィールドを持つオブジェクトはレジスタに保存できません
volatile フィールドを持つオブジェクトをレジスタに保存するのは不正です。

cannot reload integer constant operand in 'asm' – 整数オペランドを 'asm' の中でリロードできません
asm ステートメントが無効です。

cannot specify both near and far attributes – near 属性と far 属性の両方は指定できません
near 属性と far 属性は互いに排他的関係にあります。1つの関数または変数に使用できるのはいずれか1つのみです。

cannot take address of bit-field 'identifier' – ビットフィールド 'identifier' のアドレスを取得できません
ビットフィールドのアドレス取得を試行するのは不正です。

can't open 'file' for writing – 書き込み用の 'file' が開けません
システムは指定した 'file' を開けません。考えられる理由としては、ファイルを開くディスク容量が不足している、ディレクトリが存在しない、または保存先ディレクトリの書き込み許可がないことが挙げられます。

can't set 'attribute' attribute after definition – 定義後に 'attribute' 属性を設定できません
記号が定義されている場合、'attribute' 属性を使用する必要があります。

case label does not reduce to an integer constant – case ラベルを整数として判定できません
case ラベルはコンパイル時には整数である必要があります。

case label not within a switch statement – switch ステートメント内でない case ラベルです
case ラベルは switch ステートメント内に含める必要があります。

cast specifies array type – キャストが配列タイプを指定しています
キャストによる配列タイプの指定は許可されません。

cast specifies function type – キャストが関数タイプを指定しています
キャストによる関数タイプの指定は許可されません。

cast to union type from type not present in union – ユニオン内に存在しないタイプからユニオンタイプをキャストしています
ユニオンタイプにキャストする時はユニオン内に存在するタイプからキャストしてください。

char-array initialized from wide string – wide 文字列で初期化された char-array です
char-array は wide 文字列で初期化してはいけません。通常の文字列を使用してください。

file: compiler compiler not installed on this system – file: compiler コンパイラはこのシステムにインストールされていません

C コンパイラのみが配布されており、その他の高水準言語はサポートされていません。

complex invalid for ‘identifier’ – 複素数が ‘identifier’ に対して無効です

整数タイプと浮動小数点タイプにのみは複素数修飾子が適用されます。

conflicting types for ‘identifier’ – ‘identifier’ に矛盾したタイプです

identifier に複数の矛盾した宣言が存在します。

continue statement not within loop – continue ステートメントがループ内ではありません

continue ステートメントはループ内でのみ使用できます。

conversion to non-scalar type requested – 非スカラータイプへの変換がリクエストされました

タイプ変換は、スカラー（集合ではない）タイプへの変換でなくてはなりません。

D

data type of ‘name’ isn’t suitable for a register – ‘name’ のデータタイプがレジスタに適してません

データタイプがリクエストされたレジスタに適していません。

declaration for parameter ‘identifier’ but no such parameter – パラメータ ‘identifier’ が宣言されましたが、そのようなパラメータは存在しません

パラメータリスト内のパラメータのみが宣言可能です。

declaration of ‘identifier’ as array of functions – 関数の配列として ‘identifier’ が宣言されました

関数の配列を持つことは不正です。

declaration of ‘identifier’ as array of voids – void の配列として ‘identifier’ が宣言されました

void の配列を持つことは不正です。

‘identifier’ declared as function returning a function – ‘identifier’ が関数を返す関数として宣言されました

関数は関数を返すべきではありません。

‘identifier’ declared as function returning an array – ‘identifier’ が配列を返す関数として宣言されました

関数は配列を返すべきではありません。

decrement of pointer to unknown structure – ポインタが未知の構造体まで減分しています

ポインタを未知の構造体まで減分しないでください。

‘default’ label not within a switch statement – switch ステートメント内でない ‘default’ ラベルです

default の case ラベルは switch ステートメント内に含める必要があります。

‘symbol’ defined both normally and as an alias – ‘symbol’ が通常とエイリアスの両方に定義されています

すでに定義されている ‘symbol’ は他の記号のエイリアスに使用できません。

‘defined’ cannot be used as a macro name – ‘defined’ はマクロ名として使用できません

マクロ名を ‘defined’ とするべきではありません。

dereferencing pointer to incomplete type – 不完全なタイプへのデリファレンスポインタです

デリファレンスポインタは不完全なタイプへのポインタにしないでください。

division by zero in #if – #if 内でゼロの除算が行われました

ゼロの除算は計算できません。

duplicate case value – 重複した case 値です

case 値は一意である必要があります。

duplicate label 'identifier' – 重複したラベル 'identifier' です

ラベルはそのスコープ内で一意である必要があります。

duplicate macro parameter 'symbol' – 重複したマクロパラメータ 'symbol' です

'symbol' がパラメータリストで 2 回以上使用されています。

duplicate member 'identifier' – 重複したメンバー 'identifier' です

構造体は重複したメンバーを持つことはできません。

duplicate (or overlapping) case value – 重複した (またはオーバーラップした) case 値です

case は重複した、またはオーバーラップした値であってはなりません。'this is the first entry overlapping that value' というエラーメッセージが最初の重複値またはオーバーラップ値が発生した個所に表示されます。case の範囲は MPLAB C30 の ANSI 標準の拡張です。

E

elements of array 'identifier' have incomplete type – 配列 'identifier' のエレメントが不完全なタイプを含んでいます

配列エレメントは完全なタイプを含む必要があります。

empty character constant – 空の文字定数です

空の文字定数は不正です。

empty file name in '#keyword' – '#keyword' 内に空のファイル名があります

指定した #keyword の引数として指定したファイル名が空です。

empty index range in initializer – イニシャライザ内に空のインデックス範囲があります

イニシャライザ内で空のインデックス範囲を使用しないでください。

empty scalar initializer – 空のスカラーイニシャライザです

スカラーイニシャライザは空にしないでください。

enumerator value for 'identifier' not integer constant – 'identifier' の列挙型の値が整数定数ではありません

列挙値は整数定数にする必要があります。

error closing 'file' – 'file' クローズのエラーです

システムは指定した 'file' を閉じることができません。考えられる理由はファイルに書き込むためのディスク容量の不足か、ファイルサイズが大きすぎるかのいずれかです。

error writing to 'file' – 'file' 書き込みエラーです

システムは指定した 'file' に書き込みできません。考えられる理由はファイルに書き込むためのディスク容量の不足か、ファイルサイズが大きすぎるかのいずれかです。

excess elements in char array initializer – char 配列イニシャライザ内のエレメントが多すぎます

イニシャライザ値が示す以上の数のエレメントがリスト内にあります。

excess elements in struct initializer – 構造体イニシャライザ内のエレメントが多すぎます

イニシャライザ構造体内で過剰な数のエレメントを使用しないでください。

expression statement has incomplete type – 表現式ステートメントが不完全なタイプを含んでいます

表現式のタイプが不完全です。

extra brace group at end of initializer – イニシャライザの末尾に不要な中括弧グループがあります

イニシャライザの末尾に不要な中括弧グループを付けないでください。

extraneous argument to ‘option’ option – ‘option’ オプションとは無関係な引数です

指定したコマンドラインオプションにある引数が多すぎます。

F

‘identifier’ fails to be a typedef or built in type – ‘identifier’ は typedef タイプまたは built in タイプにできませんでした

データタイプは typedef または built-in にする必要があります。

field ‘identifier’ declared as a function – フィールド‘identifier’が関数として宣言されました

フィールドは関数として宣言されるべきはありません。

field ‘identifier’ has incomplete type – フィールド‘identifier’が不完全なタイプを含んでいます

フィールドは完全なタイプを含む必要があります。

first argument to __builtin_choose_expr not a constant – __builtin_choose_expr への最初の引数が定数ではありません

最初の引数はコンパイル時に決定される定数表現でなくてはなりません。

flexible array member in otherwise empty struct – フレキシブル配列メンバーか、空の構造体です

フレキシブル配列メンバーは 1 つ以上の名前が付いたメンバーを持つ構造体の最後のエレメントである必要があります。

flexible array member in union – フレキシブル配列メンバーがユニオン内にあります

フレキシブル配列メンバーはユニオン内では使用できません。

flexible array member not at end of struct – フレキシブル配列メンバーが構造体の末尾ではありません

フレキシブル配列メンバーは構造体の最後のエレメントである必要があります。

‘for’ loop initial declaration used outside C99 mode – ‘for’ ループの初期宣言が C99 モード外で使用されています

‘for’ ループの初期宣言は C99 モード外では無効です。

format string arg follows the args to be formatted – フォーマット文字列引数がフォーマットされる引数の後に付いています

フォーマット属性への引数が矛盾しています。フォーマット文字列引数のインデックスは最初の引数のインデックスより短くする必要があります。

format string arg not a string type – フォーマット文字列引数が文字列タイプではありません

フォーマット属性のフォーマット文字列インデックス引数が文字列タイプではないパラメータを指定しています。

format string has invalid operand number – フォーマット文字列は無効なオペランド数を含んでいます

フォーマット属性のオペランド番号引数はコンパイル時には定数でなければなりません。

**function definition declared ‘register’ – ‘register’ として宣言された関数定義です
関数定義で ‘register’ 宣言してはなりません。**

**function definition declared ‘typedef’ – ‘typedef’ として宣言された関数定義です
関数定義で ‘typedef’ としてはなりません。**

function does not return string type – 関数が文字列タイプを返しません

format_arg 属性は戻り値が文字列タイプの関数でのみ使用できます。

function ‘identifier’ is initialized like a variable – 関数 ‘identifier’ が変数のように初期化されています

関数を変数のように初期化することは不正です。

function return type cannot be function – 関数戻りタイプは関数にはなりません

関数の戻りタイプは関数にはなりません。

G

global register variable follows a function definition – グローバルレジスタ変数が関数定義の後に付いています

グローバルレジスタ変数は関数定義の前に付きます。

global register variable has initial value – グローバルレジスタ変数は初期値を含んでいます

グローバルレジスタ変数に初期値を指定しないでください。

global register variable ‘identifier’ used in nested function – グローバルレジスタ変数 ‘identifier’ がネストされた関数に使用されています

ネストされた関数内でグローバルレジスタ変数を使用しないでください。

H

‘identifier’ has an incomplete type – ‘identifier’ が不完全なタイプを含んでいます

指定した ‘identifier’ に不完全なタイプを使用するのは不正です。

‘identifier’ has both ‘extern’ and initializer – ‘identifier’ が ‘extern’ とイニシャライザの両方を含んでいます

‘extern’ として宣言された変数は初期化できません。

hexadecimal floating constants require an exponent – 16 進法浮動小数点定数は指数を求めています

16 進法浮動小数点定数には指数を含む必要があります。

I

implicit declaration of function ‘identifier’ – 関数 ‘identifier’ の暗示的宣言です

関数識別子は先行のプロトタイプ宣言または関数定義なしに使用しています。

impossible register constraint in 'asm' – 'asm'に含まれる不可能なレジスター制約です

asm ステートメントが無効です。

incompatible type for argument *n* of 'identifier' – 'identifier' の引数 *n* と互換性のないタイプです

C 言語内で関数をコールする際、実際の引数タイプが正式なパラメータタイプと一致していることを確認してください。

incompatible type for argument *n* of indirect function call – 間接的関数コールの引数 *n* と互換性のないタイプです

C 言語内で関数をコールする際、実際の引数タイプが正式なパラメータタイプと一致していることを確認してください。

incompatible types in operation – operation 内に互換性のないタイプがあります

operation 内で使用されるタイプは互換性のあるものでなくてはなりません。

incomplete 'name' option – 不完全な 'name' オプションです

コマンドラインパラメータ *name* のオプションが不完全です。

inconsistent operand constraints in an 'asm' – 'asm'に含まれる矛盾したオペランド制約です

asm ステートメントが無効です。

increment of pointer to unknown structure – ポインタが未知の構造体まで増分しています

ポインタを未知の構造体まで増分しないでください。

initializer element is not computable at load time – イニシャライザエレメントがロード時に計算できません

イニシャライザエレメントはロード時に計算可能である必要があります。

initializer element is not constant – イニシャライザエレメントが定数ではありません

イニシャライザエレメントは定数にする必要があります。

initializer fails to determine size of 'identifier' – イニシャライザが 'identifier' のサイズ計測ができませんでした

配列イニシャライザがサイズ計測に失敗しました。

initializer for static variable is not constant – 静的変数のイニシャライザが定数ではありません

静的変数イニシャライザは定数にする必要があります。

initializer for static variable uses complicated arithmetic – 静的変数のイニシャライザが複合演算を使用しています

静的変数イニシャライザは複合演算を使用しないでください。

input operand constraint contains 'constraint' – 'constraint' を含むオペランド制約です

指定した制約は入力オペランドには無効です。

int-array initialized from non-wide string – wide 文字列以外の文字列で初期化された int-array です

Int-array は wide 文字列以外の文字列で初期化してはいけません。

interrupt functions must not take parameters – 割り込み関数はパラメータをとるべきではありません

割り込み関数はパラメータを受け取れません。引数リストが空であることを明示的に宣言するため *void* を使用する必要があります。

interrupt functions must return void – 割り込み関数は void を返すべきです

割り込み関数は *void* の戻りタイプを持つ必要があります。他のタイプは許可されません。

interrupt modifier 'name' unknown – 割り込み修飾子 'name' が不明です

コンパイラは *'irq'*、*'altirq'* または *'save'* を割り込み属性修飾子として求めています。

interrupt modifier syntax error – 割り込み修飾子構文エラー

割り込み属性修飾子に構文エラーがあります。

interrupt pragma must have file scope – 割り込み pragma はファイル範囲を持つ必要があります

#pragma はファイル範囲内である必要があります。

interrupt save modifier syntax error – 割り込み保存修飾子構文エラー

割り込み属性の *'save'* 修飾子に構文エラーがあります。

interrupt vector is not a constant – 割り込みベクターが定数ではありません

割り込みベクトルの値は整数である必要があります。

interrupt vector number n is not valid – 割り込みベクターの数値 n が無効です

有効な割り込みベクトルの数値が必要です。

invalid #ident directive – 無効な #ident ディレクティブ

#ident の後に引用符付きの文字列リテラルを付ける必要があります。

invalid arg to '__builtin_frame_address' – '__builtin_frame_address' への無効な引数

引数は関数のコーラーのレベル (0 が現在の関数のフレームアドレスを表し、1 が現在の関数のコーラーのフレームアドレスを表し、後続の数字もこれに続く) で整数リテラルである必要があります。

invalid arg to '__builtin_return_address' – '__builtin_return_address' への無効な引数

レベル引数は整数リテラルである必要があります。

invalid argument for 'name' – 'name' への無効な引数

コンパイラは *'data'* または *'prog'* を空間属性パラメータとして求めています。

invalid character 'character' in #if – #if 内に無効な文字 'character' があります

このメッセージは制御文字などの印刷されない文字が *#if* の後に付いた場合に表示されます。

invalid initial value for member 'name' – メンバー 'name' の初期値が無効です

ビットフィールド *'name'* は整数でのみ初期化可能です。

invalid initializer – 無効なイニシャライザ

無効なイニシャライザを使用しないでください。

Invalid location qualifier: 'symbol' – 無効なローケーション修飾子: 'symbol'

ローケーション修飾子として、dsPIC DSC デバイスでは無視される *'sfr'* または *'gpf'* を求めています。

invalid operands to binary 'operator' – バイナリ 'operator' への無効なオペランド

そのバイナリ演算子へのオペランドは無効です。

Invalid option 'option' – 無効なオプション 'option'

指定したコマンドラインオプションが無効です。

Invalid option '*symbol*' to interrupt pragma – 割り込み pragma に無効なオプション '*symbol*'

割り込み pragma へのオプションとして shadow および / または save を求めています。

Invalid option to interrupt pragma – 割り込み pragma に無効なオプション

pragma の末尾の不要部分です。

Invalid or missing function name from interrupt pragma – 割り込み pragma の無効か見つからない関数名です

割り込み pragma にはコールされる関数名が必要です。

Invalid or missing section name – 無効または見つからないセクション名

セクション名は必ず文字または下線 ('_') で始め、連続した文字列、下線および / または数字を後に付ける必要があります。'*access*'、'*shared*'、'*overlay*' などの名前には特別な意味があります。

invalid preprocessing directive #'*directive*' – 無効なプリプロセッシングディレクティブ #'*directive*'

有効なプリプロセッシングディレクティブではありません。スペルをチェックしてください。

invalid preprologue argument – 無効なプリプロローグ引数

プリプロローグオプションがアセンブリステートメントまたは二重引用符でくくられた引数のステートメントを求めています。

invalid register name for '*name*' – '*name*' の無効なレジスタ名

ファイル範囲変数 '*name*' が不正なレジスタ名を持つレジスタ変数として宣言されました。

invalid register name '*name*' for register variable – レジスタ変数の無効なレジスタ名 '*name*'

指定した名前はレジスタ名ではありません。

invalid save variable in interrupt pragma – 割り込み pragma 内に無効な保存変数があります

保存には 1 つ以上の記号が必要です。

invalid storage class for function '*identifier*' – 関数 '*identifier*' の無効なストレージクラス

関数は 'register' ストレージクラスを持つことはできません。

invalid suffix '*suffix*' on integer constant – 整数の無効な接尾辞 '*suffix*'

整数の接尾辞に使用できる文字は 'u'、'U'、'l'、'L' のみです。

invalid suffix on floating constant – 浮動小数点定数の無効な接尾辞です

浮動小数点定数に使用できる文字は 'f'、'F'、'l'、'L' のみです。'L' が 2 つある場合、これらは隣接であり同じケースである必要があります。

invalid type argument of '*operator*' – '*operator*' の無効なタイプの引数

operator への引数のタイプが無効です。

invalid type modifier within pointer declarator – ポインタ宣言子内に無効なタイプの修飾子があります

ポインタ宣言子内でタイプ修飾子として使用できるのは const か volatile のタイプのみです。

invalid use of array with unspecified bounds – 指定されていない境界を持つ配列の使用法は正しくありません

指定されていない限界を持つ配列は正しい方法で使用してください。

invalid use of incomplete typedef ‘typedef’ – 不完全なタイプ定義 ‘typedef’ が正しく使用されていません

指定した *typedef* は無効な方法で使用されていますので、許可されません。

invalid use of undefined type ‘type identifier’ – 未定義のタイプ ‘type identifier’ の無効な使用

指定した *type* は無効な方法で使用されていますので、許可されません。

invalid use of void expression – void 表現の無効な使用

void 表現は使用しないでください。

“name” is not a valid filename – “name” は無効なファイル名です

#line は有効なファイル名を求めています。

‘filename’ is too large – ‘filename’ が大きすぎます

指定したファイルは処理するにはサイズが大きすぎます。サイズが 4 GB を超えている可能性があるため、プリプロセッサがサイズの大きいファイル処理を拒否しています。ファイルサイズは 4GB 未満にして下さい。

ISO C forbids data definition with no type or storage class – ISO C ではタイプなしまたはストレージクラスなしのデータ定義を禁止しています

タイプ限定子またはストレージクラス限定子が ISO C 内でのデータ定義に必要です。

ISO C requires a named argument before ‘...’ – ISO C は ‘...’ の前に名前つき引数を付けることを求めています

ISO C は ‘...’ の前に名前つき引数を付けることを求めています。

L

label *label* referenced outside of any function – ラベル ‘label’ が関数の外側で参照されました

ラベルは関数内でのみ参照できます。

label ‘*label*’ used but not defined – ラベル ‘label’ が使用されましたが定義されていません

指定したラベルは使用されましたが定義されていません。

language ‘name’ not recognized – 言語 ‘name’ は認識されません

許可される言語は C、アセンブラ、None です

***filename*: linker input file unused because linking not done – ファイル名: リンクは実行されないためリンカ入力ファイルが使用されていません**

指定したファイル名がコマンドラインで指定されましたが、リンカ入力ファイルとして認識されました (他のものとして認識されないため)。ただし、リンクは実行されていません。よって、このファイルは無視されました。

long long long is too long for GCC – long long long は GCC には大きすぎます

MPLAB C30 は long long より長い整数に対応していません。

long or short specified with char for ‘*identifier*’ – long または short が ‘*identifier*’ の char で指定されました

long 修飾子および short 修飾子は char タイプと一緒に使用できません。

long or short specified with floating type for ‘*identifier*’ – long または short が ‘*identifier*’ の浮動小数点タイプで指定されました

long 修飾子および short 修飾子は浮動小数点タイプと一緒に使用できません。

long, short, signed or unsigned invalid for ‘*identifier*’ – ‘*identifier*’ には、long、short、signed、unsigned は無効です

long、short、signed 修飾子は整数タイプでしか使用できません。

M

macro names must be identifiers – マクロ名は識別子にしてください

マクロ名は必ず文字か下線で始め、その後にさらに文字、数字、下線などを付けます。

macro parameters must be comma-separated – マクロパラメータはコンマで区切る必要があります

パラメータリスト内では、パラメータの間にコンマが必要です。

macro 'name' passed *n* arguments, but takes just *n* – マクロ 'name' は引数を *n* 個引き渡されましたが、*n* 個しか受け取っていません

マクロ 'name' に渡された引数が多すぎます。

macro 'name' requires *n* arguments, but only *n* given – マクロ 'name' は *n* 個の引数を要求しましたが、*n* 個しか与えられていません

マクロ 'name' に渡された引数が少なすぎます。

matching constraint not valid in output operand – 出力オペランド内では一致制約は無効です

asm ステートメントが無効です。

'symbol' may not appear in macro parameter list – マクロパラメータリスト内に 'symbol' が含まれるべきではありません

'symbol' はパラメータとして認められません。

Missing '=' for 'save' in interrupt pragma – 割り込み pragma 内の 'save' には '=' を付けていません

save パラメータはリストされている変数の前に等符号 (=) を付けることを要求します。例えば、`#pragma interrupt isr0 save=var1,var2` のようになります。

missing '(' after predicate – 述語の後に '(' が見つかりません

`#assert` または `#unassert` は answer の前後に括弧を付けるよう要求します。例えば、次のようになります。`ns#assert PREDICATE (ANSWER)`

missing '(' in expression – 表現内に '(' が見つかりません

括弧付けが一致しません。始まりの括弧が必要です。

missing ')' after "defined" – "defined" の後に ')' が見つかりません

閉じ括弧が必要です。

missing ')' in expression – 表現内に ')' が見つかりません

括弧付けが一致しません。閉じ括弧が必要です。

missing ')' in macro parameter list – マクロパラメータリストに ')' が見つかりません

マクロはカッコにくくられ、コンマで区切られたパラメータを要求しています。

missing ')' to complete answer – answer を完全にするための ')' が見つかりません

`#assert` または `#unassert` は answer の前後に括弧を付けるよう要求します。

missing argument to 'option' option – 'option' オプションへの引数が見つかりません

指定したコマンドラインオプションは引数を要求しています。

missing binary operator before token 'token' – 'token' の前にバイナリ演算子が見つかりません

'token' の前に演算子が必要です。

missing terminating 'character' character – 終了文字'character'が見つかりません
一重引用符'、二重引用符"または右山括弧>などの終了文字が見つかりません。

missing terminating > character – 終了文字>が見つかりません

#include ディレクティブ内には終了文字>が必要です。

more than *n* operands in 'asm' – 'asm' 内に *n* 個以上のオペランドがあります

asm ステートメントが無効です。

multiple default labels in one switch – 1 つの switch 内に複数の default ラベルがあります

各 switch に指定できる default ラベルは1つだけです。

multiple parameters named 'identifier' – 複数のパラメータに 'identifier' という名前が付いています

パラメータ名は一意である必要があります。

multiple storage classes in declaration of 'identifier' – 'identifier' 宣言の中に複数のストレージクラスがあります

各宣言に含めるストレージクラスは1つだけです。

N

negative width in bit-field 'identifier' – ビットフィールド 'identifier' が負の幅です
ビットフィールド幅は負の値にしないでください。

nested function 'name' declared 'extern' – ネストされた関数'name'が'extern'宣言されました

ネストされた関数には'extern'を宣言できません。

nested redefinition of 'identifier' – 'identifier' のネストされた再定義です

ネストされた再定義は不正です。

no data type for mode 'mode' – モード 'mode' のデータタイプがありません

モード属性に指定された引数モードは認識できる GCC マシンモードですが、MPLAB C30 では実装されていません。

no include path in which to find 'name' – 'name' を見つけるためのインクルードパスがありません

インクルードファイル 'name' が見つかりません。

no macro name given in '#directive' directive – '#directive' ディレクティブ内にマクロ名がありません

マクロ名の前には#define, #undef, #ifdef または #ifndef ディレクティブを付ける必要があります。

nonconstant array index in initializer – イニシャライザ内に非定数配列インデックスがあります

イニシャライザ内で使用できるのは定数配列インデックスのみです。

non-prototype definition here – 非プロトタイプ定義があります

関数プロトタイプがプロトタイプなしの定義の後に続き、2つの間で引数の数に矛盾が生じた場合、このメッセージは非プロトタイプ定義の行番号を示します。

number of arguments doesn't match prototype – 引数の数がプロトタイプと一致しません

関数の引数の数は、その関数のプロトタイプと一致する必要があります。

O

operand constraint contains incorrectly positioned '+' or '='. – オペランド制約が誤った位置の '+' または '=' を含んでいます

asm ステートメントが無効です。

operand constraints for 'asm' differ in number of alternatives – 'asm' のオペランド制約の数が異なります

asm ステートメントが無効です。

operator "defined" requires an identifier – 演算子 "defined" には識別子が必要です
"defined" は識別子を求めています。

operator 'symbol' has no right operand – 演算子 'symbol' に右のオペランドがありません

プリプロセッサ演算子 'symbol' の右にオペランドが必要です。

output number *n* not directly addressable – 出力番号 *n* は直接アドレスできません

asm ステートメントが無効です。

output operand constraint lacks '=' – 出力オペランド制約に '=' がありません

asm ステートメントが無効です。

output operand is constant in 'asm' – 'asm' 内の出力オペランドが定数です

asm ステートメントが無効です。

overflow in enumeration values – 列挙型値内のオーバーフローです

列挙型値は 'int' の範囲内でなくてはなりません。

P

parameter 'identifier' declared void – パラメータ 'identifier' は void を宣言しました

パラメータは void を宣言してはいけません。

parameter 'identifier' has incomplete type – パラメータ 'identifier' が不完全なタイプを含んでいます

パラメータは完全なタイプを含む必要があります。

parameter 'identifier' has just a forward declaration – パラメータ 'identifier' は前方宣言のみを含んでいます

パラメータは完全なタイプを含む必要があります。前方宣言は不完全です。

parameter 'identifier' is initialized – パラメータ 'identifier' が初期化されました

パラメータを初期化するのは不正です。

parameter name missing – パラメータ名が見つかりません

マクロはパラメータ名を要求しています。間に名前が入っていない2つのコンマがないか確認してください。

parameter name missing from parameter list – パラメータ名がパラメータリストから見つかりません

パラメータリスト内にはパラメータ名を含める必要があります。

parameter name omitted – パラメータ名が省かれています

パラメータ名は省くべきではありません。

param types given both in param list and separately – パラメータタイプがパラメータリスト内と個別に両方の形で存在します

パラメータタイプはパラメータリストか、個別に指定します。両方で指定できません。

parse error – 構文解釈エラー

ソース行は構文解析できません。エラーを含んでいます。

pointer value used where a complex value was expected – 複素数値が必要なところでポインタ値が使用されています

複素数値が必要なところでポインタ値を使用しないでください。

pointer value used where a floating point value was expected – 浮動小数点値が必要なところでポインタ値が使用されています

浮動小数点が必要なところでポインタ値を使用しないでください。

pointers are not permitted as case values – ポインタは case 値として許可されません

case 値は整数または定数表現でなくてはなりません。

predicate must be an identifier – 述語は識別子でなくてはなりません

#assert または #unassert には述語として識別子が 1 つ必要です。

predicate's answer is empty – 述語の answer が空です

#assert または #unassert は述語と括弧を持っていますが、括弧内に必要な answer がありません。

previous declaration of 'identifier' – 'identifier' の以前の宣言です

このメッセージは以前の識別子の宣言の場所を特定し、現在の宣言とは矛盾します。

identifier previously declared here – identifier が以前ここで宣言されました

このメッセージは以前の識別子の宣言の場所を特定し、現在の宣言とは矛盾します。

identifier previously defined here – identifier が以前ここで定義されました

このメッセージは以前の識別子の定義の場所を特定し、現在の定義とは矛盾します。

prototype declaration – プロトタイプ宣言

関数プロトタイプが宣言された場所を行番号で特定します。他のエラーメッセージと一緒に使用されます。

R

redeclaration of 'identifier' – 'identifier' の再宣言

identifier が重複して宣言されています。

redeclaration of 'enum identifier' – 'enum identifier' の再宣言

Enums は再宣言しないでください。

'identifier' redeclared as different kind of symbol – 'identifier' は異なる種類の記号として再宣言されました

identifier に複数の矛盾した宣言が存在します。

redefinition of 'identifier' – 'identifier' の再定義

identifier が重複して定義されています。

redefinition of 'struct identifier' – 'struct identifier' の再定義

Structs は再定義するべきではありません。

redefinition of 'union identifier' – 'union identifier' の再定義

Unions は再定義するべきではありません。

register name given for non-register variable 'name' – 非レジスタ変数 'name' にレジスタ名が付与されました

レジスタとしてマークされていない変数へレジスタのマッピングが試みられました。

register name not specified for 'name' – 'name' にレジスタ名が指定されていません

ファイル範囲変数 'name' がレジスタの提供なしにレジスタ変数として宣言されました。

register specified for 'name' isn't suitable for data type – 'name' に指定されたレジスタがデータタイプに適してません

配列またはその他制約があるため、リクエストされたレジスタが使用できません。

request for member 'identifier' in something not a structure or union – 構造体またはユニオンではない何かのメンバー 'identifier' のリクエスト

メンバーを持っているのは構造体またはユニオンのみです。そのため、それ以外のメンバーを参照するのは不正です。

requested alignment is not a constant – リクエストされた整列引数が定数ではありません

aligned 属性の引数はコンパイル時には定数でなければなりません。

requested alignment is not a power of 2 – リクエストされた整列引数が 2 の乗数ではありません

aligned 属性の引数は 2 の乗数でなければなりません。

requested alignment is too large – リクエストした整列引数が大きすぎます

リクエストした整列サイズがリンカが許可するサイズを超えています。サイズは 4096 以下の 2 の乗数にしてください。

return type is an incomplete type – 戻りタイプが不完全なタイプです

戻りタイプは完全である必要があります。

S

save variable 'name' index not constant – 保存変数 'name' のインデックスが定数ではありません

配列 'name' のサブスクリプトが整数ではありません。

save variable 'name' is not word aligned – 保存変数 'name' がワード整列ではありません

保存されるオブジェクトはワード整列にしてください。

save variable 'name' size is not even – 保存変数 'name' のサイズに端数があります
保存するオブジェクトのサイズは端数があってははいけません。

save variable 'name' size is not known – 保存変数 'name' のサイズが不明です

保存するオブジェクトのサイズは明確でなくてはなりません。

section attribute cannot be specified for local variables – section 属性はローカル変数には指定できません

ローカル変数は常にレジスタ内またはスタック上に割り当てます。よって、ローカル変数をセクションに配置するのは不正です。

section attribute not allowed for identifier – identifier には section 属性は許可されていません

section 属性は関数または変数でしか使用できません。

section of identifier conflicts with previous declaration – identifier のセクションが以前の宣言と矛盾します

同じ識別子の複数の宣言が section 属性を指定している場合、属性の値は一貫している必要があります。

sfr address 'address' is not valid – SFR アドレス 'address' が有効ではありません
有効なアドレスは 0x2000 未満です。

sfr address is not a constant – SFR アドレスが定数ではありません
SFR アドレスは定数でなくてはなりません。

'size of' applied to a bit-field – ビットフィールドに 'size of' が適用されています
'sizeof' はビットフィールドに適用しないでください。

size of array 'identifier' has non-integer type – 配列のサイズ 'identifier' は整数以外の
のタイプを含んでいます

配列サイズは整数タイプでなくてはなりません。

size of array 'identifier' is negative – 'identifier' のサイズが負の値です
配列サイズは負の値にしないでください。

size of array 'identifier' is too large – 'identifier' のサイズが大きすぎます
指定した配列が大きすぎます。

size of variable 'variable' is too large – 変数 'variable' のサイズが大きすぎます
変数の最大サイズは 32768 バイトです。

storage class specified for parameter 'identifier' – ストレージクラスがパラメータ
'identifier' に指定されました

ストレージクラスはパラメータの指定に使用できません。

storage size of 'identifier' isn't constant – 'identifier' のストレージサイズが定数で
はありません

ストレージサイズはコンパイル時には定数である必要があります。

storage size of 'identifier' isn't known – 'identifier' のストレージサイズが不明です
identifier のサイズが不完全に指定されています。

stray 'character' in program – プログラム内にストレー 'character' が存在します
ソースプログラムにストレー 'character' 文字を配置しないでください。

strftime formats cannot format arguments – strftime フォーマットは引数で形成さ
れません

archetype パラメータが strftime の場合、属性の 3 番目のパラメータは、最初のパラ
メータがフォーマット文字列に一致するよう指定され、値は 0 になります。strftime
スタイルの関数はフォーマット文字列に一致する入力値を持ちません。

structure has no member named 'identifier' – 構造は 'identifier' という名前のメン
バーを持ちません

'identifier' という名前の構造メンバーが参照されましたが、参照先の構造にそのよ
うなメンバーは存在しません。そのような参照は許可されません。

subscripted value is neither array nor pointer – サブスクリプト値が配列、ポイン
タのどちらでもありません

配列またはポインタのみがサブスクリプトとして使用できます。

switch quantity not an integer – switch の数量が整数ではありません
switch の数量は整数にする必要があります。

symbol 'symbol' not defined – 記号 'symbol' が定義されていません
記号 'symbol' は pragma 内で使用される前に宣言される必要があります。

syntax error – 構文エラー

指定した行に構文エラーがあります。

syntax error ‘:’ without preceding ‘?’ – 構文エラー ‘:’ の前に ‘?’ がない ‘:’

‘:’ の前には ‘?’ 演算子内の ‘?’ を付ける必要があります。

T

the only valid combination is ‘long double’ – 有効な組み合わせは ‘long double’ のみです

long 修飾子は double タイプと一緒に使用できる唯一の修飾子です。

this built-in requires a frame pointer – この built-in にはフレームポインタが必要です

`__builtin_return_address` はフレームポインタが必要です。

`-fomit-frame-pointer` オプションは使用しないでください。

this is a previous declaration – 以前の宣言です

ラベルが重複している場合、このメッセージは以前の宣言の行数を知らせます。

too few arguments to function – 関数への引数が少なすぎます

C 内で関数を呼び出す場合は、関数が要求するより少ない引数を指定しないでください。多く指定してもいけません。

too few arguments to function ‘*identifier*’ – 関数 ‘*identifier*’ への引数が少なすぎます

C 内で関数を呼び出す場合は、関数が要求するより少ない引数を指定しないでください。多く指定してもいけません。

too many alternatives in ‘asm’ – ‘asm’ 内の選択肢が多すぎます

asm ステートメントが無効です。

too many arguments to function – 関数への引数が多すぎます

C 内で関数を呼び出す場合は、関数が要求するより多い引数を指定しないでください。少なく指定してもいけません。

too many arguments to function ‘*identifier*’ – 関数 ‘*identifier*’ への引数が多すぎます

C 内で関数を呼び出す場合は、関数が要求するより多い引数を指定しないでください。少なく指定してもいけません。

too many decimal points in number – 数字に 10 進小数点が多すぎます

必要な 10 進小数点は 1 つのみです。

top-level declaration of ‘*identifier*’ specifies ‘auto’ – ‘*identifier*’ の最上位宣言は ‘auto’ を指定しました

オート変数は関数内でのみ宣言可能です。

two or more data types in declaration of ‘*identifier*’ – ‘*identifier*’ の宣言に 2 つ以上のデータタイプがあります

各識別子は 1 つのデータタイプしか持てません。

two types specified in one empty declaration – 1 つの空の宣言内に 2 つのタイプが指定されています

1 つ以上のタイプを指定しないでください。

type of formal parameter *n* is incomplete – 仮パラメータのタイプ *n* が不完全です
表示されたパラメータには完全なタイプを指定してください。

type mismatch in conditional expression – 条件式内のタイプが一致しません

条件式内のタイプは不一致であってははいけません。

typedef 'identifier' is initialized – typedef 'identifier' が初期化されました
typedef を初期化するのは不正です。代わりに `__typeof__` を使用してください。

U

'identifier' undeclared (first use in this function) – 'identifier' が宣言されていません (この関数内では初めて使用)

指定された識別子を宣言する必要があります。

'identifier' undeclared here (not in a function) – ここでは 'identifier' が宣言されていません (関数内以外)

指定された識別子を宣言する必要があります。

union has no member named 'identifier' – ユニオンは 'identifier' という名前のメンバーを持っていません

'identifier' という名前のユニオンメンバーが参照されましたが、参照先のユニオンに該当するメンバーが存在しませんので、参照は許可されません。

unknown field 'identifier' specified in initializer – イニシャライザ内で未知のフィールド 'identifier' が指定されました

イニシャライザ内で未知のフィールドを使用しないでください。

unknown machine mode 'mode' – 未知のマシンモード 'mode'

モード属性に指定された引数 *mode* は認識可能なマシンモードではありません。

unknown register name 'name' in 'asm' – 'asm' 内に未知のレジスタ名があります

asm ステートメントが無効です。

unrecognized format specifier – 認識されないフォーマット規定子です

フォーマット属性への引数が無効です。

unrecognized option '-option' – 認識されないオプション '-option'

指定したコマンドラインオプションが認識されません。

unrecognized option 'option' – 認識されないオプション 'option'

'option' は未知のオプションです。

'identifier' used prior to declaration – 宣言の前に 'identifier' が使用されています

識別子はその宣言の前に使用されています。

unterminated #'name' – 終端のない #'name'

#endif は #if、#ifdef または #ifndef 条件の終端である必要があります。

unterminated argument list invoking macro 'name' – 終端のない引数リストがマクロ 'name' を呼び出しています

関数マクロの評価で、マクロ拡張完了前にファイルの終わりが検出されました。

unterminated comment – 終端のないコメント

コメントターミネータをスキャンしている間にファイルの終わりに達しました。

V

'va_start' used in function with fixed args – 固定引数を持つ関数内で 'va_start' が使用されました

変数引数リストを持つ関数内でのみ 'va_start' を使用できます。

variable 'identifier' has initializer but incomplete type – 変数 'identifier' はイニシャライザを持っていますが、タイプが不完全です

不完全なタイプを持つ変数を初期化するのは不正です。

variable or field 'identifier' declared void – 変数またはフィールド 'identifier' は void を宣言しました

変数およびフィールドは void を宣言してはいけません。

variable-sized object may not be initialized – 変数サイズのオブジェクトは初期化できません

変数サイズのオブジェクトを初期化することは不正です。

virtual memory exhausted – 仮想メモリが不足しています

エラーメッセージを記述するのに必要なメモリが不足しています。

void expression between '(' and ')' – '('and')' の間に void 表現があります

定数表現を期待していますが、括弧の間に void 表現があります。

'void' in parameter list must be the entire list – パラメータリスト内の 'void' は完全なリストでなくてはなりません

'void' パラメータがパラメータリストに使用される場合、他のパラメータが存在してはなりません。

void value not ignored as it ought to be – void 値が通常のように無視されませんでした

void 関数の値は表現式には使用できません。

W

warning: -pipe ignored because -save-temps specified – 警告: -save-temps が指定されたため、-pipe が無視されました

-pipe オプションは -save-temps オプションと一緒に使用できません。

warning: -pipe ignored because -time specified – 警告: -time が指定されたため、-pipe が無視されました

-pipe オプションは -time オプションと一緒に使用できません。

warning: '-x spec' after last input file has no effect – 警告: 最後の入力ファイルの後の '-x spec' は効果がありません

'-x' コマンドラインオプションは、そのコマンドライン上で名前が後ろに付けられたファイルにのみ影響を及ぼします。そのようなファイルがない場合、このオプションには効果がありません。

weak declaration of 'name' must be public – 'name' の弱型宣言はパブリックにする必要があります

弱型記号は外部から見えるようにする必要があります。

weak declaration of 'name' must precede definition – 'name' の弱型宣言は定義より前に来る必要があります

'name' は定義されてから弱型宣言をしています。

wrong number of arguments specified for attribute attribute – attribute 属性に誤った引数の数が指定されています

'attribute' という名前の属性に与えられている引数が少なすぎるか、多すぎます。

wrong type argument to bit-complement – bit-complement の引数のタイプが誤っています

この演算子には、誤ったタイプの引数は使用できません。

wrong type argument to decrement – 減分の引数のタイプが誤っています

この演算子に誤ったタイプの引数を使用しないでください。

wrong type argument to increment – 増分の引数のタイプが誤っています

この演算子に誤ったタイプの引数を使用しないでください。

wrong type argument to unary exclamation mark – 単項感嘆符の引数のタイプが誤っています

この演算子に誤ったタイプの引数を使用しないでください。

wrong type argument to unary minus – 単項マイナスの引数のタイプが誤っています

この演算子に誤ったタイプの引数を使用しないでください。

wrong type argument to unary plus – 単項プラスの引数のタイプが誤っています

この演算子に誤ったタイプの引数を使用しないでください。

Z

zero width for bit-field '*identifier*' – ビットフィールド '*identifier*' の幅がゼロです
ビットフィールドの幅はゼロにできません。

B.3 警告

記号

'/*' within comment – コメント内に '/*' があります

コメントマークがコメント内で見つかりました。

'\$' character(s) in identifier or number – 識別子または数字内に '\$' が含まれていません

識別子名内のドル記号は、標準の機能拡張です。

#'directive' is a GCC extension – #'directive' は GCC の機能拡張です

#warning、#include_next、#ident、#import、#assert、#unassert ディレクティブは GCC の機能拡張であり、ISO C89 の機能拡張ではありません。

#import is obsolete, use an #ifndef wrapper in the header file – #import は廃止されています。ヘッダーファイル内では #ifndef を使用してください

#import ディレクティブは廃止されています。#import は、インクルードされていないファイルをインクルードするのに使用されていました。代わりに #ifndef ディレクティブを使用してください。

#include_next in primary source file – プライマリソースファイル内に #include_next があります

#include_next は現在のファイルが見つかったディレクトリの後にあるヘッダーファイルディレクティブのリストを検索します。この場合、それ以前にはヘッダーファイルはないため、プライマリソースファイルから検索が始まります。

#pragma pack (pop) encountered without matching #pragma pack (push, <n>) – #pragma pack (push, <n>) とのマッチングなしに、#pragma pack (pop) が見つかりました

pack(pop) pragma は、ソースファイル内では前にある pack(push) pragma とペアになる必要があります。

#pragma pack (pop, identifier) encountered without matching #pragma pack (push, identifier, <n>) – #pragma pack (push, identifier, <n>) とのマッチングなしに #pragma pack (pop, identifier) が見つかりました

pack(pop) pragma は、ソースファイル内では前にある pack(push) pragma とペアになる必要があります。

#warning: message – # 警告 : メッセージ

ディレクティブ #warning はプリプロセッサに警告を発生させプリプロセスを続行させます。#warning の後に続くトークンは警告メッセージとして使用されます。

A

absolute address specification ignored – 絶対アドレス仕様が無視されました

MPLAB C30 でサポートされていないため、#pragma ステートメント内のコードセクションの絶対アドレス仕様を無視します。アドレスはリンクスクリプト内で指定するか、コードセクション内で __attribute__ キーワードで指定する必要があります。

address of register variable 'name' requested – レジスタ変数 'name' のアドレス請求

レジスタ指定が変数のアドレスを取得するのを防ぎます。

alignment must be a small power of two, not n – 整列は n ではなく、2 の小さな乗数にする必要があります

パック pragma の整列パラメータは 2 の乗数でなければなりません。

anonymous enum declared inside parameter list – 匿名 enum がパラメータリスト内で宣言されました

匿名 enum が関数パラメータリスト内で宣言されました。通常のよりよいプログラミング慣習としては、パラメータリスト外で enum を宣言します。パラメータリスト内で定義される場合、完全なタイプとなりません。

anonymous struct declared inside parameter list – 匿名構造体がパラメータリスト内で宣言されました

匿名構造体が関数パラメータリスト内で宣言された。通常のよりよいプログラミング慣習としては、パラメータリスト外で構造体を宣言します。パラメータリスト内で定義される場合、完全なタイプとなりません。

anonymous union declared inside parameter list – 匿名ユニオンがパラメータリスト内で宣言されました

匿名ユニオンが関数パラメータリスト内で宣言された。通常のよりよいプログラミング慣習としては、パラメータリスト外でユニオンを宣言します。パラメータリスト内で定義される場合、完全なタイプとなりません。

anonymous variadic macros were introduced in C99 – 匿名 variadic マクロが C99 に導入されました

引数の変数を受け入れるマクロは C99 の機能です。

argument 'identifier' might be clobbered by 'longjmp' or 'vfork' – 引数 'identifier' は 'longjmp' または 'vfork' によって修飾される可能性があります

引数は longjmp へのコールにより変更される可能性があります。この警告はコンパイルの最適化時にのみ生成されます。

array 'identifier' assumed to have one element – 配列 'identifier' はエレメントを1つ持つと仮定しました

配列の長さは明示的に指定されていません。情報がない場合、コンパイラは配列は1つのエレメントを持つと仮定します。

array subscript has type 'char' – 配列添え字はタイプ 'char' を持ちます

配列添え字はタイプ 'char' を持ちます。

array type has incomplete element type – 配列タイプは不完全なエレメントタイプを持ちます

配列タイプは不完全なエレメントタイプを持つことはできません。

asm operand *n* probably doesn't match constraints – asm オペランド *n* は制約とマッチしない可能性があります

指定した拡張 asm オペランドは制約とマッチしない可能性があります。

assignment of read-only member 'name' – 読み取り専用メンバー 'name' の割り当てです

メンバー 'name' が const として宣言され、割り当てでは変更できません。

assignment of read-only variable 'name' – 読み取り専用変数 'name' への代入です

'name' が const として宣言され、代入では変更できません。

'identifier' attribute directive ignored – 'identifier' 属性ディレクティブが無視されました

属性が不明か、サポートされないため無視されました。

'identifier' attribute does not apply to types – 'identifier' 属性はタイプに適用されません

属性はタイプには使用できないため、無視されます。

'identifier' attribute ignored – 'identifier' 属性が無視されました

属性が与えられた文脈では意味を成さないため無視されました。

'attribute' attribute only applies to function types – 'attribute' 属性は関数タイプにのみ適用されます

指定された属性は関数の戻りタイプにのみ適用され、他の宣言には適用されません。

B

backslash and newline separated by space – スペースで区切られたバックスラッシュと復帰改行文字です

エスケープシーケンスの処理中に、スペースで区切られたバックスラッシュと復帰改行文字が検出されました。

backslash-newline at end of file – ファイルの最後にバックスラッシュ-復帰改行文字があります

エスケープシーケンスの処理中に、ファイルの最後にあるバックスラッシュと復帰改行文字が検出されました。

bit-field 'identifier' type invalid in ISO C – ビットフィールド 'identifier' タイプは ISO C では無効です

指定した識別子で使用されているタイプは ISO C では有効ではありません。

braces around scalar initializer – スカラーイニシャライザの前後に中括弧がありません

イニシャライザの前後に冗長な中括弧が付いています。

built-in function 'identifier' declared as non-function – built-in 関数 'identifier' が非関数として宣言されました

指定された関数は built-in 関数と同じ名前を持っていますが、関数以外のものとして宣言されました。

C

C++ style comments are not allowed in ISO C89 – C++ スタイルコメントは ISO C89 では許可されていません

C スタイルコメント `/*` と `*/` を C++ スタイルコメント `//` の代わりに使用してください。

call-clobbered register used for global register variable – グローバルレジスタ変数に call-clobbered が使用されています

関数コールで普通に保存、復帰されるレジスタ (W8-W13) を選択してください。ライブラリルーチンがそれらを上書きせずに済みます。

cannot inline function 'main' – 関数 'main' をインライン化できません

関数 `main` が `inline` 属性と一緒に宣言されました。個別にコンパイルされる C スタートアップコードからメインを呼び出さなくてはならないため、これはサポートされていません。

can't inline call to 'identifier' called from here – ここからコールされた 'identifier' へのコールをインライン化できません

コンパイラは指定された関数へのコールをインライン化できませんでした。

case value 'n' not in enumerated type – 列挙型タイプでないケース値 'n' です

`switch` ステートメントの制御表現は列挙型ですが、`case` 表現が列挙型の値に対応しない値 `n` を持っています。

case value 'value' not in enumerated type 'name' – 列挙型タイプ 'name' がない case 値 'value' です

'value' は追加の switch case で列挙型タイプ 'name' のエレメントではありません。

cast does not match function type – キャスティングが関数タイプとマッチしません

関数の戻りタイプは関数タイプとマッチしないタイプにキャストされます。

cast from pointer to integer of different size – ポインタから異なるサイズの整数へのキャストです

16 ビット幅ではない整数にポインタがキャストされます。

cast increases required alignment of target type – キャストによりターゲットタイプの必要な整列が増加します

-Wcast-align コマンドラインオプションでコンパイルする際、コンパイラはキャストによりターゲットタイプに必要な整列が増加していないことを確認します。例えば、この警告メッセージはポインタが char へのキャストを int としてキャストされると生成されます。char 整列 (バイト整列) が int が求める整列 (ワード整列) より少ないためです。

character constant too long – 文字定数が長すぎます

文字定数は長すぎではありません。

comma at end of enumerator list – 列挙子リストの末尾にカンマがあります

列挙子リストの末尾にカンマは不要です。

comma operator in operand of #if – #if のオペランド内にカンマ演算子があります

#if ディレクティブにカンマ演算子は不要です。

comparing floating point with == or != is unsafe – 浮動小数点を == または != での比較は安全ではありません

浮動小数点の値は非常に正確な実数の近似となります。同等性をテストする代わりに、関係演算子を使用して 2 つの値の範囲がオーバーラップしているかを見ます。

comparison between pointer and integer – ポインタと整数間の比較です

ポインタタイプは整数タイプと比較されました。

comparison between signed and unsigned – 符号付きと符号無しの比較です

比較のオペランドのうち一方が符号付きで、もう一方が符号無しです。符号付きオペランドは符号無しの値として取り扱われ、比較は正しくない可能性があります。

comparison is always n – 比較は常に n です

比較は定数表現のみであるため、コンパイラは比較のランタイム結果を予測できません。結果は常に n です。

comparison is always n due to width of bit-field – ビットフィールドの幅に従い、比較は常に n です

ビットフィールドを含む比較は常に n を評価します。これは、ビットフィールドの幅のためです。

comparison is always false due to limited range of data type – データタイプの制限範囲に従い、比較は常に false です

データタイプの範囲のため実行時の比較は常に false と評価されます。

comparison is always true due to limited range of data type – データタイプの制限範囲に従い、比較は常に true です

データタイプの範囲のため実行時の比較は常に true と評価されます。

comparison of promoted ~unsigned with constant – 昇格 ~unsigned と定数の比較です

比較のオペランドのうち一方が昇格 ~unsigned で、もう一方が定数です。

comparison of promoted ~unsigned with unsigned – 昇格 ~unsigned と符号無し の比較です

比較のオペランドのうち一方が昇格 ~unsigned で、もう一方が符号無しです。

comparison of unsigned expression ≥ 0 is always true – 符号無し表現 ≥ 0 の比較は常に true です

比較表現は符号無しの値とゼロを比較します。符号無しの値は 0 未満にはならないため、実行時の比較は常に true と評価されます。

comparison of unsigned expression < 0 is always false – 符号無し表現 < 0 の比較は常に false です

比較表現は符号無しの値とゼロを比較します。符号無しの値は 0 未満にはならないため、実行時の比較は常に false と評価されます。

comparisons like $X <= Y <= Z$ do not have their mathematical meaning – $X <= Y <= Z$ のような比較に数学的な意味はありません

C 表現は対応する数学的表現と常に同じ意味を持つ必要はありません。特に、C 表現 $X <= Y <= Z$ は数学的表現 $X \leq Y \leq Z$ と同等ではありません。

conflicting types for built-in function 'identifier' – built-in 関数 'identifier' と競合するタイプです

指定された関数は built-in 関数と同じ名前を持っていますが、競合するタイプと宣言されました。

const declaration for 'identifier' follows non-const – 'identifier' の non-const 宣言の後には const と宣言された

指定した識別子は non-const として宣言された後に const として宣言されました。

control reaches end of non-void function – 制御が non-void 関数の末尾に達しました

non-void 関数からは適切な値を返す必要があります。コンパイラは明示的な戻り値なしに non-void 関数の末尾を検出します。したがって、戻り値は予想できない場合があります。

conversion lacks type at end of format – フォーマットの末尾にタイプが欠落しています

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列のフォーマットフィールドに型指定がないことを検出しました。

concatenation of string literals with `__FUNCTION__` is deprecated – 文字列リテラルと `__FUNCTION__` は連結しています

`__FUNCTION__` は `__func__` (ISO 標準 C99 で定義) と同じように取り扱われません。`__func__` は変数で、文字列リテラルではありません。そのため、他の文字列リテラルとは連結しません。

conflicting types for 'identifier' – 'identifier' に矛盾したタイプです

指定した identifier が複数の矛盾する宣言を含んでいます。

D

data definition has no type or storage class – データ定義がタイプまたはストレージクラスを持っていません

データ定義にタイプとストレージクラスがないことが検出されました。

data qualifier ‘qualifier’ ignored – データ修飾子 ‘qualifier’ が無視されました

‘access’、‘shared’、‘overlay’ などのデータ修飾子は MPLAB C30 では使用しませんが、MPLAB C17 および C18 との互換性のために備えています。

declaration of ‘identifier’ has ‘extern’ and is initialized – ‘identifier’ の宣言は ‘extern’ を持ち、初期化されています

Extern は初期化できません。

declaration of ‘identifier’ shadows a parameter – ‘identifier’ の宣言はパラメータをシャドウします

指定した *identifier* 宣言はパラメータをシャドウし、パラメータをアクセス不能にします。

declaration of ‘identifier’ shadows a symbol from the parameter list – ‘identifier’ の宣言はパラメータリスト内の記号をシャドウします

指定した *identifier* 宣言はパラメータリスト内の記号をシャドウし、記号をアクセス不能にします。

declaration of ‘identifier’ shadows global declaration – ‘identifier’ の宣言はグローバル宣言をシャドウします

指定した *identifier* 宣言はグローバル宣言をシャドウし、グローバルをアクセス不能にします。

‘identifier’ declared inline after being called – ‘identifier’ はコールされた後にインラインを宣言しました

指定した関数はコールされた後インラインを宣言しました。

‘identifier’ declared inline after its definition – ‘identifier’ は定義の後インラインを宣言しました

指定した関数は定義された後インラインを宣言しました。

‘identifier’ declared ‘static’ but never defined – ‘identifier’ は ‘static’ を宣言しましたが、定義されていません

指定した関数は static を宣言しましたが、定義されていません。

decrement of read-only member ‘name’ – 読み取り専用メンバー ‘name’ の減分です メンバー ‘name’ が const として宣言され、減分変更できません。

decrement of read-only variable ‘name’ – 読み取り専用変数 ‘name’ の減分です ‘name’ が const として宣言され、減分変更できません。

‘identifier’ defined but not used – ‘identifier’ が定義されましたが、使用されていません

指定した関数は定義されましたが、使用されていません。

deprecated use of label at end of compound statement – 複合ステートメント末尾でラベル使用が重複しています

ラベルはステートメントの末尾に置くべきではありません。ステートメントの前に付けます。

dereferencing ‘void *’ pointer – ‘void*’ ポインタを間接参照しています

‘void*’ ポインタを間接参照するのは不正です。参照する前に、適切なタイプのポインタをキャストしてください。

division by zero – ゼロによる割り算

コンパイル時にゼロによる割り算が検出されました。

duplicate ‘const’ – 重複した ‘const’

‘const’ 修飾子は宣言に 1 度だけ適用します。

duplicate 'restrict' – 重複した 'restrict'

'restrict' 修飾子は宣言に 1 度だけ適用します。

duplicate 'volatile' – 重複した 'volatile'

'volatile' 修飾子は宣言に 1 度だけ適用します。

E

embedded '\0' in format – フォーマットに '\0' が組み込まれています

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列に '\0' (ゼロ) が組み込まれていることを検出しました。これはフォーマット文字列処理の早期終了を引き起こすことがあります。

empty body in an else-statement – else- ステートメント内に空の本文があります

else ステートメントは空です。

empty body in an if-statement – if- ステートメント内に空の本文があります

if ステートメントは空です。

empty declaration – 空の宣言

宣言には宣言する名前が含まれていません。

empty range specified – 空の範囲が指定されました

case 範囲内の値の範囲が空であるため、低位表現の値が上位表現より高く扱われています。case 範囲の構文は `case low ... high:`。

'enum identifier' declared inside parameter list – 'enum identifier' がパラメータリスト内で宣言されました

指定した enum が関数パラメータリスト内で宣言されました。通常のよりよいプログラミング慣習としては、パラメータリスト外で enum を宣言します。パラメータリスト内で定義される場合、完全なタイプとなりません。

enum defined inside parms – パラメータ内で enum が定義されました

enum が関数パラメータリスト内で定義されました。

enumeration value 'identifier' not handled in switch – 列挙型の値 'identifier' は switch 内では取り扱われません

switch ステートメントの制御表現は列挙型タイプですが、すべての列挙型の値は case 表現を持つとは限りません。

enumeration values exceed range of largest integer – 列挙型の値が最大整数の範囲を超えています

列挙型の値は整数として表現されます。列挙型範囲はそのような最大フォーマットを含む MPLAB C30 整数フォーマットで表現不可能なことをコンパイラが検出しました。

excess elements in array initializer – 配列イニシャライザ内のエレメント数を超えています

イニシャライザリスト内のエレメント数が一緒に宣言された配列のエレメントより多くなっています。

excess elements in scalar initializer – スカラーイニシャライザ内のエレメント数を超えています

スカラー変数に必要なイニシャライザは 1 つのみです。

excess elements in struct initializer – 構造イニシャライザ内のエレメント数を超えています

イニシャライザリスト内のエレメント数が一緒に宣言された構造のエレメントより多くなっています。

excess elements in union initializer – union イニシャライザ内のエレメント数を超えています

イニシャライザリスト内のエレメント数が一緒に宣言された union のエレメントより多くなっています。

extra semicolon in struct or union specified – struct または union 内に余分なセミコロンが指定されています

構造体タイプまたは union タイプが余分なセミコロンを含んでいます。

extra tokens at end of #‘directive’ directive – #‘directive’ ディレクティブの末尾に余分なトークンがあります

コンパイラは #‘directive’ ディレクティブを含むソース行で余分なテキストを検出しました。

F

-ffunction-sections may affect debugging on some targets – -ffunction- セクションは一部ターゲットのデバッグに影響を及ぼす可能性があります

-g オプションと -ffunction-sections オプションの両方を指定するとデバッグに問題が発生する可能性があります。

first argument of ‘identifier’ should be ‘int’ – ‘identifier’ の最初の引数は‘int’にする必要があります

指定した識別子の最初の引数宣言はタイプ int である必要があります。

floating constant exceeds range of ‘double’ – 浮動小数点定数が‘double’ の範囲を超えます

浮動小数点定数は‘double’ として表現するには大きすぎるか、小さすぎます（絶対値）。

floating constant exceeds range of ‘float’ – 浮動小数点定数が‘float’ の範囲を超えます

浮動小数点定数は‘float’ として表現するには大きすぎるか、小さすぎます（絶対値）。

floating constant exceeds range of ‘long double’ – 浮動小数点定数が‘long double’ の範囲を超えます

浮動小数点定数は‘long double’ として表現するには大きすぎるか、小さすぎます（絶対値）。

floating point overflow in expression – 表現内で浮動小数点オーバーフローします

浮動小数点定数表現を畳み込む際に、コンパイラは表現がオーバーフローしていることを検出しました。したがって、これは浮動小数点としては表現できません。

‘type1’ format, ‘type2’ arg (arg ‘num’) – ‘type1’ フォーマット、‘type2’ 引数（引数‘num’）

フォーマットはタイプ‘type1’ ですが、渡される引数はタイプ‘type2’ です。問題の引数は、‘num’ 引数です。

format argument is not a pointer (arg n) – フォーマット引数はポインタ (arg n) ではありません

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラは指定した引数 *n* はフォーマット指定が指示しているとおりのポインタではないことを検出しました。

format argument is not a pointer to a pointer (arg n) – フォーマット引数はポインタ (arg n) へのポインタではありません

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラは指定した引数 *n* はフォーマット指定が指示しているとおりのポインタではないことを検出しました。

fprefetch-loop-arrays not supported for this target – fprefetch-loop-arrays はこのターゲットをサポートしていません

メモリプリフェッチオプションはこのターゲットではサポートされていません。

function call has aggregate value – 関数コールが集合数を持っています

関数の戻り値が集合数です。

function declaration isn't a prototype – 関数宣言がプロトタイプではありません

`-Wstrict-prototypes` コマンドラインオプションをコンパイルする際、コンパイラは関数プロトタイプがすべての関数を指定していることを確認します。この場合、前に付くプロトタイプがない関数定義を検出できます。

function declared 'noreturn' has a 'return' statement – 'noreturn' 宣言の関数は 'return' を持っています

関数が `noreturn` 属性と宣言されました。これは関数が戻り値を返さないことを意味しますが、関数は戻りステートメントを含んでいます。よって、この宣言は矛盾しています。

function might be possible candidate for attribute 'noreturn' – 関数は属性 'noreturn' 候補である可能性があります

コンパイラは関数が戻り値を返さないことを検出しました。関数が `noreturn` 属性と宣言されていれば、コンパイラはよりよいコードを作成できます。

function returns address of local variable – 関数はローカル変数のアドレスを返します

関数はローカル変数のアドレスを返すべきではありません。関数がこの値を返すと、ローカル変数が再割り当てされるためです。

function returns an aggregate – 関数は集合数を返します

関数の戻り値が集合数です。

**function 'name' redeclared as inline
previous declaration of function 'name' with attribute noinline –
関数 'name' がインラインとして再宣言されました
以前の関数 'name' の宣言では、属性は noinline でした**

関数 `'name'` は2度目の宣言ではキーワード `'inline'` で宣言され、インライン化が行われます。

**function 'name' redeclared with attribute noinline
previous declaration of function 'name' was inline –
関数 'name' が属性 noinline で宣言され
以前の関数 'name' の宣言では、inline でした**

関数 `'name'` は2度目の宣言では `noinline` 属性で宣言され、インライン化できません。

function 'identifier' was previously declared within a block – 関数 'identifier' は以前ブロック内で宣言されました

関数は以前、ブロック内で明示的に宣言されましたが、現在の行で暗示的宣言も持っています。

G

GCC does not yet properly implement '['*'] array declarators – GCC は '['*'] 配列宣言を適切に実行できません

可変長の配列は現在、コンパイラではサポートされていません。

H

hex escape sequence out of range – hex エスケープシーケンスが範囲外です

hex シーケンスは 16 進法で 100 未満 (10 進法で 256) にする必要があります。

I

ignoring asm-specifier for non-static local variable '*identifier*' – 非静的ローカル変数 '*identifier*' の asm 規則子を無視します

asm 規則子は普通の非レジスタローカル変数と使用すると無視されます。

ignoring invalid multibyte character – 無効なマルチバイト文字を無視します

マルチバイト文字を構文解析している際に、コンパイラはそれを無効と判断しました。無効な文字は無視されます。

ignoring option '*option*' due to invalid debug level specification – オプション '*option*' はデバッグレベル仕様が無効なため無視します

デバッグオプションが有効でないデバッグレベルとともに使用されています。

ignoring #pragma *identifier* – #pragma 識別子を無視します

指定した pragma は MPLAB C30 コンパイラではサポートされていないため、無視されます。

imaginary constants are a GCC extension – 虚定数は GCC 拡張です

ISO C は虚定数を許可しません。

implicit declaration of function '*identifier*' – 関数 '*identifier*' の暗黙の宣言です

指定した関数は以前の明示的宣言 (定義または関数プロトタイプ) を含んでいないため、コンパイラがその戻りタイプおよびパラメータを予測しました。

increment of read-only member '*name*' – 読み取り専用メンバー '*name*' の増分です

メンバー '*name*' が const として宣言され、増分変更できません。

increment of read-only variable '*name*' – 読み取り専用変数 '*name*' の増分です

'*name*' が const として宣言され、増分変更できません。

initialization of a flexible array member – フレキシブル配列メンバーの初期化です

フレキシブル配列メンバーは静的にはなく、動的に割り当てられるようになっています。

'*identifier*' initialized and declared '*extern*' – '*identifier*' が初期化され '*extern*' と宣言された

Extern は初期化すべきではありません。

initializer element is not constant – イニシャライザエレメントが定数ではありません

イニシャライザエレメントは定数にする必要があります。

inline function '*name*' given attribute *noinline* – インライン関数 '*name*' に属性 *noinline* が付与されています

関数 '*name*' はインラインとして宣言されましたが、*noinline* 属性が関数のインライン化を妨げています。

inlining failed in call to 'identifier' called from here – ここからコールされた 'identifier' へのコールをインライン化できません

コンパイラは指定した関数へのコールをインライン化できませんでした。

integer constant is so large that it is unsigned – 整数定数が大きすぎるため unsigned と見なされます

整数定数は明示的 unsigned 修飾子無しでソースコード中に表示されますが、sign int としてこの数字を表すことはできません。したがって、コンパイラは自動的に unsigned int としてこれを扱います。

integer constant is too large for 'type' type – 整数数が 'type' タイプには大きすぎます

整数数は符号無し long int では $2^{32} - 1$ 、long long int では $2^{63} - 1$ もしくは符号無し long long int では $2^{64} - 1$ を超えないようにしてください。

integer overflow in expression – 整数が表現内でオーバーフローしています

整数表現を畳み込む際に、コンパイラは表現がオーバーフローしていることを検出しました。したがって、これは int としては表現できません。

invalid application of 'sizeof' to a function type – 関数タイプへの 'sizeof' 適用が無効です

sizeof 演算子を関数タイプに適用するのはお奨めできません。

invalid application of 'sizeof' to a void type – void タイプへの 'sizeof' 適用が無効です

sizeof 演算子は void タイプに適用すべきではありません。

invalid digit 'digit' in octal constant – 無効な数字 'digit' が 8 進定数内にあります

すべての数字は使用されている基数内にする必要があります。例えば、8 進法で使用できるのは 0 から 7 の数字のみです。

invalid second arg to __builtin_prefetch; using zero – __builtin_prefetch の第二引数に無効な値 が使用されています ; 0 にします

第二引数は 0 または 1 にする必要があります。

invalid storage class for function 'name' – 関数 'name' の無効なストレージクラス

'auto' ストレージクラスは上位レベルで関数定義として使用するべきではありません。関数が上位レベルで定義されていない場合は、'static' ストレージクラスは使用するべきではありません。

invalid third arg to __builtin_prefetch; using zero – __builtin_prefetch への無効な第三引数です ; 0 にします

第三引数は 0、1、2 または 3 にする必要があります。

'identifier' is an unrecognized format function type – 'identifier' は認識されないフォーマット関数タイプです

フォーマット属性として指定した identifier は printf, scanf, strftime 用のフォーマットタイプとして認識できません。

'identifier' is narrower than values of its type – 'identifier' の幅がそのタイプの値より狭くなっています

構造体のビットフィールドメンバーは列挙型のタイプを持っていますが、フィールドの幅がすべての列挙値を表現するのに不十分です。

'storage class' is not at beginning of declaration – 'storage class' が宣言の最初にありません

指定したストレージクラスが宣言の最初にありません。宣言の最初にはストレージクラスが来る必要があります。

ISO C does not allow extra ';' outside of a function – ISO C は関数外の余分な ';' を許可しません

余分な ';' が関数の外で見つかりました。ISO C では許可されません。

ISO C does not support '++' and '--' on complex types – ISO C は複素数型の '++' および '--' をサポートしません

増分演算子と減分演算子は ISO C の複素数型ではサポートされていません。

ISO C does not support '~' for complex conjugation – ISO C は複合接合の '~' をサポートしません

ビットの否定演算子は ISO C の複合接合では使用できません。

ISO C does not support complex integer types – ISO C は複合整数タイプをサポートしません

`__complex__ short int` などの複合整数タイプは ISO ではサポートされていません。

ISO C does not support plain 'complex' meaning 'double complex' – ISO C は 'double complex' を意味する単純 'complex' はサポートしません

他の修飾子を持たない `__complex__` は 'complex double' と見なされ、ISO C ではサポートされていません

ISO C does not support the 'char' 'kind of format' format – ISO C は 'char' 'kind of format' フォーマットをサポートしません

ISO C は指定した 'kind of format' の指定文字 'char' をサポートしません。

ISO C doesn't support unnamed structs/unions – ISO C は名前のない struct/union をサポートしません

ISO C ではすべての構造体および/またはユニオンに名前をつける必要があります。

ISO C forbids an empty source file – ISO C は空のソースファイルを禁じています
ファイルに関数またはデータがありません。これは ISO C では許可されません。

ISO C forbids empty initializer braces – ISO C は空のイニシャライザ中括弧を禁じています

ISO C は中括弧内にイニシャライザの値を入れるよう要求します。

ISO C forbids nested functions – ISO C はネストされた関数を禁じています

関数が他の関数内で定義されています。

ISO C forbids omitting the middle term of a ?: expression – ISO C は ?: 表現の中間項の省略を禁じています

条件式は '?' と ':' の間に中間式を要求します。

ISO C forbids qualified void function return type – ISO C は修飾された void 関数戻りタイプを禁じています

void 関数戻りタイプには修飾子を使用しないでください。

ISO C forbids range expressions in switch statements – ISO C は switch ステートメント内の範囲表現を禁じています

ISO C では、単一ケースラベル内に連続した値の範囲指定を禁じています。

ISO C forbids subscripting 'register' array – ISO C は 'register' 配列をサブスクリプトすることを禁じています

ISO C では、'register' 配列のサブスクリプトを禁じています。

ISO C forbids taking the address of a label – ISO C はラベルのアドレスを取得することを禁じています

ISO C では、ラベルのアドレスを取得することは禁じています。

ISO C forbids zero-size array ‘name’ – ISO C はゼロサイズの配列 ‘name’ を禁じています

‘name’ の配列サイズは 0 より大きいである必要があります。

ISO C restricts enumerator values to range of ‘int’ – ISO C は ‘int’ の範囲に列挙型の値を制限しています

列挙型の値の範囲は int タイプの範囲を超えてはなりません。

ISO C89 forbids compound literals – ISO C89 は複合リテラルを禁じています

ISO C89 では、複合リテラルは無効です。

ISO C89 forbids mixed declarations and code – ISO C89 は宣言とコードの混合を禁じています

コードが書き込まれる前に宣言をする必要があります。宣言をコードに含めないでください。

ISO C90 does not support ‘[*]’ array declarators – ISO C90 は ‘[*]’ 配列宣言をサポートしていません

可変長の配列は ISO C90 ではサポートされていません。

ISO C90 does not support complex types – ISO C90 は複合タイプをサポートしません

`__complex__ float x` などの複合タイプは ISO C90 ではサポートされていません。

ISO C90 does not support flexible array members – ISO C90 はフレキシブル配列メンバーをサポートしません

フレキシブル配列メンバーは C99 の新しい機能です。ISO C90 ではサポートされていません。

ISO C90 does not support ‘long long’ – ISO C90 は ‘long long’ はサポートしません

`long long` タイプは ISO C90 ではサポートされていません。

ISO C90 does not support ‘static’ or type qualifiers in parameter array declarators – ISO C90 はパラメータ配列宣言子内の ‘static’ またはタイプ修飾子をサポートしません

配列を関数のパラメータとして使用する際、ISO C90 は配列宣言子が `static` またはタイプ修飾子を使用するのを許可しません。

ISO C90 does not support the ‘char’ ‘function’ format – ISO C90 は ‘char’ ‘function’ フォーマットはサポートしません

ISO C は指定した関数フォーマットとしての指定文字 ‘char’ はサポートしません。

ISO C90 does not support the ‘modifier’ ‘function’ length modifier – ISO C90 は ‘modifier’ ‘function’ 長さ変更子をサポートしません

指定された修飾子はその関数の長さ修飾子としてはサポートされていません。

ISO C90 forbids variable-size array ‘name’ – ISO C90 は可変長の配列 ‘name’ を禁じています

ISO C90 では、配列内のエレメント数は整数表現で指定する必要があります。

L

label ‘identifier’ defined but not used – ラベル ‘identifier’ が定義されましたが、使用されていません

指定したラベルは定義されましたが参照されていません。

large integer implicitly truncated to unsigned type – サイズの大きな整数が暗示的に unsigned タイプに短縮されました

整定数値は明示的 unsigned 修飾子無しでソースコード中に表示されますが、signed int としてこの数字を表すことはできません。したがって、コンパイラは自動的に unsigned int としてこれを扱います。

left-hand operand of comma expression has no effect – カンマ表現の左側のオペランドは効果がありません

比較のオペランドのうち一方が昇格 ~unsigned で、もう一方が unsigned です。

left shift count >= width of type – 左シフトカウント >= タイプの幅

シフトのカウントはシフトされたタイプ内のビット数を超えてはいけません。そうでないと、シフトは無意味になり、結果が不確定です。

left shift count is negative – 左シフトのカウントが負の数です

シフトのカウントは正の数でなくてはなりません。左シフトのカウントが負の数でも、右のシフトを意味しませんので、実行しても意味がありません。

library function 'identifier' declared as non-function – ライブラリ関数 'identifier' は non-function として宣言されました

指定された関数はライブラリ関数と同じ名前を持っていますが、関数以外のものとして宣言されました。

line number out of range – 行数が範囲外です

#line ディレクティブの行数の制限は、C89 で 32767、C99 で 2147483647 です。

'identifier' locally external but globally static – 'identifier' はローカルでは外部ですが、グローバルでは静的です

指定した 'identifier' はローカルでは外部ですが、グローバルでは静的です。これは疑わしい状態といえます。

location qualifier 'qualifier' ignored – ロケーション修飾子 'qualifier' が無視されました

'grp'、'sfr' などのロケーション修飾子は MPLAB C30 では使用されませんが、MPLAB C17 および C18 との互換性として備えてあります。

'long' switch expression not converted to 'int' in ISO C – ISO C では、'long' スイッチ表現は 'int' に変換されません

ISO C は 'long' スイッチ表現を 'int' に変換しません。

M

'main' is usually a function – 'main' は通常、関数です

識別子 main は通常、アプリケーションのメインエントリーポイントの名前に使用されます。コンパイラはこの識別子が変数の名前など他の方法で使用されていることを検出しました。

'operation' makes integer from pointer without a cast – 'operation' はキャスト無しでポインタから整数を作成します

ポインタは暗示的に整数に変換されます。

'operation' makes pointer from integer without a cast – 'operation' は整数からキャスト無しでポインタを作成します

整数は暗示的にポインタに変換されます。

malformed '#pragma pack-ignored' – 不正な形式の '#pragma pack-ignored' です pack pragma の構文が正しくありません。

malformed '#pragma pack(pop[,id])-ignored' – 不正な形式の '#pragma pack(pop[,id])-ignored' です

pack pragma の構文が正しくありません。

malformed '#pragma pack(push[,id],<n>-ignored' – 不正な形式の '#pragma pack(push[,id],<n>-ignored' です

pack pragma の構文が正しくありません。

malformed '#pragma weak-ignored' – 不正な形式の '#pragma weak-ignored' です

weak pragma の構文が正しくありません。

'identifier' might be used uninitialized in this function – この関数で初期化されていない 'identifier' が使用されている可能性があります

コンパイラは初期化する前に指定した識別子を使用した可能性のある関数を介した制御パスを検出しました。

missing braces around initializer – イニシャライザの前後に中括弧がありません

イニシャライザの前後に必要な中括弧がありません。

missing initializer – イニシャライザが見つかりません

イニシャライザが見つかりません。

modification by 'asm' of read-only variable 'identifier' – 'asm' を使用した読み取り専用変数 'identifier' の変更です

const 変数が 'asm' ステートメント内代入の左端にあります。

multi-character character constant – 複数文字 character 定数です

文字定数が 2 つ以上の文字を含んでいます。

N

negative integer implicitly converted to unsigned type – 負の整数が暗示的に unsigned タイプに変換されました

負の整数値はソースコード中に表示されますが、signed int としてこの数字を表すことはできません。したがって、コンパイラは自動的に unsigned int としてこれを行います。

nested extern declaration of 'identifier' – 'identifier' のネストされた extern 宣言です

指定した *identifier* のネストされた extern 定義があります。

no newline at end of file – ファイルの最後が復帰改行ではありません

ソースファイルの最後の行が復帰改行文字で終了していません。

no previous declaration for 'identifier' – 'identifier' の事前の宣言がありません

-Wmissing-declarations コマンドラインオプションでコンパイルする際、コンパイラは関数が定義の前に宣言されていることを確認します。このケースでは、関数宣言なしに関数定義が検出されました。

no previous prototype for 'identifier' – 'identifier' の事前のプロトタイプがありません

-Wmissing-prototypes コマンドラインオプションでコンパイルする際、コンパイラは関数プロトタイプがすべての関数に指定されていることを確認します。この場合、事前の関数プロトタイプなしに関数定義が検出されました。

no semicolon at end of struct or union – struct または union の末尾にセミコロンがありません

構造体またはユニオン宣言の末尾にセミコロンがありません。

non-ISO-standard escape sequence, 'seq' – ISO 標準ではないエスケープシーケンス 'seq' です

'seq' が '\e' または '\E' で、ISO 標準の拡張機能です。シーケンスは文字列または文字定数で使用され、ASCII 文字 <ESC> を表します。

non-static declaration for 'identifier' follows static – 静的宣言の後に 'identifier' の静的ではない宣言が付いています

指定した識別子は静的として宣言された後に非静的として宣言されました。

'noreturn' function does return – 'noreturn' 関数が戻り値を返します

noreturn 属性で宣言された関数が戻り値を返します。これは矛盾しています。

'noreturn' function returns non-void value – 'noreturn' 関数が非 void 値を返します

noreturn 属性で宣言された関数が非 void 値を返します。これは矛盾しています。

null format string – null フォーマット文字列

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列がないことを検出しました。

O

octal escape sequence out of range – 8 進数エスケープシーケンスが範囲外です

8 進数シーケンスでは 400 未満 (10 進法で 256) にする必要があります。

output constraint 'constraint' for operand *n* is not at the beginning – オペランド *n* の出力制約 'constraint' が文頭にありません

拡張 asm の出力制約は文頭に来る必要があります。

overflow in constant expression – 定数表現でオーバーフローが発生しています

定数表現の長さがその型の相当値の範囲を超えています。

overflow in implicit constant conversion – 暗示的定数変換でオーバーフローが発生しています

暗示的定数変換結果の数字は、signed int としてこの数字を表すことはできません。したがって、コンパイラは自動的に unsigned int としてこれを扱います。

P

parameter has incomplete type – パラメータが不完全なタイプを含んでいます

関数パラメータが不完全なタイプを含んでいます。

parameter names (without types) in function declaration – 関数宣言内のパラメータ名 (タイプ無し) です

関数宣言はタイプではなくパラメータ名をリストしています。

parameter points to incomplete type – パラメータは不完全なタイプを指しています

関数パラメータは不完全なタイプを指しています。

parameter 'identifier' points to incomplete type – パラメータ 'identifier' は不完全なタイプを指しています

指定した関数パラメータは不完全なタイプを指しています。

passing arg 'number' of 'name' as complex rather than floating due to prototype – プロトタイプに基づき、浮動値ではなく 'name' の引数 'number' を複合値として渡しました

プロトタイプは引数 'number' を複合体として宣言しましたが、浮動値が使用されたため、コンパイラはこれを複合値に変換してプロトタイプと適合するようにしました。

passing arg 'number' of 'name' as complex rather than integer due to prototype
– プロトタイプに基づき、整数ではなく 'name' の引数 'number' を複合値として渡しました

プロトタイプは引数 'number' を複合値として宣言しましたが、整数値が使用されたため、コンパイラはこれを複合値に変換してプロトタイプと適合するようにしました。

passing arg 'number' of 'name' as floating rather than complex due to prototype
– プロトタイプに基づき、複合値ではなく 'name' の引数 'number' を浮動値として渡しました

プロトタイプは引数 'number' を浮動値として宣言しましたが、複合値が使用されたため、コンパイラはこれを浮動値に変換してプロトタイプと適合するようにしました。

passing arg 'number' of 'name' as 'float' rather than 'double' due to prototype
– プロトタイプに基づき、'double' ではなく 'name' の引数 'number' を 'float' として渡しました

プロトタイプは引数 'number' を浮動値として宣言しましたが、double が使用されたため、コンパイラはこれを浮動値に変換してプロトタイプと適合するようにしました。

passing arg 'number' of 'name' as floating rather than integer due to prototype
– プロトタイプに基づき、整数ではなく 'name' の引数 'number' を浮動値として渡しました

プロトタイプは引数 'number' を浮動値として宣言しましたが、整数が使用されたため、コンパイラはこれを浮動値に変換してプロトタイプと適合するようにしました。

passing arg 'number' of 'name' as integer rather than complex due to prototype
– プロトタイプに基づき、複合値ではなく 'name' の引数 'number' を整数として渡しました

プロトタイプは引数 'number' を整数として宣言しましたが、複合値が使用されたため、コンパイラはこれを整数に変換してプロトタイプと適合するようにしました。

passing arg 'number' of 'name' as integer rather than floating due to prototype
– プロトタイプに基づき、浮動値ではなく 'name' の引数 'number' を整数として渡しました

プロトタイプは引数 'number' を整数として宣言しましたが、浮動値が使用されたため、コンパイラはこれを整数に変換してプロトタイプと適合するようにしました。

pointer of type 'void *' used in arithmetic – タイプ 'void *' のポインタが演算に使用されました

タイプ 'void' のポインタはサイズがないので、演算には使用しないでください。

pointer to a function used in arithmetic – 関数へのポインタが演算に使用されました

関数へのポインタは演算に使用できません。

previous declaration of 'identifier' – 'identifier' の以前の宣言です

この警告メッセージは他の警告メッセージに関連して表示されます。以前のメッセージは疑わしいコードの場所を示します。このメッセージは *identifier* の最初の宣言または定義を示します。

previous implicit declaration of 'identifier' – 'identifier' の以前の暗示的宣言です

この警告メッセージは、“type mismatch with previous implicit declaration”に関連して表示されます。このメッセージは、明示的宣言と矛盾する識別子の暗示的宣言の場所を示します。

R

"name" re-asserted – "name" が再表明されました

"name" の解が重複しています。

“name” redefined – “name” が再定義されました

“name” は以前定義されていましたが、再度定義されました。

redefinition of ‘identifier’ – ‘identifier’ の再定義

指定した identifier が複数の互換性のない宣言を含んでいます。

redundant redeclaration of ‘identifier’ in same scope – 同じ範囲内での ‘identifier’ の冗長な再宣言

指定した識別子は同じ範囲内で再宣言されました。これは冗長です。

register used for two global register variables – 2 つのグローバルレジスタ変数にレジスタが使用されています

2 つのグローバルレジスタ変数が同じレジスタを使用するように定義されました。

repeated ‘flag’ flag in format – フォーマット内の繰り返しの ‘flag’ フラグ

strftime へのコールの引数リストを確認する際、コンパイラはフォーマット文字列内に繰り返されているフラグがあることを検出しました。

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラはフラグ {+,#,0,-} の 1 つがフォーマット文字列内で繰り返されていることを検出しました。

return-type defaults to ‘int’ – 戻りタイプは ‘int’ のデフォルトです

明示的関数戻りタイプ宣言がないため、コンパイラは関数の戻り値を int と想定しました。

return type of ‘name’ is not ‘int’ – ‘name’ の戻りタイプが ‘int’ ではありません

コンパイラは ‘name’ の戻りタイプとして ‘int’ を期待しています。

‘return’ with a value, in function returning void – void 関数が値を返しました

関数は void として宣言されましたが、値を返しました。

‘return’ with no value, in function returning non-void – 非 void 戻り値を返す関数に戻り値はありません

非 void 値を返すことを宣言した関数が、値のない戻りステートメントを含んでいます。これは矛盾しています。

right shift count >= width of type – 右シフトカウント >= タイプの幅

シフトのカウントはシフトされたタイプ内のビット数を超えてはいけません。シフトは無意味になり、結果が不確定です。

right shift count is negative – 右シフトのカウントが負の数です

シフトのカウントは正の数でなくてはなりません。右シフトのカウントが負の数でも、左のシフトを意味しませんので、実行する意义がありません。

S

second argument of ‘identifier’ should be ‘char **’ – ‘identifier’ の 2 番目の引数は ‘char **’ にする必要があります

指定した識別子の 2 番目の引数はタイプ ‘char **’ である必要があります。

second parameter of ‘va_start’ not last named argument – ‘va_start’ の 2 番目のパラメータは最後の引数ではありません

‘va_start’ の 2 番目のパラメータは最後の引数である必要があります。

shadowing built-in function ‘identifier’ – built-in 関数 ‘identifier’ がシャドウされます

指定した関数は built-in 関数と同じ名前を持っているので、built-in 関数をシャドウします。

shadowing library function 'identifier' – ライブラリ関数 'identifier' がシャドーされます

指定した関数はライブラリ関数と同じ名前を持っているので、ライブラリ関数をシャドウします。

shift count >= width of type – シフトカウント >= タイプの幅

シフトのカウントはシフトされたタイプ内のビット数を超えてはいけません。そうでないと、シフトは無意味になり、結果が不確定です。

shift count is negative – シフトのカウントが負の数です

シフトのカウントは正の数でなくてはなりません。負のカウント数の左シフトは右シフトを意味しません。負のカウント数の右シフトも左シフトを意味しません。これらは無意味です。

size of 'name' is larger than *n* bytes – 'name' のサイズが *n* バイトより大きいです

-Wlarger-than-len を使用する場合、'name' のサイズが定義された *len* バイトより大きい場合に上記の警告が生成されます。

size of 'identifier' is *n* bytes – 'identifier' のサイズは *n* バイトです

指定した識別子のサイズ (*n* バイト) が -Wlarger-than-len コマンドラインオプションで指定したサイズより大きいです。

size of return value of 'name' is larger than *n* bytes – 'name' の戻り値のサイズが *n* バイトより大きいです

-Wlarger-than-len を使用する場合、'name' の戻り値のサイズが定義された *len* バイトより大きい場合に上記の警告が生成されます。

size of return value of 'identifier' is *n* bytes – 'identifier' の戻り値のサイズは *n* バイトです

指定した識別子の戻り値のサイズは *n* バイトで、-Wlarger-than-len コマンドラインオプションで指定したサイズより大きいです。

spurious trailing '%' in format – フォーマットに偽の '%' があります

printf、scanf などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列内に偽の '%' 文字を検出しました。

statement with no effect- 効果のないステートメントです

効果のないステートメントです。

static declaration for 'identifier' follows non-static – 'identifier' の非静的宣言の後には静的宣言が付いています

指定した識別子は非静的として宣言された後に静的として宣言されました。

string length '*n*' is greater than the length '*n*' ISO C*n* compilers are required to support – 文字列長 '*n*' は ISO C*n* コンパイラでサポートする '*n*' より長いです

ISO C89 の最長の文字列の長さは 509 です。ISO C99 の文字列の最長は 4095 です。

'struct identifier' declared inside parameter list – 'struct identifier' がパラメータリスト内で宣言されました

指定した struct が関数パラメータリスト内で宣言されました。通常よりもよいプログラミング慣習としては、パラメータリスト外で struct を宣言します。パラメータリスト内で定義される場合完全なタイプとなりません。

struct has no members – struct はメンバーを持っていません

構造体は空です。メンバーを持っていません。

structure defined inside parms – パラメータ内で構造体が定義されました

union が関数パラメータリスト内で定義されました。

style of line directive is a GCC extension – line ディレクティブのスタイルは GCC 拡張です

伝統的 C では、フォーマット '#line linenum' を使用します。

subscript has type 'char' – 添え字はタイプ 'char' を持っています

配列添え字はタイプ 'char' を持っています。

suggest explicit braces to avoid ambiguous 'else' – 曖昧な 'else' を回避するため、明示的中括弧を推奨します

ネストされた if ステートメントは曖昧な else 項を持っています。曖昧さをなくするため、中括弧の使用をお奨めします。

suggest hiding #directive from traditional C with an indented # – # を字下げした #directive を伝統的 C から隠すようお奨めします

指定したディレクティブは伝統的 C ではなく、# を字下げした場合「隠されます」。ディレクティブは、その # がカラム 1 にない限り無視されます。

suggest not using #elif in traditional C – 伝統的 C で #elif を使用しないようお奨めします

#elif は伝統的 K&R C では使用しないでください。

suggest parentheses around assignment used as truth value – 真 (truth) の値として使用される代入文の前後に括弧を付けるようお奨めします

代入文が真 (truth) の値として使用される場合、ソースプログラムのリーダーにその意図を明確にするため、括弧で囲む必要があります。

suggest parentheses around + or - inside shift – シフト内の + または - の前後に括弧を付けるようお奨めします

suggest parentheses around && within || – || 内の && の前後に括弧を付けるようお奨めします

suggest parentheses around arithmetic in operand of | – | のオペランド内の演算の前後に括弧を付けるようお奨めします

suggest parentheses around comparison in operand of | – | のオペランド内の比較の前後に括弧を付けるようお奨めします

suggest parentheses around arithmetic in operand of ^ – ^ のオペランド内の演算の前後に括弧を付けるようお奨めします

suggest parentheses around comparison in operand of ^ – ^ のオペランド内の比較の前後に括弧を付けるようお奨めします

suggest parentheses around + or - in operand of & – & のオペランド内の + または - の前後に括弧を付けるようお奨めします

suggest parentheses around comparison in operand of & – & のオペランド内の比較の前後に括弧を付けるようお奨めします

演算子の優先順は C では適切に定義されていますが、明示的な括弧がなく表現の読み取りが順位ルールのみによって行なわれる場合、表現内のオペランドの優先順序を理解するのに時間がかかります。例えば、シフト内のオペレータ '+' または '-' の使用です。括弧を使用してプログラムの意図を明確に表現すれば、リーダーは不要な手間を省くことができます。プログラマやコンパイラにとっては明白な表現であっても括弧を使用するようお奨めします。

T

'identifier' takes only zero or two arguments – 'identifier' はゼロまたは 2 つの引数のみ取ります

ゼロまたは 2 つの引数のみ期待されています。

the meaning of '\a' is different in traditional C – 伝統的 C では '\a' の意味は異なります

-wtraditional オプションが使用されると、エスケープシーケンス '\a' はメタシーケンスとして認識されません。値は 'a' のみです。非伝統的なコンパイルでは、'\a' は ASCII BEL 文字を意味します。

the meaning of '\x' is different in traditional C – 伝統的 C では '\x' の意味は異なります

-wtraditional オプションが使用されると、エスケープシーケンス '\x' はメタシーケンスとして認識されません。値は 'x' のみです。非伝統的なコンパイルでは、'\x' は 16 進数エスケープシーケンスを示します。

third argument of 'identifier' should probably be 'char **' – 'identifier' の 3 番目の引数は 'char **' である可能性があります

指定した識別子の 3 番目の引数はタイプ 'char **' である必要があります。

this function may return with or without a value – この関数は値を返す時と返さない時があります

non-void 関数からのすべての exit パスは適切な値を返す必要があります。コンパイラが non-void 関数末端から明示的な戻り値がある時とない時があることを検出しました。したがって、戻り値は予想できません。

this target machine does not have delayed branches – このターゲットマシンは遅延分岐を持っていません

-fdelayed-branch オプションがサポートされていません。

too few arguments for format – フォーマットの引数が少なすぎます

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラは実際の引数の数がフォーマット文字列に必要な数より少ないことを検出しました。

too many arguments for format – フォーマットの引数が多すぎます

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラは実際の引数の数がフォーマット文字列に必要な数より多いことを検出しました。

traditional C ignores #'directive' with the # indented – 伝統的 C は字下げした # が付く #'directive' を無視します

伝統的に、ディレクティブはその # がカラム 1 内にない限り無視されます。

traditional C rejects initialization of unions – 伝統的 C は union の初期化を拒否します

伝統的 C では、union は初期化できません。

traditional C rejects the 'ul' suffix – 伝統的 C は接尾辞 'ul' を拒否します

接尾辞 'u' は伝統的 C では有効ではありません。

traditional C rejects the unary plus operator – 伝統的 C は単項+演算子を拒否します

単項+演算子は伝統的 C では有効ではありません。

trigraph ??char converted to char – 3 重文字 ??char は char に変換されます

3 文字の連続、3 重文字はキーボードにない記号を表現するのに使用します。3 重文字シーケンスは以下のように変換されます。

??(=[??)=]	??<={	??>=}	??=#	??/=\\	??='^	??!=	??-=~
-------	-------	-------	-------	------	--------	-------	------	-------

trigraph ??char ignored – 3 重文字 ??char は無視されました

3 重文字シーケンスが無視されました。char は、(、)、<、>、=、\、!、または - になります。

type defaults to 'int' in declaration of 'identifier' – 'identifier' の宣言内ではタイプのデフォルトは 'int' になります

指定した *identifier* の明示的タイプ宣言がないため、コンパイラはタイプを int と想定しました。

type mismatch with previous external decl of 'identifier' – タイプが以前の 'identifier' の外部宣言とマッチしません

指定した識別子のタイプが以前の宣言とマッチしません。

type mismatch with previous implicit declaration – タイプが以前の暗示的宣言とマッチしません

明示的宣言が以前の暗示的宣言と矛盾します。

type of 'identifier' defaults to 'int' – 'identifier' のタイプのデフォルトは 'int' です
明示的タイプ宣言がないため、コンパイラは識別子のタイプを int と想定しました。

type qualifiers ignored on function return type – 関数戻りタイプの修飾子は無視されます

関数戻りタイプに使用されるタイプ修飾子は無視されます。

U

undefining 'defined' – undefined の 'defined' です

'defined' はマクロ名として使用できないので、undefined できません。

undefining 'name' – #undef の 'name' です

#undef ディレクティブが以前定義されたマクロ名 'name' に使用されました。

union cannot be made transparent – union は透過できません

transparent_union 属性が union に適用されましたが、指定した変数が属性の要件を満たしていません。

'union identifier' declared inside parameter list – 'union identifier' がパラメータリスト内で宣言されました

指定した union が関数パラメータリスト内で宣言されました。通常よりよいプログラミング慣習としては、パラメータリスト外で union を宣言します。パラメータリスト内で定義される場合、完全なタイプとはなりません。

union defined inside parms – パラメータ内でユニオンが定義されました

ユニオンが関数パラメータリスト内で定義されました。

union has no members – ユニオンはメンバーを持っていません

ユニオンは空であり、メンバーを持っていません。

unknown conversion type character 'character' in format – フォーマット内に未知の変換タイプ文字 'character' があります

printf、scanf などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列内の変換文字の 1 つが無効（認識不可能）であることを検出しました。

unknown conversion type character 0xnumber in format – フォーマット内に未知の変換タイプ文字 0xnumber があります

printf、scanf などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列内の変換文字の 1 つが無効（認識不可能）であることを検出しました。

unknown escape sequence 'sequence' – 未知のエスケープシーケンス 'sequence' です

'sequence' は有効なエスケープコードではありません。エスケープコードは '\' で始まり、n、t、b、r、f、b、\、'、"、a、?、のいずれか1つの文字を使用するか、8進または16進の数的シーケンスである必要があります。8進数では、数的シーケンスは8進法で400未満にする必要があります。16進数では、数的シーケンスは'x'で始まり16進法の100未満にする必要があります。

unnamed struct/union that defines no instances – インスタンスを定義しない名前無しの struct/union

struct/union は空であり、名前を持っていません。

unreachable code at beginning of identifier – identifier の先頭に到達不可能なコード

指定した関数の先頭に到達不可能なコードがあります。

unrecognized gcc debugging option: char – 認識できない gcc デバッグオプション: char

-dletters デバッグオプションでは、'char' は有効な文字ではありません。

unused parameter 'identifier' – 使用されていないパラメータ 'identifier'

指定した関数パラメータはその関数内では使用されていません。

unused variable 'name' – 使用されていない変数 'name'

指定した変数は定義されましたが使用されていません。

use of '*' and 'flag' together in format – フォーマット内で '*' と 'flag' が一緒に使用されています

printf、scanf などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列内で '*' と 'flag' の両方が使用されていることを検出しました。

use of C99 long long integer constants – C99 long long 整数が使用されていません

ISO C89 では、整数を long long として宣言することは許可されません。

use of 'length' length modifier with 'type' type character – 'length' 長さ修飾子が 'type' タイプと一緒に使用されています

printf、scanf などへのコールの引数リストを確認する際、コンパイラは指定したタイプにおいて、指定した長さは正しくないことを検出しました。

'name' used but never defined – 'name' は使用されていますが、定義されていません

指定した関数は使用されましたが、定義されていません。

'name' used with 'spec' 'function' format – 'name' は 'spec' 'function' フォーマットで使用されています

指定した関数のフォーマットの中では変換仕様 'spec' と一緒に使用される 'name' は有効ではありません。

useless keyword or type name in empty declaration – 空の宣言の中には無用キーワードまたはタイプ名が含まれています

空の宣言には無用キーワードまたはタイプ名が含まれています。

V

__VA_ARGS__ can only appear in the expansion of a C99 variadic macro –

__VA_ARGS__ は C99 variadic マクロの拡張でのみ使用されます

定義済みマクロ __VA_ARGS__ は省略記号を使用したマクロ定義の代入部分で使用します。

value computed is not used – 計算された値は使用されていません

計算された値は使用されていません。

variable 'name' declared 'inline' – 変数 'name' に 'inline' を宣言しました

キーワード 'inline' は関数でのみ使用すべきです。

variable '%s' might be clobbered by 'longjmp' or 'vfork' – 変数 '%s' は 'longjmp' または 'vfork' によって変更される可能性があります

非揮発性自動変数は longjmp へのコールにより変更される可能性があります。この警告はコンパイルの最適化時にのみ生成されます。

volatile register variables don't work as you might wish – 揮発性レジスタ変数が期待通りに動作しません

変数を引数として渡す場合、引数転送用レジスタ (w0-w7 でない場合) を指定したにも関わらず、変数は別のレジスタ (w0-w7) に移行される可能性があります。または、コンパイラは指定したレジスタに適切ではない命令を生成する場合、その値を一時的に別の場所に移動することがあります。これらの問題は指定したレジスタが非同期的 (つまり、ISR を介して) に変更された場合にのみ発生します。

W

-Wformat-extra-args ignored without -Wformat – -Wformat がいないため

-Wformat-extra-args が無視されました

-Wformat-extra-args を使用するには、-Wformat を指定する必要があります。

-Wformat-nonliteral ignored without -Wformat – -Wformat がいないため

-Wformat-nonliteral が無視されました

-Wformat-nonliteral を使用するには、-Wformat を指定する必要があります。

-Wformat-security ignored without -Wformat – -Wformat がいないため

-Wformat-security が無視されました

-Wformat-security を使用するには、-Wformat を指定する必要があります。

-Wformat-y2k ignored without -Wformat – -Wformat がいないため -Wformat-y2k が無視されました

-Wformat を指定する必要があります。

-Wid-clash-LEN is no longer supported – -Wid-clash-LEN は現在サポートされていません

オプション -Wid-clash-LEN は現在サポートされていません。

-Wmissing-format-attribute ignored without -Wformat – -Wformat がいないため

-Wmissing-format-attribute が無視されました

-Wmissing-format-attribute を使用するには、-Wformat を指定する必要があります。

-Wuninitialized is not supported without -O – -Wuninitialized は -O なしではサポートされません

-Wuninitialized オプションを使用するには、最適化をオンにする必要があります。

'identifier' was declared 'extern' and later 'static' – 'identifier' は 'extern' として宣言され、その後 'static' として宣言されました

指定した識別子は 'extern' として宣言された後に静的として宣言されました。

'identifier' was declared implicitly 'extern' and later 'static' – 'identifier' は暗示的に 'extern' として宣言され、その後 'static' として宣言されました

指定した識別子は暗示的に 'extern' として宣言された後に静的として宣言されました。

'identifier' was previously implicitly declared to return 'int' – 'identifier' は 'int' 戻り値として以前暗示的に宣言されました

以前の暗示的宣言に対して不一致があります。

'identifier' was used with no declaration before its definition – 'identifier' はその定義の前に宣言されていません

-Wmissing-declarations コマンドラインオプションでコンパイルする際、コンパイラは関数が定義の前に宣言されていることを確認します。この場合、事前宣言の無い関数定義が検出されます。

'identifier' was used with no prototype before its definition – 'identifier' はその定義の前にプロトタイプ指定されていません

-Wmissing-prototypes コマンドラインオプションでコンパイルする際、コンパイラは関数プロトタイプがすべての関数に指定されていることを確認します。この場合、コールされる前に関数プロトタイプが無いことを検出しました。

writing into constant object (arg n) – 定数オブジェクト (arg n) への書き込みです

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラは指定した引数 *n* はフォーマット指定で書き込み先として指定した *const* オブジェクトであることを検出しました。

Z

zero-length identifier format string – ゼロ長 identifier フォーマット文字列です

printf、*scanf* などへのコールの引数リストを確認する際、コンパイラはフォーマット文字列が空(“”)であることを検出しました。

付録 C. MPLAB C18 と MPLAB C30 C コンパイラ

C.1 はじめに

本章の目的は、MPLAB C18 と MPLAB C30 C コンパイラの違いの明確化です。MPLAB C18 コンパイラの詳細については、*MPLAB® C18 C Compiler User's Guide* (DS51288) をご参照ください。

本章では、2つのコンパイラの相違点を、以下の項目について説明します。

- データフォーマット
- ポインタ
- ストレージクラス
- スタックの使い方
- ストレージ修飾子
- 定義されたマクロ名
- 整数拡張
- 文字列定数
- 匿名構造体
- アクセスメモリー
- インラインアセンブリー
- Pragma
- メモリモデル
- コール規則
- スタートアップコード
- コンパイラで管理されるリソース
- 最適化
- オブジェクトモジュールフォーマット
- インプリメンテーション定義された動作
- ビットフィールド

C.2 データフォーマット

表 C-1: データフォーマットで使用されるビット数

データフォーマット	MPLAB® C18 ⁽¹⁾	MPLAB C30 ⁽²⁾
char	8	8
int	16	16
short long	24	-
long	32	32
long long	-	64
float	32	32
double	32	32 or 64 ⁽³⁾

- 注 1: MPLAB C18 は独自のデータフォーマットを使用し、それは IEEE-754 フォーマットに似ていますが、上位 9 ビットが回転します (表 C-2 をご参照ください)。
2: MPLAB C30 は IEEE-754 フォーマットを使用します。
3: セクション 5.5 「浮動小数点」をご参照ください。

表 C-2: MPLAB® C18 FLOATING-POINT と MPLAB C30 IEEE-754 フォーマット

標準	バイト 3	バイト 2	バイト 1	バイト 0
MPLAB C30	seeeeeee1	e ₀ ddd dddd16	dddd dddd8	dddd dddd0
MPLAB C18	eeeeeeee0	sddd dddd16	dddd dddd8	dddd dddd0

凡例: s = 符号ビット、d = 仮数、e = 指数

C.3 ポインタ

表 C-3: ポインタで使用されるビット数

メモリアイプ	MPLAB® C18	MPLAB C30
プログラムメモリ - Near	16	16
プログラムメモリ - Far	24	16
データメモリ	16	16

C.4 ストレージクラス

MPLAB C18 では、変数、および、関数引数用の auto もしくは static といった非 ANSI ストレージクラス overlay 指定子が使用できます。

MPLAB C30 ではこれらの指定子は使用できません。

C.5 スタックの使い方

表 C-4: 使用されるスタックのタイプ

スタック上の項目	MPLAB® C18	MPLAB C30
戻りアドレス	ハードウェア	ソフトウェア
ローカル変数	ソフトウェア	ソフトウェア

C.6 ストレージ修飾子

MPLAB C18 は非 -ANSI の `far`、`near`、`rom` および `ram` のタイプの修飾子を使用します。

MPLAB C30 は非 -ANSI の `far`、`near` および `space` の属性を使用します。

例 C-1: `near` 変数を定義する

```
C18: near int gVariable;
C30: __attribute__((near)) int gVariable;
```

例 C-2: `far` 変数を定義する

```
C18: far int gVariable;
C30: __attribute__((far)) int gVariable;
```

例 C-3: プログラムメモリ内の変数を作成する

```
C18: rom int gArray[6] = {0,1,2,3,4,5};
C30: __attribute__((section(".romdata"), space(prog)))
      int gArray[6] = {0,1,2,3,4,5};
```

C.7 定義されたマクロ名

MPLAB C18 は、選択されたメモリモデルにより、`__18CXX`、`__18F242`、... (`__prefix` を持ったほかのプロセッサ) と `__SMALL__` もしくは `__LARGE__` を定義します。

MPLAB C30 は `__dsPIC30` を定義します。

C.8 整数拡張

MPLAB C18 は、両方のオペランドが `int` より小さくても、一番大きいオペランドのサイズで整数拡張を実行します。MPLAB C18 には標準に適合するように `-Oi+` オプションがあります。

ISO で必須とされている `int` 精度もしくはそれより大きい値で整数拡張を実行します。

C.9 文字列定数

MPLAB C18 はプログラムメモリ内の文字列定数を、`.stringtable` セクション内でも保持します。MPLAB C18 は文字列関数のいくつかの変数をサポートします。例えば、`strcpy` 関数は4つの変数を持ち、文字列の、データ/プログラムメモリへの、もしくはデータ/プログラムメモリからのコピーが実行できます。

MPLAB C30 は、その他のデータがアクセスされるように、データメモリもしくは PSV ウィンドウを通して、プログラムメモリからの文字列定数のアクセスができません。

C.10 匿名構造体

MPLAB C18 は、共用体の中で、非 ANSI の匿名構造体をサポートします。
MPLAB C30 はサポートしません。

C.11 アクセスメモリー

dsPIC30F デバイスにはアクセスメモリーがありません。

C.12 インラインアセンブリー

MPLAB C18 は、インラインアセンブリーのブロックを識別するために、非 ANSI の `_asm` と `_endasm` を使用します。

MPLAB C30 は、関数コールのように見える、非 ANSI の `asm` を使用します。
MPLAB C30 での `asm` 文の使用に関して詳細は、[セクション 8.4 「インラインアセンブリ言語を使用する」](#) に述べられています。

C.13 PRAGMA

MPLAB C18 は、セクション用 (`code`、`romdata`、`udata`、`idata`)、割り込み用 (最優先と低い優先) および変数ロケーション用 (バンク、セクション) の `pragma` を用います。

MPLAB C30 は、`pragma` の代わりに非 ANSI 属性を使用します。

表 C-5: MPLAB® C18 Pragma と MPLAB C30 属性

Pragma (MPLAB C18)	属性 (MPLAB C30)
<code>#pragma udata [name]</code>	<code>__attribute__((section("name")))</code>
<code>#pragma idata [name]</code>	<code>__attribute__((section("name")))</code>
<code>#pragma romdata [name]</code>	<code>__attribute__((space(prog)))</code>
<code>#pragma code [name]</code>	<code>__attribute__((section("name"))),</code> <code>__attribute__((space(prog)))</code>
<code>#pragma interruptlow</code>	<code>__attribute__((interrupt))</code>
<code>#pragma interrupt</code>	<code>__attribute__((interrupt, shadow))</code>
<code>#pragma varlocate bank</code>	NA*
<code>#pragma varlocate name</code>	NA*

*dsPIC® DSC デバイスはバンクを持ちません。

例 C-4: データメモリ内のユーザセクションにある初期化されない変数の指定

```
C18: #pragma udata mybss
      int gi;
C30: int __attribute__((__section__(".mybss"))) gi;
```

例 C-5: データメモリ内のアドレス 0x100 に変数 Mabonga を置く

```
C18: #pragma idata myDataSection=0x100;
      int Mabonga = 1;
C30: int __attribute__((address(0x100))) Mabonga = 1;
```


例 C-6: プログラムメモリに置かれる変数を指定する

```
C18: #pragma romdata const_table
      const rom char my_const_array[10] =
          {0,1,2,3,4,5,6,7,8,9};
C30: const __attribute__((space(const)))
      char my_const_array[10] = {0,1,2,3,4,5,6,7,8,9};
```

注: MPLAB C30 コンパイラは、プログラム空間の変数をアクセスすることを、直接にはサポートしていません。そのように割り当てられた変数は、通常はテーブルアクセスインラインアセンブリ命令を用いるかもしくはプログラム空間可視化 (PSV) ウィンドウを用いて、プログラマで明確にアクセスしなければなりません。PSV ウィンドウの詳細については、**セクション 4.15 「Program Space Visibility (PSV) の使用方法」**をご参照ください。

例 C-7: 関数 PrintString をプログラムメモリのアドレス 0x8000 に配置する

```
C18: #pragma code myTextSection=0x8000;
      int PrintString(const char *s){...};
C30: int __attribute__((address(0x8000))) PrintString
      (const char *s) {...};
```

例 C-8: コンパイラが変数 var1 と var2 を自動的に保存、回復する

```
C18: #pragma interrupt isr0 save=var1, var2
      void isr0(void)
      {
          /* perform interrupt function here */
      }
C30: void __attribute__((__interrupt__(__save__(var1,var2))))
      isr0(void)
      {
          /* perform interrupt function here */
      }
```

C.14 メモリモデル

MPLAB C18 は非 ANSI のスモールとラージのメモリモデルを使用します。スモールは 16-ビットのポインタを使用し、プログラムメモリを 64 KB (32 KB ワード) 以下に制限します。

MPLAB C30 は非 ANSI のスモールコードとラージコードのモデルを使用します。スモールコードはプログラムメモリを 96 KB (32 KB ワード) 以下に制限します。ラージコードでは、ポインタはジャンプテーブル内を検索します。

MPLAB® C30 ユーザーズガイド

C.15 コール規則

MPLAB C18 と MPLAB C30 のコール規則については、多くの差があります。MPLAB C30 のコール規則については、**セクション 4.12 「関数コール規則」**をご参照ください。

C.16 スタートアップコード

MPLAB C18 は 3 つのスタートアップルーチンを提供します。1 つはユーザデータ初期化しないもの、1 つはイニシャライザを持つ変数のみを初期化するもの、もう 1 つはすべての変数を初期化するもの（イニシャライザを持たないものは ANSI 標準で要求されるようにゼロに設定します）です。

MPLAB C30 は 2 つのスタートアップルーチンがあります。1 つはユーザデータ初期化を行わないもの、1 つは、**Persistent** データセクション内の変数以外のすべての変数を初期化するもの（イニシャライザを持たないものは ANSI 標準で要求されるようにゼロに設定します）です。

C.17 コンパイラで管理されるリソース

MPLAB C18 は以下のような管理リソースがあります：PC、WREG、STATUS、PROD、セクション .tmpdata、セクション MATHDATA、FSR0、FSR1、FSR2、TBLPTR、TABLAT。

MPLAB C30 には以下のような管理リソースがあります：W0-W15、RCOUNT、SR。

C.18 最適化

以下の最適化がそれぞれのコンパイラの一部です。

MPLAB® C18	MPLAB® C30
分岐 (-Ob+) コード整頓化 (-Os+) 最後部コードの併合 (-Ot+) 未到達コードの除去 (-Ou+) コピー伝播 (-Op+) 冗長保存の除去 (-Or+) 不要コードの除去 (-Od+)	最適化設定 (-O、n は 1, 2, 3 もしくは s) ⁽¹⁾
同一文字列の統合 (-Om+)	-fwritable-strings
バンキング (-On+)	なし - バンキングは未使用
WREG コンテンツトラッキング (-Ow+)	すべてのレジスタは自動的にトラックされます
過程抽象 (-Opa+)	過程抽象 (-mpa)

注 1: MPLAB C30 では、これらの最適化設定はほとんどの要求を満たします。詳細調整「fine-tuning」用に追加のフラグが使用できます。詳細については、**セクション 3.5.6 「最適化を制御するオプション」**をご参照ください。

C.19 オブジェクトモジュールフォーマット

MPLAB C18 と MPLAB C30 は、互換性のない異なる COFF ファイルフォーマットを使用します。

C.20 インプリメンテーション定義された動作

負の符号がついた整数値の右シフトに関して、

- MPLAB C18 は符号ビットを保持しません。
- MPLAB C30 は符号ビットを保持します。

C.21 ビットフィールド

MPLAB C18 のビットフィールドはバイトストレージ境界を越えることはできません。したがって、サイズが 8 ビットより大きな値をとることができません。

MPLAB C30 はどんなビットサイズでも、その基底タイプのサイズまでサポートします。どんな整数タイプもビットフィールドにできます。割り当ては、その基底タイプに備わったビット境界を越えることはできません。

例:

```
struct foo {
    long long i:40;
    int j:16;
    char k:8;
} x;

struct bar {
    long long I:40;
    char J:8;
    int K:16;
} y;
```

`struct foo` は、MPLAB C30 を使用する場合、10 バイトのサイズを持ちます。i はビットオフセット 0 (から 39 まで) に割り当てられます。j の前に 8 ビットのパディングがあり、j はビットオフセット 48 に割り当てられます。もし、j が次に利用できるビット 40 に割り当てられたら、16 ビット整数のストレージ境界を越えることとなります。j のあとのビットオフセット 64 に、k が割り当てられます。構造体は、配列の場合でも、必要な境界整列を保持するために、8 ビットのパディングを含みます。構造体内の一番大きい境界整列は 2 バイトなので、境界整列は 2 バイトになります。

`struct bar` は MPLAB C30 を使用する場合、8 バイトのサイズを持ちます。I はビットオフセット 0 (から 39 まで) に割り当てられます。J は、char のストレージ境界を越えず、ビットオフセット 40 に割り当てられるので、パッドする必要はありません。K はビットオフセット 48 から割り当てられ、空間を浪費することなく、構造体を完結できます。

メモ:

付録 D. 使用を廃止した機能

D.1 はじめに

次に説明する機能は、使用されなくなつたと考えられ、より高度な機能に置き換えられました。使用を廃止した機能に依存するプロジェクトは、次に列挙する言語ツールのバージョンで適切に動作します。使用を廃止した機能を使用すると、警告が出されます。使用を廃止した機能への依存を取り除くためにプロジェクトを変更することをおすすめします。こうした機能のサポートは、将来のバージョンの言語ツールでは完全になくなる可能性があります。

D.2 ハイライト

本章では、次の使用を廃止した機能について説明します。

- 事前定義定数

D.3 事前定義定数

次のプリプロセッシング記号が、MPLAB C30 コンパイラによって定義されます。

記号	-ansi コマンドライン オプションで定義されているか？
dsPIC30	いいえ
__dsPIC30	はい
__dsPIC30__	はい

ELF 特有のコンパイラ バージョンでは、次のプリプロセッシング記号が定義されます。

記号	-ansi コマンドライン オプションで定義されているか？
dsPIC30ELF	いいえ
__dsPIC30ELF	はい
__dsPIC30ELF__	はい

COFF 特有のコンパイラ バージョンでは、次のプリプロセッシング記号が定義されます。

記号	-ansi コマンドライン オプションで定義されているか？
dsPIC30COFF	いいえ
__dsPIC30COFF	はい
__dsPIC30COFF__	はい

最新の情報については、[セクション 3.7 「事前定義制約」](#) を参照してください。

メモ:

付録 E. ASCII 文字セット

表 E-1: ASCII 文字セット

		上位							
Hex	0	1	2	3	4	5	6	7	
0	NUL	DLE	Space	0	@	P	'	p	
1	SOH	DC1	!	1	A	Q	a	q	
2	STX	DC2	"	2	B	R	b	r	
3	ETX	DC3	#	3	C	S	c	s	
4	EOT	DC4	\$	4	D	T	d	t	
5	ENQ	NAK	%	5	E	U	e	u	
6	ACK	SYN	&	6	F	V	f	v	
7	Bell	ETB	'	7	G	W	g	w	
8	BS	CAN	(8	H	X	h	x	
9	HT	EM)	9	I	Y	i	y	
A	LF	SUB	*	:	J	Z	j	z	
B	VT	ESC	+	;	K	[k	{	
C	FF	FS	,	<	L	\	l		
D	CR	GS	-	=	M]	m	}	
E	SO	RS	.	>	N	^	n	~	
F	SI	US	/	?	O	_	o	DEL	

下位

メモ:

付録 F. GNU 無料ドキュメントライセンス

GNU 無料ドキュメントライセンス

バージョン 1.2 2002 年 11 月

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

このライセンスドキュメントのコピーおよび逐語的コピーの配布は許可されていますが、変更は許可されません。

0. 前書き

このライセンスの目的は、マニュアルやテキスト、その他機能的および役に立つドキュメントを、自由と言う意味において自由に作成するために、修正して、もしくは修正せずに、商用目的であれ非商用目的であれ、誰でもコピーしたり再配布したりすることの自由を保証するものです。このライセンスは、著者や出版者のために、その作品に対する信用を得るための方法を保持するものであり、他人によりなされた修正に責任を持つとは見なされません。

このライセンスは、「コピーレフト」に類するものであり、ドキュメントの派生物はそれ自体が同様に無料でなければなりません。それは、GNU General Public License を補足するものであり、無料ソフトウェア用に設計されたコピーレフトライセンスです。

このライセンスは、無料ソフトウェア用のマニュアルとして用いるために作成されています。無料ソフトウェアには無料のドキュメントが必要であり、無料のプログラムは、同様に自由である無料のマニュアルと一緒に供給されねばならないからです。しかし、このドキュメントは、ソフトウェアマニュアルだけには限定されず、主題となる内容もしくは印刷された本として出版されるか否かにかかわらず、原文に関するどんな作品にも用いられます。このライセンスを、主として、教育的もしくは参考的な目的として推奨します。

1. 適用範囲と定義

このライセンスは、どんなメディアであれ、著作権保有者による、このライセンス条項のもとで配布される、という注意書きを含むいかなるマニュアルやその他の作品にも適用されます。そのような注意書きは、全世界で、ロイヤリティ無しのライセンスを、期限無しで、ここに述べられる条件のもとで作品を使用する権利を与えます。以下に示す「ドキュメント」は、そのようなマニュアルもしくは作品を示します。どんな人でもライセンスを受けることができ、「あなた」として述べられます。著作権法のもと、許可を得たうえで、コピーしたり、修正したり、作品を配布する場合、このライセンスを受けることができます。

ドキュメントの「修正版」は、ドキュメントもしくはその一部を含むあらゆる著作物を意味し、逐語複写されたものもしくは、修正および/もしくは他の言語に翻訳されたもののいずれかです。

「第二のセクション」は名前を付けられた付録もしくはドキュメントの表書セクションで、ドキュメント全体の主題（もしくは関連事項に関して）出版者と著者の関係を専門に扱う部分であり、全体の主題に直接関わるものではありません（したがって、もしドキュメントが数学の教科書なら、第二のセクションは数学の説明をする

ものではありません)。関係とは、主題もしくは関連する項目との歴史的繋がりのある事柄、もしくは、それらに関する法律的、商業的、哲学的、倫理的、もしくは政治的地位の事柄である場合もあります。

「不変のセクション」は、ある種の第二のセクションで、そのタイトルが不変のセクションであり、ドキュメントがこのライセンスのもとに発行されていると述べた注意が記載されています。もしセクションが、上記 **Secondary** の定義に適合しなければ、不変のと記載するのは許されません。ドキュメントによっては、不変のセクションの無いものもあります。もしドキュメントが不変のセクションを認識しないのであれば、それは無いからです。

「カバーテキスト」は、表書 (Front-Cover Texts) もしくはあとがき (Back-Cover Texts) と、リストアップされた短い文章で、ドキュメントがこのライセンスのもとに発行されていると述べた注意が記載されています。表書は多くても 5 語、あとがきは多くても 25 語です。

ドキュメントの「透明な」コピーとは、マシンで読み込むことのできるコピーを意味し、その表現フォーマットは一般大衆に利用できるものであり、基本的なテキストエディタもしくは (ピクセルで構成された画像用に) 基本的なペイントプログラムもしくは (描画用の) 広く利用可能な描画エディタで直接的に改訂するのに適しており、テキストフォーマットへの入力に適したもの、もしくはテキストフォーマットへの入力に適する種々のフォーマットへの自動変換に適したものです。別の透明なファイルフォーマットで作られたコピーで、そのマークアップが (もしくはマークアップのないものもあるが) 読み取り器による引続き行われる修正を阻止するようなコピーは、透明ではありません。画像フォーマットは、たくさんの量のテキスト用に使用されるとしたら、透明ではありません。透明でないコピーは「不透明」と呼びます。

透明なコピー用の適切なフォーマットの例は、マークアップを持たない普通の ASCII、Texinfo 入力フォーマット、LaTeX 入力フォーマット、一般に利用可能な DTD を用いた SGML もしくは XML、および、人手で修正可能なように設計された、標準に準拠した簡単な HTML、PostScript もしくは PDF です。透明な画像フォーマットには、PNG、XCF および JPG があります。不透明なフォーマットには、独自のワードプロセッサによってのみ読んだり編集したりすることのできる独自のフォーマットを持ったもので、DTD および / もしくはプロセッシングツールが一般的に利用できない SGML もしくは XML、および、出力目的専用のワードプロセッサで生成された、HTML、PostScript もしくは PDF があります。

「タイトルページ」は、印刷された著作物用として、タイトルページ自体、プラス、タイトルページに明記することをライセンスが必要としているものをはっきりと保持するために必要なページを意味します。そのようなタイトルページを持たないフォーマットの作品では、「タイトルページ」は、文章本体の始まりより以前にあり、作品のタイトルが最も大きく表示される場所に近いテキストを意味します。

セクション「XYZ と題をつけられた」は、タイトルがまさしく XYZ もしくは、別の言語で XYZ と訳されるテキストに引続く括弧内に XYZ を含むようなタイトルを持つドキュメントのサブユニットを意味します (ここで、XYZ は以下に示すような特定のセクションを意味し、それらは謝辞、献呈の辞、裏書、もしくは履歴を意味します)。ドキュメントを修正する際にそのようなセクションのタイトルを保存すること (「Preserve the Title」) は、この定義にしたがって、「XYZ と題をつけられた」セクションを保持することを意味します。

ドキュメントは、このライセンスをそのドキュメントに適用するという注意書きの次に免責事項を含む場合もあります。この免責事項はこのライセンスのリファレンスによって含まれると見なされますが、責任放棄に関してのみであり、これらの免責事項が持つであろうその他の暗示については無効であり、このライセンスの意味には影響を与えません。

2. 逐語的複製

このライセンス、著作権の注意書き、および、ライセンスがドキュメントに適用されることを述べる注意書きがすべての複製物で再製造されるのであれば、またこのライセンスの条件に他のどんな条件をも追加しないのであれば、ドキュメントを、どのようなメディアにおいてでも、商用であれ非商用であれ、複製し配布することができます。複製し配布する複製物を読んだりもしくは更なる複製をすることを、妨害したりコントロールしたりする技術的な手段を用いてはなりません。しかしながら、複製物と引き換えに補償を受けることができます。もし大量の複製物を配布するのであれば、第3章の条件に従わねばなりません。

複製物を、上記で述べられているのと同じ条件で、貸与したり、公衆に展示することもできます。

3. 大量の複製

ドキュメントの印刷された複製物（もしくは印刷されたカバーを共通に持つメディアに複製されたもの）を100部以上出版し、ドキュメントのライセンス注意書きがカバーテキストを必要とするならば、すべてのカバーテキスト、表紙の表書きや裏表紙の後書きにはっきりと、そのライセンス注意書きを含めなければなりません。両方のカバーははっきりとあなたがそれらの複製物の出版者であることが認識できるようにしなければなりません。その他の記事をカバーに追加することもできます。カバーに限り修正を加えて複製を行うことは、それがドキュメントのタイトルを保持しこれらの条件を満たす限り、別の面から見ても逐語的な複製として扱われます。もしどちらかのカバー用に要求される文が、あまりにも多くて明確性に適合しないなら、最初の文を実際のカバーに（合理的に適合するようできるだけ多く）リストアップし、残りを近くのページに続けるようにしなければなりません。

もしドキュメントの不透明な複製物を100部以上出版もしくは配布するのであれば、それぞれの不透明な複製物にマシンで読める透明なコピーと一緒に含めるか、それぞれの不透明な複製物内もしくはそれに添えて、一般のネットワークを使用する大衆が、公衆の標準であるネットワークプロトコルを用いて、ドキュメントの完全に透明な複製物を、添付物無しに、ダウンロードするためにアクセスするロケーションを記述しなければなりません。後者のオプションを使用する場合は、多くの不透明な複製物の配布を開始する時は、この透明な複製物が、記述されたロケーションでアクセスできる状態を、公衆向けの版の不透明な複製物を（直接もしくは代理店もしくは小売店を経由して）最後に配布した後すくなくとも1年は保持することを保証するという、合理的に注意深い手順を取らねばなりません。

強制ではありませんが、ドキュメントの著者には、大量の複製物を再配布する場合は十分な時間をみて事前にコンタクトし、ドキュメントの最新版を供給する了承を得るべきです。

4. 修正

上記第2章、第3章の条件のもと、このライセンスに忠実に従い、ドキュメントの役割を満たし、したがって、修正版の複製物を所有する誰に対しても、修正版の配布と修正する権利をライセンスすることのできる修正版を発行することを条件に、ドキュメントの修正版を複製し配布することができます。さらに、修正版に関して、以下のことをしなければなりません。

- a) タイトルページ（およびもしあるならカバー）に、ドキュメントのタイトルから、および以前の版（もしあるなら、ドキュメントの履歴セクションにリストアップされているような）のタイトルとは異なったタイトルを使用します。もしオリジナルの出版者が許可を与えるのなら、以前の版と同じタイトルを使用できます。
- b) タイトルページに、著者として、修正版の中で修正の権利に責任のある1人もしくはそれ以上の人物もしくは実体のあるものを、（もしリストアップするという要求が著者たちにより免除されない限り）ドキュメントの主な著者のうち少なくとも5人（5人より少なければすべての著者）と一緒にリストアップします。

- c) タイトルページに、出版者として、修正版の出版者の名前を述べます。
- d) ドキュメントのすべての著作権を保持します。
- e) その他の著作権注意書きに近接する、あなたの修正に対する適切な著作権を追加します。
- f) 著作権注意書きの直後に、以下の補遺に示される形式で、このライセンスの条項において修正版を使用する公の許可をライセンスする注意書きを含みます。
- g) そのライセンス注意書きには、不変のセクションおよび、ドキュメントのライセンス注意書きにあるような要求されるカバーテキストのすべてのリストを保持します。
- h) このライセンスの改変されていない複製を含みます。
- i) 「履歴」と題されたセクションを保持し、そのタイトルを保持し、タイトルページで与えられる修正版の、少なくともタイトル、年、新しい著者および出版者を述べた項目を、それに追加します。もしドキュメントに「履歴」と題されたセクションがない場合は、タイトルページで与えられる修正版の、少なくともタイトル、年、新しい著者および出版者を述べた項目を作成し、以前の文章で述べられたように、修正版を説明している項目を追加します。
- j) もしあれば、ドキュメント内で与えられ、ドキュメントの透明な複製に公的にアクセスするためのネットワークロケーション、および、同様にドキュメント内で与えられる、そのドキュメントの元になっている、以前の版用のネットワークロケーションを保持します。これらは、「履歴」セクションに置かれています。ドキュメント自体より少なくとも4年前に出版された作品のネットワークロケーション、もしくは、参照している原出版者の許可があれば、ネットワークロケーションは省略できます。
- k) 「謝辞」もしくは「献呈」と題されたセクション用に、セクションのタイトルを保持し、ここで与えられる、貢献者に対する謝辞および/もしくは献呈のすべての中身や論調を、セクション内に保持します。
- l) その文やタイトル内で変更されない、ドキュメントのすべての不変のセクションを保持します。セクション番号もしくはそれに相当するものは、セクションタイトルの一部だとはみなされません。
- m) 「裏書」と題されたセクションを消去します。そのようなセクションは、修正版には含まれません。
- n) 存在するセクションに再タイトル化して、「裏書」と題されるようにするとか、もしくは不変のセクションを持ったタイトルと競合するようにはいけません。
- o) 免責事項を保持します。

もし修正版が新しい前付セクションもしくは付録を含みそれが、第二のセクションとして認められ、ドキュメントからのマテリアル複製を含まないなら、あなたの判断でそれらのいくつかもしくはすべてのセクションを不変として指名できます。これを行うには、修正版のライセンス通知の中の不変セクションのリストに、そのタイトルを追加します。これらのタイトルは、その他のセクションタイトルとは異なるものでなければなりません。

もし「裏書」と題されたセクションが、種々の団体による修正版の裏書のみを含んでいる場合、例えば、よく見たレビュー文もしくは文が標準の権威ある定義としての機関に承認されているという文言であれば、裏書を追加できます。

最大5文字までの1節をフロントカバーテキストとして、また最大25文字までの1節をバックカバーテキストとして、修正版の中のカバーテキストのリストの終わりに追加できます。1つの団体によって（もしくはアレンジされることにより）フロントカバーテキストの1つの節および、バックカバーテキストの1つの節のみを追加できます。もしドキュメントが、同じカバーにカバーテキストをすでに含んでいる場合、以前追加されたかもしくは、その団体のために行動しているの団体によって準備された場合は、さらなる追加はできませんが、旧版を追加した以前の出版者からの明確な許可があれば、旧版と置き換えることができます。

ドキュメントの著者と出版者は、そのライセンスにより、彼らの名前を宣伝に用いたり、修正版の裏書を明言もしくは暗示するしたりする許可を与えてはなりません。

5. ドキュメントを結合する

ドキュメントを、すべてのオリジナルドキュメントのすべての不変のセクションを、修正無しで含め、ライセンス通知の中に、結合された作品の不変のセクションをリストアップし、すべての免責事項を保持すれば、このライセンスのもと（修正版用としては、上の第4章で定義された条項のもと）で発行された他のドキュメントと一緒に結合することができます。

結合された作品は、このライセンスの複製を1つだけ含めばよく、複数の同じ不変のセクションは、1つの複製で置き換えられます。もし同じ名前だが異なる内容を持った複数の不変のセクションがあるならば、その終わりの部分に、もし分かるようであれば、セクションのオリジナル著者もしくは出版者を、もしくは他の一意の番号を、括弧の中に入れて追加することで1つ1つのセクションのタイトルを一意にします。同様な調整を、結合された作品のライセンス通知の中の不変のセクションのリスト内のセクションタイトルに行います。

結合にあたっては、種々のオリジナルドキュメント内の「履歴」と題されたセクションを結合して、1つの「履歴」と題されたセクションにしなければなりません。同様に、「謝辞」や「献呈」と題されたセクションについても、1つのセクションにしなければなりません。「裏書」と題されたセクションはすべて消去しなければなりません。

6. ドキュメントの収集

もし他のすべての面において、ドキュメントの1つ1つを逐語的に複製するために、このライセンスの規則に従うのであれば、ドキュメントと、このライセンスのもとで発行された他のドキュメントから成るコレクションを生成し、種々のドキュメント内にあるこのライセンスの個々の複製を、コレクションに含まれる1つの複製で置き換えることができます。

もし、抽出されたドキュメントにこのライセンスの複製を挿入し、そのドキュメントを逐語的に複製することに関するすべてのその他の面においてこのライセンスに従うのであれば、そのようなコレクションから1つのドキュメントを抽出し、このライセンスのもとで個別に配布できます。

7. 独立作品の集合

もし結合の結果としての著作権が、個々の著作物が許可する範囲を超えて結合物のユーザの法律上の権利を制限するために使用されないのであれば、ドキュメントもしくはその派生物をその他の分けられた独立のドキュメントもしくは作品と、ストレージもしくは配布メディアのボリューム内もしくはボリューム上で、結合することを、「集合」と呼びます。

もし、第3章のカバーテキストで要求される物がドキュメントの複製物に適用されるのであれば、ドキュメントが全体の集合の半分より小さいのであれば、ドキュメントのカバーテキストは、集合内でドキュメントを括弧でくるカバー上、もしくはもしドキュメントが電子形態であれば電子的にカバーに該当する場所に置かれず。そうでなければ、集合全体を括弧でくくった印刷されたカバーの上に表示されなければなりません。

8. 翻訳

翻訳は修正と見なされますので、第4章の条項のもとで、ドキュメントの翻訳物を配布することができます。不変のセクションを翻訳に置き換えるには、著作権所有者からの特別な許可を必要としますが、それらの不変のセクションのオリジナル版に追加して、いくつかのもしくはすべての不変のセクションの翻訳を含めることができます。もし、このライセンスのオリジナルの英語版およびそれらの通知や放棄のオリジナル版を含めるならば、このライセンスの翻訳、ドキュメント内のすべて

のライセンス通知およびすべての免責事項を含めることができます。このライセンスもしくは通知もしくは放棄の翻訳とオリジナル版の間に不一致がある場合は、オリジナル版が優先されます。

このドキュメント内のセクションが「謝辞」「献呈」もしくは「履歴」と題されたものであるなら、そのタイトル（第1章）を保持するという要求（第4章）は、典型的に実際のタイトルを変更することを要求します。

9. 失効

このライセンスではっきりと与えられていない場合、ドキュメントを複製したり修正したりサブライセンスを与えることはできません。ドキュメントを複製したり修正したりサブライセンスを与えるということ以外の試み、行為は、無効であり、このライセンスのもとで自動的に権利を失効させます。しかし、このライセンスのもとで、あなたから複製や権利を受けた当事者は、ライセンスに忠実に従い続ける限りは、ライセンスを失効されることはありません。

10. このライセンスの将来の改定

Free Software Foundation は、時により、GNU 無料ドキュメントライセンスの新しい改訂版を出版します。そのような新版は、意図は現行版に似ていますが、新しい問題もしくは事柄を述べているもので、詳細は異なります。<http://www.gnu.org/copyleft/>を参照下さい。

ライセンスのそれぞれの版は、区別するための版番号を与えられています。もしドキュメントがこのライセンスの特定の番号の版もしくはより後の版が適用されることを指定しているとすると、Free Software Foundation により（草案としてではなく）出版されている、その指定された版もしくはより後の版のどちらかの条項と条件に従うオプションを選択します。もしドキュメントがこのライセンスの版番号を指定しないのであれば、これまで Free Software Foundation により（草案としてではなく）出版されたどの版でも選択できます。

用語集

ANSI

米国規格協会 (American National Standards Institute) は合衆国内での標準を作成、承認することに責任を持つ機関です。

ASCII

情報交換用米国標準コード (American Standard Code for Information Interchange) は、それぞれの文字を表示するために 7 バイナリビットを用いてコード化された文字のことです。大文字、小文字、アラビア数字、シンボルおよび制御文字を含みます。

C

記述が少なく、現代のコントロールフローとデータ構造、豊富なオペレータセットを特徴とする、汎用プログラム言語。

COFF

Common Object File Format。このフォーマットのオブジェクトファイルは機械コード、デバッグおよびその他の情報を含みます。

DSC

デジタルシグナルコントローラを参照してください。

DSP

デジタルシグナルプロセッサを参照してください。

GPR

汎用レジスタ (General Purpose Register)。デバイスのデータメモリ (RAM) の一部で、汎用に用いることができます。

Hex コード

16 進のフォーマットコードでストアされる実行可能命令。Hex コードは Hex ファイルに含まれます。

Hex ファイル

デバイスをプログラムするのに適した、16 進のアドレスや値 (Hex コード) を含んだ ASCII ファイル。

IDE

統合開発環境 (Integrated Development Environment)。MPLAB IDE は Microchip の統合された開発環境です。

IEEE

Institute of Electrical and Electronics Engineers。

IRQ

割り込み要求をご参照ください。

ISO

国際標準化機関 (International Organization for Standardization) をご参照ください。

ISR

割り込みサービスルーチンをご参照ください。

L 値

検証や変更が可能なオブジェクトを参照する式。L 値式は、代入の左側で使用されません。

MPLAB ASM30

マイクロチップ社の、dsPIC30F デジタルシグナルコントローラデバイス用のリロケータブルマクロアセンブラ。

MPLAB C1X

マイクロチップ社の MPLAB C17 と MPLAB C18 C 両方を指します。MPLAB C17 は PIC17CXXX デバイス用の C コンパイラで、MPLAB C18 C は PIC18CXXX と PIC18FXXXX デバイス用の C コンパイラです。

MPLAB C30

dsPIC30F デジタルシグナルコントローラデバイス用のマイクロチップ社の C コンパイラ。

MPLAB IDE

マイクロチップ社の集積化された統合開発環境です。

MPLAB LIB30

MPLAB LIB30 アーカイバ/ライブラリアンは、MPLAB ASM30 もしくは MPLAB C30 C コンパイラのどちらかを用いて生成される COFF オブジェクトモジュールとともに使用されるオブジェクトライブラリアンです。

MPLAB LINK30

MPLAB LINK30 は、マイクロチップ社の MPLAB ASM30 アセンブラとマイクロチップ MPLAB C30 C コンパイラ用のオブジェクトリンカです。

PICmicro MCUs

PIC マイクロコントローラ (MCUs) はマイクロチップ社のマイクロコントローラファミリを指します。

RAM

ランダムアクセスメモリ (データメモリ)。情報にどんな順番でもアクセスできるようなメモリ。

ROM

リードオンリーメモリ (プログラムメモリ)。変更できないメモリ。

SFR

特殊関数レジスタを参照下さい。

アーカイバ

ライブラリを生成、操作する道具。

アーカイブ

再配置可能なオブジェクトモジュールの集合。複数のソースファイルをオブジェクトファイルにアセンブルし、アーカイバを用いてオブジェクトファイルを 1 つのライブラリファイルに束ねることで生成されます。ライブラリはオブジェクトモジュールと他のライブラリとリンクされ、実行可能コードを生成します。

アクセスメモリ (PIC18 のみ)

バンク選択レジスタ (BSR) の設定に関わらずアクセスできる PIC18XXXXX デバイスの特殊レジスタ。

アセンブラ

アセンブリ言語ソースコードを機械コードに変換する言語ツールです。

アセンブリ言語

バイナリ機械コードをシンボリック形式で記述するプログラム言語です。

アドレス

メモリ内の位置を示す値。

アルファニューメリック

アルファニューメリック文字は、アルファベット文字と 10 進数 (0,1,...,9) から構成されます。

アルファベット文字

アルファベット文字とは、アラビア文字のことです。(a, b, ...,z, A, B, ...,Z)。

エピローグ

スタック領域の割り当て解除、レジスタの復元、およびランタイム モデルで指定されたその他のマシン固有の条件を実行する、コンパイラによって生成されたコードの一部。このコードは、特定の関数のユーザー コードの後、関数の戻りの直前に実行されます。

エラー

エラーは、ユーザーのプログラムの処理を続行できなくする問題を報告します。可能な場合は、問題が認められるソース ファイル名と行番号を明らかにします。

演算子

プラス符号 '+' やマイナス符号 '-' のようなシンボルで、十分に定義された表現で記述する際に使用されます。それぞれの演算子は評価の順序を決定するために指定された優先権を持ちます。

エンディアンネス

マルチ バイト オブジェクト内のバイトの順序を記述します。

オブジェクト ファイル

機械語コードとデバッグ情報もある場合もあるが、それらを含むファイル。すぐに実行可能かもしれないが、完全な実行可能なプログラムを生成するには、他のオブジェクトファイル、例えばライブラリ、とリンクが必要な、再配置可能なファイルです。

オペコード

実行コード。ニーモニックを参照ください。

記憶域クラス

オブジェクトの寿命を決定します。

警告

警告は、問題かもしれないものの、処理を停止しない状態を報告します。MPLAB C30 では、警告メッセージはソース ファイル名と行番号を報告しますが、エラーメッセージと区別するために「warning:」というテキストが含まれます。

高水準言語

アセンブリよりもさらにプロセッサから離れた、プログラムを作成する言語です。

国際標準化機構

コンピューティングや通信を含め、多くの商業や技術における標準を設定する機関。

コマンドライン インターフェース

テキストの入力、出力に基づいた、プログラムとそのユーザーとの間の通信手段。

再帰呼び出し

直接的または間接的に自らを呼び出す関数。

再配置

リンカによって実行される処理で、絶対アドレスが再配置可能セクションに割り当てられ、再配置可能セクション内のすべての記号が新しいアドレスに更新されます。

再配置可能

セクションがメモリ内の固定位置に割り当てられていないオブジェクトファイル。

式

算術演算子または論理演算子によって分離された定数や記号の組み合わせ。

識別子

関数名または変数名。

シミュレータ

デバイスの動作をモデル化したソフトウェアプログラム。

初期化済みデータ

初期値とともに定義されるデータ。C では、

```
int myVar=5;
```

は、初期化済みデータ セクションに存在する変数を定義します。

指令

言語ツールの動作の制御を行う、ソースコード内のステートメント。

実行可能コード

実行するためにロードする準備ができていないソフトウェア。

スタック、ソフトウェア

戻りアドレス、関数パラメータおよびローカル変数をストアするために、アプリケーションにより使用されるメモリ。このメモリは高級言語内でコードを開発する際にコンパイラにより管理されます。

ストレージ修飾子

オブジェクトの特別なプロパティを示します(例えば、volatile)。

セクション

コードまたはデータの名前付きシーケンス。

ソースコード

その中に、プログラマによりコンピュータプログラムが書かれる形式。ソースコードは、翻訳されるか、機械語コードか、もしくは翻訳器により実行されるようないくつかの公式プログラム言語で書かれます。

ソースファイル

ソースコードを含む ASCII テキストファイル。

属性

マシン固有のプロパティを記述するのに使用される C プログラム内の変数や関数の特性。

データメモリ

Microchip の MCU と DSC デバイスでは、データメモリ (RAM) は、汎用レジスタ (GPRs) と特殊関数レジスタ (SFR) から成ります。いくつかのデバイスでは、EEPROM データメモリも持っています。

デジタルシグナルコントローラ

デジタル信号処理機能を備えたマイクロコントローラ デバイス(つまり、マイクロチップ社 dsPIC DSC デバイス)。

デジタル シグナル プロセッサ

デジタル信号処理で使用するよう設計されたマイクロプロセッサ。

デジタル信号処理

通例、デジタル形式に変換された (サンプリングされた) アナログ信号 (音声または画像) である、デジタル信号のコンピュータ操作。

デバイス プログラマ

マイクロコントローラのような電氣的にプログラム可能な半導体デバイスのプログラムに使用されるツールです。

特殊機能レジスタ

I/O プロセッサ関数、I/O ステータス、タイマもしくはその他のモードや周辺機器を制御する専用のレジスタで、データメモリ (RAM) の一部。

匿名構造体

無名の構造体。

トライグラフ

すべて ?? で始まる 3 文字のシーケンス。ISO C で 1 文字の置換として定義されています。

ニーモニック

直接機械語コードに翻訳できるテキスト命令。Op コードとも言われます。

ヒープ

動的なメモリ割り当てに使用されるメモリの領域で、実行時に決定される任意の順序でメモリ ブロックが割り当ておよび解放されます。

非初期化データ

初期値を持たずに定義されるデータ。C では

```
int myVar;
```

は、初期化されないデータセクションに駐在する変数を定義します。

ファイル レジスタ

チップに内蔵されたデータメモリで、汎用レジスタ (GPR) と特殊関数レジスタ (SFR) を含みます。

フリースタンディング

複雑な型を使用せず、ISO ライブラリ条項に規定される機能の使用が標準ヘッダ <float.h>、<iso646.h>、<limits.h>、<stddef.h>、および <stdint.h> の内容に限定された、厳密に準拠するプログラムを受け付ける C コンパイラの実装。

フレーム ポインタ

スタックベースの引き数とスタックベースのローカル変数とを区別する、スタック上の位置を参照するポインタ。現在の関数のローカル変数およびその他の値にアクセスする、便利な基点となります。

プラグマ

特定のコンパイラに対して意味を持つ指令。多くの場合、プラグマは、コンパイラに対して実装定義情報を伝えるために使用されます。MPLAB C30 は、属性を使用してこの情報を伝えます。

プログラム カウンタ

現在実行中の命令のアドレスを含む位置。

プログラム メモリ

命令がストアされる、デバイス内のメモリ領域。

プロローグ

スタック領域の割り当て、レジスタの保存、およびランタイム モデルで指定されたその他のマシン固有の条件を実行する、コンパイラによって生成されたコードの一部。このコードは、特定の関数のユーザー コードの前に実行されます。

ベクタ

リセットや割り込みなどの特定のイベントが発生したときにアプリケーションが実行を開始するメモリ ロケーション。

マイクロコントローラ (MCU)

CPU、RAM、プログラムメモリ、I/O ポートおよびタイマを含む、高度集積チップ。

マクロ

マクロ命令のこと。短縮された形式の一連の命令を表現する命令。

マシン言語

特定の中央処理装置用の命令のセットで、翻訳無しにプロセッサで使用できるように設計されたもの。

マシンコード

プロセッサにより実際に読み込まれ解釈されるコンピュータプログラムの表現。バイナリ機械語コードのプログラムは一連の機械語命令（データが挿入されることもある）で構成されます。特定のプロセッサ用のすべての可能性のある命令の集合は、「命令セット」として知られています。

命令

中央処理装置 (CPU) に特定の動作を行うよう指示し、その動作に用いられるデータを含むことができるビットのシーケンス。

命令セット

特定のプロセッサが理解する機械語命令の集合。

メモリ モデル

アプリケーションで利用できるメモリの表現。

優先順位

式内での評価順を定義するルール。

ライブラリ

アーカイブをご参照ください。

ライブラリアン

アーカイバをご参照ください。

ランタイム モデル

ターゲット アーキテクチャのリソースの使用を記述します。

リトル エンディアンネス

マルチバイト データのデータ順序体系で、最下位のバイトが下位アドレスに格納されます。

リンカ

オブジェクトファイルとライブラリを結合し実行可能コードを生成する言語ツールで、あるモジュールから別のモジュールへの参照を分解します。

リンカ スクリプト ファイル

リンカスクリプトファイルはリンカのコマンドファイル。リンカオプションを定義し、ターゲットプラットフォーム上の利用できるメモリを記述します。

レイテンシ

イベントとその応答の間の時間。

割り込み

動作中のアプリケーションの実行を中断し、イベントが処理されるために割り込みサービスルーチン (ISR) に制御を渡すことを行う、CPU への信号。

割り込みサービス ルーチン

割り込みが発生したときに起動される関数。

割り込みハンドラ

割り込みが発生した時に、特別のコードを処理するルーチン。

割り込み要求

プロセッサが一時的に通常の命令実行を中断し、割り込みハンドラの実行を開始させるようなイベント。いくつかのプロセッサでは、異なる優先度の割り込みを許可する、いくつかの割り込みリクエストイベントを持ちます。

メモ:

索引

記号

#define	47
#ident	54
#if	40
#include	48, 49, 79, 81
#line	50
#pragma	37, 121, 178
.bss	15, 62, 121
.const	61, 63, 75
.data	15, 61, 121
.dconst	62
.dinit	62, 63
.nbss	62
.ndata	61
.ndconst	62
.pbss	62, 63
.text	22, 32, 61, 67, 121
.tmpdata	180

A

-A	47
abort	22, 124
address 属性	12, 19
alias 属性	19
aligned 属性	13
-ansi	23, 34, 50
ANSI C、MPLAB C30 との違い	11
ANSI C、厳密	35
ANSI C 標準	9
ANSI-89 extension	77
ANSI 標準ライブラリ サポート	9
ASCII 文字セット	185
asm	13, 109, 178
atomic 動作	102
auto_psv 領域	31
Automatic 変数	37, 38, 69
-aux-info	34

B

-B	52, 55
----	--------

C

-C	47
-c	33, 51
C、アセンブリとの混用	107
Case 範囲	28
Cast	37, 38, 39
char	14, 34, 35, 72, 74, 77
COFF	7, 8, 58, 80, 180
const 属性	20

CORCON	63, 79, 80
C スタック使用方法	69
C ヒープの使用方法	71
C 方言制御オプション	34
-ansi	34
-aux-info	34
-ffreestanding	34
-fno-asm	34
-fno-builtin	34
-fno-signed-bitfields	34
-fno-unsigned-bitfields	34
-fsigned-bitfields	34
-fsigned-char	34
-funsigned-bitfields	34
-funsigned-char	34
-fwritable-strings	34, 180
-traditional	23

D

-D	47, 48, 50
-dD	47
deprecated 属性	13, 20, 39
-dM	47
-dN	47
double	54, 72, 74, 78, 176
dsPIC DSC 特有のオプション	31
dsPIC 特有のオプション	
-mconst-in-code	31
-mconst-in-data	31
-merrata	31
-mlarge-code	31
-mlarge-data	31
-mno-isr-warn	32
-mno-pa	31
-momf=	32
-mpa	31
-mpa=	31
-msmall-code	32
-msmall-data	32
-msmall-scalar	32
-msmart-io	32
-mtext=	32
DWARF	32

E

-E	33, 47, 49, 50, 51
EEDATA	83, 84
EEPROM、データ	83
ELF	7, 32
endian	77

errno	124	-fregmove	44
exit	124	-frename-registers	44
extern	24, 39, 46, 54	-frerun-cse-after-loop	44, 45
F		-frerun-loop-opt	44
-falign-functions	43	-fschedule-insns	44
-falign-labels	43	-fschedule-insns2	44
-falign-loops	43	-fshort-enums	54
-fargument-alias	53	-fsigned-bitfields	34
-fargument-noalias	53	-fsigned-char	34
-fargument-noalias-global	53	FSRn	180
far 属性	13, 20, 61, 62, 66, 110, 177	-fstrength-reduce	44, 45
far データ空間	66	-fstrict-aliasing	42, 43, 45
-fcaller-saves	43	-fsyntax-only	35
-fcall-saved	53	-fthread-jumps	42, 45
-fcall-used	53	-funroll-all-loops	43, 45
-fcse-follow-jumps	43	-funroll-loops	42, 43, 45
-fcse-skip-blocks	43	-funsigned-bitfields	34
-fdata-sections	43	-funsigned-char	34
-fdefer-pop。-fno-defer 参照		-fverbose-asm	54
-fexpensive-optimizations	43	-fvolatile	54
-ffixed	25, 53	-fvolatile-global	54
-fforce-mem	42, 46	-fvolatile-static	54
-ffreestanding	34	-fwritable-strings	34, 180
-ffunction-sections	43	G	
-fgcse	44	-g	41
-fgcse-lm	44	getenv	125
-fgcse-sm	44	H	
-finline-functions	23, 39, 42, 46	-H	48
-finline-limit	46	--heap	71
-finstrument-functions	21, 53	--help	33
-fkeep-inline-functions	23, 46	Hex ファイル	58
-fkeep-static-consts	46	I	
float	14, 54, 72, 74, 78	-I	48, 50, 55
-fmove-all-movables	44	-I-	48, 50
-fno	46, 53	-idirafter	48
-fno-asm	34	IEEE 754	176
-fno-builtin	34	-imacros	48, 50
-fno-defer-pop	44	imag	26
-fno-function-cse	46	-include	48, 50
-fno-ident	54	Inline Functions	23
-fno-inline	47	int	14, 72, 74, 77
-fno-keep-static-consts	46	interrupt 属性	21, 22, 89, 100, 178
-fno-peephole	44	-iprefix	48
-fno-peephole2	44	IRQ	90
-fno-short-double	54	ISR	
-fno-show-column	47	記述	88
-fno-signed-bitfields	34	記述するためのガイドライン	88
-fno-unsigned-bitfields	34	記述するための構文	88
-fno-verbose-asm	54	コーディング	89
-fomit-frame-pointer	42, 47	宣言	84
-foptimize-register-move	44	ISR のコーディング	89
-foptimize-sibling-calls	47	ISR を記述するためのガイドライン	88
format_arg 属性	21	ISR を記述するための構文	88
format 属性	20	-isystem	48, 52
-fpack-struct	54	-iwithprefix	48
-fpcc-struct-return	54	-iwithprefixbefore	49
-freduce-all-givs	44		

L

-L	51, 52
-l	51
little endian	77
LL、接尾辞	26
long	14, 72, 74, 77
long double	14, 54, 72, 74, 78
long long	14, 39, 74, 77, 176
long long int	26

M

-M	49
Mabonga	67, 178
MATH_DATA	180
-mconst-in-code	31, 61, 62, 63, 65
-mconst-in-data	31, 65
-MD	49
-merrata	31
-MF	49
-MG	49
-mlarge-code	31, 65
-mlarge-data	31, 61, 62, 65
-MM	49
-MMD	49
-mno-isr-warn	32
-mno-pa	31
mode 属性	14
-momf=	32
-MP	49
-mpa	31
-mpa=	31
MPLAB C18 と MPLAB C30	175
MPLAB C18、MPLAB C30 との違い	175
MPLAB C30	7, 9
ANSI C との違い	11
MPLAB C18 との違い	175
コマンドライン	29
MPLAB C30 と ANSI C	11
-MQ	49
-msmall-code	32, 65, 66
-msmall-data	32, 61, 62, 65, 66
-msmall-scalar	32, 65
-msmart-io	32
-MT	50
-mtext=	32

N

near コードと far コード	66
near 属性	14, 21, 61, 62, 66, 110, 177
near データ空間	111
near データセクション	65
near データと far データ	65
no_instrument_function 属性	21, 54
-nodefaultlibs	51
noload 属性	14, 21
noreturn 属性	22, 39
-nostdinc	48, 50
-nostdlib	51

O

-O	41, 42
-o	33, 58
-O0	42
-O1	42
-O2	42, 46
-O3	42
-Os	42

P

-P	50
packed 属性	15, 54
PATH	57
PC	180
-pedantic	35, 39
-pedantic-errors	35
persistent 属性	15
Persistent データ	63, 83, 180
PIC30_C_INCLUDE_PATH	55, 57
PIC30_COMPILER_PATH	55
PIC30_EXEC_PREFIX	52, 55
PIC30_LIBRARY_PATH	55
PIC30_OMF	55
pic30-gcc	29
pointer	72, 74
PROD	180
PSV ウィンドウ	60, 61, 65, 75, 79, 84
PSV の使用方法	75, 84

Q

-Q	41
----	----

R

RAW 依存	44
RCOUNT	180
real	26
reverse 属性	15

S

-S	33, 51
-s	51
-save-temps	41
section 属性	15, 22, 61, 67, 178
SFR	9, 57, 60, 79, 80, 81
sfr 属性	16
SFR の使用	81
shadow 属性	22, 89, 178
short	72, 74, 77
short long	176
signed char	77
signed int	77
signed long	77
signed long long	77
signed short	77
space 属性	16, 177, 178
-specs=	52
SPLIM	68
SR	180
STATUS	180

strerror	125	-Wlong-long	39
switch	36	-Wmain	35
T		-Wmissing-braces	35
-T	80	-Wmissing-declarations	39
TABLAT	180	-Wmissing-format- 属性	39
TBLPTR	180	-Wmissing-noreturn	39
TBLRD	85	-Wmissing-prototypes	39
TMPDIR	55	-Wmultichar	36
tmpfile	124	-Wnested-externs	39
-traditional	23, 34	-Wno-	35
transparent_union 属性	17	-Wno-deprecated-declarations	39
-trigraphs	50	-Wno-div-by-zero	35
typeof	26	-Wno-long-long	39
U		-Wno-multichar	36
-U	47, 48, 50	-Wno-sign-compare	38, 40
-u	51	-Wpadded	39
ULL、接尾辞	26	-Wparentheses	36
-undef	50	-Wpointer-arith	40
unordered 属性	17	-Wredundant-decls	40
unsigned char	77	WREG	180
unsigned int	77	-Wreturn-type	36
unsigned long	77	-Wsequence-point	36
unsigned long long	77	-Wshadow	40
unsigned long long int	26	-Wsign-compare	40
unsigned short	77	-Wstrict-prototypes	40
unused 属性	17, 22, 37	-Wswitch	36
V		-Wsystem-headers	37
-v	33	-Wtraditional	40
void	74	-Wtrigraphs	37
volatile	54	-Wundef	40
W		-Wuninitialized	37
-W	35, 37, 38, 40, 127	-Wunknown-pragmas	37
-w	35	-Wunreachable-code	40
W14	69, 180	-Wunused	37, 38
W15	69, 180	-Wunused-function	37
-Wa	50	-Wunused-label	37
-Waggregate-return	38	-Wunused-parameter	37
-Wall	35, 37, 38, 40	-Wunused-value	37
-Wbad-function-cast	38	-Wunused-variable	37
-Wcast-align	39	-Wwrite-strings	40
-Wcast-qual	39	WWW アドレス	5
-Wchar-subscripts	35	W レジスタ	72, 107
-Wcomment	35	X	
-Wconversion	39	-x	33
-Wdiv-by-zero	35	-Xlinker	51
weak 属性	17, 22	あ	
-Werror	39	アーカイバ	8
-Werror-implicit-function-declaration	35	アクセス メモリ	178
-Wformat	20, 35, 39	アセンブラ	8
-Wimplicit	35	アセンブラ オプション	50
-Wimplicit-function-declaration	35	-Wa	50
-Wimplicit-int	35	アセンブリ言語と C 変数と関数の混用	107
-Winline	23, 39	アセンブリ、C との混用	107
-Wl	51	アセンブリ、インライン	109, 178
-Wlarger-than-	39	値としてのラベル	27
		アドレス空間	59

アンダースコア	88, 107
い	
インクルードファイル	52
インターネット アドレス	5
インライン	23, 39, 42, 46, 47, 54, 109, 178
インラインアセンブリ言語を使用する	109
インラインアセンブリ使用	83
え	
エスケープ文字列	117
エラー制御オプション	
-pedantic-errors	35
-Werror	39
-Werror-implicit-function-declaration	35
お	
オブジェクトファイル	7, 8, 43, 49, 51, 57, 61
オブジェクトモジュールフォーマット	180
オプション	
C 方言制御	34
dsPIC DSC 特有	31
アセンブラ	50
コード生成規則	53
最適化の制御	42
出力制御	33
ディレクトリ検索	52
デバッグ	41
プリプロセッサ制御	47
リンク	51
ワーニングとエラー制御	35
か	
開発ツール	7
外部シンボル	107
拡張子	49
カスタマ サポート	6
過程抽象	180
環境	116
環境変数	55
PIC30_C_INCLUDE_PATH	55
PIC30_COMPILER_PATH	55
PIC30_EXEC_PREFIX	55
PIC30_LIBRARY_PATH	55
PIC30_OMF	55
TMPDIR	55
関数	
コール規則	72
コール、レジスタの保存	74
属性	19
パラメータ	72
ポインタ	65
き	
キーワードの違い	11
記号	51
共通部分式	46
共通部分式除去	20, 43, 44, 45

く

グローバルレジスタ変数	24
グローバルレジスタ変数の定義	24

け

結合	120
----------	-----

こ

構造体	72, 120
構造体、匿名	178
高プライオリティ割り込み	87, 102
構文をチェック	35
コードサイズ、節約	31, 42
コードサイズの節約	31, 42
コード生成規則オプション	53
コード生成変換用オプション	
-fargument-alias	53
-fargument-noalias	53
-fargument-noalias-global	53
-fcall-saved	53
-fcall-used	53
-ffixed	53
-finstrument-functions	53
-fno-ident	54
-fno-short-double	54
-fno-verbose-asm	54
-fpack-struct	54
-fpcc-struct-return	54
-fshort-enums	54
-fverbose-asm	54
-fvolatile	54
-fvolatile-global	54
-fvolatile-static	54
コードとデータ セクション	61
コードとデータの配置	67
コール規則	180
顧客への通知サービス	5
コマンドライン オプション	30
コマンドライン コンパイラ	29
コマンドライン シミュレータ	7, 8, 9
コメント	35, 47
コンパイラ	8
概要	7
コマンドライン	29
ドライバ	8, 9, 29, 52, 57
コンパイラで管理されるリソース	180
コンフィギュレーションビット設定	83

さ

最適化	9, 180
最適化、ピープホール	44
最適化、ループ	20, 44

最適化の制御オプション	42	修飾子	120
-falign-functions	43	出力制御オプション	33
-falign-labels	43	-c	33
-falign-loops	43	-E	33
-fcaller-saves	43	--help	33
-fcse-follow-jumps	43	-o	33
-fcse-skip-blocks	43	-S	33
-fdata-sections	43	-v	33
-fexpensive-optimizations	43	-x	33
-fforce-mem	46	条件文	28
-ffunction-sections	43	使用されていない関数パラメータ	37
-fgcse	44	使用されていない変数	37
-fgcse-lm	44	省略オペランド	28
-fgcse-sm	44	省略されたオペランドを持った条件文	28
-finline-functions	46	初期化された変数	61
-finline-limit	46	初期化されない変数	62
-fkeep-inline-functions	46	信号	123
-fkeep-static-consts	46	す	
-fmove-all-movables	44	スカラ	65
-fno-defer-pop	44	スケジュール	44
-fno-function-cse	46	スタートアップ	
-fno-inline	47	コード	180
-fno-peephole	44	初期化	63
-fno-peephole2	44	モジュール	69
-fomit-frame-pointer	47	モジュール、初期	63
-foptimize-register-move	44	モジュール、代替	63
-foptimize-sibling-calls	47	スタック	60, 100
-freduce-all-givs	44	C 使用方法	69
-fregmove	44	使用方法	176
-frename-registers	44	ソフトウェア	68, 69
-frerun-cse-after-loop	44	ポインタ リミット レジスタ (SPLIM)	63, 68
-frerun-loop-opt	44	ポインタ (W15)	53, 63, 68, 69
-fschedule-insns	44	ステートメント	120
-fschedule-insns2	44	ストリーム	123
-fstrength-reduce	44	ストレージクラス	176
-fstRICT-aliasing	45	ストレージ修飾子	177
-fthread-jumps	45	スモール コード モデル	9, 32, 78
-funroll-all-loops	45	スモール データ モデル	9, 32, 61, 62
-funroll-loops	45	せ	
-O	42	整数	77, 110
-O0	42	拡張	177
-O1	42	タイプ、複素	25
-O2	42	動作	118
-O3	42	倍長語	26
-Os	42	静的	54
三重字	37, 50	整列	13, 15, 72, 120
し		セクション	43, 61, 180
識別子	117	セクション、コードとデータ	61
システム	125	接頭辞	48, 52
システム ヘッダ ファイル	37, 49	接尾辞 LL	26
事前定義されたマクロ名	177	接尾辞 ULL	26
事前定義定数	56, 183	宣言子	120
実行可能な	57	そ	
実行時環境	59	属性	12, 19, 178
実装時定義動作	115, 180		
指定レジスタ内の変数	24		
シミュレータ、コマンドライン	7, 8, 9		

属性、関数	19	データメモリ割り当て	83
address	19	データの表現	77
alias	19	手続きの抽象化	31
const	20	デバイスサポートファイル	79
deprecated	20	デバイス特有のヘッダファイル	57
far	20	デバッグオプション	41
format	20	-g	41
format_arg	21	-Q	41
interrupt	21, 89, 100	-save-temps	41
near	21	デバッグ情報	41
no_instrument_function	21, 54	伝統的なC	40
noload	21	と	
noreturn	22, 39	ドキュメント	
section	22, 67	凡例	3
shadow	22, 89	レイアウト	2
unused	22	特殊関数レジスタ	57, 79
weak	22	特殊レジスタ	100
属性、変数	12	特徴	9
address	12	匿名構造体	178
aligned	13	は	
deprecated	13	倍長整数	26
far	13, 61, 62, 66	バイナリ Radix	28
mode	14	配列およびポインタ	119
near	14, 61, 62, 66	パラメータ、関数	72
noload	14	汎用レジスタ	110
packed	15	ひ	
persistent	15	ヒープ	60
reverse	15	ピープホール最適化	44
section	15	ヒープ、Cでの使用方法	71
sfr	16	ビットフィールド	34, 103, 120, 181
space	16	ビット反転とモジュロアドレッシング	75
transparent_union	17	標準I/O関数	9
unordered	17	ふ	
unused	17	ファイル	123
weak	17	ファイル拡張子	30
ソフトウェアスタック	22, 68, 69	ファイル名規則	30
た		複数の関数コールにまたがるレジスタの保存	74
タイプ変換	39	複数ファイルのコンパイル	58
て		複素	25
定数		数	25
事前定義	56, 183	整数タイプ	25
バイナリ	28	データタイプ	25
文字列	177	浮動小数点タイプ	25
低プライオリティ割り込み	87, 102	浮動	78
ディレクトリ	48, 49, 50, 57	浮動小数点	78, 118
ディレクトリ検索オプション	52	浮動小数点タイプ、複素	25
-B	52, 55	フラグ、正と負	46, 53
-specs=	52	プラグマ	178
データタイプ	14, 77	プリプロセッサ	52
整数	77	プリプロセッサ指令	121
複素	25		
浮動小数点	78		
ポインタ	78		
データフォーマット	176		
データメモリ空間	31, 32, 60, 71		
データメモリ空間、Near	14		

プリプロセッサ制御オプション	47	マクロ	24, 47, 48, 50, 83
-A	47	ISR 宣言	84
-C	47	インラインアセンブリ使用	83
-D	47	構成ビット設定	83
-dD	47	マクロ データ メモリ割り当て	83
-dM	47	マクロの使用	83
-dN	47	マクロ名、事前定義された	177
-fno-show-column	47	め	
-H	48	メモリ	124
-I	48	メモリ、アクセス	178
-I-	48	メモリ モデル	9, 65, 179
-idirafter	48	-mconst-in-code	65
-imacros	48	-mconst-in-data	65
-include	48	-mlarge-code	65
-iprefix	48	-mlarge-data	65
-isystem	48	-msmall-code	65
-iwithprefix	48	-msmall-data	65
-iwithprefixbefore	49	-msmall-scalar	65
-M	49	メモリ空間	64
-MD	49	も	
-MF	49	文字	117
-MG	49	文字列	34
-MM	49	文字列定数	177
-MMD	49	戻り値	74
-MQ	49	戻り値の型	36
-MT	50	ゆ	
-nostdinc	50	ユーザー定義のデータ セクション	67
-P	50	ユーザー定義のテキストセクション	67
-trigraphs	50	ら	
-U	50	ラージコード モデル	31, 78
-undef	50	ラージデータ モデル	31, 61, 62
フレーム ポインタ (W14)	47, 53, 69	ライブラリ	51, 57
プログラム メモリ ポインタ	65	ANSI 標準	9
プログラム メモリ空間	60	関数	122
プログラム空間可視化ウィンドウ。「PSV ウィンドウ」参照		ライブラリアン	8
プロセッサ ヘッダ ファイル	57, 79, 81	り	
文献、推奨	4	リセット	90, 100, 101
文の差異	27	リセット ベクタ	60
へ		リンカ	8, 51
ベクタ、リセットと例外	60	リンカ スクリプト	57, 68, 80, 81
ヘッダ ファイル	30, 48, 49, 50, 55	リンクオプション	51
プロセッサ	57, 79, 81	-L	51, 52
変換	116	-l	51
変数属性	12	-nodfaultlibs	51
ほ		-s	51
ポインタ	40, 78, 176	-u	51
関数	65	-Wl	51
スタック	53	-Xlinker	51
フレーム	47, 53	リンク用オプション	
ま		-nostdlib	51
マイクロチップ社のインターネット		る	
ウェブサイト	5	ループ解除	45
		ループ最適化	44
		ループの最適化	20

れ

例外ベクタ	60, 90
レイテンシ	100
レジスタ	24, 25
規則	74
定義ファイル	80
動作	119
列挙	120

ろ

ローカル レジスタ変数	24, 25
ローカル変数用のレジスタ指定	25

わ

ワーニング、禁止	35
ワーニング禁止	35
ワーニングとエラーの制御オプション	35
-fsyntax-only	35
-pedantic	35
-pedantic-errors	35
-W	38
-w	35
-Waggregate-return	38
-Wall	35
-Wbad-function-cast	38
-Wcast-align	39
-Wcast-qual	39
-Wchar-subscripts	35
-Wcomment	35
-Wconversion	39
-Wdiv-by-zero	35
-Werror	39
-Werror-implicit-function-declaration	35
-Wformat	35
-Wimplicit	35
-Wimplicit-function-declaration	35
-Wimplicit-int	35
-Winline	39
-Wlarger-than-	39
-Wlong-long	39
-Wmain	35
-Wmissing-braces	35
-Wmissing-declarations	39
-Wmissing-format-attribute	39
-Wmissing-noreturn	39
-Wmissing-prototypes	39
-Wmultichar	36
-Wnested-externs	39
-Wno-long-long	39
-Wno-multichar	36
-Wno-sign-compare	40
-Wpadded	39
-Wparentheses	36
-Wpointer-arith	40
-Wredundant-decls	40
-Wreturn-type	36
-Wsequence-point	36
-Wshadow	40

-Wsign-compare	40
-Wstrict-prototypes	40
-Wswitch	36
-Wsystem-headers	37
-Wtraditional	40
-Wtrigraphs	37
-Wundef	40
-Wuninitialized	37
-Wunknown-pragmas	37
-Wunreachable-code	40
-Wunused	37
-Wunused-function	37
-Wunused-label	37
-Wunused-parameter	37
-Wunused-value	37
-Wunused-variable	37
-Wwrite-strings	40

割り込み

関数	107
高プライオリティ	87, 102
サービスルーチンのコンテキスト保存	100
低プライオリティ	87, 102
ネスティング	100
ハンドリング	107
プライオリティ	100
ベクタ	90
ベクタ、記述	90
保護	104
有効化 / 無効化	101
要求	90
レイテンシ	100
割り込みサービスルーチンを記述する	88
割り込みのネスティング	100
割り込みの有効化 / 無効化	101
割り込みベクタを記述する	90

世界各国での販売およびサービス

北米

本社

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
テクニカルサポート:
http://support.microchip.com
ウェブアドレス:
www.microchip.com

アトランタ

Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

ボストン

Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

シカゴ

Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

ダラス

Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

デトロイト

Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

ココモ

Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

ロサンゼルス

Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

サンタクララ

Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

トロント

Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

アジア / 太平洋

アジア太平洋支社

Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

オーストラリア - シドニー

Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

中国 - 北京

Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 - 成都

Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 - 福州

Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

中国 - 香港 SAR

Tel: 852-2401-1200
Fax: 852-2401-3431

中国 - 青島

Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 - 上海

Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 - 瀋陽

Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 - 深川

Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 - 順徳

Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

中国 - 武漢

Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 - 西安

Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

アジア / 太平洋

インド - バンガロール

Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

インド - ニューデリー

Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

インド - プネ

Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 - 横浜

Tel: 81-45-471-6166
Fax: 81-45-471-6122

韓国 - 亀尾

Tel: 82-54-473-4301
Fax: 82-54-473-4302

韓国 - ソウル

Tel: 82-2-554-7200
Fax: 82-2-558-5932 または
82-2-558-5934

マレーシア - ペナン

Tel: 60-4-646-8870
Fax: 60-4-646-5086

フィリピン - マニラ

Tel: 63-2-634-9065
Fax: 63-2-634-9069

シンガポール

Tel: 65-6334-8870
Fax: 65-6334-8850

台湾 - 新竹

Tel: 886-3-572-9526
Fax: 886-3-572-6459

台湾 - 高雄

Tel: 886-7-536-4818
Fax: 886-7-536-4803

台湾 - 台北

Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

タイ - バンコク

Tel: 66-2-694-1351
Fax: 66-2-694-1350

ヨーロッパ

オーストリア - ヴェルス

Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

デンマーク - コペンハーゲン

Tel: 45-4450-2828
Fax: 45-4485-2829

フランス - パリ

Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

ドイツ - ミュンヘン

Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

イタリア - ミラノ

Tel: 39-0331-742611
Fax: 39-0331-466781

オランダ - ドリユーン

Tel: 31-416-690399
Fax: 31-416-690340

スペイン - マドリッド

Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 - ウォーキングガム

Tel: 44-118-921-5869
Fax: 44-118-921-5820