SN SYSTEMS
Sony Computer Entertainment Group

ユーザー ガイド

SNC PPU C/C++ Compiler

SN Systems Limited
バージョン 310.1
2009年11月12日

目次

1: SNC PPU C/C++コンパイラの基礎

ドキュメント変更履歴

Ver.	日付	変更
310.1	2009年11月	Bz72524: 「-Xdebugvtbl」を追加。 Bz73443: 「-Xignoreeh」を追加。 Bz73531: 「-Xswbr」「-Xswmaxchain」を追加。 Bz75846: 「コンパイラのドライバ オプション」(-M1、-MMD)を更新。 Bz77554: 「C++言語定義」を更新。 Bz78710: 「コントロール変数リファレンス」を更新。 Bz79132: 「-Xnosyswarn」を更新。 Bz79339: 「-Xnewalign」を更新。 Bz79992: 「__unaligned キーワード」を追加。
300.1	2009年9月	「-Xsaverestorefuncs」を追加。 Bz71611: 「-Werror」を追加。 Bz71617: 「-Xtrigraphs」を追加。 Bz71997: 「診断プラグマ」を更新。 Bz72120: 制御変数デフォルト値を修正。. Bz73153: 「-Xpreprocess」を更新。 Bz73254: 「ライブラリ検索」を追加。 Bz75225: 「-Xfastlibc」及び「-Xhostarch」を追加。 Bz76450: 「適切なポインタ アラインメントの前提」を追加。 Bz76452: プラグマにおける情報を改善。 Bz77007: 「一般的な定義済みシンボル」及び「GNU モード シンボル」を更新。 Bz77971: 「C言語定義」及び「-Xc」を更新。
280.1	2009年6月	「-Xgnuversion」及び「-Xuseatexit」を追加。 Bz57204: 複数ファイルにまたがるプロシージャ間最適化へのリファレンスを削除。 Bz68731:「diaglimit: 診断メッセージの制限数」及び「-Xdiaglimit」を追加。 Bz72162:「-Xnewalign」を追加。 Bz72460:「-Xreorder」を追加。 Bz71088: -Xfltedge、-Xfltfold と -Xintedge を修復。 Bz71260:「ビット フィールド実装制御」を追加。 Bz71903:「-Xalignfunctions」を追加。
270.1	2009年3月	「コントロール変数のテスト」を追加。「callmod」を削除。「-Xc」を修正(rtti デフォルトをオンに修正)。「reg: レジスタの割り当て」及び「-Xreg」を変更。 「-Xbranchless」、「-Xunrollssa」及び「-Xuseintcmp」を追加。 「デバッグオプション」(-gfull スイッチ) を追加。 Bz55741/57204: 細部を修正。 Bz56064: 「C言語定義」を更新。 Bz57204: 「コントロール グループ O の最適化」を更新。 Bz66946: 「Microsoft __fastcall と __stdcall 拡張」を追加。 Bz67268: 「定義済みのマクロの使用」及び「定義済みのマクロ リファレンス」を追加。

		Bz68733: 「-Xpredefinedmacros」を追加。 Bz68751: その他細部を多数修正。 Bz68733: 「-Xcallprof」を追加。 Bz69277: Xalnref、-Xcih、-Xfcm、-Xflttp、-Xjoin、 -Xmopt、-Xxopt、-Xxref を削除。
250.3	2008年11月	-Xc=c99 の定義を修正。
250.1	2008年10月	「PCH ファイルの同一ディレクトリ チェックの変更」、「-Xnoident」、「-Xnotocrestore」、「-Xpch_override」、「-Xreserve」及び「-Xzeroinit」を追加。
240.1	2008年6月	大部分の修正 。'アセンブラオプション' (アセンブラコマンドライン構文にパラメータを指定せず ps3ppuas を起動)を削除。2番目のデフォルトに関する文書を削除 (現在でも動作するが、その使用は非推奨)。新しい章「最適化の手法」を追加。「-Xassumecorrectalignment」、「-Xassumecorrectsign」及び「-Xinlinehotfactor」を追加。
220.1	2008年3月	「-Xasmreg」、「-Xautovecreg」、「-Xcf」、「-Xdepmode」、「-Xfastmath」、「-Xforcevtbl」、「-Xlinkoncesafe」、「-Xmemlimit」、「-Xnosyswarn」、「-Xoveralign」及び「-Xpreprocess」を追加。内容を更新し、タイトルを変えた「診断プラグマ」を追加。

概要

このマニュアルは、Windows XP環境でのSNCコンパイラの使用法について説明します。

このマニュアルに記載されている情報

- プログラムをコンパイル、アセンブル、リンクする方法。
- コンパイル時にコンパイラの動作を制御する方法。
- コンパイラが実行する最適化の種類。
- SNC コンパイラで使用可能な言語と、それらの言語の業界標準の定義との比較。
- SNC コンパイラ使用時のプログラム上の制限、およびそれらの制限に反するプログラムの処理方法。
- SNC コンパイラ特有のターゲット別固有機能の使用方法。

このマニュアルに記載されていない情報

- 一般的なプログラム記述方法。
- プログラムのコンパイルや実行に間接的に関与する、プログラミングの各種アスペクト (以下)。
  - コンパイラに入力されるプログラム ファイルの準備
  - コンパイルのプロセスを自動化するツールの使用方法
  - デバッガーの使用方法
  - パフォーマンス モニタリング機能の使用方法
  - プログラムの実行によって出力されるファイルの処理方法

SNCコンパイラをすぐに使い始める方法については10ページの「SNCコンパイラのクイックスタートガイド」を参照してください。

このマニュアル全体を通し、CおよびC++を「C/C++」として集合的に扱っています。それぞれの言語に特有な場合は、どの言語が対象であるか示してあります。上記いずれかの言語に「SN」が付加されている場合は、該当のSNCコンパイラを指すことを意味します。

下表は、このマニュアル全体を通して使用される用語の定義です。

用語	定義
コントロール変数	プログラミング言語における変数の概念に似ているが、コンパイル中にだけ存在し、コンパイラの動作を制御するために使用されるものを言う。「コントロール変数」という用語は、その他の一般的な使用、たとえば「ループ制御変数. . .」などに使われる制御変数とは異なることに注意。
ホストコンピュータ	コンパイラが実行される特定の種類とモデルのコンピュータ。(必ずではないが)通常は*ターゲットコンピュータ*と同じである。
組み込み関数	関数が供給されなくてもユーザーがその関数をコールできるように、使用中のソース言語の定義の一部としてその効果が定義されている*関数*。
メイン関数	プログラム実行の開始ポイントとなるようにコーディングされている関数。C/C++では、main()という名前の関数のこと。
プログラム	一緒に実行されるように編成された*関数*の集まり。各関数には*メイン関数*が1つだけあり、いくつかの非メイン関数を持つこともある。
プログラム障害	コンパイル済みプログラムの動作で、誤った解答を生成するもの、または実行を正常終了できないもの。プログラムの成功、失敗の判断基準の正しさの概念は、標準の言語定義、あるいは他のコンパイラでコンパイルされた場合はプログラムの動作に基づく。
関数	サブルーチン、またはプロシージャ。「関数」という用語には、値を返す関数と返さない関数、単一のまたは複数のエントリポイントを持つ関数、ユーザーによって書かれたものや、組み込み*関数*も含まれる。 C/C++では、ユーザーによって書かれた関数またはライブラリ関数が*関数*である。プリプロセッサマクロは関数ではない。
ターゲットコンピュータ	コンパイラがコンパイル済みコードを生成する対象であり、その動作環境となる特定のモデルのコンピュータ。

SNCコンパイラのクイックスタートガイド

このセクションは、SNCコンパイラ使い始める方法を簡単に説明します。

オプションの指定

SNCコンパイラは、オプションの指定方法については従来のUNIX方式にできる限り従います。確立された従来方式がない場合には、SNCコンパイラに固有のオプションが使用されます。

- SNCコンパイラのオプションは、UNIXスタイルの '-' 接頭部またはWindowsスタイルの '/' 接頭部を使って指定できます。

SNCコンパイラが受け入れる一般的なUNIXコンパイラオプションは、-c、-g、-l、-o、-w、-A、-C、-D、-E、-H、-I、-L、-O、-S、および-Uです。詳細は、15ページの「コンパイラのドライバ オプション」を参照してください。

コンパイラを詳細に制御することができる非従来型のオプションについては、15ページにある「コマンドライン書式 」表の-Xオプション、および 73 ページの「コントロール変数リファレンス」に記載されているコントロール変数の表を参照してください。

最適化制御

最適化制御には-O*n*オプションを使用し、*n*は0～3です。デフォルト設定は -O0 で、これは最適化なし、およびインライン化なしを意味します(強制インライン化を除く)。-O だけを指定した場合、これは -O2 と同じことを意味し、完全な最適化とインライン化が行われます。レベル -O3 は現在 -O2 と同じですが、将来的にはさらなる最適化が追加される予定です。デバッグ可能な最適化済みコードをさらに生成するには -Od を指定しますが、最適化済みコードのデバッグにはいくつかの課題があることに注意が必要です。

注:最善の結果を得るには、プロシージャ間最適化 (-Xipa) などの他オプションを有効化する前に、最適化パフォーマンスのベースを得るため、最適化スイッチ -O (または -O2) を使用してプログラムをコンパイルすることをお勧めします。この簡単なアプローチにより、一般に、最小限の努力で優れたパフォーマンスが得られます。

C/C++言語サポート

SNC-Cは、従来型のCとANSI Cとの相違に対処するため数種類のモードを提供します。デフォルトのモードは、いくらか要件が緩和されたANSI準拠です。その他のモードは従来型Cのサポートを提供します (52 ページの「言語の定義」を参照)。

SNC-C++は、cfront類似C++と、ARM (*The Annotated C++ Reference Manual*) またはANSI C++との相違に対処するため数種類のモードを提供します。デフォルトのモードは、いくつか拡張が追加されたANSI準拠です。その他のモードはcfront類似C++のサポートを提供します (52 ページの「言語の定義」を参照)。

SNCコンパイラと最適化を行うプリプロセッサはそれぞれ、完全最適化を適用するために、(ANSI/ISO規格で指定される通り)プログラミングの使用に対する特定の制約が満たされることを要求します。プログラムの中には、これらのANSI制約の潜伏違反を含んでいるものがあります。そのようなプログラムは、高度な最適化が適用されると動作しないことがあります。この場合は、最良のプログラムパフォーマンスが得られるように、エラーに対して系統だった修正や対応策を講じる必要があります。

SNCコンパイラは標準のCコールシーケンスを使用するので、他のコンパイラによってコンパイルされたモジュールを混合してリンクすることができます。

組み込み関数とインライン アセンブリ

我々は低レベルコードへの組み込み関数ベースのアプローチを好むため、このコンパイラリリースはGNU形式のインラインアセンブリのサポートは提供しません。組み込み関数には、コンパイラの最適化機能との高度な統合性があり、C/C++ とインライン アセンブリとの混合アプローチより優れたコードの作成が可能になっています。用意されている組み込み関数では実装が不可能と思われるコードがある場合は、当社までご連絡ください。

コンパイラによる更なる干渉を一切せずに直接アセンブラに渡す機能である、'raw' asmをサポートします。

組み込み関数に関する詳細は、97ページの「組み込み関数リファレンス」を参照してください。

コンパイルシステム

コンパイルシステムは、たとえば次のように、プログラムの実行をいろいろな方法で準備するために使用します。

1. UNIXスタイルのユーティリティ*make*を使ってさまざまなビルドツールを起動し、実行可能プログラムを生成する。
2. Visual Studio .NETのSNC統合を使ってプロジェクトを管理した後、必要なビルドツールを起動して実行可能プログラムを生成する。

SNCコンパイラは、ソースプログラムのコンパイルと実行に使われるいくつかのコンポーネントを持つコンパイルシステムの一部です。このツールの集まりを*ビルドツール*と呼びます。

ビルドツールのコンポーネントは以下の通りです。

SNコンパイラドライバ	このプログラムはコマンドラインパラメータを受け取り、それらをチェックし、関連のビルドツールコンポーネントを実行する。
SNC C/C++コンパイラ	このプログラムはC/C++ソースファイルを受け取り、それらをコンパイルして、コンパイル対象のプログラムをシンボリックマシン語形式で表現したアセンブリファイルを生成する。このコンパイラには、C/C++前処理言語のプリプロセッサが含まれている。SNCコンパイラには2つの形式がある。
SNアセンブラ	このプログラムはアセンブリファイルを受け取り、それらをアセンブルしてオブジェクトファイルを生成する。オブジェクトファイルは、コンパイル済みプログラムをバイナリのマシン言語形式で表現する。
SNリンカ	このプログラムはいくつかのオブジェクトファイルとライブラリファイルを受け取り、実行可能プログラムをターゲット形式（ELF）で生成する。
SNアーカイブライブラリアン（SNARL）	このユーティリティは、オブジェクトファイルのライブラリの作成と管理に使用する。

通常、プログラムは1つまたは複数のソースファイルから構成されます。これらのソースファイルは、C、C++、またはアセンブリ言語で書くことができます。さらに、各ソースファイルには1つまたは複数の関数を含めることができます。

コンパイラのルートコンポーネントは*ドライバ*と呼ばれます。これは、コンパイルシステムを開始するコマンドラインから呼び出され、呼び出された後は単一プロセスとして動作します。ドライバは、渡されたオプションのセットと、渡された各ファイル名の拡張子を見ます。これらのファイルとオプションがドライバの動作を指示し、ドライバの動作によって、システムの残りの部分の動作が呼び出され、また指示されます。一般に、コンパイラのコンポーネントを直接コールするのではなく、ドライバを使用すべきです。

コンパイルシステムのコンポーネントは、一時ファイルを読み書きして相互に連絡します。たとえば、高水準言語のソースファイルをコンパイルするとき、結果のアセンブリ出力は一時ファイルに置かれ、その後これがアセンブラによって処理されます。デフォルトでは、一時ファイルはすべて、システムコンフィグレーションでの指定通りWindowsの一時ディレクトリに置かれます。環境変数TMPDIRをディレクトリのパス名に設定することで、他の希望のディレクトリを代わりに使用することもできます。

コンパイラドライバ使用法のシナリオ

コンパイルシステムの呼び出しに使用するコマンドラインについては、15ページの「コマンドライン書式」で詳しく解説しています。以下の例は、その解説の予備知識となります。

例 1:

最初の例では、ドライバにC、C++、アセンブリのソースファイルを混ぜて渡し、コマンドラインオプションは指定しません（従って、ドライバのデフォルト動作が起動されます）。この状況では、ドライバは次のように動作します。

- 与えられたCソースファイルのそれぞれをC前処理とコンパイル用にコンパイラに渡す。

- 与えられたC++ソースファイルのそれぞれをC++前処理とコンパイル用にコンパイラに渡す。
- 結果のすべてのアセンブリファイルとすべてのアセンブリソースファイルをアセンブリ用にアセンブラに渡す。
- 結果のすべての再配置可能オブジェクトファイルをリンカに渡し、結合された1つの実行可能オブジェクトファイルを生成する。

例 2:

2番目の例では、ドライバに1つのCソースファイルを渡し、コマンドラインオプション-cを指定します(-cオプションは、リンカをコールする前に停止するようドライバに指示します)。この状況では、ドライバは次のように動作します。

- 与えられたCソースファイルをコンパイル用にCコンパイラに渡す。
- 結果のアセンブリファイルをアセンブリ用にアセンブラに渡し、結果の再配置可能オブジェクトファイルをカレントディレクトリに残す。

これは一般に、ソースファイルやそのいずれかの先行ファイルが変更された場合にリコンパイルを起動するため、makefileの中で使用します。

例 3:

3番目の例では、ドライバに一組の再配置可能オブジェクトファイルを渡し、オプションは指定しません。この状況では、ドライバは次のように動作します。

- 与えられたファイルをリンカに渡し、リンカはそれらを結合して1つの実行可能オブジェクトELFファイルを生成する。

これは一般に、再配置可能オブジェクトファイルのいずれかが変更された場合に実行可能オブジェクトファイルを作成するため、makefileの中で使用します。

コンパイラ動作の制御

SNCコンパイラでは、コンパイル中の動作を柔軟に制御できます。たとえば、コンパイラ動作の次のような状態を制御できます。

- 実行可能オブジェクトファイルの生成における進行状況(上記の説明参照)。
- コンパイラによって適用される最適化と、それらが適用される範囲。
- ソースプログラムに使用されたソース言語の規格別表現、および/またはソースプログラムをエンコードするために使用されたファイル形式。
- リスト作成、診断、シンボリックデバッグ情報、またはコンパイラが生成するその他の出力。
- コンパイラに課せられるリソース利用制限。

コンパイラ動作制御の一部の状態については、制御メカニズムの詳細を規定する従来方式が確立されています。これは特に、上述のようなコンパイルシステムとしてのコンパイラの編成と、コンパイルシステムの動作を指示するため使用されるオプション(リンカの呼び出し前に動作を停止させる-cオプションなど)に適用されます。

コンパイラ動作制御のその他の状態については、確立された従来方式はなく、SNCコンパイラ特有のメカニズムが使用されます。

コンパイラ動作の制御は次の2か所で実行できます。

- コマンドラインでのコマンドラインオプションの指定と、コンパイラに渡されたファイルの性質によって。

- ソースファイル内で、言語の拡張を適切に記述することによって。SNCコンパイラでは、この記述にはプラグマディレクティブを用います。

コンパイラ動作の状態の中には、コマンドラインのレベルでのみ制御できるものがあります。これらの状態については、適切なコマンドラインオプションがあります。ただし、コンパイラ動作のほとんどの状態は、状況に応じてコマンドラインかソースファイルのどちらかから適切に制御できます。たとえば、適用する最適化の程度について考えてみます。これは通常、コマンドラインから最も都合よく制御できます。しかし、ある関数の性質によって、その関数に対してある種の最適化を無効にすることが要求される場合は、無効にする指示をその関数で設定する方が便利です。これとはまったく逆の例もあります（つまり、通常はソースファイルの中に制御を設定するのが便利ですが、コマンドラインに制御を置く方が便利な場合もあります）。

この2種類の使用ニーズを満たすため、制御をコマンドラインまたは、ソースファイルの中のどちらからでも、自由に選択して実行することができる制御スキームが使用されます。この基本的な発想は、一組の*コントロール変数*という概念です。

コンパイラ動作の制御可能状態のそれぞれにコントロール変数が1つあり、この変数に現在割り当てられている値がコンパイラの動作を規定します。これらの変数にはコマンドラインで初期値を与えることができ、それらの値は、適切なプラグマディレクティブによりソースファイル内の任意のポイントで変更できます。

詳細は、23ページの「コンパイラの制御」を参照してください。

2: コマンドライン書式

コマンドライン書式

コンパイラドライバのコマンドライン書式は以下の通りです。

```
ps3ppusnc [オプション] [ファイル]
```

ここで、[オプション]はオプションのリストで、[ファイル]はファイル名のリストです。コンパイラドライバ オプションの一覧については、15ページの「コンパイラのドライバ オプション」を参照してください。ファイル名の取り扱いについては、21ページの「ファイル名」を参照してください。

コンパイラのドライバ オプション

コンパイラのデフォルトの動作を、ファイル名リストの前に配置するオプションによって変更することができます。以下のオプションを指定できます。それ以外のオプションを指定すると、それらはコンパイラによって無視され、リンカが呼び出された場合はリンカに渡されます。

下表は、さまざまなコンパイラオプションをその種類によってグループ分けしています。

ヘルプ

オプション	アクション
[なし]	引数なしでプログラム名をタイプすると、主なコンパイラオプションを含め、一般的な使用法情報が出力される。
--help	このオプションは現在利用可能なスイッチをプリントアウトします。

プリコンパイル済みヘッダ

オプション	アクション
--pch	プリコンパイル済みヘッダを自動的に使用および/または作成する(詳細は、59ページの「プリコンパイル済みヘッダ」を参照)。このオプションに続いて、コマンドラインに--use_pchまたは--create_pch(手動プリコンパイル済みヘッダモード)があった場合、その効果は無効になる。
--create_pch=*filename*	他の条件が満たされた場合に、プリコンパイル済みヘッダファイルを、指定された名前で作成する。このオプションに続いて、コマンドラインに--pch(自動プリコンパイル済みヘッダモード)または--use_pchがあった場合、その効果は無効になる。
--use_pch=*filename*	指定された名前のプリコンパイル済みヘッダファイルを現行コンパイルの一部として使用する。このオプションに続いて、コマンドラインに--pch(自動プリコンパイル済みヘッダモード)または--create_pchがあった場合、その効果は無効になる。
--pch_dir=	プリコンパイル済みヘッダを検索および/または作成する対象のディ

directory-name	レクトリ。自動(--pch)または手動(--create_pchまたは--use_pch)プリコンパイル済みヘッダモードで使用できる。
--pch_messages --no_pch_messages	現行コンパイルでプリコンパイル済みヘッダファイルが作成または使用されたことを示すメッセージの表示を有効または無効にする。
--pch_verbose	自動pchモードにおいて、現行コンパイルに使用できないプリコンパイル済みヘッダファイルのそれぞれについて、ファイルが使用不能であることの理由を示すメッセージを表示する。

プロセス制御と出力

オプション	アクション
-c	オブジェクトファイルにコンパイルする。出力ファイルを(-oオプションで)指定した場合、すべての出力がこのファイルに送られる。それ以外の場合、出力ファイルには、新しい拡張子.oの付いた入力ファイル名を用いる。
-C	C/C++プリプロセッサ出力内のコメントを維持する。
-dryrun	コンパイル中に呼び出される各プロセスの名前と引数、およびリンク解除される各一時ファイルの名前を標準エラー出力に出力するが、実際のコンパイルは行わない。
-E	プリプロセスのみ。プリプロセッサ出力を標準出力に出力し、コンパイルを停止する。C/C++プリプロセッサ出力では、デフォルトでコメントはこの結果から削除されるが(ただし、上記の-Cを参照)、行番号情報は含まれる。
-H	インクルードされているファイルのパス名を標準出力に出力し、コンパイルを停止する。前処理されるべきソースファイルはすべてプリプロセッサを通して渡されるが、通常の前処理結果ファイルは生成されない。代わりに、前処理中に含まれていたすべてのファイルのパス名のリストが標準出力に出力される。下記の-Mオプションも参照。
-keeptemp	コンパイル中に作成された一時ファイルを削除しない。
-M	各オブジェクト ファイルの依存を意味するルール (「make」に適切) が出力されます。また依存情報は、stdout に書き込まれます。
-M1	-M オプションと同様ですが、依存情報にはユーザー ヘッダ ファイルのみが含まれ、システム ヘッダ ファイルは含まれません。システム ヘッダ ファイルは、ソース ファイルと同じディレクトリにはありませんが、-I オプションを使用しなくてもインクルードすることが可能です。これは、インクルード ディレクトリにあるヘッダ ファイルは黙示的にコンパイラに認識されることを意味します。このインクルード ディレクトリは、$CELL_SDK内の一連のディレクトリを指します。
-Map <file>	<file>というマップファイルを作成する。
-MD	各オブジェクト ファイルの依存を意味するルール (「make」に適切) が出力されます。依存情報は、入力ファイルと同じ名前のファイルに書き込まれ、拡張子が「.d」になります。
-MMD	-MD オプションと同様ですが、依存情報にはユーザー ヘッダ ファイルのみが含まれ、システム ヘッダ ファイルは含まれません。システム ヘッダ ファイルは、ソース ファイルと同じディレクトリにはありませんが、-I オプションを使用しなくてもインクルードすることが可能です。これは、インクルード ディレクトリ内のヘッダ ファイルは黙示的にコンパイラに認識されることを意味します。このインクルード ディレク

	トリは、$CELL_SDK 内の一連のディレクトリを指します。.
-o <file>, -o<file>	出力ファイル名にデフォルトを使用するのではなく、<file>を指定する。このオプションは、デフォルトルールで生成されるもの以外の出力ファイルを指定することを許可する。リンカの呼び出し前にコンパイルが停止された場合、<file>の拡張子に対していくらか制約が強制される。これは、たとえばソースファイルが誤って上書きされることを防ぐためである。
-P	このオプションはC/C++ソースファイルにのみ適用される。C/C++ソースファイルはすべて単に前処理されるだけで、各ファイルの前処理結果は、ソースファイルのファイル名拡張子を.iで置き換えた名前のファイルに書き込まれる。コメントはデフォルトでこの結果から削除され（上記の-Cを参照）、行番号情報も除外される（上記の-Eと比較）。-C/C++コンパイラは前処理結果に対して呼び出されない。
-S	アセンブラソースにコンパイルする。アセンブラを呼び出す前にコンパイルを停止し、コンパイルで生成されたすべてのアセンブラソースファイルをカレントディレクトリに残す。
-Tc	通常のファイル拡張子.cを持たないソースファイルを、Cソースファイルとして扱うことを指定する。
-Tp	通常のファイル拡張子.cppを持たないソースファイルを、C++ソースファイルとして扱うことを指定する。
-V	標準出力に呼び出された各プロセスのバージョン番号を出力する。
-v -verbose -#	詳細モード - 実行前にすべてのコマンドを表示する。コンパイル中に呼び出された各プロセスの名前と引数を標準エラー出力に出力し、削除された各一時ファイルの名前も出力する。makefileで-#を使用する場合は、#文字をエスケープする必要がある。
-X*cprog*	任意の数のコントロール変数に初期値を割り当てる。ここで、*cprog* はコントロールプログラム。たとえば: -Xdiag=2, inline=joe, unroll=8 これは、コントロール変数diagに初期値2を、コントロール変数inlineに初期値"joe"を、コントロール変数unrollに初期値8を割り当てる。 -Xオプションは繰り返し可能。-Xオプションの完全セット、-XXオプション、および-Xオプションの省略形は左から右の順に処理され、割り当てが重複する場合には右端の割り当てが優先される。（注：-gはこのルールの例外であり、その相対位置に関わらず、最初に処理される。）コントロールグループにはいくつかのコントロール変数への暗黙の割り当てが含まれるため、コントロールグループを使用するときは特別の注意が必要である。 コマンドライン上でこの方法で割り当てられる初期値は、コンパイルシステムによって処理される各ソースファイルの開始点において有効値として確立される。有効値は、各ソースファイルの一部または全部について、そのファイル内に存在するプラグマディレクティブにより変更できる。 コントロールプログラムの詳しい説明については、23ページの「コンパイラの制御」を参照。各コントロール変数の正確な意味については、35ページの「コントロール変数の定義」を参照。また、73 ページの「コントロール変数リファレンス」に記載されている全コントロール変数の表を参照。

-XX*cprog*	任意の数のコントロール変数に変更不能値を割り当てる。ここで、*cprog*はコントロールプログラム。このオプションが上記の-Xと異なるのは、プラグマディレクティブや他のコマンドラインオプションがあるかどうかに関わらず、-*cprog*で割り当てた値が(このコンパイルでは)変わらないという点である。-XXオプションは繰り返し可能で、-Xオプションと混ぜることができる。上記の-Xオプションにある左から右へのルールを参照。
-Yc,<dir>	*c*で指定されるプロセスの場所の新しいパス名<dir>を指定する。ここで、*c*は以下のいずれか1つまたは複数である。 p C/C++プリプロセッサ f フロントエンド i プロシージャ間アナライザ b バックエンド(オプティマイザ/コードジェネレータ) a アセンブラ l (小文字の"L")リンカ S 起動関数が含まれているディレクトリ I デフォルトのインクルードディレクトリ L リンカが最初に検索するデフォルトライブラリ U リンカが2番目に検索するデフォルトライブラリ。 呼び出されることのないようなプロセスの新しいパス名が指定された場合は、次の場合を除きこのパス名は無視される。 SNC-C/C++コマンドラインでは、C/C++プリプロセッサは個別のプロセスとしては実装されない。前処理機能はC/C++フロントエンドに統合される。ただし、-Yp,<dir>オプションが指定された場合、ドライバは、ディレクトリ<dir>にある'cpp'という名前のファイルをプリプロセッサとして使用する。
-##	-#と同様であるが、実際のコンパイルは行わない。makefileでは、#文字をエスケープする必要がある。

C/C++言語オプション

オプション	アクション
-K	CのKernighan & Ritchie (K&R) 規格別表現を受け入れる。これは、-Xc=knrの省略形。
-noex	これは、-Xc-=exceptionsの省略形。

警告オプション

オプション	アクション
-w	すべての警告を無効にする。これは、-Xdiag=0の省略形。これはプリプロセッサからの警告を抑制するが、アセンブラまたはリンカからの警告は抑制しない。
-Werror	すべての警告をエラーとして扱います。警告が発生した場合、ビルドは終了されます。これは -Xquit=1 設定に相当します。(43 ページの「quit: 診断終了レベル」を参照。)
--diag_error=<list>	カンマ区切りのリスト <list> に含まれる診断コードまたはタグ名を、エラーとして発行されるようにします。
--diag_remark=<list>	カンマ区切りのリスト <list> に含まれる診断コードまたはタグ名を、

	注意レベルのメッセージとして発行されるようにします。
--diag_suppress= <list>	カンマ区切りのリスト <list> に含まれる診断コードまたはタグ名を、診断は発行されません。
--diag_warning= <list>	カンマ区切りのリスト <list> に含まれる診断コードまたはタグ名を、警告として発行されるようにします。

診断メッセージを制御する別の方法については、31 ページの「診断プラグマ」を参照してください。

デバッグオプション

オプション	アクション
-g	ソース レベルのデバッグ用のデバッグ情報を生成する。これには、アセンブリ ファイルのシンボル デバッグ情報が含まれます。-g デバッグ オプションでは、プログラム内で使用されるタイプ、変数、関数、名前領域などに関するシンボル デバッグ情報が生成されます。使用されないプログラム エレメントについては、デフォルトでデバッグ情報が一切生成されません (-gfull を参照)。 注：-gオプションは、ProDG Debuggerを使用する場合に必須。
-gfull	-g と同様ですが、すべてのプログラム エレメントに関するシンボル情報が生成されます。

最適化オプション

オプション	アクション
-O*n*	レベル*n*での最適化をオンにする。n には0（ゼロ）から3、もしくは d または s (以下を参照) が入ります。 このオプションは、-XO=nの省略形 (コントロールグループ O の詳細は、96 ページの「最適化グループ (O)」を参照)。最適化の指定がない場合、結果は-O0と同等になる。
-O0 [ゼロ]	最適化なし、およびインライン化なし (強制インライン化は除く)。
-O1	最適化なし、インライン化を許可。
-O2	完全最適化。
-Oまたは-O3	完全最適化。より時間のかかる最適化を有効にします (現在のリリースにおいてはこのカテゴリに属するものはなし)。
-Od	デバッグ可能な最適化コード (スケジューリングなしなど)。
-Os	パフォーマンスとコード サイズの両方が最適化され、-O2 の場合よりもコード サイズへの考慮比重が大きくなります。たとえば、インライン化は -O2 よりも -Os の方が少なくなります。

プリプロセッサオプション

オプション	アクション
-D<name>	プリプロセッサのシンボル<name>を定義する。このオプションはC/C++プリプロセッサを通して渡されるソースファイルにのみ適用さ

	れる。<name>は値1で定義される。-Dオプションは-Uオプションよりも優先度が低い。下記参照。
-D<name>=<def>	プリプロセッサのシンボル<name>を値<def>で定義する。このオプションはC/C++プリプロセッサを通して渡されるソースファイルにのみ適用される。<name>は、対応する#defineステートメントがプログラムの最初の行に出現した場合とまったく同様に、値<def>で定義される。-Dオプションは-Uオプションよりも優先度が低い。下記参照。
-I<dir> -I <dir>	このパスを、インクルードファイルを検索する対象のディレクトリのリストに追加する。
-include <file>	コンパイルの始めに<file>のソースファイルをインクルードする。これは、標準マクロ定義などの確立に使用できる。このファイルは、インクルード検索リストにあるディレクトリ内で検索される。
-nostdinc	「標準」インクルードファイルが含まれるディレクトリに関連してドライバが生成するすべての-Iオプションを抑制する。ユーザー指定の-Iオプションは通常通りコンパイラに渡される。
-nostdinc++	「標準」C++インクルードファイルが含まれるディレクトリに関連してドライバが生成するすべての-Iオプションを抑制する。ただし、C++に関連するファイルだけ。ユーザー指定の-Iオプションは通常通りコンパイラに渡される。-nostdincが指定された場合、このオプションは何もしない。
-U<name>	前処理の前にシンボル<name>を定義解除する。このオプションはC/C++プリプロセッサを通して渡されるソースファイルにのみ適用される。<name>の初期定義はすべて削除される。そのような初期定義は-Dオプションで作成できる、または、特定の環境内で定義済みのシンボルの1つである場合もある。-Uオプションは、コマンドラインでのオプションの順序に関わらず、同じ名前の-Dオプションを上書きする。
-Wf,...	フロントエンドのオプションを指定する。
-Wi,...	プロシージャ間アナライザのオプションを指定する。
-Wb,...	バックエンド(オプティマイザ/コードジェネレータ)のオプションを指定する。

-Iオプションは、#includeステートメントで指定されたファイルを見つけるための検索順序を変更します。#includeステートメントでは、この検索順序は次のようになります。

- 絶対パス名であるファイル名については、指名されたファイルだけを使用する。
- 絶対パス名ではないファイル名で、引用符で囲まれたものについては、以下のディレクトリに関してリスト順に検索する。
  1. コントロール変数inclpathが値`absolute`を持つ場合は、主ソースファイルが含まれているディレクトリ。コントロール変数inclpathが値`relative`を持つ場合は、#includeステートメントを含むファイルが含まれているディレクトリ(これら2つのディレクトリは、ネストされた#includeステートメントの場合にのみ異なる)。
  2. -Iオプションでリストされたディレクトリ。コマンドラインでのオプションの出現順序で。
- 絶対パス名ではないファイル名で、山括弧で囲まれたものについては、以下のディレクトリに関してリスト順序に検索する。
  1. -Iオプションでリストされたディレクトリ。コマンドラインでのオプションの出現順序で。
  2. SN供給のインクルードファイルがインストールされているディレクトリ。

リンカオプション

オプション	アクション
-l<library> [小文字の'L']	リンク時に、指定されたライブラリ<library>を含める。このオプションはリンカに渡された後、リンカに対して<library>という名前のライブラリを検索するよう指示する。動的ライブラリと静的ライブラリのどちらが検索されるかに応じて、さまざまな拡張子が<library>に適用される。-lオプションが他のオプションと異なるのは、ファイル名と混在でき、ファイル名の間での相対位置が重要であるという点である。
-L<dir>	このパスを、ライブラリを検索する対象のディレクトリのリストに追加する。このオプションはリンカに渡された後、リンカに対して、標準ライブラリのディレクトリを検索する前に、<dir>でライブラリを検索するよう指示する。
-nolib-nostdlib	ドライバが生成するようなすべての-lオプションを抑制する。ユーザー指定の-lオプションは通常通りリンカに渡される。
-Wl,...	リンカに渡すオプションを指定する。リンカー コマンドライン構文については、ProDG リンカー ドキュメントを参照。

ファイル名

コンパイラは以下のタイプのファイルを入力として受け入れ、ファイル名の拡張子に応じたアクションを適用します。

ファイルタイプ	拡張子	アクション
Cソース	.C	前処理、コンパイル、アセンブル、リンク
C++ソース	.CC、.CPP、.CXX	前処理、コンパイル、アセンブル、リンク
前処理されたCソース	.I	コンパイル、アセンブル、リンク
コンパイラソースのアセンブラ	.S	前処理、コンパイル、アセンブル、リンク
コンパイラソースのアセンブラ	.SX	前処理、アセンブル、リンク
ユーザーソースのアセンブラ	.ASM	前処理、アセンブル
オブジェクトファイル	.O、.OBJ	リンクのみ

- 特定のファイルタイプを示すものと認識さない拡張子の付いたファイルは、オブジェクトファイルとして処理され、リンカにのみ渡されます。これには、標準のオブジェクトファイル拡張子の.oファイルが含まれます。
- 使用する拡張子のタイプの数に制限はありません。コンパイラは多数のCファイルとC++ファイルを1回の呼び出しでコンパイルし、それぞれに対して正しいコンパイラを適用します。

行われるアクションは、自動リンクを省略する-cオプションなど、制御オプションによっても異なります。

例:

```
ps3ppusnc -c -O2 main.c objects.c pluscode.cpp
```

これは、main.c、objects.c、pluscode.cppを前処理し、コンパイルしてアセンブルし、3つのオブジェクトファイルを生成します。コンパイルには最適化が適用され、デバッグ情報は含まれません。これらのファイルのmain.cとobjects.cはCコンパイラでコンパイルされますが、pluscode.cppはC++コンパイラでコンパイルされます。

これらのファイルはカレントディレクトリにある必要はありません。絶対または相対のパス名を使用できます。

コンパイルシステムのデフォルトの前提は、渡されたファイルは全体として、ユーザが実行準備をしたい1つのプログラムを構成するということです。従って、デフォルト動作として、渡されたすべてのファイルをそのタイプに応じて適切に処理した後、結果を結合して1つの実行可能オブジェクトファイルを生成します。

以下の一定のステップにより、これが達成されます。

1. すべての高水準言語のソースファイルがコンパイルされてアセンブリソースファイルが生成され、これが一時ディレクトリに置かれる。コンパイル前に、必要なすべてのソースファイルが適切なプリプロセッサによって前処理される(これは.iファイルには冗長であるが、無害)。
2. すべてのアセンブリソースファイル(ステップ1で生成されたものか、入力として与えられたもの)がアセンブルされ、カレントディレクトリにオブジェクトファイルが生成される。それぞれの名前は、前のファイル名の拡張子が.oで置き換えられたものとなる。同じ名前のファイルが存在していた場合、それは削除される。
3. すべてのオブジェクトファイル(ステップ2で生成されたものか、入力として与えられたもの)がリンカに渡され、リンカはそれらを結合してリンクし、a.selfという名前の1つの実行可能オブジェクトファイルをカレントディレクトリに生成する。
4. ソースファイルであった1つだけのファイルがコンパイルシステムに渡され、エラーが発生しなかった場合は、そのソースファイルから作成された.oファイルが削除される。メモ: この動作はUnixのそれとは異なります。

コンパイルの制約

個別にコンパイルされた複数のファイルが最終的に一緒にリンクされるようにするため、コマンドラインオプションの使用などにおいて2～3の制約に従う必要があります。これらの制約には、個々のコントロール変数のスコープによって強制されるものがありますが、制約がコンパイルシステムの個別の呼び出しに完全に関連しているためにコンパイラによって強制できないものもあります。

3: コンパイラの制御

コンパイラの制御

このセクションでは、SNCコンパイラで提供される各種の制御について説明します。

コントロール変数

*コントロール変数*の基本的な発想は、変数に値を割り当てることによってコンパイラの動作を制御することです。この章では、コントロール変数の概念について説明します。各コントロール変数の正確な意味については、35ページの「コントロール変数の定義」を参照してください。また、73ページの「コントロール変数リファレンス」に記載されている全コントロール変数の表も参照してください。

コンパイラ動作の状態で制御可能なもののそれぞれについて、コントロール変数が1つあります。コンパイル中は、これらの変数に現在割り当てられている値が、その時点でのコンパイラの動作を管理します。コントロール変数はプログラミング言語における変数と概念的に似ています。特に、コントロール変数には以下の特性があります。

- 各コントロール変数には固有の名前がある。(この名前は大文字小文字を区別し、必ず小文字で構成される。)
- コントロール変数の名前セットは固定である(コントロール変数名の例:diag)。新しいコントロール変数名を作成することはできない。
- 各コントロール変数は、正規に割り当てることのできる一定の値セットが存在するという意味で特定のタイプを持つ。
- 各コントロール変数は、各C/C++ソースファイル全体に渡りどのポイントにおいても明確な割り当て値を持つ。この値は(ソースファイル内でのプラグマディレクティブの出現に応じて)ソースファイル内の異なるポイントで変更できる。
- 各コントロール変数は、その値への変更が有効となるソースファイル範囲を管理する特定のスコープを持つ。
- コントロール変数はコンパイル中にのみ存在し、実行時には存在しない。

ソースファイル内のどのポイントでコントロール変数に割り当てられた値も、以下のルールで確立されます。

- **各**ソースファイルの開始点で、コマンドライン処理によって確立された値がコントロール変数に割り当てられる。-Xと-XXのコマンドラインオプションをこの目的で使用する。
- この値が-XXオプションで確立された場合、ソースファイル全体に渡りこの値は変わらない。それ以外の場合は、ソースファイル中を順次に進み、コントロール変数に値を割り当てるプラグマディレクティブに遭遇した場合は新しい割り当て値が確立され、別の割り当てによって変更されるか、ソースファイルの最後に到達するまでこれが維持される。

コントロール変数のスコープの観念は、ある重要な点において、プログラミング言語の変数のスコープの観念に似ています。つまり、プログラムにおいて変数が有効である範囲を管理します。ただし、コントロール変数のスコープは、この目的をどのように達成するかという点において、プログラミング

言語の変数のスコープと大きく異なります。特に、コントロール変数のスコープの観念には以下の特性があります。

- 各コントロール変数は次の5つのスコープのいずれかを持つ。
  1. コンパイルスコープ
  2. ファイルスコープ
  3. 関数スコープ
  4. ループスコープ
  5. ラインスコープ
- コントロール変数のスコープは、その変数の対応する*スコープポイント*のセットを以下のように決定する。
  1. *コンパイルスコープ*を持つコントロール変数では、コンパイル開始時に、コマンドラインによるコントロール変数の割り当てを処理した後で、かつ、すべてのソースファイルのソーステキストを処理する前にスコープポイントがある。
  2. *ファイルスコープ*を持つコントロール変数では、各ファイルの開始時に、最初の非プラグマ非コメントソース言語テキストのトークンに遭遇したとき（つまり、真のソース言語テキストのこの最初のトークンに先行するプラグマディレクティブを処理した後）にスコープポイントがある。
  3. *関数スコープ*を持つコントロール変数では、各関数の開始時に、新しい関数を定義する最初のテキストのトークンに遭遇したときにスコープポイントがある。
  4. ループスコープを持つコントロール変数では、最初の反復ソース言語ステートメントのトークンに遭遇したときにスコープポイントがある。（C/C++では、for、while、およびdoステートメント。）
  5. *ラインスコープ*を持つコントロール変数では、各ソース行の開始時に、先行行におけるプラグマディレクティブの処理が完了した後にスコープポイントがある。
- コントロール変数のスコープポイントは、コンパイラがそのコントロール変数に現在割り当てられている値を読み取る唯一のポイントであり、コンパイラはこの値を使って（以降の）その動作を管理する。

コントロール変数の割り当てとコントロール変数のスコープに関するこれらのルールには、以下の効果があります。

- コントロール変数に値を割り当てるプラグマディレクティブは、その制御値のスコープに関わらず、プラグマディレクティブを書くことのできるソースファイル内の任意の位置で書くことができる。
- コントロール変数に値を割り当てるプラグマディレクティブは、そのコントロール変数のスコープに関わらず、指定された割り当てを行い、コントロール変数の現行値を確立するという効果を持つ。これには1つ例外があり、コマンドラインで-XXオプションを使ってコントロール変数が確立された場合、この割り当ては無視される。
- コントロール変数に対して確立された現行値は、そのコントロール変数の次のスコープポイントまで、コンパイラの動作には何も影響しない。この次のスコープポイントで、現在確立されている値がコンパイラによって読み取られ、さらに次のスコープポイントに遭遇するまで、コンパイラの動作を管理するために保存される。
- コンパイルスコープを持つコントロール変数は、常に、-XXオプションを使って設定されたかのように動作する。
- プラグマディレクティブによりコントロール変数に割り当てられた値で、そのコントロール変数のファイル中最後のスコープポイントの後に出現するものは、決してコンパイラによって適用されない。

コントロールグループ

さまざまなコントロール変数が豊富にあり、それぞれのコントロール変数がコンパイラ動作の特定の詳細状態を管理します。この方法により、必要な場合には柔軟性が提供されますが、同時に、多くのコントロール変数値を設定するという余計な負担がプログラマに課せられます。この負担を軽減するため、コントロール変数をグループ化する機能が提供されています。詳細は、73ページの「コントロール変数リファレンス」を参照。

コントロールグループには以下の特性があります。

- 各コントロールグループには固有の名前がある。(この名前は大文字小文字を区別し、必ず1つの大文字である。)
- コントロールグループに属するものとして、特定のコントロール変数セットがある。
- 各コントロールグループは、正規に割り当てることのできる一定の値セットが存在するような特定のタイプを持つ。
- コントロールグループは、コントロール変数の場合と同じ方法では値を持たない。あるコントロール変数をあるコントロールグループに割り当てることが、ある特定の値セットをそのグループ内のコントロール変数に割り当てることの省略形として解釈される。

コントロールグループにはデフォルト値がありません。コントロールグループの指定がなく、そのコントロールグループ内に他に割り当てがない場合は、そのコントロール変数のデフォルト値が適用されます。

コントロール式

コントロール変数には、整数、名前、ペア、名前リスト、ペアリストのいずれかのタイプの値を割り当てることができます。これらのタイプの値は、コントロール式を評価して作成されます。コントロール式は以下のように形成します。

- 10進表記で書かれた整数定数は、整数値として使用できる。たとえば、"1"、"47"などの整数値がある。
- 整数式は、+と-の演算子を使って形成できる。たとえば、"1+4+8-2"は"11"となる。括弧は使用できない。また、評価は厳格に左から右へ行われる。たとえば、"5-1+3"と"11-1-3"はどちらも"7"となる。
- 名前値は任意の文字を使って書くことができるが、等号("=")、コンマ(",")、プラス("+")、マイナス("-")、コロン(":")は使用できない。最初の文字はパーセント("%")または数字であってはならない。これにより、ほとんどの高水準言語の識別子ルールを使って名前を形成できること、また、ほとんどのファイル名やパス名が許可されることに注目。たとえば、"simple3"、"gorp/foo_bar"などの名前値がある。
- コロン(":")を区切り文字として使って書かれるペア演算子を使い、ペア値を形成できる。ペア値は、値の前部は名前でなければならないが、値の後部には名前または整数を持つことができる。たとえば、"a:2"、"b:1"、"c:joe"などの名前値がある。
- プラス文字("+")を使って書かれるリスト追加演算子を使い、リスト値を形成できる。リスト内の項目には、名前かペア、またはその混合を使用できる。たとえば、"a+b+c"は3つの名前が含まれるリストであるが、"a:2+b:1+c:joe+d"は3つのペアと1つの(ペアでない)名前が含まれるリストである。リスト内で名前が重複する場合には、右端の名前が優先される。たとえば、"a:10+b+a:5"は"b+a:5"となり、"a:10+b+a"は"b+a"となる。
- マイナス文字("-")を使って書かれるリスト削除演算子を使い、リスト値の中から項目を削除してリスト値を形成できる。たとえば、"a+b+c-a"は"b+c"となる。リスト内に項目が見つからない

場合、削除は無視される。たとえば、"a+b-c"は"a+b"となる。括弧は使用できない。また、評価は厳格に左から右へ行われる。たとえば、"a-a+b"は"b"となり、"a-b-c+b"は"a+b"となる。

- パーセント文字("%")を使って書かれる「値抽出」演算子を使い、任意のコントロール変数の現行値を抽出できる。たとえば、"%inline+a"は、コントロール変数inlineに現在割り当てられているリストに名前"a"を追加したリストとなる。
- 特殊トークン"%all"は、それが割り当てられているコントロール変数に適用可能なすべての可能な名前のセットを表す名前として使用できる。

コントロール割り当て

コントロール変数またはコントロールグループに値を割り当てるには、以下の形式のコントロール割り当てを記述します。

コントロール変数=コントロール式

または

コントロールグループ=コントロール式

コンパイラ呼び出しコマンドラインから、-Xスイッチを使ってそのようなコントロール割り当てを次のように指定できます。

-Xコントロール変数=コントロール式

または

-Xコントロールグループ=コントロール式

簡単な例として、値2をdiagという名前のコントロール変数に割り当てるとします。次のどちらの形式も許可されます。

```
diag=2          diag2
```

コンパイラ呼び出しコマンドラインから、-Xスイッチを使ってこれらのどちらのコントロール割り当てでも指定できます。

```
-Xdiag=2  -Xdiag2
```

同様に、値4をOという名前のコントロールグループに割り当てるとします。次のどちらの形式も許可されます。

```
O=4             O4
```

コントロール式が名前値で始まる場合、等号は必須です。たとえば、cという名前のコントロール変数にansiという名前を割り当てるには、次の形式だけが許可されます。

```
c=ansi
```

コントロールプログラム

*コントロールプログラム*は、コントロール割り当てのシーケンスとして書かれます。

コントロールプログラム内では必要に応じてスペースを挿入できますが、以下の制約があります。

- スペースは、名前または数字の中には挿入できない。
- また、スペースはコマンドラインオプションの区切り文字として使われるため、コマンドラインに指定されるコントロールプログラムではまったく使用できない。
- コントロールプログラム内では、コントロール割り当ては通常コンマで区切られる。

- しかし、コマンドライン上でなければ、スペースをコンマの代わりに使用できる。
- また、整数定数で終わるコントロール割り当ての後では、コンマは省略可能である。

たとえば、3をdiagに割り当て、joe+peteをinlineに割り当てる場合、以下のすべての形式が許可されます("Δ"はスペースを表す)。

```
diag=3,inline=joe+pete inline=joe+pete,diag=3
diag=3Δinline=joe+pete inline=joe+peteΔdiag=3
diag=3inline=joe+pete  inline=joe+peteΔdiag3
diag3inline=joe+pete   inline=joe+pete,diag3
```

など

この例をさらに拡張し、3をコントロールグループOに割り当てることも必要である場合は、以下の形式が許可されます。

```
O=3,diag=3,inline=joe+pete    inline=joe+pete,diag=3,O=3
O3diag=3Δinline=joe+pete      inline=joe+pete,diag=3O=3
diag=3O=3inline=joe+pete      inline=joe+pete,O=3diag=3
O3diag3inline=joe+pete        inline=joe+pete,diag3O3
```

など

コントロールプログラムはすべて、左から右へ処理されます。従って、割り当ての重複があった場合は、右端の割り当てが優先されます。たとえば、Oに3を割り当てることによってaliasに3の値を割り当てるとします。この場合:

```
O=3,alias=1
```

はaliasに1を割り当て、一方、

```
alias=1,O=3
```

はaliasに3を割り当てます。

以下の文法は、コントロールプログラムを書く際に許可される形式に関するこれらのルールを要約しています。

コントロールプログラム:	:=	コントロール割り当てリスト
コントロール割り当てリスト:	:=	コントロール割り当て | コントロール割り当てリスト [区切り文字] コントロール割り当て {制約:区切り文字は、コントロール割り当てリストが整数値で終わるときにのみ省略可能。}
区切り文字:	:=	"," | "Δ"{注:Δはスペース文字を表す。}
コントロール割り当て:	:=	コントロール変数名 ["="] コントロール式 | コントロールグループ名 ["="] コントロール式 {制約: "="は、コントロール割り当てリストが整数値で始まるときにのみ省略可能。}
コントロール変数名:	:=	{35ページの「コントロール変数の定義」および 73 ページの「コントロール変数リファレンス」にリストされているコントロール変数名のいずれか。}
コントロールグループ名:	:=	{35ページの「コントロール変数の定義」および 73ページの「コントロール変数リファレンス」にリストされているコントロールグループ名のいずれか。}
コントロール式:	:=	項 | コントロール式 プラス演算子 項
項:	:=	整数値 | 名前 | ペア
整数値:	:=	数字 | 整数値 数字

数字:	:=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
名前:	:=	名前値 | "%all" | "%none" |"%" コントロール変数名
名前値:	:=	{等号、コンマ、＋、－、コロンを含まず、%または数字で始まらない任意の文字列。}
ペア:	:=	名前 ":" 名前 | 名前 ":" 整数値
プラス演算子:	:=	"+" | "-"

プラグマ命令

プラグマディレクティブ(または単に「プラグマ」)はプログラムのソースコード内のステートメントであり、書式的にはコメントと同等ですが、コンパイルシステムと情報を交換することができます。SNCコンパイラでのプラグマの使用法の1つは、コントロール変数および/またはコントロールグループの値を操作することです。

プラグマディレクティブはCとC++どちらでも記述できます。

構文

```
#pragma <命令>
```

または

```
_Pragma <命令>
```

_Pragma 形式は、C99 または GNU 拡張 (gnu_ext) モードが有効な場合にのみ使用できます。gnu_ext モードはデフォルトで有効に設定されています。詳細は、75 ページの「-Xc」を参照してください。

マクロ内でプラグマ命令を使用する必要がある場合を除き、両形式とも置き換えが可能です (マクロ内では `_Pragma` 形式を使用する)。

ライブラリ検索

```
#pragma comment (lib,"<library>")
```

このプラグマは、オブジェクト ファイルにライブラリ検索リクエストを発行します。<library> には、リンカーによってインクルードされるライブラリ名を指定します。ライブラリ名は、リンカーの -l オプションでインクルードされるのと同様のルールに従います。すなわち、「lib」が接頭辞として付けられるので、以下のようになります。

```
#pragma comment (lib,"foo")
```

これはリンカーのコマンドラインに「-lfoo」を配置した場合と同等で、リンカーでは「libfoo.a」と名付けられたライブラリの検索が行われます。

リンカーで、自動的に複数のライブラリがインクルードされるようにするため、オブジェクト ファイル内に複数の「lib」コメントを使用することもできます。

セグメント制御用プラグマ

SNC コンパイルでは、セグメント (セクションなど) を制御し、特定のエンティティを生成するためのプラグマを使用できます。プログラマーは、この方法で生成された特別なセクションが、プログラム ELF 内で正しくレイアウトされていることを確実にする必要があります (通常はリンカー スクリプトを使用)。

Cコード セグメントの制御

コード セグメントを制御するプラグマは、以下の形式で使用します。

```
#pragma code_seg ("<segname>")
```

上記の <segname> は、目的のセクション名になります。このプラグマは、関数レベルで宣言し、後続するすべての関数用のセグメントを示します。

通常のコード セグメントへの生成は、セグメント名の部分をブランクにすることによって指定します (以下を参照)。

```
#pragma code_seg ("")
```

文字列セグメントの制御

文字列セグメントを制御するプラグマは、以下の形式で使用します。

```
#pragma str_seg ("<segname>")
```

上記の <segname> は、目的のセクション名になります。このプラグマは、ステートメント レベルで宣言し、後続するすべての定数文字列用の目的のセグメントを示します。これは、デバッグ コード (アサートなど) で使用する文字列の定数を、通常のプログラム文字列の生成から分離する際によく使用します。

通常の文字列セグメントへの生成は、セグメント名の部分をブランクにすることによって指定します (以下を参照)。

```
#pragma str_seg ("")
```

ビット フィールド実装制御

以下は、SNC におけるビット フィールドの実装方法に影響を与えるプラグマです。

`#pragma ms_struct on`	ビット フィールドと非ビット フィールド間でワードが共有されない (Microsoft のコンパイラ ビット フィールド割当てルールを使用)。
`#pragma ms_struct off`	Microsoft のコンパイラ ビット フィールド ルールを使用しない。
`#pragma reverse_bitfields on`	リバース ビット フィールドを有効にする。
`#pragma reverse_bitfields off | #pragma reverse_bitfields reset`	リバース ビット フィールドを無効にする。

通常、ビット フィールドは左から右の順に割り当てられます (最上位ビットから最下位ビットの順)。リバース ビット フィールドが有効な場合、ビット フィールドは右から左の順に割り当てられます (最下位ビットから最上位ビットの順)。#pragma reverse_bitfields と #pragma ms_struct のデフォルト ステータスは「OFF (無効)」です。

制限:

以下の PPU Lv2 GCC コンパイラ実装の SNC におけるリバース ビット フィールドの有効化は、#pragma ms_structs と共に使用された場合のみ、効果を発揮します。つまり、reverse_bitfields をオン (有効) にするなら、ms_struct もオン (有効) にする必要があるということです (逆も同様)。

例:

```
#include <stdio.h>

#pragma ms_struct on
#pragma reverse_bitfields on

union u01 {
  struct s {
```

```
    /**
       [reverse bit-field]
        msb                                    lsb
        00000000 00000000 00000000 000 00 00 0
                                  |s4 |s3|s2|s1|
    **/
    unsigned int s1: 1;
    unsigned int s2: 2;
    unsigned int s3: 2;
    unsigned int s4: 3;
  } u1;
  unsigned int i;
};

#pragma ms_struct off
#pragma reverse_bitfields off

union u02 {
  struct ss {
    /**
       [normal bit-field]
        msb                                  lsb
        0  00 00 000 00000000 00000000 00000000
       |s1|s2|s3|s4 |
    **/
    unsigned int s1: 1;
    unsigned int s2: 2;
    unsigned int s3: 2;
    unsigned int s4: 3;
  } u1;
  unsigned int i;
};

int main(void) {
  union u01 uu1; /* リバース */
  union u02 uu2; /* 通常 */

  uu1.i = 0;
  uu2.i = 0;

  uu1.u1.s1 = 0x1;
  uu1.u1.s2 = 0x2;
  uu1.u1.s3 = 0x3;
  uu1.u1.s4 = 0x4;
  /**
     [リバース ビット フィールド]
      msb                                    lsb
      00000000 00000000 00000000 100 11 10 1
                                |s4 |s3|s2|s1|
  **/

  printf("%x\n", uu1.i); /* 9dが出力される */


  uu2.u1.s1 = 0x1;
  uu2.u1.s2 = 0x2;
  uu2.u1.s3 = 0x3;
  uu2.u1.s4 = 0x4;
  /**
     [通常のビット フィールド]
```

```
      msb                                       lsb
      1  10 11 100 00000000 00000000 00000000
     |s1|s2|s3|s4 |
  **/

  printf("%x\n", uu2.i); /* dc000000 が出力される*/

  return(0);
}
```

テンプレート インスタンス化プラグマ

テンプレートのインスタンス化制御を支援するプラグマが、3 つ存在します。

```
#pragma instantiate argument
```

上記のプラグマでは、このコンパイル処理で *argument* がインスタンス化されます。これは -Xtmpl=none モードで使用し、明示的に指定した必要なインスタンス化のみを実行することができます。

```
#pragma do_not_instantiate argument
```

上記のプラグマでは、このコンパイル処理で *argument* がインスタンス化されません。

```
#pragma can_instantiate argument
```

上記のプラグマでは、その時点の変換ユニットの *argument* が、必要に応じてインスタンス化されます。

上記の各ケースでは、*argument* は、テンプレート クラス名、メンバーの関数名、静的データのメンバー名、メンバー関数の宣言、関数の宣言になります。クラス名が指定された場合は、クラスのすべてのメンバー関数および静的データ メンバーにその命令が適用されます。

インライン プラグマ

インライン化を明示的に制御する場合は、2 つのプラグマを使用できます (以下を参照)。

`#pragma inline`	関数を、コールされる箇所で強制的にインライン化する。
`#pragma noinline`	関数を一切インライン化しない。

このプラグマは、関数単位での指定であるため、関数の宣言の前に配置する必要があります。また、自動インライン化のレベルが 1 以上 (-Xautoinlinesize > 0) にセットされている場合にのみ有効になります。

診断プラグマ

コンパイラが出力する多くの診断メッセージは、ソース プラグマを使用してそのカテゴリーを変更できます。これにより、個々のメッセージの重要度を、行単位で上下させることができます。警告と備考には任意の診断レベルを割り当てることができ、「任意エラー」と呼ばれる特定エラーのみは、診断レベルを下げることができます。この任意エラーには、診断表示行のエラー コード番号に「-D」という接尾辞が付きます。一部のエラー コードは、特定の条件においてのみ任意エラーとなります。任意エラーではないエラーは、ソース言語やコンパイラでの処理に実行不可能な変更を加えてしまう可能性があるため、診断レベルを下げることができません。

この診断プラグマは、以下の形式になります。

```
#pragma diag_<category>=<idlist>
```

上記の <category> には、診断メッセージに設定する目的のカテゴリー (以下を参照) を入力します。

suppress	診断メッセージを出力しない。
remark	備考の診断メッセージを出力する。
warning	警告の診断メッセージを出力する。
error	エラーの診断メッセージを出力する。
default	診断をデフォルトのカテゴリーにセットする。

<idlist> は、診断番号または診断タグ名のカンマ区切りのリストとなります。診断タグとその番号に関しては、help\err_doc.htm のエラーに関するドキュメントを参照してください。

警告セット全体のステータスをローカル スタックに一時的に保存し、これらをスタックから復元することもできます。診断スタック プラグマは以下の形式で使用します。

```
#pragma diag_push           // 警告セット全体のスタータスを保存
#pragma diag_pop            // 警告セット全体のステータスを復元
```

診断プラグマを使用して警告を無効にする

診断プラグマの使用法:

```
#pragma diag_default=942
```

これは、非 void 関数からの戻り値がないことに対する診断レベルを「警告」に戻します。

選択されたコード ブロックに対する個々の警告を、以下のプラグマを使用して出力しないようにすることも可能です。

```
#pragma diag_push
#pragma diag_suppress=942
     <code block>
#pragma diag_pop
```

これによってコード ブロックの間、警告 942 が無効になり、その後このメッセージの以前の状態に戻ります。

すべての警告を抑制する制御プラグマを使用する

これに関連するやり方として、コントロール プラグマ スタックと diag コントロールを使用し、コンパイラの警告や通知のすべてを出力しないようにする方法もあります。

```
#pragma control %push diag
#pragma control diag=0
     <code block>
#pragma control %pop diag
```

これによってコード ブロックの間、すべての通知や警告が出力されなくなり、その後ブロックの終わりで、diag コントロール変数の以前の状態に戻ります。なお、「control %push」と「control %pop」プラグマはコントロール プラグマに適用されるため、診断プラグマに影響を与えることはありません。

制御プラグマ

制御プラグマ命令の構文は以下のようになります。

```
#pragma control <cprog>
```

ここで:

- controlは大文字小文字を区別し、
- <cprog>はコントロールプログラムであり、
- これらのフィールドはペースで区切られる。スペースとは、1つまたは複数の空白文字および/またはタブ文字のシーケンスです。

コンパイラは次のように診断を発行します。

- プラグマトークン(#pragma)が認識されたが、制御トークンが存在しない場合は、注釈診断メッセージが発行される (42ページの「診断用コントロール変数」を参照)。
- プラグマトークン(#pragma)が認識され、制御トークンが存在するが、コントロールプログラムの形式が不正な場合は、エラーが発行される。

コントロール プラグマを使用するとき、値をスタックに保存し、スタックから取り出すことができます。スタックに値を追加するための書式は次の通りです。

```
#pragma control %push <cprog>
```

ここで、<cprog>はコントロールプログラムです。

スタックから値を取り出すための書式は次の通りです。

```
#pragma control %pop <cprog>
```

スタックは、値を一時的に変更し、後で元のコードの値に戻したい場合などに使用すると便利です。

以下の例は、コードの1行に対してゼロ除算の警告を無効にする方法を示しています。

```
#pragma control %push diag
#pragma control diag=0
   return 1/0;
#pragma control %pop diag
```

pushプラグマを拡張して、前の値を保存するだけでなく、新しい値を設定することもできます。次の書式を使用します。

```
#pragma control %push <cprog>=0
```

たとえば、上記の例を簡略形で書くと次のようになります。

```
#pragma control %push diag=0
   return 1/0;
#pragma control %pop diag
```

定義済みのマクロの使用

コンパイラでは、定義済みの多数のマクロが内部で宣言されます。これらマクロとその現在値のリストを取得するには、-Xpredefinedmacros コントロール変数を使用します。

使用例:

出力例:

```
#define __SIGNED_CHARS__ 1
#define __DATE__ "Feb 18 2009"
#define __TIME__ "14:51:11"
...
```

詳細は、88ページの「-Xpredefinedmacros」を参照してください。

コンパイラ バージョンの取得

使用されている SNC のバージョンを知るには、定義済みのマクロ __SN_VER__ を使用します。たとえば、バージョン 300.1.x のコンパイラの場合、__SN_VER__ では 30001 への評価が行われます。

これは条件コンパイルに対して使用できます。たとえば、ヘッダー ファイルで新しいコンパイラ機能が利用される場合に、SNC の前バージョンでコンパイルすると、この機能によってファイル エラーが引き起こされてしまう状況などです。

コントロール変数のテスト

コードで __option プリプロセッサ マクロを使用することにより、整数値を取るコントロール変数のコンパイル時の値をテストできます。無効な (非整数の) コントロール変数が指定された場合は、コンパイラによってコンパイラ エラーが発行されます。

構文:

```
__option(control-variable)
```

control-variable には、整数値を取るコントロール変数の名前から接頭辞「-X」を取り除いた名前が入ります。

例:

```
#if __option(notocrestore)
... // -Xnotocrestore が非ゼロの場合に依存するコードを実行
#else
... // -Xnotocrestore がゼロの場合に依存するコードを実行
#endif
```

制限

- __option プリプロセッサ マクロは、プリコンパイル済みヘッダーとの互換性がありません。詳細は、59 ページの「プリコンパイル済みヘッダ」を参照してください。

-Xc コントロール変数オプションのサポート

__option マクロでは、-Xc コントロール変数で指定される単独 C/C++ 言語モードを引数として取ることもできます。詳細は、44 ページの「c:C/C++言語モード」を参照してください。

例:

```
#if __option(rtti)
... // -Xc+=rtti に依存するコードを実行
#else
... // -Xc-=rtti に依存するコードを実行
#endif
```

4: コントロール変数定義

コントロール変数の定義

このセクションでは、各コントロール変数の定義および意味を説明します。説明は、関連する関数を持つコントロール変数のクラス別に記載されています。

73 ページの「コントロール変数リファレンス」には、すべてのコントロール変数がアルファベット順でリストされた表が記載されています。また、名前、省略形、スコープ、値のタイプや範囲、デフォルト値、コントロール変数に割り当て可能な各値の意味の簡単な説明など、それぞれの重要な特性も記載されています。

コントロール変数の機能に関する説明が必要な場合はこのセクションを、特定のコントロール変数の機能を確認する場合は 73 ページの「コントロール変数リファレンス」を参照してください。この両方の章では、コントロール グループに関しても記載されています。

最適化のコントロール変数

このセクションで説明するコントロール変数は、コンパイラによって実行される最適化の種類と範囲を決定するものです。これらのコントロール変数のサブセットは、「O」という名前の付いたコントロール グループの変数を構成します。

最適化について

SNC コンパイラは、コンパイルするプログラムに高度な最適化を適用できるコンパイラです。最適化機能のないコンパイラは、ソース プログラムをマシン語に単に翻訳するのみのため、その翻訳時にプログラムの構造 (実行される演算処理と制御の論理フローなど) も維持されます。一方、最適化機能を搭載したコンパイラでは、以下が可能となっています。

1. ソース プログラムを詳細に分析し、プログラムの変数とステートメントの各種基本特性を解明する。
2. プログラムに対して、最適化と呼ばれる変換を実行し、変換後のマシン語は、同じ結果を高速に出すことができる。しかし最適化されたプログラムの構造は、元のソース プログラムの構造と大幅に異なる場合もある。

この分析と最適化は、コンパイラの設計によって決められた順番で実行されます。SNC コンパイラでは、分析と最適化が交互にミックスされており、一部の最適化が一部の分析の後に実行される場合などがあります。さらに、最適化後に別の最適化が可能になる場合もあるため、一部の分析や最適化が繰り返し実行されることもあります。

関数がコンパイルされる際、分析や最適化が適用される前に、その関数は「基本ブロック」と呼ばれるユニットに分割されます。具体的に基本ブロックとは、シーケンスの最初でのみ開始し、シーケンスの最後でのみ終了する (その後、0 個、1 個または複数の別の基本ブロックに転送できる) 演算命令のシーケンスです。最適化機能においては、ブロックのいずれかの演算が実行されると、すべての演算が実行されるということが、基本ブロックの重要な特性です。基本ブロックは、ソース プログラムのステートメントと異なり、各基本ブロックに部分的なソース ステートメントのみが含まれることがあり、またひとつの基本ブロックに複数のソース ステートメントが含まれることもあります。各基

本ブロックを個別に検証/変更することによって適用される分析や最適化は、「ローカル分析」や「ローカル最適化」と呼ばれ、複数の基本ブロックが関連するものは「グローバル分析」や「グローバル最適化」と呼ばれます。また、2 つ以上の関数が関係する分析または最適化は、プロシージャ間分析または最適化と呼ばれます。

alias: エイリアスの分析

エイリアス分析は、プログラムにおける 2 つのメモリ参照が、実行時に同じオブジェクトを参照するかどうかの判定に関するものです。エイリアス分析の結果は、最適化全体にわたって各ポイントで使用され、多くの異なる最適化の結果に影響を与えます。たとえば、以下の 2 つのステートメントを考えてください。

```
X = 4*A*B/(2*C-D)+E*F
Y = M/(N+O-P)-5*Q
```

この場合、X がメモリ内の M、N、O、P、Q とは完全に別のオブジェクトであることが判明すると、最適化機能は、X に結果がストアされるまえに 2 つ目の式の評価を開始するコードをコンパイルします。また Y が、A、B、C、D、E、F のいずれとも異なる場合、2 つのステートメントは完全に交換することができ、または他の条件が満たされた場合は、他方のステートメントをループ外に巻き上げることができない場合でも、そのステートメントが巻き上げられることがあります。

2 つのメモリ参照が、常にメモリ内の特定オブジェクトを参照すると判定された場合、その参照は独立したものであると考えられます。またこの判定ができない場合、この参照は相互干渉している可能性があります。この判定における相違点を理解するため、以下の C フラグメントを参照してください。

```
float x,y;
union p{float u[10], v[5]};
float a,b,c,d,e,f,g,h;
int i,j;
...
x = a + b;
y = c - d;
...
p.u[5] = e*f;
p.v[j] = g*h;
...
p.u[i] = g/h;
p.u[i+1] = e/f;
```

このプログラムでは、以下がわかります。

- x と y の宣言を検証するのみで、x と y に対する参照が独立したものである。
- u と v の宣言を検証することにより、および 添字 u が固定値 5 (j の値が v の範囲外の要素を指さないという、ANSI/ISO C 言語基準の制限が前提) であることから、u[5] と v[j] に対する参照が独立したものである。
- プログラムのフローで、2 つの添字が別々の値を持つ (2 つのステートメント間に i=i-1 ステートメントがないなど) と判定された場合は、u[i] と u[i+1] に対する参照が独立したものである。

エイリアスのコントロール変数は、実行するエイリアス分析の程度を制御するために使用します (以下を参照)。

-Xalias=0	エイリアス分析を実行しない。コンパイラは、すべてのメモリ参照が、オプションが適用されているコード セクション内の他のすべてのメモリ参照と干渉する可能性があると想定します。これは、最適化に大きな影響を及ぼし、ほとんどの最適化が制限されます。
-Xalias=1	宣言に基づきエイリアス分析を実行。参照に使用されている変数の宣言は、干渉が可能かどうかが検証されます。独立している場合の多くは、このレベルの分析で検出されます。

-Xalias=2	宣言と固定添字に基づきエイリアス分析を実行する。共通の添字位置に異なる定数値を持つ配列の非ポインタ参照は、独立したものとしてマークされます。このレベルの分析は、数値ベースの関数、特に内部ループで複数次元の配列を使用する関数で重要になります。その他の関数では、重要ではなく、利用価値もない場合があります。
-Xalias=3	添字同士のフロー依存情報を使用することにより、エイリアス分析を強化する。このレベルの分析は、数値ベースの関数、特に内部ループで配列を使用する関数で重要になります。その他の関数では、重要ではなく、利用価値もない場合があります。

alias コントロール変数は、関数スコープを持ち O コントロール グループのメンバーで、0 ～ 3 の値を使用します。デフォルト値は alias=0。

flow: 制御フローの最適化

制御フロー最適化では、プログラムの制御フローが向上します。この最適化では、処理が及ばないコードの削除、GOTO に転送される GOTO の省略、隣接する基本ブロックの統合 (可能な場合)、分岐の簡素化が行われます。これらの最適化は通常、プログラムに対する別の最適化 (テスト条件への定数の適用など) が適用された後にのみ使用します。

制御フロー最適化の重要な欠点は、プログラムの制御構造が変更されるため、プログラムのデバッグが非常に困難になるという点です。

制御フロー最適化は、flow コントロール変数で制御します (以下を参照)。

-Xflow=0	制御フロー最適化を実行しない。
-Xflow=1	制御フロー最適化を実行する。

flow コントロール変数は、関数スコープを持ち O コントロール グループのメンバーで、0 または 1 の値を使用します。デフォルト値は flow=0。

fltedge: 浮動小数点の限度

浮動小数点値は、数値の場合や非数値 (NaN や INF など) の場合があります。また、非数値が関連する浮動小数点の演算は、「信号」や「信号なし」になる場合があり、IEEE Standard 754 の規則および目的のプロセッサの規則によって、例外発生の原因になる場合やならない場合があります。

最適化では、非数値が関連するプログラムの動作が変化する場合があります。たとえば、信号を発する演算が「死んで」おり、その結果が使用されない場合、最適化によってその演算が削除され、例外も発生しなくなります。また IEEE の規則では、非数値が関連する「信号なし」の比較が常にエラーを発生するように決められているため、「A は A と等しい」の比較を削除するオプションでは、A に非数値が含まれている場合、結果が変化します。「A は B を超えない」を「A は B 以下」に変更する最適化でも、A または B に非数値が含まれている場合、結果が変化します。

この場合は、fltedge コントロール変数を使用して、部分的に制御することができます (以下を参照)。

-Xfltedge=1	非数値が発生し、「信号なし」の演算でそれが使用された場合にプログラムの動作を変化させる最適化を実行しない。このモードの使用は、SNC コンパイラではあまり適していません。場合によっては、比較演算が変更され、その動作が変化することもあります。たとえば、式 (!(a>b)) が (a<=b) に変更された場合、a と b の順番を変更しない限り不正となります。
-Xfltedge=2	非数値が発生し、「信号なし」の演算でそれが使用される場合にプログラムの動

	作を変化させる可能性のある最適化を実行する。このモードでは、変数テスト (変数自体に対する等式や不等式) の特殊ケースは最適化されません。このモードは、通常の最適化を許可するためのものですが、非数値のテストをプログラムする機能も搭載しています。
-Xfltedge=3	非数値が発生し、「信号なし」の演算でそれが使用される場合にプログラムの動作を変化させる可能性のある最適化を実行する。

fltedge コントロール変数は関数スコープを持ち、1 ～ 3 の値を使用します。デフォルト値は fltedge=2。

メモ: fltedge コントロール変数は、-Xfastmath=1 を使用して fastmath が有効にされない限り、効力を生じません (79ページの「-Xfastmath」を参照)。

fltfold: 浮動小数点の定数のフォールド

定数のフォールドは、実行時ではなく、コンパイル時に定数が関連する式を評価する最適化です。浮動小数点の定数に対するこの最適化は、fltfold コントロール変数を使用して制御します (以下を参照)。

-Xfltfold=0	コンパイル時に、浮動小数点定数が関連する式を評価しない。
-Xfltfold=1	コンパイル時に、浮動小数点定数および演算子が関連する式を評価する。浮動小数点定数に適用されている組み込み関数が関連する式は、評価されません。
-Xfltfold=2	コンパイル時に、浮動小数点定数が関連する式を評価する。

fltfold コントロール変数は関数スコープを持ち、0 ～ 2 の値を使用します。デフォルト値は fltfold=2。

intedge: 整数の限度

一部の最適化は、関連する変数の値が許容範囲の限度に近くないことが判明している場合にのみ実行できます。整数変数では、intedge コントロール変数でこれを制御します (以下を参照)。

-Xintedge=0	整数の演算時に整数のオーバーフローが発生すると想定される場合において、オーバーフローが発生した際にプログラムの動作が変化する最適化を実行しない。
-Xintedge=1	整数の演算時に発生する整数のオーバーフローの影響が、最適化の適用において無視できる程度の場合。

intedge コントロール変数は関数スコープを持ち、0 または 1 の値を使用します。デフォルト値は intedge=0。

notocrestore: TOC オーバーヘッドの削減

PS3 PPU ABI では、関数のコールに対応させるために使用する TOC (Table of Contents) という機能が定義および使用されます。ABI に従うと、その関数コールが以下の状態になっている必要があります。

- 関数へのコールには、リンカがコードをパッチするために、コール命令自体の後にスペースがある必要があります。
- 関数へのポインタ経由のコールは、中間ストラクチャの「.opd」エントリを使用する必要があります。このストラクチャは、ターゲット コードで使用される TOC 範囲のアドレス、およびターゲット コード自体のアドレスで構成されます。

SN コンパイラではこの TOC が使用されないため、関数コールの後に nop 命令を省略するようコンパイラに指示することができます。また、関数へのポインタを経由するコール用の TOC をロードするためのコードを省略するようコンパイラに指示することもできます。これは、コントロール変数「-Xnotocrestore」を指定することによって行います。

notocrestore をオンにしてビルドされたオブジェクト ファイルはSN Linker 240.0.2992.0 以降を使用し、リンカのコマンドライン スイッチ「--notocrestore」および「--no-multi-toc」を指定してリンクする必要があります。このオプションは、TOC データのサイズ合計が 64 KB を越えないという 1 つの条件—これはリンカーが厳密に強制する制限です—の下で、SNC および GCC でコンパイルされたコードが混在していても、まったく安全に使用できます。

プラグマで PRX 関数への間接コールが使用される場合、notocrestore 制御変数の使用には制限があります。詳細は、『*ProDG Linker for PlayStation 3 ユーザー ガイド*』の「TOC 情報」を参照してください。

notocrestore の最適化設定は、「#pragma control %push」オプションを使用することによって関数単位で変更することができます。詳細は、71 ページの「関数単位の最適化」を参照してください。

reg: レジスタの割り当て

レジスタ割り当ては、目的のプロセッサにおける高速レジスタの使用の最適化に関連したものです。レジスタからの数量の参照は、メモリからの参照に要する時間の数分の 1 で済むため、これは重要になります。通常は、数量を維持するレジスタより、レジスタに効率的に格納される数量のほうが多く、実際にレジスタに格納するこれらの数量のベストな組み合わせの選択が非常に困難な問題となるため注意が必要です。

関数がコールされた際の最初の 2 ～ 3 個の引数など、数個の数量をレジスタに割り当てることはできますが、それ以上の場合、レジスタで維持する候補となる数量には、2 つの種類があります。

- 式の評価またはその言語のステートメントの実行に関連する中間値。これには、評価される式のすべての部分式や、アドレス指定式に関連するすべての数量などが含まれます。
- *レジスタ候補*変数
- C/C++ では、変数がスカラーで自動保管期間があり、そのアドレスが & 演算子で指定されていない場合、その変数はレジスタ候補となります。SNC-C コンパイラと SNC-C++ コンパイラでは、レジスタの宣言が無視されます。

SNC コンパイラでのレジスタ割り当ては、プロシージャ間 (関数の間)、グローバル (1 つの関数内)、ローカル (1 つの基本ブロック内) の 3 つのレベルで行われます。プロシージャ間でのレジスタ割り当ては、コールグラフ ツリーのボトムアップ式走査によって行われます (可能な場合)。また、コールする側とされる側の関係が明確な場合、コールされる側がコールする側より先に処理されます。これにより、コール サイトに関連するローカルとグローバルの割り当てで、コールされる側で既に行われた割り当てを反映されることができます。このためコール側は、コールの規則によって通常は消されてしまうレジスタを使用することができるようになります。

グローバルのレジスタ割り当ては、優先度ベースのグラフ カラーリング アルゴリズムを使用して行われます。この割り当てアルゴリズムを適切に進めるために、レジスタ干渉グラフが構築されます。ローカルのレジスタ割り当ては、グローバルのレジスタ割り当ての後に実行されます。

レジスタ割り当てとスケジューリングの関係に関しては、sched コントロール変数に関するセクション (40 ページの「sched: スケジューリング」) を参照してください。

多くの場合、候補の値をすべて格納するためのレジスタが不足します。この場合は、レジスタの値をメモリとの間で移動させる spill コードを挿入します。

レジスタ割り当ては、reg コントロール変数で制御します (以下を参照)。

-Xreg=0	レジスタ候補の変数をレジスタに割り当てない。
-Xreg=1	レジスタ候補の変数をレジスタに割り当て、グローバルおよびローカルのレジスタ割り当てを実行する。
-Xreg=2	レジスタ候補の変数をレジスタに割り当て、グローバルおよびローカルのレジスタ割り当てを実行する。より積極的なレジスタ最適化を実行。

reg コントロール変数は、関数スコープを持ち O コントロール グループのメンバーで、0 ～ 2 の値を使用します。デフォルト値は reg=0。

sched: スケジューリング

最近のほとんどの RISC プロセッサには、ある程度の命令レベルでの並列機能が搭載されています。並列処理では、特定命令の実行が、それに近い他の命令の実行と時間的にオーバーラップします。この並列の程度と状況は、プロセッサによって大きく異なりますが、特定の命令の並びが他の並びより高速になるという特徴があります。スケジューリングとは、目的のマシンで使用できる命令レベルの並列処理を活かして命令を並べ替える最適化です。当然ながらこの最適化は、マシンに大きく依存します。

スケジューリングを、レジスタ割り当ての前に実行するか後に実行するかというのは、コンパイラにおける典型的なジレンマです。レジスタ割り当ての後にスケジューリングした場合は、スケジューラが実行できる並べ替えの範囲に対する制限が発生します。レジスタ割り当ての前にスケジューリングした場合は、レジスタに対する負荷が増加し、spill コードが増えることになります。このため、SNC コンパイラでは、この処理が分割されています。最初にグローバル レジスタ割り当て、レジスタ負荷を考慮した 1 つ目のスケジューリング、ローカル レジスタ割り当て、そして最後に 2 つ目のスケジューリング (必要に応じて) が実行されます。

スケジューリングは、sched コントロール変数で制御します (以下を参照)。

-Xsched=1	1 つ目のスケジューリングのみを実行する。
-Xsched=2	両方のスケジューリングを実行する。

sched コントロール変数は、関数スコープを持ち O コントロール グループのメンバーで、0 ～ 2 の値を使用します。デフォルト値は sched=0。

unroll: ループのアンロール

ループ アンロール最適化では、特定のループを使用してそのループ内のコードを数回複製します。これにより、コード サイズが拡大しますが、ループ条件テストのオーバーヘッドを低減し、他の最適化の適用可能範囲を広げることができます。

このアンロールは、特定のループに対してのみ実行できます。

ループ アンロール最適化は、unroll コントロール変数で制御します (以下を参照)。

-Xunroll=0	ループをアンロールしない。
-Xunroll=1	自動制御でループをアンロールする。
-Xunroll=*n*	*n* > 1 の場合、アンロール可能なループを、常に *n* 回アンロールする。

unroll コントロール変数は、ループスコープを持ち、O コントロール グループのメンバーで、整数値を使用します。デフォルト値は unroll=0。

コントロール グループ O の最適化

コントロール グループに関しては、25 ページの「コントロールグループ」を参照してください。O コントロール グループには、最適化を制御するためのコントロール変数の値をセットする、便利な方法が用意されています。

O には、6 つの最適化レベルが用意されています。

-XO=0	最適化なし、およびインライン化なし (強制インライン化を除く)。
-XO=1	最適化なし、インライン化を許可。
-XO=2	完全最適化。
-XO=3	O=2に加え、より時間のかかる最適化 (現在は該当なし)
-XO=d	デバッグ可能な最適化済みコード (スケジューリングなしなど)
-XO=s	O=2、ただしインライン化の程度は抑えたもの。

これらの各 O 値において、メンバーのコントロール変数に割り当てる値に関しては、96 ページの「最適化グループ (O)」に記載されている O コントロール グループの表を参照してください。

関数のインライン化: inline、noinline、deflib

インライン化の最適化では、コールされる側の関数のコードが、コールする側のコードのコールが発生する箇所に直接挿入されます。これにより、コードの全体的なサイズが拡大しますが、以下のような特長もあります。

- 関数のコールとリターン、および引数を渡す際のオーバーヘッドが軽減される。
- 別の最適化が可能になる場合が多い。たとえば、複数の異なるケースを処理する関数、およびそのケースを区別するための定数値を想定してください。このような関数がインライン化された場合、定数伝播や制御フロー最適化などの最適化により、実行されるコードを削減できることがあります。また、コールがループ内にある場合は、インライン化後に、関数の一部をループ外に移動することができます。

SNC-C/C++ では、組み込み関数とユーザー 関数の両方をインライン化できます。プログラム内の関数がコールされる箇所は、「コールサイト」と呼ばれます。コールされる関数をインライン化するかどうかは、各コールサイトで個々に決定されます。このため、同じ関数があるコールサイトでインライン化され、他のコールサイトではインライン化されないという場合もあります。この決定は、以下の要因に左右されます。

1. コールされる側の関数の特徴
   - 常にインライン化される組み込み関数や、展開可能な形式でコンパイラに提供されない組み込み関数がある。
   - ユーザー 関数では、関数のソース コードが利用できる場合とできない場合がある (コールする側の関数と同じファイル内にある場合にのみ利用可能)。
   - コールされる側の関数のサイズ。
   - その関数がコールされる箇所の数。
2. コールサイトの特徴
   - コールは、C/C++ のポインタ経由した間接的なものである場合がある。このため、実際にコールされる関数をコンパイル時に特定することはできません。

- コールサイトのループのネスト レベル。
- コールする側の関数のサイズ (その前にインライン化された関数を含む)。

3. コールサイトにおける **inline** コントロール変数の値 (42 ページの「inline」を参照)。
4. コールサイトにおける **noinline** コントロール変数の値 (42 ページの「noinline」を参照)。
5. 組み込み関数では、コールサイトにおける **deflib** コントロール変数の値 (42 ページの「deflib」を参照)。

inline

inline コントロール変数では、名前のリストや、名前と整数で構成されるペアなどのリストを使用します。リストのコールサイト値に、コールされる側の関数の名前がある場合は、コンパイラの自動規則に応じて、それがその時点でのインライン化の候補になります。整数値が与えられた場合には、自動規則はそれを指定されたルーチンの優先順位として使います。*n* が大きいほど、関数がインライン化される可能性が高くなります。

inline コントロール変数はラインスコープを持ち、上記のリスト値を使用します。デフォルトの値は空のリストです。

noinline

noinline コントロール変数では、名前のリストを使用します。リストのコールサイト値に、コールされる側の関数の名前がある場合、この関数はインライン化されません。

noinline コントロール変数はラインスコープを持ち、上記のリスト値を使用します。デフォルトの値は空のリストです。

deflib

deflib コントロール変数では、以下の値を使用します。

`-Xdeflib=0`	デフォルトで、組み込み関数をインライン化しない。
`-Xdeflib=1`	デフォルトで、自動制御で組み込み関数をインライン化する。
`-Xdeflib=2`	デフォルトで、可能な限り組み込み関数をインライン化する。

deflib コントロール変数はラインスコープを持ち、0 ～ 2 の値を使用します。デフォルト値は deflib=1。

診断用コントロール変数

コンパイラが生成する各診断メッセージは、以下のカテゴリーに分類されます。

備考	備考メッセージは、一部の言語の使用がコンパイラには対応しているが、一般的ではない使用方法として認識されるという診断メッセージです。
警告	警告メッセージは、一部の言語の使用がコンパイラには対応しているが、問題のある使用方法として認識されるという診断メッセージです。
エラー	エラー メッセージは、コンパイルしている言語の構文や動作に対する違反を示すメッセージです。この場合、オブジェクト コードは生成されませんが、別のエラーがある場合もあるため、コンパイラはエラー箇所を越えて処理を継続します。

致命的なエラー	致命的なエラーは、その箇所以降のコンパイル処理が継続できないような深刻な問題を示すメッセージです。この場合、オブジェクト コードは生成されません。
内部エラー	内部エラーは、コンパイラ自体のロジックに関する問題を示すメッセージです。内部エラーが生じた場合は適当なサポートグループへご報告ください(SN Systems へ連絡)。その際は、内部エラーが発生した際に使用したソース コードをサポートに提供し、メッセージを再現できるようにしてください。

これらのメッセージに対する処理は、コントロール変数 diag と quit を使用して制御します。

diag: 診断出力レベル

コンパイラからの診断メッセージの出力レベルは、コントロール変数に設定した値を使用して制御します (以下を参照)。

`-Xdiag=0`	エラーと致命的なエラーのメッセージのみを出力。備考や警告は出力されません。
`-Xdiag=1`	警告、エラー、致命的なエラーのメッセージのみを出力。備考は出力されません。
`-Xdiag=2`	備考、警告、エラー、致命的なエラーのメッセージを出力。

これらのメッセージは、stderr に出力されます。また、エラーと致命的なエラーのメッセージを抑制することはできません。

diag コントロール変数はラインスコープを持ち、0 ～ 2 の値を使用します。デフォルト値は diag=1。

メモ: コマンドラインでの -w オプションは、-Xdiag=0 の省略形です。

diaglimit: 診断メッセージの制限数

SNC では他コンパイラと比較した場合、より徹底的な警告がデフォルトで発行される傾向にあります。これら他のコンパイラから初めてソースを移植する際には、追加警告が生成されるため、移植プロセスが分かりにくくなる場合があります。このスイッチでは、特定の診断それぞれに対し、発行される診断最大数を設定できます。

`-Xdiaglimit=`*n*	各診断に対して最初の *n* メッセージ のみを出力。

diaglimit 制御変数にはファイル スコープが存在し、整数値が許可されます。デフォルト値は 0 (無制限) です。

quit: 診断終了レベル

メッセージが生成される状況がない場合、コンパイラはコンパイル処理の最後で正常に終了 (終了ステータスは 0) します。診断メッセージが出力された時点での終了ステータスは、quit コントロール変数に設定した値によって制御します (以下を参照)。

`-Xquit=0`	エラーまたは致命的なエラーのメッセージが出力される状況が発生した場合は、異常終了する (終了ステータス＝1)。それ以外の場合は正常に終了します。
`-Xquit=1`	警告、エラー、致命的なエラーのいずれかのメッセージが出力される状況が発生した場合は、異常終了する (終了ステータス＝1)。それ以外の場合は正常に終了します。
`-Xquit=2`	備考、警告、エラー、致命的なエラーのいずれかのメッセージが出力される状況が発生した場合は、異常終了する (終了ステータス＝1)。それ以外の場合は正常

	に終了します。

これらの終了は、メッセージが出力されるかどうかではなく、メッセージの対象となる状況が発生するかどうかが基準となります。このコントロールの影響は、diag コントロール変数の設定とは無関係です。

quit コントロール変数はコンパイルスコープを持ち、0 ～ 2 の値を使用します。デフォルト値は quit=0。

C/C++コンパイル

このセクションでは、C/C++に関連するコントロール変数について説明します。

c:C/C++言語モード

SNC-C/C++には6つの基本モードがあり、その3つはコンパイラが受け入れるC言語の方言の指定と制限、あとの3つはC++言語の方言の指定と制限を行います。これらのモードは、以下のようにcコントロール変数で制御されます。

-Xc=ansi	このモードでは、コンパイラは、ANSIとISOのC規格（ANSI X3.159-1989およびISO/IEC 9899:1990(E)）の「規格合致ホスト処理系」に完全に従う。つまり、そのすべての言語と標準ヘッダファイルをサポートする。
-Xc=knr	このモードでは、コンパイラは、*The C Programming Language*（Kernighan & Ritchie著）（邦訳: プログラミング言語C）で与えられるC言語の定義の大部分と互換性があり、UNIX pccコンパイラと綿密に互換性がある。
-Xc=mixed	このモードでは、既存のK&RコードをANSIに移植する作業を容易にするためいくつかの拡張が追加されたことを除き、コンパイラは本質的にはANSIコンパイラである。これら拡張に関する詳細は、52 ページの「言語定義」を参照してください。
-Xc=arm	このモードでは、コンパイラは、*The Annotated C++ Reference Manual*（Margaret A. Ellis & Bjarne Stroustrup著）のC++言語と、C++規格（ISO/IEC 14882:2003を受け入れる。
-Xc=cp	armに似ているが、一部の旧式規格を受け入れ、比較的制約が少ない。armモードとcpモードでコンパイルされるプログラムはまったく同一の動作となる。
-Xc=cfront	このモードでは、コンパイラは、AT&T Cfrontコンパイラが受け入れるC++言語を受け入れ、互換性のあるオブジェクトコードを生成する。このオプションは:21か:30いずれかの追加の値を取る。-Xc=cfront:21はAT&T Cfront 2.1との互換性を有効にし、-Xc=cfront:30はAT&T Cfront 3.0との互換性を有効にする。-Xc=cfrontは-Xc=cfront:30と等価である。

cコントロール変数はファイルスコープを持ち、名前値として、ansi、knr、mixed、arm、cp、cfront、c99、const、volatile,、signed、noknr、inline、c_func_decl、array_nd、rtti、wchar_t、bool、old_for_init、exceptions、gnu_extおよびmsvc_extを取ります。SNC Cコンパイラデフォルト値はc=mixedです。SNC C++コンパイラデフォルト値はc=cpです。コマンドラインで指定する-Kは-Xc=knrの省略形です。

const、volatile、signed

これらの7つの基本モードに加え、3つの名前、const、volatile、signedの任意のサブセットをコントロール変数cの値に追加して、リスト値を形成することができます。基本モードがc=knrである場合にこれらのいずれかの名前を使用すると、ANSI C言語での対応する修飾子が認識されることを示

します。たとえば、c=knr+const+volatileはK&R互換性を表しますが、ANSI Cのconstとvolatileのタイプ修飾子も認識されます。

noknr

追加の値noknrをmixedモードまたはansi Cモードに追加できます（たとえば、-Xc=mixed+noknr）。この値により、プロトタイプのない関数の宣言と定義に対して、コンパイラが警告を生成するようになります。noknrモードを有効にすると、まだ宣言または定義されていない関数の使用にコンパイラが遭遇した場合にも警告が生成されます。

inline

追加の値inlineを、Cモードのansi、knr、mixedで与えることができます。これらのモードではデフォルトはオフであり、コンパイラにinlineをキーワードとして認識しないよう指示します。Cプログラムの中でinlineをキーワードとして認識させるには、inlineをコントロール変数cに追加します（たとえば、-Xc =mixed+inline）。inlineはC++モードでは常にキーワードとして認識されます。c99モードでは、inlineはデフォルトでオンです。

コントロール変数cの値としてのinline（上述の説明）と、コントロール変数inlineとの区別に注意してください (41 ページの「関数のインライン化: inline、noinline、deflib」を参照)。

c_func_decl

追加の値c_func_declを、すべてのC++モードで与えることができます。この値は、C++言語のプロトタイプ要件を、extern "C"ブロック内で宣言された関数については、C言語のプロトタイプ要件にまで緩和します。この値は、ユーザーコード内で直接使用するためのものではなく、Cスタイルのシステムインクルードファイルの使用をC++環境で有効にするためのものです。

array_nd

追加の値array_ndを、すべてのC++モードで与えることができます。デフォルトはオンであり、コンパイラにarray newとarray deleteの演算子を認識するよう指示します。array newとarray deleteを認識しないようにするには、array_ndをコントロール変数cから削除します（たとえば、-Xc-=array_ndまたは-Xc=arm-array_nd）。

rtti

追加の値rttiを、すべてのC++モードで与えることができます。オンに設定すると、コンパイラはRTTI（runtime type identification：実行時タイプ情報）キーワードを認識するようになり、RTTI動作が有効になります。RTTIを有効にした後でこれを無効にするには、rttiをコントロール変数cから削除します（たとえば、-Xc-=rttiまたは-Xc=cp-rtti）。デフォルト設定は「オン」です。

wchar_t

追加の値wchar_tを、すべてのC++モードで与えることができます。デフォルトはオンで、コンパイラにwchar_tをキーワードとして認識するよう指示し、また、-D_WCHAR_Tと-D__WCHAR_T_IS_KEYWORDを組み込み定義済みプリプロセッサマクロとして追加するよう指示します。前者のマクロは、コンパイラから提供されるさまざまなインクルードファイルの中で使用され、多くて1つのwchar_tタイプの定義が必ず認知されるようにします。後者のマクロは、wchar_tが特殊なC++タイプであることに依存する（たとえば、すべての組み込みタイプに対してテンプレートをインスタンス化する場合などの）コードをプログラマが保護できるように提供されています。wchar_tがキーワード（特殊タイプ）として認識されるのを無効にするには、wchar_tをコントロール変数cから削除します（たとえば、-Xc-=wchar_t）。

コントロール変数cの値としてのwchar_t（上述の説明）と、コントロール変数wchartとの区別に注意してください。47ページの「sizetおよびwchart：size_tとwchar_tのC/C++タイプ定義」を参照してください。

bool

追加の値boolを、すべてのC++モードで与えることができます。デフォルトはオンで、コンパイラにboolをキーワードとして認識するよう指示し、また、-D_BOOL_DEFINEDと-D__BOOL_IS_KEYWORDを組み込み定義済みプリプロセッサマクロとして追加するよう指示します。前者のマクロは、自分のboolの定義を保護するため、たとえば次のように使用できます。

```
#ifndef _BOOL_DEFINED
typedef unsigned char bool;
#define _BOOL_DEFINED 1
#endif
```

後者のマクロは、boolが特殊なC++組み込みタイプであることに依存する（たとえば、すべての組み込みタイプに対してテンプレートをインスタンス化する場合などの）コードをプログラマが保護できるように提供されています。boolがキーワードとして認識されるのを無効にするには、boolをコントロール変数cから削除します（たとえば、-Xc=cp-bool）。

old_for_init

追加の値old_for_initを、すべてのC++モードで与えることができます。cfrontモードでのデフォルトはオンですが、cpモードとarmモードではデフォルトがオフであり、forループのinitステートメントで宣言された変数のスコープを、ループ自身のスコープに制限するようコンパイラに指示します（スコープの制限は ISO/IEC 14882:2003 C++ 標準で必要です）。より大きなスコープを持つ変数を仮定するコードを書くときで、cpモードかarmモードを使いたい場合は、この変数をコントロール変数cに追加する必要があります（たとえば、-Xc=cp+old_for_init）。

exceptions

追加の値exceptionsを、すべてのC++モードで与えることができます。デフォルトはすべてのモードでオフです。オンに設定した場合exceptionsは、C++ 例外の使用をサポートするため必要なすべてのデータ構造を生成するようコンパイラに指示します。これにより、自分のコードを通過する例外を他のコードがスローした場合に、自分のコードが（例外を使用していなくても）保護されます。例外処理に関する詳細は、53 ページの「例外処理」を参照してください。

tmplname

追加の値 tmplname は、すべての C++ モードで使用できます。この値により、non-templated 関数に与えられた名前とは異なる、マングルされた名前を持つテンプレートが作成されます。

gnu_ext

追加の値 gnu_ext は、すべての C モードおよび C++ モードで使用できます。この値により、GNU GCC の拡張機能を C/C++ 言語で使用 (対応しているもののみ) できるようになります。これには、従来の GCC コードで一般に使用される 'attribute' 機能が含まれます。デフォルトはオン。

msvc_ext

追加の値 msvc_ext は、すべての C モードおよび C++ モードで使用できます。この値により、Microsoft Visual Studio の拡張機能をC/C++ 言語で使用 (対応しているもののみ) できるようになります。

char: C/C++ の char の符号

ANSI C/C++ には、char、signed char、unsigned char の異なる 3 種類の文字タイプがあります。規格からこれらは、2 つの式が同じタイプがどうかを判定するなどの目的では、3つの別々のタイプであることは明白です。しかし規格では「処理系定義」とされており、char タイプとして宣言されている数量を、符号ビットありの表現を付けて実装するかどうかの選択が必要になります。SNC-C/C++ ではこの選択を、char コントロール変数の値で制御します (以下を参照)。

-Xchar=signed	char の表現を、signed char と同じとする。
-Xchar=unsigned	char の表現を、unsigned char と同じとする。

char コントロール変数はファイルスコープを持ち、符号付きまたは符号なしの名前値を使用します。デフォルトの値は'signed'です。

sizetおよびwchart：size_tとwchar_tのC/C++タイプ定義

CとC++において、コンパイラがsize_tタイプとwchar_tタイプについて、それらが定義されていない場合であっても、情報を知る必要がある場合が生じます。このため、コンパイラには、これらのタイプが定義される方法についての予想が組み込まれています。コンパイラの予想とプログラムでの定義の間に不一致があると、プログラムが正常に動作しないことがあります。

通常、これらのタイプは1つまたは複数のシステムインクルードファイル内で定義されます。コンパイラの組み込み予想は、予想される環境における標準システムインクルードファイルでの定義に一致するよう設定されています。しかし、何らかの理由で非標準のシステムインクルードファイルのセットが使用された場合は、下記のオプションにより、それらのインクルードファイルでの設定に一致するようにコンパイラの組み込み予想を変更できます。

コントロール変数sizetとwchartは以下の値を取ります。

-Xsizet=uint	size_tの定義はunsigned int
-Xsizet=ulong	size_tの定義はunsigned long
-Xsizet=ushort	size_tの定義はunsigned short
-Xwchart=uint	wchar_tの定義はunsigned int
-Xwchart=ulong	wchar_tの定義はunsigned long
-Xwchart=ushort	wchar_tの定義はunsigned short
-Xwchart=uchar	wchar_tの定義はunsigned char
-Xwchart=int	wchar_tの定義はint
-Xwchart=long	wchar_tの定義はlong
-Xwchart=short	wchar_tの定義はshort
-Xwchart=char	wchar_tの定義はchar
-Xwchart=schar	wchar_tの定義はsigned char

注: sizetには符号付きタイプは許可されません。

注: コントロール変数wchart（上述の説明）と、コントロール変数cの値としてのwchar_tとの区別に注意してください。44ページの「c:C/C++言語モード」を参照してください。

どちらのコントロール変数もコンパイルスコープを持ち、上記に説明する名前値を受け入れます。デフォルト値は、sizet=uint および wchart=ushort です。

inclpath: include ファイルの検索

#include ステートメント内のファイル名に対するディレクトリ検索の順番には、広く普及している UNIX の習慣がありますが、ひとつだけ変わったケースがあります。このケースは、それ自体が別の #include ステートメントでインクルードされているファイル内にある #include ステートメント内の、引用符に囲まれた箇所に相対ファイル名がある場合です。最近の UNIX の実装ではこの検索は通常、処理されている #include ステートメントのあるファイルを含むディレクトリから開始されます。以前の UNIX の実装ではこの検索が、元の (最上位の) ソース ファイルを含むディレクトリから開始されていました。また、ANSI C の規格に記載されている規格委員会のコメントには、以前のアプローチが「原則として」が好ましいとされていますが、実際は規格で指定はされていません。

SNC-C では、inclpath コントロール変数を使用してこれを選択します (以下を参照)。

`-Xinclpath=relative`	C では、引用符で区切られた相対ファイル名を使用している #include ステートメントで指定されたファイルを検索する際に、処理中の #include ステートメントを含むファイルが格納されているディレクトリを、最初に検索します。
`-Xinclpath=absolute`	C では、引用符で区切られた相対ファイル名を使用している #include ステートメントで指定されたファイルを検索する際に、元の (最上位の) ソース ファイルが格納されているディレクトリを最初に検索します。

inclpath コントロール変数はファイルスコープを持ち、relative または absolute の値を使用します。デフォルト値は inclpath=relative。

C++コンパイル

このセクションでは、特にC++に関連するコントロール変数について説明します。

C++規格別表現

コンパイラが認識するC++の規格別表現はコントロール変数cによって制御され、これについては44ページの「c:C/C++言語モード」で説明しています。53ページの「C++言語定義」も参照してください。

一般的なコード制御

このセクションでは、プログラム コードの一般的な制御に関連するコントロール変数について説明します。

bss: .bss セクションの使用

リンク時に、特定のゼロ以外の値での初期化を必要としないデータ アイテムは、.data セクションや .bss セクションに配置することができます。バイナリ プログラムでは、このようなアイテムを .bss に配置することによってサイズが小さくなりますが、互換性を考慮すると、.data セクションに配置する必要があります。これは、bss コントロール変数を使用して制御します (以下を参照)。

-Xbss=0	すべてのデータを .data セクションに配置する。
-Xbss=1	初期化されていない静的データおよびゼロで初期化されたデータを、コンパイラ内の自動規則に従って .data セクションまたは .bss セクションに配置する。
-Xbss=2	初期化されていない静的データを .bss セクションに配置し、ゼロで初期化されたデータを、可能な限り .bss セクションに配置する。

bss コントロール変数は関数スコープを持ち、0 ～ 2 の値を使用します。デフォルト値は bss=1。

<reg>reserve: マシン レジスタの予約

SNC コンパイラでは、割り当てプールから個々のレジスタを削除することができます。これにより、これらのレジスタを使用するコードをコンパイラが生成することを抑制できます。そうすると asm ステートメントにレジスタ番号を埋め込むなどの方法で、そのレジスタを自由に使用できるようになります。

メモ: この機能は、その時点でのコンパイル ユニットに対してのみ有効になります。別のユニットやライブラリをコールした場合は、予約したこれらのレジスタを使用するコードが実行される可能性があります。

この方法では、「一般的」な用途のレジスタのみを予約できます。ターゲットの構造によっては、一部のレジスタの用途が決められているため、この方法で予約できない場合もあります。たとえば、スタック ポインタに関連付けられたGPR は、その用途が、ターゲット構造のコード生成に対する固有のもののため、予約することはできません。特別な目的を持つレジスタを予約すると、以下のようなエラー メッセージが表示されます。

```
Command line error: Illegal attempt to reserve <regclass> register number
<num>
```

上記の <regclass> は {fpr | gpr }、<num> はレジスタ番号になります。

ここでは、以下のコントロール変数を使用します。

-Xfprreserve=*list*	*list* で指定した浮動小数点レジスタを予約する。
-Xgprreserve=*list*	*list* で指定した汎用整数レジスタを予約する。
上記の*list* は、レジスタ番号のリストまたは範囲ペアとなります。範囲ペアにはコロンを使用し、リストの複数エレメントはプラス記号で区切ります。たとえば「-Xgprreserve=4+21:27」では、汎用レジスタ 4 および 21～27 を予約します。	

<reg>reserve コントロール変数はファイルスコープを持ちます。デフォルトの値は<empty list>すなわち、どのレジスタも予約されていません。

g: シンボルのデバッグ

シンボルのデバッグ情報は、以下のように g コントロール変数を使用することにより、コンパイラで生成されるアセンブリ ファイルに含まれます。

-Xg=0	シンボル デバッグ情報をアセンブリ ファイルに含めない。
-Xg=1,2	SN シンボル デバッガーで使用するシンボル デバッグ情報を、アセンブリ ファイルに含める。

g コントロール変数はコンパイルスコープを持ち、0 ～ 2 の値を使用します。デフォルト値は g=0。

writable_strings: 文字列の読み取り専用ステータスの設定

文字列の処理は、特に、メモリ文字列を配置するセクション、およびそのセクションが読み取り専用か書き込み可能かの点において、言語やターゲットによってその方法が異なります。また、「小規模データ」セクションのあるホストでは、そのセクションへのデータの配置を制御するための機能が用意されている場合があります。writable_strings コントロール変数では、文字列の配置先を制御することができます。

writable_strings は、他のすべてのコントロール変数と同様に、コマンドラインや、コードに挿入するプラグマで使用できます。すべての文字列を読み取り専用や書き込み可能にする場合は、コマンドラインでの使用が特に便利です。プラグマで使用した場合は、個々の文字列の書き込み可能ステータスを正確に制御できるため、ほとんどの文字列を読み取り専用メモリに配置する (一部のシステムでは特に便利) と同時に、一部の文字列を書き込み可能にし、コードによってその文字列が実際に変更される場合のメモリ エラーを防止することができます。

-Xwritable_strings=0	文字列を、読み取り専用のデータ セクション (.rdata など) に強制的に配置する。
-Xwritable_strings=1	文字列を、ターゲットに応じたデータ セクションに配置する。コントロール変数 c の値が knr の場合、これは通常 .data になり、それ以外の場合は .string セクション (ターゲット上に存在する場合) に配置されます。ターゲット上に .string セクションがない場合、ターゲットの規則に従って、文字列は .data セクションまたは .rdata セクションに配置されます。
-Xwritable_strings=2	文字列を、書き込み可能のデータ セクション (.data など) に強制的に配置する。

writable_strings コントロール変数はラインスコープを持ち、0 ～ 2 の値を使用します。デフォルト値は writable_strings=0。

その他の制御

このセクションでは、コンパイル システムの一般的な制御を行うためのコントロール変数について説明します。

mserrors: エラー/警告のソース行の非表示

SNC コンパイラではデフォルトで、すべてのエラーと警告にソース行がプリントされますが、Visual Studio ではこれが不要であると同時に、Visual Studio のタスク リストでエラーを確認することが困難になります。mserrors コントロール変数を使用することにより、エラーと警告でのソース行表示を抑制できます (以下を参照)。

-Xmserrors=0	エラーと警告にソース行をプリントする。
-Xmserrors=1	エラーと警告にソース行をプリントしない。

このコントロール変数は、Visual Studio の SNC コンパイラのすべてのビルドで自動的に有効化されますが、SNC コンパイラをコールするカスタムのビルド ステップを作成した場合は、手動でこのスイッチを追加してください。

たとえば、このスイッチを使用しない場合は以下のようになります。

```
test.c(11,6): warning: variable "a" was declared but never referenced
    int a =1 ;
        ^
```

-Xmserrors スイッチを使用した場合は、同じ警告が以下のようになります。

```
test.c(11,6): warning: variable "a" was declared but never referenced
```

progress: コンパイルのステータス

コンパイルにステータスに関しては、処理中の最適化に関する情報に加え、補足情報を表示することもできます。これらの補足情報は、progress コントロール変数を使用して制御します。

この情報には 2 つの基本タイプがあり、1 つはソース コードのコンパイル状況に応じた補足情報の表示に関するもの、もう 1 つは、ソース コードに対して実行された最適化に関する情報となっています (以下を参照)。

-Xprogress=files	各ファイルのコンパイル開始時に進行状況を表示する。
-Xprogress=functions	各関数のコンパイル開始時に進行状況を表示する。
-Xprogress=phases	コンパイルの各フェーズ開始時に進行状況を表示する。
-Xprogress=subphases	コンパイルの各サブフェーズ開始時に進行状況を表示する。
-Xprogress=actions	コンパイラでの各主要処理 (インライン化など) の終了時に進行状況を表示する。
-Xprogress=failures	コンパイラでの各主要処理のエラー (関数のインライン化失敗など) を表示する。
-Xprogress=templates	テンプレート関数のインスタンス化を表示する。
-Xprogress=memory	進行状況の表示に、コンパイラのメモリ使用率情報を含める。
-Xprogress=sizes	進行状況の表示に、内部のコンパイラ データ構造のサイズ情報を含める。
-Xprogress=realtime	進行状況の表示に、コンパイラで使用された実時間を含める。
-Xprogress=rtime	進行状況の表示に、コンパイラで使用された実時間を含める (realtime と同じ)。
-Xprogress=usertime	進行状況の表示に、コンパイラで使用されたユーザー時間を含める。
-Xprogress=utime	進行状況の表示に、コンパイラで使用されたユーザー時間を含める (usertime と同じ)。
-Xprogress=%all	コンパイルのすべての可能なポイントで進行状況を表示する。
-Xprogress=%none	コンパイラの進行状況を表示しない。

「名前」タイプのその他のコントロールと同様に、このコントロールでも複数の値に対して「-Xprogress=actions+failures+files」などのリスト形式を使用できます。

progress コントロール変数はコンパイルスコープを持ち、上記リストの値を使用します。デフォルト値は progress=files。

show: コントロール変数の値出力

指定したコントロール変数の値は、-Xshow コマンドライン スイッチを使用して stdout に配置することができます。このコントロール変数はコマンドラインでのみ指定でき、プラグマで指定した場合は無効になります。

show コントロール変数はコンパイルスコープを持ち、値を表示する他のコントロール変数の名前のリストを、値として使用します。デフォルトの値は空のリストです。

5: 言語定義

言語の定義

このセクションでは、SNC コンパイラで可能な、プログラム言語 C および C++ の定義に対する制御について説明します。

C言語定義

SNC-C には、コンパイラで使用できる C 言語の「方言」を、c コントロール変数の値に応じて制御するための、3 つのモードが用意されています。

- ANSI モード。このモードでは、コンパイラはANSI C 規格 (ANSI X3.159-1989 および ISO/IEC 9899:1990(E))の「規格合致ホスト処理系」に完全に従います。つまり、すべての標準ヘッダ ファイルに対応します。
- K&R モード。このモードは、『*The C Programming Language*』(Kernighan & Ritchie 著)（邦訳: プログラミング言語C）に記載されている C 言語の定義と高い互換性があり、UNIX pcc コンパイラとの互換性も高くなっています。
- ミックス モード。このモードのコンパイラは、既存の K&R コードを ANSI に移植する作業を支援するための拡張機能がいくつか追加されているという点を除き、基本的に ANSI コンパイラとなります。

デフォルトのミックス モードは、ANSI モードと以下の点で異なります。

- 一部のメッセージがエラーから警告に降格される。
- alloca 関数が組み込み関数として認識され、通常の K&R 定義を使用して実装される。

値 c99 を ansi、K&R、ミックス モードに追加することにより、ISO/IEC 9899:1999 C プログラミング標準規格で追加された C 言語機能を有効にできます。デフォルトはオン。

K&R モードには const、volatile、signed の値を追加でき、コンパイラではこれがキーワードとして認識されます。C モードには inline 値を追加でき、コンパイラではこれがキーワードとして認識され、C++ と同様に処理されます。

上記 3 つのすべての C モードでは、23 ページの「コンパイラの制御」に記載されているプラグマ ステートメントを使用できます。

ANSI モードとミックス モードでは、noknr の値を追加 (-Xc= ansi+noknr など) することにより、プロトタイプ化されていない関数の各定義や各宣言で、コンパイラから警告を出力できるようになります。noknr モードを有効にした場合は、宣言または定義されていない関数の使用を検出した場合も警告が出力されます。このモードは、すべての関数を検索してプロトタイプ化されたバージョンに変更し、コードを「きれい」にする場合に使用できます。

ANSI C/C++ では、int タイプのビットフィールドが「実装時に定義」となります。ビット フィールドの動作は、PlayStation®3 PPU ABI の仕様に従います。

ANSI C/C++ では、char の表現が「実装時に定義」となります。SNC-C/C++ では、char コントロール変数を使用して符号付き char と符号なし char を切り替えます。

C++言語定義

このセクションでは、C++言語の定義を制御する方法について説明します。

コンパイラのフロント エンドでは、ISO/IEC 14882:1998 規格で定義され、この規格に対して TC1 で改訂された C++ 言語が受け入れられます。また、フロント エンドには 4 つの「方言」互換モードが用意されているため、これら方言を使用するプログラマーはそれぞれの既存コードを継続してコンパイルできます。ただし、完全な互換性は保証されておらず、また意図もされていません。特に、SNC コンパイラの使用時には、ネイティブで生成されるコンパイラ エラーが、異なるエラーとなることや、エラーがまったく発生しないこともあり得ます。

方言

SNC-C++ には、コンパイラで使用できる C 言語の「方言」を、c コントロール変数の値に応じて制御するための、4 つのモードが用意されています。

- ARM モード。これは、SNC-C++ における厳密な ANSI モードです。このモードには当初、C++ 規格の ISO/IEC 14882:2003 のベースとなった『*The Annotated C++ Reference Manual* (the ARM)』(Margaret A. Ellis & Bjarne Stroustrup 著) の記載に沿った言語として実装されました。またこのモードは、コンパイラ ドライバの -Xc=arm オプションを使用して有効化します。
- CP モード。これは上記の ARM と同様ですが、複数の制限が緩和されています。このモードは、SNC C コンパイラ (.c ファイルと .C ファイル) および SNC C++ コンパイラ (.cpp ファイル) での デフォルト値となっています。またこのモードは、-Xc=cp オプションを使用して有効化します。
- Cfront 2.1 モード。このモードでは、コンパイラが AT&T Cfront version 2.1 に対応します。またこのモードは、-Xc=cfront:21 オプションを使用して有効化します。
- Cfront 3.0 モード。このモードでは、コンパイラが AT&T Cfront version 3.0 に対応します。またこのモードは、-Xc=cfront または -Xc=cfront:30 を使用して有効化します。

これらすべての方言では、テンプレート化されていないバージョンのライブラリがデフォルトで使用されます。

この C++ コンパイラは、規格にほぼ対応したものとなっており、例外、RTTI (実行時のタイプ認識)、テンプレート、名前空間、ライブラリ (STL：Standard Template Library を含む) に対応しています。また、ブールのキーワードと wchar_t も認識されます。認識する言語定義の制御や変更に関しては、44 ページの「c:C/C++言語モード」を参照してください。

デフォルトでは、いくつかの新機能が有効になっていますが、c コントロール変数の値を変更することによって個々に無効化できます。その場合に使用できる値は、すべて上記のリファレンスに記載されていますが、以下のセクションに補足説明を記載します。

例外処理

例外処理には、コンパイル システムに対して以下の 2 つの大きな影響があります。

1. 明示的な例外処理のコンストラクトとキーワード (try、throw、catch など) の認識とコード生成。
2. 例外処理コンストラクトがなく、コードで例外が発生した際にクリーンアップが必要となるローカル変数のあるコードにおける、コードやテーブルの生成。これは、例外に関連しないコードのスタック フレームが、例外処理の一部としてアンワインド (巻き戻し) される実行スタックの部分に含まれる場合に発生します。コンパイラに対する影響は、破壊を要するローカル変数を持つすべての関数に、デストラクタ コールが必要になるということです。

上記の (1) は、例外の設定にかかわらず実行されます。例外では、(2) を実行するかどうかが制御されます。(2) の実装を使用することにより、実行時に実際には例外がスローされない場合のコストが低減します。現在、データ スペースの一部の追加テーブル、および small 整数定数のローカル変数への余分な割り当てがコストとなっています。またこのコストは、デストラクタのあるローカル変数を持つ関数のみで発生します。

SNC コンパイラでは、すべてモードで例外処理が無効 (デフォルト) になっています。例外を使用する場合は、コマンドラインで -Xc=*mode*+exceptions を指定して有効にします (44 ページの「c:C/C++言語モード」を参照)。

特殊コメント

C ソース コードに配置すてコンパイラで生成される警告メッセージを制御するためのコメントには、以下の 3 つの種類があります。

`/*NOTREACHED*/`	コンパイラが処理できないコード ブロックの最初に挿入した場合、警告メッセージが抑制される。
`/*VARARGSn*/`	後続する関数の宣言における、通常の可変引数の数のチェックが抑制される。最初の n 引数のデータ タイプがチェックされますが、n が省略された場合は値としてゼロが使用されます。
`/*ARGSUSED*/`	関数内の使用されない引数に関する警告が抑制される。

上記 3 つのコメントではすべて、大文字と小文字が区別されます。

定義済みシンボル

特定のプロセッサ シンボルは、コンパイラによって定義済みとなっています (詳細は、33 ページの「定義済みのマクロの使用」を参照)。これらのシンボルの一部 (`__STDC__` など) は、目的のコンピュータ環境を問わず定義されますが、その他のシンボルは、適切な環境のみで定義されます。言語モードに関する詳細は、44 ページの「c:C/C++言語モード」を参照してください。

すべての言語モード (C および C++) で定義済み

- `__STDC__` シンボルを除き、ANSI C 規格で指定されているすべての定義済みシンボル、および `__SNC__` シンボル

knr C モードを除くすべての言語モードで定義済み

- `__STDC__` (ansi C モード、arm C++ モード、cp C++ モードでは 1 の値で定義、mixed C モードおよび cfront C++ モードでは 0 の値で定義).

すべての C++ モードで定義済み

- `__cplusplus`

グローバルな静的インスタンス化の順番の制御

起動時にインスタンス化を必要とするグローバル オブジェクトに対しては、*init_priority* 属性を使用することにより、コンストラクタがコールされる順番を制御することができます。範囲は 101-65535 となっており、割り当てた値が低いほどコンストラクトの優先度が高くなります。また、デフォルト値は 65535 (init_priority 属性なし) で、0-100 の値は予約済みです。

以下はその構文です。

```
<object type> <object name> __attribute__((init_priority( x )));
```

例

```
foo myfoo1 __attribute__((init_priority( 110 )));
foo myfoo2 __attribute__((init_priority( 101 )));
foo myfoo3;       // effective priority is 65535
```

これらのオブジェクトは、myfoo2、myfoo1、myfoo3 の順番で、起動時にインスタンス化されます。

__restrict キーワード

SNC は __restrict キーワードの拡張形式に対応しています。このため、C または C++ ソースコードでポインタを使用する場合に、エイリアスの問題を制御できます。

例:

```
void VectorAdd ( float *Result, const float *Src1, const float *Src2 )
{
  Result[0] = Src1[0] + Src2[0] ;
  Result[1] = Src1[1] + Src2[1] ;
  Result[2] = Src1[2] + Src2[2] ;
} ;
```

この関数はシンプルに見えますが、Result が Src1 および Src2 と同じまたは重複したメモリを指した状態でこのコードを呼び出すことを、この言語は許しています。このため、結果の一部をストアすると 1 つまたは両方のソース配列が上書きされる可能性があるため、コンパイラは加算演算を個別に生成する必要があります。こうした条件では、Result は Src1 と Src2 をエイリアスする可能性があると言います。Result を保存しても入力値が変更されないことをコンパイラが理解している場合は、高速なコードを生成することができます。

SNC を使うと、キーワード '__restrict' を C および C++ ソースの両方で使用し、ポインタのエイリアシングを制御できます。さらに、コンパイラの制御によって関数パラメータが暗黙の __restrict 修飾子を持っているいるものとして自動的にタグ付けすることができます。__restrict キーワードを使用する際には、C99 標準 'restrict' キーワードの構文と規則に従います。これは、ポインタ宣言に追加できる修飾子です。例:

```
float * __restrict pParams;
void VectorAdd ( float * __restrict Result,
const float * __restrict Src1,
const float * __restrict Src2 )
```

__restrict キーワードで実行される操作は -Xrestrict コントロール変数で制御されます。

-Xrestrict=0	__restrict キーワードの影響を受けない。ポインタが他のポインタとエイリアスすると仮定する。
-Xrestrict=1	__restrict で修飾されたポインタが他の __restrict 修飾ポインタをエイリアスしないと仮定する。
-Xrestrict=2	__restrict で修飾されたポインタが他のポインタをエイリアスしないと仮定する。

関数パラメータの自動修飾は、-Xparamrestrict コントロール変数を使って次のように制御できます。

-Xparamrestrict=0	ポインタ タイプの関数パラメータを、__restrict 修飾子で修飾しない。

-Xparamrestrict=1	ポインタ タイプの関数パラメータを、__restrict 修飾子で修飾する。

設定されると、この制御は __restrict 修飾子の付いたポインタタイプの関数パラメータを自動的に装飾します。この制御は pragma 機能を使ってソース内で変更することもできます。例:

```
#pragma control %push paramrestrict
#pragma control paramrestrict=1

// this function will assume __restrict on its parameters

void qaz ( float * dest, float * src, float * src2 )

{

  dest[0] = src[0] + src2[0] ;
  dest[1] = src[1] + src2[1] ;
  dest[2] = src[2] + src2[2] ;

}

#pragma control %pop paramrestrict

// this function will not assume __restrict

void qaz0 ( float * dest0, float * src0, float * src02 )

{

  dest0[0] = src0[0] + src02[0] ;
  dest0[1] = src0[1] + src02[1] ;
  dest0[2] = src0[2] + src02[2] ;

}
```

__unaligned キーワード

__unaligned キーワードは、ポインタ定義におけるタイプ修飾子です。ポインタが __unaligned 修飾子を使用して宣言されるとコンパイラでは、このポインタが、不適切にアラインされたデータを指すと仮定されます。__unaligned で宣言されたポインタを通じてデータへのアクセスが行われる場合に、アライメント エラーを引き起こすことなくデータの読み書きが行えるようにするため、必要な追加コードがコンパイラによって生成されます。この追加コードの使用はパフォーマンス面に悪影響を与えるため、可能な限り、データが適切にアラインされるようにすることが最善策となります。

Cell PPU プロセッサにおける浮動小数点やベクター データ タイプの読み書きでは、正しくアラインされていることが必要なため、正しくない場合は例外が発生します。このハードウェアでは、不適切なアラインの整数タイプに対しても、正しくアクセスが行われます。

この修飾子は、アドレスが指定されたデータのアラインメントのみが対象となり、ポインタ自体はアラインされていると仮定されます。

たとえば、以下のコード部分では、不正にアラインされたアクセスが意図的に作成されますが、_unaligned キーワードを使用することにより、コードは問題なく実行されます。

```
float read (float __unaligned * f)
{
    return *f;
}

char x [5];
float f;

void foo (void)
{
   f = read (&x [1]);
}
```

__may_alias__ 属性

__may_alias__ 属性でマークされたタイプのオブジェクトへのアクセスは、タイプ ベースのエイリアス分析の影響を受けませんが、代わりに char タイプと同様に、その他任意のタイプのオブジェクトをエイリアスすることができると想定されます。これは事実上、__restrict と正反対になります。-Xrelaxalias=2 によって有効化される、より厳密なタイプ ベース (C99) のエイリアシング ルールでコンパイルする際に、__may_alias__属性は、エイリアスするポインタを意図的にマークするために使用できます。

例:

```
typedef int __attribute__((__may_alias__))  int_a;

int main ()
{
    int local;
    int_a *local_ptr = &local;
}
```

Microsoft __fastcall と __stdcall 拡張

SNC コンパイラでは、__fastcall と __stdcall 修飾子に対応しています。以下は構文例です。

```
// 関数プロトタイプ
void __fastcall foo();

// fastcall 関数ポインタ
void(__fastcall *call_to_foo)();
```

PS3 ターゲットに対する fastcall 命令では、プロシージャ記述子の使用回避と、ポインタによる関数アドレスへの直接関数コールが実行されます。Fastcall 命令を使用すると、コールへの間接性が削除されるため、および TOC 復元に必要な命令が削除されることによってコード サイズが削減されるため、パフォーマンスを向上させることが可能になります。

関数記述子は 2 語から成るデータ構造で、関数のエントリ ポイント アドレスを説明する 1 語と、関数の TOC ベース アドレスを説明する 1 語が含まれます。これらの関数記述子は、オブジェクト ファイルの opd セクションに含まれます。関数記述子について詳しくは、PS3 PPU ABI ドキュメント (cell\SDK_doc\en\pdf\OS_lowlevel\PPU_ABI-Specifications_e.pdf) を参照してください。

メモ: __fastcall コール仕様は GCC との互換性がないことに注意が必要です。また、fastcall 関数ポインタは、GCC でコンパイルされたコードに渡すことはできません

- 「stdcall」では、プロシージャ記述子を通じて間接的に関数をコールする。
- 「fastcall」では、ポインタ経由で関数を直接コールする。

標準的な関数コールの例:

```
Code:
int bar();
typedef int (*func_ptr)();
func_ptr ptr;
int foo()
{
  return ptr();
}
Output:
.Z8foov:
...
  std     %rtoc,40(%sp)   # 現在の TOC 値を保存
  lwz     %r5,0(%r4)      # 関数記述子から関数アドレス
                          # をロード
  lwz     %rtoc,4(%r4)    # 関数記述子から TOC 値
                          # をロード
  mtctr   %r5             # CTR を関数アドレスにセット
  bcctrl 20,30            # 関数への分岐
  ld      %rtoc,40(%sp)   # TOCを復元
...
```

__fastcall 関数コールの例:

```
Code:
int __fastcall bar();
typedef int (__fastcall *func_ptr)();
func_ptr ptr;
int foo()
{
  return ptr();
}
Output:
.Z8foov:
...
  lwz     %r4,fastptr@l(%r4)   # 関数アドレスをロード
  mtctr   %r4                  # CTR を関数アドレスにセット
  bcctrl 20,30                 # 関数への分岐
...
```

6: プリコンパイル済みヘッダ

プリコンパイル済みヘッダ

ヘッダファイルのセットをリコンパイルすることを避けたい場合がよくあります。特に、ヘッダファイルによって多数のコード行が含まれるようになり、それらをインクルードする主ソースファイルが比較的小さい場合などがそうです。EDGフロンエンドはこのメカニズムを提供し、実際には特定の時点におけるコンパイルの状態のスナップショットを保存し、コンパイル完了前にそれをディスクファイルに書き込みます。その後、同じソースファイルをリコンパイルするときや、別のファイルを同じヘッダファイルセットでコンパイルするときには、この「スナップショット」が認識され、対応するプリコンパイル済みヘッダ（PCH）ファイルが再使用されてそれが読み込まれます。状況が正しければ、これによりコンパイル時間が著しく向上します。そのトレードオフは、PCHファイルが大量のディスクスペースを使用するという点です。

--pchスイッチの全一覧は、15ページの「プリコンパイル済みヘッダ」を参照してください。

自動によるプリコンパイル済みヘッダ処理

コマンドラインに--pchがあると、プリコンパイル済みヘッダの自動処理が有効になります。これにより、フロントエンドが、読み込み可能な有効なプリコンパイル済みヘッダファイルを自動的に検索し、また、後続のコンパイルで使用するためこれを自動的に新規作成するようになります。

PCHファイルには、ヘッダ停止ポイントより前のすべてのコードのスナップショットが含まれます。ヘッダ停止ポイントは通常、主ソースファイル内の前処理ディレクティブに属しない最初のトークンですが、（それより先を指定する場合は）#pragma hdrstopで直接指定できます。

たとえば：

```
#include "xxx.h"
#include "yyy.h"

int i;
```

ヘッダ停止ポイントはint（最初の非プリプロセッサトークン）であり、PCHファイルには、xxx.hとyyy.hのインクルードが反映されたスナップショットが書き込まれます。最初の非プリプロセッサトークンまたは#pragma hdrstopが#ifブロック内にある場合、ヘッダ停止ポイントは、ブロックを囲む最も外側の#ifになります。

たとえば：

```
#include "xxx.h"
#ifndef YYY_H
#define YYY_H 1
#include "yyy.h"
#endif

#if TEST
int i;
#endif
```

ここでは、前処理ディレクティブに属さない最初のトークンはやはりintですが、ヘッダ停止ポイントは、それを含む#ifブロックの開始点です。PCHファイルにはxxx.hのインクルードが反映され、条件

付きでYYY_Hの定義とyyy.hのインクルードが反映されます。しかし、#if TESTにより生成される状態は含まれません。

PCHファイルは、ヘッダ停止ポイントとそれに先行するコード（主に、ヘッダファイル自体）が、以下の一定の条件を満たす場合にのみ生成されます。

1. ヘッダ停止ポイントはファイルスコープで出現しなければならない。ヘッダ停止ポイントが、ヘッダファイルによって確立された閉じていないスコープ内にあってはならない。たとえば、次の場合にはPCHファイルは作成されない。

```
// xxx.h
class A {
// xxx.C
#include "xxx.h"
int i; };
```

2. ヘッダ停止ポイントは、ヘッダファイル内で開始された宣言の内部にあってはならず、（C++では）リンク指定の宣言リストの一部であってはならない。たとえば、次の例ではヘッダ停止ポイントはintであるが、これは新しい宣言の開始ではないため、PCHファイルは作成されない。

```
// yyy.h
static
// yyy.C
#include "yyy.h"
int i;
```

3. 同様に、ヘッダ停止ポイントは、ヘッダファイル内で開始された#ifブロックまたは#defineの内部にあってはならない。
4. ヘッダ停止ポイントの先行部分の処理にエラーがあってはならない。

> **注**：PCHファイルを再使用するとき、警告やその他の診断は再生成されません。定義済みマクロ__DATE__または__TIME__への参照があってはなりません。

5. #line前処理ディレクティブがまったく使われていないこと。
6. #pragma no_pchが出現していないこと。
7. ヘッダ停止ポイントに先行するコードにより、プリコンパイル済みヘッダに関連するオーバーヘッドを正当化するための十分な数の宣言が提示されていること。

プリコンパイル済みヘッダファイルが生成されると、このファイルにはコンパイル状態のスナップショットに加え、どのような状況でそれを再使用できるか判断するためチェックできる情報も含まれています。これには以下の情報が含まれます。

- コンパイラのバージョン。コンパイラがビルドされた日付と時刻も含む。
- カレントディレクトリ（すなわち、コンパイルが行われているディレクトリ）。
- コマンドラインオプション。
- 主ソースファイルからの前処理ディレクティブの初めのシーケンス。#includeディレクティブも含む。
- #includeディレクティブに指定されるヘッダファイルの日付と時刻。

この情報がPCHプレフィックスを構成します。ソースファイルのプレフィックス情報がPCHファイルのプレフィックス情報と比較され、後者が現行コンパイルに対して適切かどうか決定されます。例として、次の2つのソースファイルを考えます。

```
// a.C
#include "xxx.h"
... // コードの開始
// b.C
```

```
#include "xxx.h"
... // コードの開始
```

a.Cが--pchでコンパイルされると、a.pchという名前のプリコンパイル済みヘッダファイルが作成されます。その後、b.Cがコンパイルされる(またはa.Cがリコンパイルされる)と、現行ソースファイルと比較するためにa.pchのプレフィックス部が読み込まれます。コマンドラインオプションが同一である、xxx.hが変更されていないなどの条件が満たされると、xxx.hを開いてその行を1つずつ処理する代わりに、フロントエンドがa.pchの残りを読み取り、それによってコンパイルの残りの状態が確立されます。あるコンパイルに対して複数のPCHファイルが適用可能であることがあります。その場合は、最も大きいファイル(つまり、主ソースファイルからの最も多くの前処理ディレクティブを表すもの)が使用されます。たとえば、以下のコードで始まる主ソースファイルを考えます。

```
#include "xxx.h"
#include "yyy.h"
#include "zzz.h"
```

xxx.h用に1つのPCHファイルがあり、xxx.hとyyy.h用にもう1つのPCHがあった場合、後者が選択されます(どちらも現行コンパイルに有効であると仮定する)。さらに、最初の2つのヘッダ用のPCHファイルが読み込まれ、3つ目のヘッダがコンパイルされた後、3つ全部のヘッダ用に新しいPCHファイルが作成されることもあります。

プリコンパイル済みヘッダファイルが作成されるとき、その名前として主ソースファイルの名前が取られ、その拡張子が.pchで置き換えられます。--pch_dirが指定されない限り、このファイルは主ソースファイルのディレクトリに作成されます。プリコンパイル済みヘッダファイルが作成または使用されると、次のようなメッセージが発行されます。

```
"test.C":creating precompiled header file "test.pch"
```

このメッセージは、コマンドラインオプション--no_pch_messagesを使って抑制できます。

--pch_verboseオプションを使用すると、フロントエンドにより、使用できないプリコンパイル済みヘッダファイルそれぞれについてメッセージが表示され、その理由も示されます。

自動モード(つまり、--pchを使用した場合)では、フロントエンドは以下の状況下では、プリコンパイル済みヘッダファイルが古くなったと判断し、それを削除します。

- プリコンパイル済みヘッダファイルが、現行コンパイルには適用可能であるが、少なくとも1つの古いヘッダファイルに基づいている場合。または
- プリコンパイル済みヘッダファイルが、コンパイルされているソースファイルと同じ基底名を持つが(たとえば、xxx.pchとxxx.C)、現行コンパイルには適用可能でない場合(たとえば、コマンドラインオプションが異なるため)。

これにより、一部のよくある場合が処理されます。その他のPCHファイルクリーンアップは、ユーザーが処理する必要があります。1つのコンパイルで複数のソースファイルが指定された場合、プリコンパイル済みヘッダ処理はサポートされません。コマンドラインにプリコンパイル済みヘッダ処理の要求があり、複数の主ソースファイルが指定された場合は、エラーが発行されてコンパイルは中止されます。

手動によるプリコンパイル済みヘッダの処理

- コマンドラインオプション--create_pch=*ファイル名*は、指定されたプリコンパイル済みヘッダファイルを作成するように指示する。
- コマンドラインオプション--use_pch=*ファイル名*は、指定されたプリコンパイル済みヘッダファイルをこのコンパイルに使用するように指示する。指定が無効であった場合(たとえば、そのプレフィックスが現行主ソースファイルのプレフィックスと一致しない場合)は、警告が発行され、そのPCHファイルは使用されない。

これらのオプションのいずれかを--pch_dirと組み合わせて使用すると、指定したファイル名（またはパス名）が、それが絶対パス名である場合を除き、ディレクトリ名に付加されます。

--create_pch、--use_pch、--pchのオプションを一緒に使用することはできません。これらのオプションを複数指定した場合、最後のオプションだけが適用されます。ただし、自動PCH処理の説明のほとんどが、これらのモードのいずれかに適用されます。たとえば、ヘッダ停止ポイントは同様の方法で決定され、PCHファイルの適用性も同様の方法で決定されます。

PCH ファイルの同一ディレクトリ チェックの変更

- コントロール変数「-Xpch_override」は、PCH ファイルの生成に使用されるファイルが、コンパイルされるファイルと同じディレクトリにあるかどうかをチェックする、コンパイラの機能をオフにします。

コンパイラには、PCH ファイルを使用したコンパイルが、PCH ファイルを使用しないコンパイルとまったく同様に処理されることを確実にするための、多くのチェック機能が実装されています。これらのチェック機能に関する詳細は、59 ページの「自動によるプリコンパイル済みヘッダ処理」を参照してください。しかし、1 つの一貫性チェックで、PCH ファイルで一般的に使用される慣用法が妨害されるということが確認されています。

このチェックでは、PCH ファイルの生成に使用されるファイルが、コンパイルされるファイルと同じディレクトリにあるかどうかが確認されます。これにより、異なるディレクトリにあるファイルが PCH 情報を共有することを防止することができます。この場合コンパイラでは、以下の警告メッセージが出力されます。

```
"the file being compiled needs to be in the same directory as the file
used to create the PCH file"
```

また、指定した PCH ファイルは使用されませんが、コンパイルは PCH を使用せずに継続されます。このチェックは、同じ名前で内容が異なるヘッダ ファイルが別のサブディレクトリで使用されている場合に必要になります。

以下はその例です。

```
src\
    src1\
      A.h
      foo1.cpp
    src2\
      A.h
      foo2.cpp
```

2 つのファイル「foo1.cpp」および「foo2.cpp」の両方が、この #include 行で A.h をインクルードする場合は、

```
#include "A.h"
```

PCH ファイルを使用せずにファイルをコンパイルすると、「foo2.cpp」は、「`src\src2`」にあるローカルの A.h ヘッダ ファイルを使用します。

しかし、以下のコマンドを発行した場合は、

```
ps3ppusnc -c src1\foo1.cpp -create_pch=foo1.pch
ps3ppusnc -c src2\foo2.cpp --use_pch=foo1.pch
```

2 つ目のコンパイルで PCH ファイルが使用されず、警告メッセージ (上記を参照) が出力されます。これは、PCH ファイルの生成に使用される「foo1.cpp」が、コンパイルされるファイル「foo2.cpp」と同じディレクトリにないためです。

このチェックは、コントロール変数「-Xpch_override=1」を以下のように使用することによってオーバーライドできます。

```
ps3ppusnc -c src2\foo2.cpp --use_pch=foo1.pch -Xpch_override=1
```

このコンパイルは正しく完了しますが、使用される A.h は「src2\」ディレクトリではなく「src1\」ディレクトリのものとなります。

- プロジェクトで、異なるサブディレクトリにある同じ名前のヘッダ ファイルを使用しない場合は、安全にコントロール変数 pch_override を使用することができます。

プリコンパイル済みヘッダの制御

プリコンパイル済みヘッダの作成と使用方法を制御したり調整したりする方法がいくつかあります。

#pragma hdrstopを、主ソースファイル内の前処理ディレクティブに属さない最初のトークンの前に挿入することができます。これにより、プリコンパイルの対象となるヘッダファイルのセットがどこで終了するかを指定できます。

たとえば、

```
#include "xxx.h"
#include "yyy.h"
#pragma hdrstop
#include "zzz.h"
```

この場合、プリコンパイル済みヘッダファイルには、xxx.hとyyy.hの処理状態が含まれますが、zzz.hの状態は含まれません。これは、#pragma hdrstopの後にあるコードで追加される情報が、別のPCHファイルを作成するに値しないと判断できる場合に有用です。

- #pragma no_pchを使い、特定のソースファイルのプリコンパイル済みヘッダ処理を抑制できる。
- コマンドラインオプション--pch_dir *ディレクトリ名*を使い、PCHファイルの検索や作成の対象となるディレクトリを指定できる。

パフォーマンスの問題

プリコンパイル済みヘッダファイルの書き出しと読み込みから生じる相対的なオーバーヘッドは、適度に大きいヘッダファイルに対してはかなり小さいものです。

一般に、プリコンパイル済みヘッダファイルを書き出すことの代償は、それが結局使用されない場合であっても小さく、*使用された*場合には、必ずコンパイルのスピードが速くなります。問題は、プリコンパイル済みヘッダファイルはかなり大きく、最小約250Kバイトから、数メガバイト以上にもなるということです。

従って、リコンパイルは速くなりますが、プリコンパイル済みヘッダ処理は、初めのシーケンスが不ぞろいの前処理ディレクティブのセットに対しては適切であると言えません。むしろ、複数のソースファイルが同じPCHファイルを共有できるときに、その効果を最大に利用できます。共有度が高いほど、ディスクスペースの消費が少なくなります。共有することにより、プリコンパイル済みヘッダファイルのサイズが大きいという不利な点を最小に抑え、コンパイル時間の著しいスピードアップという利点を維持できます。

従って、ヘッダファイルのプリコンパイルの利点を最大に利用するには、ソースファイルの#includeセクションを並べ直し、また、#includeディレクティブをよく使用されるヘッダファイルの中でグループにまとめるようにしてください。

それぞれのニーズは環境やプロジェクトによって異なりますが、一般には、プリコンパイル済みヘッダのサポートを最大限に利用するには、実験的に試してみることや、ソースコードに多少の変更を加えることが必要になります。

7: 最適化の手法

概要

SN Systems Compiler (SNC) には、複数の最適化フェーズが搭載されており、その多くは、PlayStation®3 専用にデザインされています。最適化用のコントロールは、GNU ツールチェーンのユーザーがすぐに慣れることができるような設計になっていますが、一部に相違点もあります。SNC の最適化機能を最大限活用するには、この特有の機能とコントロールになれることをお勧めします。この章では、この最適化機能に慣れるための方法を説明します。

多くの最適化機能は、ファイル単位でコマンドライン スイッチを使用してコントロール、または関数単位でコントロール プラグマを使用してコントロールできます。また SNC では、コードにアノテーションを付記することにより、属性やプラグマを使用して最適化機能により多くの情報を与えることもできます。GNU ツールチェーンまたは SNC の以前のバージョンを使用したことがあるユーザーにとって、これらの多くは使い慣れたものとなっていますが、一部は SNC for PlayStation®3 の新機能に関連しているため、慣れが必要になります。

メモ: SNC の新しいバージョンに移行する場合は、新しいスイッチやコード アノテーションによってコントロールできる改良点などが常に加えられているため、リリース ノートおよび重要な変更点に関するドキュメントを参照することをお勧めします。この章には、これらの変更点が反映されています。

GCC では、-O0 を使ってコンパイル、または最適化レベルを指定せずにコンパイルした場合、最適化機能は実行されませんでした。唯一実行されるインライン化は、強制的なインライン化のみでした。-O2 を使用してコンパイルした場合は多くの最適化が実行されますが、明確に指定して実行する必要がある最適かも多くあります。これは、コードにおける特定の仮定条件に最適化が依存する場合や、特定の環境でのみ最適化が有効な場合があるためです。SNC には「デバッグ可能な最適化」モードも用意されており、このモードを利用する場合は-Od を指定します。このモードでは、多くの最適化を実行できますが、ある程度のレベルでソース対応およびその他のデバッグ情報を残してください。-Os を使用してコンパイルするとサイズが最適化されますが、そのコードに対する仮定条件への依存のため、デフォルトでは一部の最適化は実行されません。

最適化を使用したビルドに対しては、サイズのパフォーマンスをバランスよく最適化するため、以下のベースライン設定をお勧めします。

```
-O2 -Xfastmath=1 -Xassumecorrectsign=1 -Xassumecorrectalignment=1
```

または

```
-Os -Xassumecorrectsign=1 -Xassumecorrectalignment=1
```

これらのスイッチは、以下のセクションで説明します。

主な最適化レベル

主な最適化レベルは、「-O<n>」スイッチ (<n> はレベル) を使用してコントロールします。

<n>	最適化
0	最適化なし。強制インライン化を除き、インライン化も実行されません。

1	一部の基本的な最適化。一部のインライン化が実行されます。
2	インライン化を含む、完全な一般的最適化。
3	インライン化を含む、完全な一般的最適化。現時点では、-O3 を使用した際に -O2 と同じ最適化が実行されますが、将来のバージョンでは、より長い時間が掛かる最適化が実行されるようになる可能性があります。
d	デバッグ可能な最適化モード。デバッグ情報の品質に影響を与えない最適化が実行されます。
s	サイズ最適化モード。コードのサイズを大きくする最適化は実行されず、インライン化の量も少なくなっています。

インライン化のコントロール

SNC には、インライン化をコントロールする 3 つの主なスイッチが用意されています。これらの値を調節することにより、コンパイル後のコードのサイズと実行速度を大幅に改善することができます。すべてのスタイルのコードに最適なデフォルト値を設定することは不可能なため、**コンパイルするコードに最適な値を見つけるために試行錯誤することを強くお勧めします。**

これらのコントロールのパラメータは、「命令」という意味での関数の最大サイズを表わします。これらは、コンパイラの内部命令のため、個々のプロセッサの命令と同じである必要はありません。

-Xautoinlinesize - 自動インライン化をコントロール

ソース コードでインラインとしてマークされていない関数を、コンパイラで自動的にインライン化する際の最大サイズを設定します。これは、ヘッダ ファイルのクラス内で定義されている C++ メソッドなど、黙示的なインライン関数には適用されません (65 ページの「-Xinlinesize -」を参照)。

詳細は、74 ページの「-Xautoinlinesize」を参照してください。

-Xinlinesize - 明示的インライン関数のインライン化をコントロール

このスイッチでは、コンパイラによってインライン化される明示的インライン関数の最大サイズを制限します。明示的インライン関数には、ヘッダー ファイルの内部クラスで定義される C++ メソッドが含まれます。

84 ページの「-Xinlinesize」を参照してください。

-Xinlinemaxsize - 任意の 1 つの関数へのインライン化の最大値をコントロール

このスイッチでは、任意の 1 つの関数へのインライン化の最大量をコントロールします。これは、個々の関数が大きくなりすぎて、最適化のほかのステージの処理速度が遅くならないようにするために使用します。この値をデフォルト値より大きくした場合、コンパイルには時間が掛かりますが、インライン化の量が増えます。その結果、パフォーマンスが向上する場合もあります。

83 ページの「-Xinlinemaxsize」を参照してください。

強制インライン化

「`__attribute__((always_inline))`」のマークが付けられている関数は、ほかのインライン化が実行されない場合、-O0 の使用時でもインライン化されます。

以下はその例です。

```
#define FORCE_INLINE __attribute__((always_inline))
FORCE_INLINE int timesTwo( int x )
{
                    return ( 2 * x );
}

int main()
{
                    return timesTwo( 3 );
}
```

最適なインライン化設定を見つける

当社のテストでは、プロジェクトに対して最適なインライン化設定を見つけることにより、コードのパフォーマンスとサイズが大幅に改善することがわかりました。最適な設定は、異なるコードベースによってさまざまに異なります。デフォルト値は、可能な限り多くのコードに共通して適度なパフォーマンスが実現するように設定されています。

- *インライン キーワードを多用するコード、またはクラス定義内で定義されている関数は、*-Xinlinesize=<n> の値を大きくすることによって効果が出る場合があります。-O2 のデフォルト値は 256 です。試行錯誤を行う場合は、512 または 1024 から開始することをお勧めします。
- *インライン化を実行するかどうかの判断をコンパイラに依存する設計のコードでは、*-Xautoinlinesize=<n> の値を大きくすることをお勧めします。-O2 のデフォルト値は 32 です。この値を大きくすると、自動インライン化が増加します。試行錯誤を行う場合は、128 から開始することをお勧めします。

これらの値を調節し、コードのサイズをパフォーマンスへの影響を確認して、値を増減してサイズとパフォーマンスの最適なバランスがとれる値を探してください。この「スイート スポット」により、サイズ、パフォーマンスともにデフォルト値より良い結果が出る場合がよくあります。

その他の最適化

最適化機能には、上記以外にも「-Xfastmath=1」スイッチを使用して実行する最適化があります。

これには、以下が含まれます。

- 浮動小数点、整数、VMX の間の変換を防ぐための VMX レジスタの自動使用。これにより、ロード ヒット ストアのペナルティを軽減できます。
- if 構文の「fsel」への変換。
- 「fdiv」の適切な除算および詳細化への変更。

これらの最適化は、「非正規」数などの浮動小数点値の限界動作に依存するコードでは適切に機能しない場合があるため、このスイッチはデフォルトでオフになっています。これは、大多数のコードには影響を与えません。

-Xfastmath=1 で有効にする最適化は、FLT_EPSILON の値未満の極端に小さい値による浮動小数点の除算に対して非常に敏感です。このため -Xfastmath=1 を有効にした場合は、除算の値が FLT_EPSILON の値より大きいことを常に確認することをお勧めします。

以下はその例です。

```
#include <float.h>

float divide( float x, float y )
{
                    if ( y < FLT_EPSILON )
                    {
                       y = FLT_EPSILON;
                    }

                    float z = x / y;

                    return z;
}
```

-Xfastmath は、最適化ビルドに対して可能な限り有効にし、このような限界条件に依存するコードが正常に機能するようにすることを強くお勧めします。

ポインタ演算の前提

PS3 のポインタ演算は、32 ビットのアドレス モデルを 64 ビットのアドレス モデルで実行するため、困難なものとなっています。これは、最終アドレスの上位 32 ビットが確実にゼロになるように、コンパイラが余分な命令を発行しなければならないことを意味します。これを行わないと、アドレス例外でコードがクラッシュします。

これはコンパイラにとって過大な負担となり、コードのサイズも顕著に肥大化します。以下のスイッチを有効にすることにより、コードは少数のシンプルなルール (多くの場合は true) に従うと仮定します。C99 言語仕様ではこれらのルールが文書化されており、このスイッチのメリットを得るためには、コードがこれらのルールを順守したものでなければなりません。

- -Xassumecorrectalignment=1 : ポインタのアラインメントが正しいことを前提とします。-Xassumecorrectalignment=1 の場合、多くのゼロ拡張を削除することができます。詳細は、67ページの「適切なポインタ アラインメントの前提」を参照してください。

最適化ビルドではこのスイッチを有効にすること、および前提に適合させるためコードを必要に応じて変更することを強くお勧めします。

適切なポインタ アラインメントの前提

PPU アーキテクチャでは、すべてのデータ タイプにメモリ内のデフォルト アラインメントが存在し、コンパイラではこれらのルールに従ってメモリ内にデータが配置されます。たとえば double は常に 8バイトでアラインされ、int は 4バイトでアラインされます。-Xassumecorrectalignment により、コンパイラでは全オブジェクトに対するこれらのルールが、ポインタ経由でアクセスされると仮定することができます。以下に例を示します。

```
double * dbl_pointer; // dbl_pointer には常に 8バイトでアラインされるアドレスが含まれる

int * int_pointer;    // int_pointer には常に 4バイトでアラインされるアドレスが含まれる
```

ただし、より小さなサイズまたは intptr_t からキャストすることによって、非アライン ポインタを作成することも可能です。

例:

```
char x[ 10 ];
int main()
{
  int *p = (int*)( x + 5 );
  *p = 0;
}
```

ここではキャストを使用し、不正にアラインされたポインタを作成、および配列「x」に 4 バイトのゼロを書き込んでいます。

これは一部のターゲットでは機能しない場合があり、一部の最適化がエラーとなる可能性もあります。

たとえば PPU では、たとえ効率的でないにしても、非アライン整数ポインタからの読み込みや、整数ポインタへの書き込みを行うことが可能です。ただし、非アライン浮動小数点ポインタに関しては行えません。

このため、以下はエラーとなります。

```
int main()
{
  float *p = (float*)( x + 5 );
  *p = 0;
}
```

こちらも同様です。

```
float f;

int main()
{
  int *p = (int*)( x + 5 );

  union { int i; float f; } u;

  u.i = *p;
  f = u.f;
}
```

ここでは、オプティマイザが p は適切にアラインされていると仮定する場合、ロード > 保存 > ロードシーケンスが直接 float ロードに置換される可能性がありますが、これは正しくありません。

従って、PPU ターゲットにおけるこの最適化を有効にするには、-Xassumecorrectalignment を使用し、すべてのポインタが確実にアラインされているようにする必要があります。

もっとも重要なのは、ベクトル化時にオプティマイザではシンプルな lvlx 命令を使用して、スカラーを vmx レジスタにロードできることです。

それ以外の場合、ベクトル化コードではシーケンスが使用されます。

```
add tmp1, addr, 16
lvlx tmp2, addr
lvrx tmp3, tmp1
vor result, tmp2, tmp3
```

これは大幅に長くなります。

-Xassumecorrectalignment が無効 (=0) で、アラインメントがコンパイラによって決定されない場合、オプティマイザでは、ポインタのアラインメント情報を前提とする変換が一切実行されません。

ポインタから整数への変換を回避する

-Xassumecorrectalignment および -Xassumecorrectsign のコントロール変数がセットされている場合は、ポインタから整数への変換を回避する必要があります。

以下はその例です。

```
void *pointers[ 100 ];
```

ポインタとして宣言されている場合でも、この配列にオフセットが含まれることがあります。このため、ロードされる値ではなく、ポインタをキャストすることによって、コンパイラにこれを認識させる必要があります (以下の例を参照)。

悪い例: `int offset = ( int )pointers[ i ];`

良い例: `int offset = ( ( int* )pointers )[ i ];`

これにより、その値が符号オフセットであるということを、最初からコンパイラが認識します。

ポインタのリロケーションを処理する

ファイルにストアされたポインタをリロケーションする際は、ポインタのベースが符号なしの int、オフセットが符号付き int であることを確実にしてください。これにより、マイナスのオフセットがオーバーフローしなくなります。

以下はその例です。

```
struct RelocateMe
{
  RelocateMe *next; // when loaded from a file, this is an offset.
};

void relocate( void *base, RelocatMe *ptr )
{
  if( ptr->next != NULL )
  {
                  ptr->next = (RelocateMe*)( (unsigned) base +
(int)ptr->next );
                  relocate( ptr->next );
  }
}
```

仮想コールの予測

多くの場合、オブジェクト志向のデザインでは仮想関数コールが便利ですが、PPU では、通常の関数コールと比べて、パフォーマンスが顕著に低下する場合があります。

当社で非常に多くのゲーム コードを分析した結果、1 つの仮想関数が、実行される多くの仮想関数コールのターゲットになることが多いということを発見しました。関数コールの予測機能では、これに該当するケースを SNC に指摘できるため、多くのケースで仮想関数のオーバーヘッドを削減できます。これは、属性「`__attribute__((likely_target))`」を使用して行います。

以下はその例です。

```
#include <stdio.h>

#if defined ( USE_LIKELY )
#    define LIKELY_TARGET __attribute__((likely_target))
#else
#    define LIKELY_TARGET
#endif

class Base
{
    public:
        virtual int foo();
};

class Wibble : public Base
{
    public:
        LIKELY_TARGET virtual int foo();
};

int Base::foo()
{
    printf( "Base foo\n" );
    return 0;
}

int Wibble::foo()
{
    printf( "Wibble foo\n" );
    return 1;
};

int bar()
{
    Wibble* w = new Wibble();

    return w->foo();
}
```

USE_LIKELY を定義してこのコードをコンパイルすると、仮想関数 Wibble::foo() にこの属性が適用されます。その後、vtable (この場合は Base::foo) のほかのバージョンの foo より多く Wibble::foo がコールされることが仮定されます。

foo に対するコールが実行されると、すぐに vtable に処理が移動してそれに関連するパフォーマンス低下が発生するのではなく、比較が行われます。この比較では、ターゲットがマーク付けされた関数かどうかが判定され、マーク付けされた関数の場合は直接分岐が選択されます。コールのターゲットがマーク付けされた関数ではない場合は、通常の仮想コールのメカニズムが使用されます。

マーク付けされたこの関数が通常のインライン化条件に適合する場合は、直接分岐がインライン化されたコピーに置き換えられるため、パフォーマンスがさらに向上します。

これは、ターゲットがマーク付けされた関数の場合は処理速度が大幅に向上する、ということを意味します。ターゲットがほかの関数の場合は、比較処理が増えるため、多少の処理速度低下が発生します。

詳細は、71 ページの「関数を「hot」としてマークする」を参照してください。

関数を「hot」としてマークする

1 つのフレームで特定の関数が多くの時間を使用するということを SNC が認識している場合は、最適化時にこれが反映され、関数のサイズが大きくなる変換を実行し、その関数のインライン化を増やします。

仮想関数を「hot」とういてマーク付けすると、仮想コールの予測 (69 ページの「仮想コールの予測」を参照) の __attribute__((likely_target)) でマーク付けした場合と同じ効果もあります。

「hot」関数のインライン化は、「-Xinlinehotfactor=<n>」スイッチでコントロールでき、「hot」関数の場合の <n> は、インライン化設定 (上記のスイッチでコントロール) の値を増加させる因数となります。詳細は、83 ページの「-Xinlinehotfactor」を参照してください。

SNC の将来のバージョンでは、「hot」関数に対してより多くの最適化を実行できるようになります。

エイリアス分析

ポインタが、ほかのポインタと同じロケーションを参照する場合、これを「ほかのポインタをエイリアスする」と言います。最適なコード スケジューリングを試行して生成するため、SNC では、ポインタがほかのポインタをエイリアスする場所を探すためにエイリアス分析が実行されます。

-O2 ではデフォルトで、コードが C99 の厳密なエイリアス ルールに反していないと前提されます。この前提を設定することによって、より積極的なスケジューリングを実行することが可能になるため、より効率的なコードが生成されます。しかしこの前提に依存すると、ルールに準拠しないコードを記述してしまう可能性もでてきます。

この前提は、コントロール変数の「-Xrelaxalias」スイッチで制御します。詳細は、90 ページの「-Xrelaxalias」を参照してください。

最適化ビルドではこの値を 2 以上に設定すること、および厳密なエイリアス ルールに反するすべてのコードをルールに準拠するように変更することをお勧めします。

-Xrelaxalias=2 に対応するようにコードを簡単に変更できない場合は、-Xrelaxalias=1 にすることをお勧めします。このレベルは、ほとんどのコードに対応します。これでもうまくいかない場合にのみ、-Xrelaxalias=0 を使用してください。

関数単位の最適化

SNC は、コード内でプラグマを使用することによって関数単位の最適化のオン/オフに完全に対応します。これは、すべての最適化に一度に適用することや、個々の最適化に適用することができます。詳細は、32 ページの「制御プラグマ」を参照してください。

一般的な使用方法は、ファイルでデバッグ対象となっている関数の最適化をオフにすることです。これにより、-O2 でコンパイルされないようにできます。

```
#define START_NOT_OPTIMIZING _Pragma("control %push O=0")
#define END_NOT_OPTIMIZING   _Pragma("control %pop O=0")

void aFunctionIWantOptimized()
{
                  //...
}

START_NOT_OPTIMIZING

void aFunctionIAmTryingToDebug()
{
                  //...
}

END_NOT_OPTIMIZING

void anotherFunctionIWantOptimized()
{
                  //...
}
```

これは、-O0 でコンパイルされているファイル内の 1 つの関数に対する最適化を有効にしても、逆方向には機能しません。これは、-O0 でファイルがコンパイルされると、処理速度向上のため最適化機能が有効にならないためです。

この機能は、特定の関数に対して特定の最適化を有効にする場合にも使用します。たとえば、プロジェクト全体に対するループのアンロール (コマンドラインで -Xunroll=1) は、コードのサイズが全体的に大きくなるため適切ではありませんが、パフォーマンスが重要な特定の関数に対しては有効な場合があります。

以下はその例です。

```
#define START_UNROLLING _Pragma("control %push unroll=1")
#define END_UNROLLING   _Pragma("control %pop unroll=1")

START_UNROLLING

int functionToUnRoll( int x )
{
                  for ( int i = 0; i < 3; ++i )
                  {
                     x += 7;
                  }

                  return x;
}

END_UNROLLING
```

8: コントロール変数リファレンス

コントロール変数リファレンス

以下の表のすべてのコントロール変数は、アルファベット順で記載されています。各コントロール変数の詳細は、35 ページの「コントロール変数の定義」を参照してください。

各コントロール変数には、以下の特性がリストされています。

- 定義の名前
- 変数のスコープ (コンパイル、ファイル、行、ループ、関数)
- 値のタイプや範囲
- デフォルト値
- コントロール変数に使用できる各値に関する 1～2文の簡単な説明

コントロール変数のスコープの詳細に関しては、23 ページの「コントロール変数」を参照してください。

各表は、以下の形式になっています。

-Xコントロール変数	スコープ	タイプ/値	デフォルト
値 #1	説明 #1		
値 #2	説明 #2		
…	…		

-Xalias

-Xalias	関数	0..3	0
0	エイリアス分析を実行しない。各メモリ参照が、すべてのメモリ参照と干渉する可能性があると想定した場合。		
1	宣言のみをベースにエイリアス分析を実行する。		
2	宣言と固定サブスクリプトをベースにエイリアス分析を実行する。		
3	上記の分析およびフロー依存項目を使用した分析を実行する。		

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xalignfunctions

-Xalignfunctions	ファイル	1..32768	4
n	*n* 以上の次の 2 の累乗に関数を割り当てる。たとえば、-Xalignfunctions=8 の場合は、関数が 8バイト境界に割り当てられる。		

-Xasmreg

-Xasmreg	ファイル	0..1	1
0	asm ステートメントでスクラッチ レジスタを強制終了しない。		
1	asm ステートメントでスクラッチ レジスタを強制終了する。		

-Xassumecorrectalignment

-Xassumecorrectalignment	関数	0..1	0
0	ポインタが正しくアラインされていると仮定しない (デフォルト)。		
1	ポインタが正しくアラインされていると仮定。		

-Xassumecorrectsign

-Xassumecorrectsign	関数	0..1	0
0	変数に正しい符号が含まれると仮定しない (デフォルト)。		
1	変数に正しい符号が含まれると仮定。		

-Xautoinlinesize

このスイッチは、ソース コードにおいてインラインとしてマークされることなく、コンパイラによって自動的にインライン化される、関数の最大サイズを制限します。これは、ヘッダファイルの内部クラスで定義されるC++ メソッドなど、暗黙的インライン関数には適用されません (84 ページの「-Xinlinesize」を参照してください。)。

-Xautoinlinesize	関数	0..50000	0
0	自動インラインニングなし。		
n	最大 *n* インストラクションまで、非マーク関数の自動インラインニングを許可。		

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xinlinesize と –Xinlinemaxsize も参照してください。

-Xautovecreg

-Xauto_vecreg	関数	0..2	0
0	vmx 最適化を自動的に実行しない。		
1	vmx レジスタを使用し、変換や integer/float キャストにおける LHS 依存性を回避する。		
2	vmx レジスタを使用し、変数シフト、小さな変数乗算、その他贅沢な操作を回避する。		

-Xbranchless

-Xbranchless	関数	0..2	0
0	分岐なし比較を使用しない		
1	3 項演算子 (例 a > b ? a : b) に対してのみ分岐なし比較を使用する		
2	潜在的なすべての整数比較に対して分岐なし比較を使用する。		

-Xbss

-Xbss	関数	0..2	1
0	すべてのデータを .data セクションに配置する。		
1	初期化されていない静的データおよびゼロで初期化されたデータを、コンパイラ内の自動規則に従って .data セクションまたは .bss セクションに配置する。		
2	初期化されていない静的データを .bss セクションに配置し、ゼロで初期化されたデータを、可能な限り .bss セクションに配置する。		

-Xc

-Xc	ファイル	名前リスト	mixed+gnu_ext+c99
ansi	C 用。コンパイラは、ANSI と ISO の C 規格 (ANSI X3.159-1989 および ISO/IEC 9899:1990(E))に従い、「準拠したホスト実装」としてに完全に従う。そのすべての言語と標準ヘッダ ファイルに対応。		
knr	C 用。コンパイラは、『*The C Programming Language*』(Kernighan & Ritchie 著)(邦訳: プログラミング言語C) に記載されている C 言語の定義と高い互換性があり、UNIX pcc コンパイラとの互換性も高い。		
mixed	(デフォルトはオン) C 用。コンパイラは、既存の K&R コードを ANSI に移植する作業を支援するための拡張機能がいくつか追加されているという点を除き、基本的には ANSI コンパイラ (拡張機能の説明を参照)。		
knr+x	上記の 3 つの基本モードに加え、c コントロール変数の値に const、volatile、signed の名前のサブセットを追加し、リスト値とすることが可能。基本モード c=knr でこのいずれかの名前を使用した場合は、対応する ANSI C 修飾子が認識		

	されるようになる。たとえば c=knr+const+volatile の場合は、K&R 互換に加え、ANSI C の const タイプと volatile タイプの修飾子も認識される。 mixed モードまたは ansi C モードには、noknr を追加できる (Xc=mixed+noknr など)。この値により、プロトタイプのない関数の宣言んや定義で警告が出力される。noknr モードが有効になっている場合は、宣言または定義されていない関数の使用をコンパイラが検出した際にも警告が出力される。また C モードの ansi、knr、mixed では、inline 値を追加 (Xc+=inline など) することにより、C++ と同様に inline をキーワードとすることができる。
c99	(デフォルトはオン) 値 c99 を ansi、K&R、ミックス モードに追加することにより、ISO/IEC 9899:1999 C プログラミング標準規格で追加された C 言語機能を有効にできます。
cfront:21	C++ 用。コンパイラが AT&T Cfront 2.1 互換になる。
cfront cfront:30	C++ 用。コンパイラが AT&T Cfront 3.0 互換になる。
arm	C++ ではコンパイラに、『*The Annotated C++ Reference Manual*』(Margaret A. Ellis & Bjarne Stroustrup 著) に記載され、C++ 規格 (ISO/IEC 14882:2003) で変更された言語が導入される。
cp	C++ 用。上位互換性があり制限が緩和されている点を除いてARM と同様。
いずれの C++ 言語モードでも、値を追加または削除可能。	
c_func_decl	(デフォルトはオフ) C 形式の関数プロトタイプが、C++ 以外のインクルード ファイルのインクルードに対応する。
array_nd	(デフォルトはオフ) 配列の new 演算子と delete 演算子の認識を有効にする。
rtti	(デフォルトはオン) RTTI 動作を有効にする。
wchar_t	(デフォルトはオフ) 個々のタイプの宣言において、wchar_t をキーワードにする。
bool	(デフォルトはオフ) 個々のタイプの宣言において、bool をキーワードにする。
old_for_init	(デフォルトはオフ) for ループの init ステートメントで宣言されている変数のスコープを拡張する。
exceptions	(デフォルトはオフ) 例外処理のコンストラクトと動作の使用を許可する。
tmplname	(デフォルトはオフ) テンプレート化されていない関数の名前とは異なる、マングルされた名前の付いたテンプレートを作成する。
gnu_ext	(デフォルトはオン) C/C++ 言語での GNU GCC 拡張機能の使用を許可する。
msvc_ext	(デフォルトはオフ) C/C++ 言語での Microsoft Visual Studio®拡張機能の使用を許可する。

-Xcallprof

-Xcallprof	関数	0..1	0
0	Tuner callprof 階層プロファイリングに対し、特別コードを生成しない。		
1	Tuner によるプロファイリングを許可する関数エントリと終了に対し、特別コードを生成する。この特別コードは、アプリケーションのパフォーマンスに対してほとんど影響しない。この新しい callprof 機能に関する詳細は、Tuner for PS3 ユーザー ガイドを参照。		

-Xcf

-Xcf	ファイル	0..1	1
0	「完全な」CF コンパイラを使用しない。		
1	「完全な」CF コンパイラを使用する。		

-Xchar

-Xchar	ファイル	名前	signed
signed	C/C++ では、char タイプがデフォルトで signed となる。		
unsigned	C/C++ では、char タイプがデフォルトで unsigned となる。		

-Xconstpool

この最適化では、各関数で使用される定数が、連続して配置されるキャッシュ メモリ ブロックにグループ化されます。これにより、キャッシュの局所性が向上し、レジスタへの定数ロードに必要とされるコードが簡略化されます (これらでは共通の High Address を共有)。

-Xconstpool	関数	0..1	0
0	プーリングなし。		
1	関数ごとに定数プールを作成 (O2 でデフォルト)。		

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xdebugvtbl

-Xdebugvtbl	関数	0..1	0
0	クラスに含まれる可能性のある C++ 仮想テーブルに対し、デバッグ データを生成しない。		
1	クラスに含まれる可能性のある C++ 仮想テーブルに対し、デバッグ データを生成する。このため、Debugger の [ウォッチ] ペインでクラスを調べると、仮想テーブル ポインタも検証される.		

-Xdeflib

-Xdeflib	行	0..2	1
0	組み込み関数をインライン化しない。		

1	自動制御で組み込み関数をzインライン化する。
2	可能な限り組み込み関数をインライン化する。

-Xdepmode

-Xdepmode	ファイル	0..1	1
0	GCC 2 スタイルの依存ファイル名を使用する。		
1	GCC 3/4 スタイルの依存ファイル名を使用する。		

-Xdiag

-Xdiag	行	0..2	1
0	エラーと致命的なエラーの各レベルでメッセージを出力。備考や警告は出力しない。		
1	警告、エラー、致命的なエラーの各レベルでメッセージを出力。備考は出力しない。		
2	備考、警告、エラー、致命的なエラーの各レベルでメッセージを出力。		

-Xdiaglimit

-Xdiaglimit	ファイル	0..1000000	0
n	各診断に対して発行されるメッセージの数を、最初の *n* オカレンスに制限する。値 0 は無制限を意味する。		

-Xdivstages

-Xfastfloat と合わせて使用されるこのスイッチでは、近似浮動小数点除算の結果を絞り込む際に使用されるイテレーション数を制御します。

標準的な 'ゲーム' アプリケーションの場合、-Xdivstages=3 で速度と正確度の適切なバランスが保たれます。

79 ページの「-Xfastfloat」を参照してください。

-Xdivstages	関数	0..5	0
0	高速近似を無効化 (fdiv を使用)		
1	fre インストラクションのみ。		
2	fre + newton-raphson 法の 1 段階 (~10 ビット)		
3	fre + newton-raphson 法の 2 段階 (~20 ビット)		
4	fre + newton-raphson 法の 3 段階 (~30 ビット)		

5	fre + newton-raphson 法の 4 段階 (fdiv 使用とほぼ同じ)

-Xfastfloat

-Xpreopt および -Xpostopt と合わせて使用されるこのスイッチは、制度に影響を与える可能性のある付加的浮動小数点最適化を有効にします。

- vmx と浮動小数演算の混合は、可能な場合、vmx 演算に変換されます (現時点では、浮動小数 -> 結合による vmx タイプ変換ですが、今後拡張される予定です)。
- 浮動小数点比較を、同等の整数演算に転換 (結果として規模は大きくなるものの、通常はより迅速なコードに)。
- 浮動小数点除算を、近似法の使用に転換 (78 ページの「-Xdivstages」を参照してください。)。

このスイッチは、デフォルトで -O2 で有効になります。

-Xfastfloat	関数	0..1	1
0	最適化なし。		
1	付加的浮動小数点最適化を有効化 (O2 でデフォルト)。		

-Xfastint

Xpreopt と -Xpostopt と合わせて使用されるこのスイッチでは、符号付き整数演算のオーバーフローに依存しないコードの最適化が可能になります。もっとも重大な作用は、符号拡張インストラクションのより積極的な削除が許可されることです。

このスイッチは、デフォルトで -O2 で有効になります。

-Xfastint	関数	0..1	1
0	最適化なし。		
1	整数オーバーフローがないと想定し、最適化を有効化 (O2 でデフォルト)。		

-Xfastlibc

-Xfastlibc	コンパイル	0..1	0
0	libc.a を libcs.a に置換しない。		
1	リンク時に libc.a (標準 C ライブラリ) ではなく libcs.a (コンパクト C ライブラリ) を使用する。.		

-Xfastmath

-Xfastmath	関数	0..1	0

0	さらなる最適化を行わない。
1	精度に影響する可能性のある、付加的な浮動小数点最適化を有効にする。 ▪ if ステートメントを「fsel」に変換。 ▪ 浮動小数点、整数値、VMX レジスタ間の変換を避けるため、VMX レジスタを自動的に使用。これにより、Load Hit Store ペナルティの数が削減されます。 ▪ 「fdiv」をおおよその分配と精度に置換。 **メモ：** このスイッチは GCC --fast-math スイッチに似ていますが、このスイッチが SNC において制御する最適化は、GCC によって実行されるものとは異なります。

-Xflow

-Xflow	関数	0..1	0
0	制御フロー最適化を実行しない。		
1	制御フロー最適化を実行する。		

メモ: この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xfltconst

-Xfltconst	ファイル	0|4|8	0
0	C において、fltconst=4 で示されたとおり、単精度浮動小数点の定数を導入する。		
4	単精度で、単精度浮動小数点の定数を導入する。		
8	値が、使用される前に倍精度に変換される (倍精度変数への割り当てなど) 状況で使用されている場合、単精度浮動小数点の定数を倍精度で導入、または、他のオペランドが倍精度の変数などになっている演算子の 1 つのオペランドとして使用する。		

-Xfltdbl

-Xfltdbl	関数	0..2	2
0	c=knr モードの C で、float オブジェクトに対して単精度の演算を実行し、float 関数の引数を変換せず値を double に返す。このモードでコンパイルされたファイルは、通常の K&R 浮動小数点モデルを使用してコンパイルされたファイルと正しくリンクできない。		
1	c=knr モードの C で、float オブジェクトに対して単精度の演算を実行し、float 関数の引数を変換して値を double に返す。このモードでコンパイルされたファイルは、通常の K&R 浮動小数点モデルを使用してコンパイルされたファイルと正しくリンクできる。		

2	c=knr モードの C で、float オブジェクトに対して倍精度の演算を実行し、float 関数の引数を変換して値を double に返す。これは、通常の K&R 浮動小数点モデルとなる。

-Xfltedge

-Xfltedge	関数	1..3	2
0	未使用。		
1	非数値が発生し、「信号なし」の演算でそれが使用された場合にプログラムの動作を変化させる最適化を実行しない。このモードの使用は、SNC コンパイラではあまり適さない。場合によっては、比較演算が変更され、その動作が変化することもある。たとえば、式 (!(a>b)) が (a<=b) に変更された場合、a と b の順番を変更しない限り不正となります。		
2	非数値が発生し、「信号なし」の演算でそれが使用される場合にプログラムの動作を変化させる可能性のある最適化を実行する。このモードでは、変数テスト (変数自体に対する等式や不等式) の特殊ケースは最適化されない。このモードは、通常の最適化を許可するためのものだが、非数値のテストをプログラムする機能も搭載している。		
3	非数値が発生し、「信号なし」の演算でそれが使用される場合にプログラムの動作を変化させる可能性のある最適化を実行する。		

-Xfltfold

-Xfltfold	関数	0..2	2
0	コンパイル時に、浮動小数点の定数に関連する式を評価しない。		
1	コンパイル時に、浮動小数点定数および演算子に関連する式を評価する。浮動小数点定数に適用されている組み込み関数が関連する式は、評価されない。		
2	コンパイル時に、浮動小数点の定数に関連する式を評価する。		

-Xforcevtbl

-Xforcevtbl	ファイル	0..1	0
0	C++ vテーブルの生成を強制しない。		
1	C++ vテーブルの生成を強制する。		

-Xfprreserve

-Xfprreserve	行	リスト	空のリスト
list	*list* で指定した浮動小数点レジスタを予約する。		

-Xg

-Xg	コンパイル	0..2	0
0	シンボル デバッグ情報を、出力ファイルに含めない。		
1,2	シンボル デバッグ情報を、出力ファイルに含める (SN シンボル デバッガーで使用)。		

-Xgnuversion

-Xgnuversion	コンパイル	400..500	411
n	SNC と互換性のある GNU コンパイラのバージョンを決定する (デフォルトでは GCC 4.1.1 との互換性がある)。GCC 4.0.2 との互換性には、-Xgnuversion=402 を使用する。		

-Xgprreserve

-Xgprreserve	行	リスト	空のリスト
list	*list* で指定した汎用整数レジスタを予約する。		

-Xhostarch

-Xhostarch	ファイル	0..65536	32
32	32ビット コンパイラを使用する。		
64	64ビット コンパイラを使用する。64ビット ホスト オペレーティング システムが必要。		

-Xinclpath

-Xinclpath	ファイル	名前	relative
relative	C は、引用符で区切られた相対ファイル名を使用している #include ステートメントで指定されたファイルを検索するが、処理中の #include ステートメントを含むファイルが格納されているディレクトリを最初に検索する。		
absolute	C は、引用符で区切られた相対ファイル名を使用している #include ステートメントで指定されたファイルを検索しますが、元の (最上位の) ソース ファイルが格納されているディレクトリを最初に検索する。		

-Xignoreeh

-Xignoreeh	関数	0..1	0
0	例外処理コンストラクトを無視しない。		
1	一切の例外処理が発生しないと前提し、例外処理コンストラクトを無視する。-Xignoreeh=1 でプログラムがコンパイルされ、実際に例外が発生した場合も、その例外は無視される。 • コンストラクト「try { body .. } catch()」の {} では、try ブロックで例外が発生しないという前提でコンパイルされるため、例外処理が回避される。 • 例外指定は無視される。 • クリーンアップ コードは一切生成されない。 • ただし、明確な「throw」ステートメントは通常通りコンパイルされる。 -Xignoreeh=1 は、構文的に例外処理コンストラクトを有効化するものではありません。このため、ファイルに明確な例外処理コンストラクト (try、throw など) が含まれる場合は、-Xignoreeh=1 を使用する前に、-Xc+=exceptions が必要となります。ただし -Xignoreeh=1 では、明確な例外処理コンストラクタを含まないファイルの動作を変更 (クリーンアップ コードの抑制など) することが可能です。 -Xignoreeh は、_NO_EX プリプロセッサ シンボルに一切影響を与えません。これは -Xc+=exceptions なしで定義され、-Xc+=exceptions では定義されません。-Xignoreeh=1 はこのルールを変えるものではありません。		

-Xinline

-Xinline	行	名前リストまたはペア	空のリスト
name	名前付き関数 (ソース 関数または組み込み関数) をインライン化する。		
name:*n*	*n* を優先度として使用し、名前付き関数 (ソース 関数または組み込み関数) をインライン化する。		

-Xinlinehotfactor

このスイッチは、「__attribute__((hot))」と同時に使用します。コールを行う側の関数が「hot」としてタグ付けされている場合は、その中で実行する追加インライン化の量を、このスイッチの値によってコントロールできます。n の値は、「hot」関数内でのインライン化における autoinlinesize および inlinesize の値を乗算する因数となります。

詳細は、71 ページの「関数を「hot」としてマークする」を参照してください。

-Xinlinehotfactor	関数	1..100	5
1	効果なし (デフォルト)。		
n	「hot」関数内でのインライン化における autoinlinesize および inlinesize の値を乗算する因数。		

-Xinlinemaxsize

このスイッチは、任意の 1 関数へのインライニング最大量を制御します。これは、個々の関数が大きくなりすぎること、およびオプティマイザにおける他の段階の減速を防止するために使用されま

す。デフォルト値からこれを増加させると、コンパイル速度は犠牲にされますが、インライニング量を増加させることができます (これに伴ってパフォーマンス向上の可能性もあり)。

このコントロールに対するパラメータは、'インストラクション' という観点において、関数の最大サイズを表します。これらは内部コンパイラ インストラクションであり、個別プロセッサ インストラクションと必ずしも同じではありません。

-Xinlinemaxsize	関数	0..50000	1000
0	インライニングなし。		
n	最大 *n* インストラクションまで、関数へのインライニングを許可。		

-Xinlinesize と -Xautoinlinesize も参照してください。

-Xinlinesize

このスイッチでは、コンパイラによってインライン化される明示的インライン関数の最大サイズを制限します。明示的インライン関数には、ヘッダー ファイルの内部クラスで定義される C++ メソッドが含まれます。

このコントロールに対するパラメータは、'インストラクション' という観点において、関数の最大サイズを表します。これらは内部コンパイラ インストラクションであり、個別プロセッサ インストラクションと必ずしも同じではありません。

-Xinlinesize	関数	0..50000	0
0	明示的インライン化なし。		
n	最大 *n* インストラクションまで、明示的インライン関数の自動インライン化を許可。		

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xautoinlinesize と -Xinlinemaxsize も参照してください。

-Xintedge

-Xintedge	関数	0..1	0
0	整数の演算時に整数のオーバーフローが発生すると想定される場合において、オーバーフローが発生した際にプログラムの動作が変化する最適化を実行しない。		
1	整数の演算時に発生する整数のオーバーフローの影響が、最適化の適用において無視できる程度の場合。		

-Xipa

このスイッチにより、プロシージャ間の分析 (IPA) を制御します。ある関数が現在のコンパイル単位に限定して他の関数によって呼び出されることがコンパイラによって判断できる場合 (通常は静的 (static) と宣言された関数について)、その関数は最適化が可能で、その関数を呼び出す関数では

レジスタ使用の改善ができ、呼び出し全体について確保する必要のあるレジスタ数が削減されます。これらの最適化によって、関数ごとに ABI が変更されるため、呼び出し箇所が全てわかっているときにのみ使用できます。

全ての関数および呼び出し箇所を分析する必要があるため、大きなファイルでは IPA の実行に時間がかかる場合があります。「統一」ビルドなど、巨大なファイル群の場合、IPA を無効にすることをお勧めします。

-Xipa	ファイル	0..1	0
0	IPA を無効化		
1	PAを有効化		

-Xlinkoncesafe

-Xlinkoncesafe	ファイル	0..1	0
0	すべての link-once 実装が同じであると見なさない。		
1	すべての link-once 実装が同じであると見なす。		

-Xmathwarn

-Xmathwarn	行	off .. on	off
off	コンパイラが演算エミュレーション ライブラリへのコールを使用している場合に、警告を出力しない。		
on	コンパイラ、演算エミュレーション ライブラリへのコールを使用している場合に、警告を出力する。		

-Xmemlimit

-Xmemlimit	ファイル	0..max int	512
n	使用可能と予測されるべきメモリ量 (単位 KB) をオプティマイザに通知する。		

-Xmserrors

-Xmserrors	コンパイル	0..1	0
0	エラーと警告にソース行をプリントする。		
1	エラーと警告にソース行をプリントしない。		

-Xmultibytechars

-Xmultibytechars	ファイル	0..1	0
0	マルチバイトでエンコードされたソース ファイルはサポートしない。		
1	UTF8 標準規格を使用してエンコードされたマルチバイト文字列を含む、ソース ファイルの使用を許可。		

-Xnewalign

-Xnewalign	関数	0..64	16
n	この値は、2 つの引数 (アラインされた) 形式の operator new が、コンパイラで使用されるポイントを決定する。この閾値以下のアラインメントのタイプのインスタンス割り当てでは、単独引数の標準関数「operator new(std::size_t)」が使用されるのに対し、この値より大きいアラインメントのタイプでは、2 引数の SCE 拡張関数「operator new (std::size_t, std::size_t)」が使用される。		

-Xnoident

-Xnoident	コンパイル	0..1	0
0	.comment セクションにコンパイラ バージョン用のエントリを生成する。		
1	.comment セクションにコンパイラ バージョン用のエントリを生成しない。		

-Xnoinline

-Xnoinline	行	名前リスト	空のリスト
name	名前付き関数 (ソース 関数または組み込み関数) をインライン化しない。		

-Xnosyswarn

-Xnosyswarn	ファイル	0..1	1
0	「システム」ヘッダ ファイルから発行された警告を表示する。システム ヘッダ ファイルは、ソース ファイルと同じディレクトリにはないが、`-I` オプションを使用しなくてもインクルードすることが可能である.。これは、インクルード ディレクトリにあるヘッダ ファイルは黙示的にコンパイラに認識されることを意味する。このインクルード ディレクトリは、$CELL_SDK 内の一連のディレクトリを指す。		
1	「システム」ヘッダ ファイルから発行された警告を抑制する (これは GCC の -Wsystem-headers スイッチをほぼ同じ)。		

-Xnotocrestore

-Xnotocrestore	関数	0..2	0

0	コンパイラにより、ABI 完全準拠コードが生成される。ポインタによって関数をコールするコードでは、呼び出される側の TOC レジスタの値が、呼び出し側のものと異なる可能性があると見なされる。 呼び出される側のコードがリンク時に別の TOC 領域に存在する場合、外部関数へのコール後に nop 命令が生成され、リンカーでは、TOC ポインタの復元が許可される。 このオプションでビルドされたコードの適切な実行には、特別なリンカー スイッチは必要ない。 これが notocrestore 制御のデフォルト値。
1	コンパイラにより、外部関数へのコール後の nop 命令が省かれるが、ポインタを通じたコールは TOC-safe であることが保証される。プログラムは、SN リンカー -notocrestore スイッチとリンクする必要がある。
2	コンパイラでは、外部関数へのコール後の nop 命令が省かれ、ポインタによるコールは常に同じ TOC 領域を使用すると見なす。プログラムは、SN リンカー -notocrestore スイッチとリンクする必要がある。

-Xoveralign

-Xoveralign	ファイル	0..1	0
0	構造体の GCCスタイル オーバーアライメントを実装しない。		
1	構造体の GCC スタイル オーバーアライメントを実装する (構造体のサイズに基づく)。		

-Xparamrestrict

-Xparamrestrict	関数	0..1	0
0	ポインタ タイプの関数パラメータを、__restrict 修飾子で修飾しない。		
1	ポインタ タイプの関数パラメータを、__restrict 修飾子で修飾する。		

-Xpch_override

-Xpch_override	ファイル	0..1	0
0	プリコンパイルされたヘッダ ファイルの生成に使用されるファイルが、コンパイルされるファイルと同じディレクトリにあることをチェックする、コンパイラの機能をオーバーライドしない。		
1	プリコンパイルされたヘッダ ファイルの生成に使用されるファイルが、コンパイルされるファイルと同じディレクトリにあることをチェックする、コンパイラの機能をオーバーライドする。		

-Xpostopt

このスイッチでは、データフロー分析に基づいた新たな最適化を制御します。

-Xpostopt	関数	0..6	0
0	最適化なし (デフォルト)。		
1	(現時点では何もしない)		
2	有効化: - さらなるグローバル定数のフォールドとプロパゲーション		
3	さらに有効化: - zero/sign extend の除外 - - ロード/ストア アドレスの簡略化 - エイリアス情報のプロパゲーション改善		
4	さらに有効化: - ロードとストア連鎖の破壊 - LHS 依存関係の除去 - 浮動小数点比較を、より短いレイテンシで整数演算に変換 - 空ループの削除		
5	さらに有効化: - より積極的なロードと格納の除去		
6	さらに有効化: - さらなる最適化		

メモ: この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xpredefinedmacros

-Xpredefinedmacros	関数	0..1	0
0	定義済みのマクロ、すなわち SNC コンパイラ フロントエンドで定義されたマクロをコンパイル中に表示しない。		
1	定義済みのマクロをコンパイル中に表示する。		

-Xpreprocess

-Xpreprocess	ファイル	0..2	0
0	前処理された出力を生成しない。		
1	前処理された出力を生成する。コンパイラでは、.i ファイル名拡張子を含むファイルが出力される。ユーザーは、入力ファイル名の拡張子に合うようにファイルの名前を変更する、またはファイルがそれぞれ C++ や C ソースとして取り扱われるべきであることをコンパイラに示すために、-Tp や -Tc コマンドライン スイッチを適宜		

	使用する必要がある。
2	値=1 と同様だが、前処理された出力がソース行情報と共に生成される。

-Xprogress

-Xprogress	関数	名前リスト	ファイル
Files	各ファイルのコンパイル開始時に進行状況を表示する。		
functions	各関数のコンパイル開始時に進行状況を表示する。		
Phases	コンパイルの各フェーズ開始時に進行状況を表示する。		
subphases	コンパイルの各サブフェーズ開始時に進行状況を表示する。		
Actions	コンパイラでの各主要処理 (インライン化など) の終了時に進行状況を表示する。		
Failures	コンパイラでの各主要処理のエラー (関数のインライン化失敗など) を表示する。		
templates	テンプレート関数のインスタンス化を表示する。		
Memory	進行状況の表示に、コンパイラのメモリ使用率情報を含める。		
Sizes	進行状況の表示に、内部のコンパイラ データ構造のサイズ情報を含める。		
Realtime	進行状況の表示に、コンパイラで使用された実時間を含める。		
Rtime	realtime と同じ。		
Usertime	進行状況の表示に、コンパイラで使用されたユーザー時間を含める。		
Utime	usertime と同じ。		
%all	コンパイルのすべての可能なポイントで進行状況を表示する。		
%none	コンパイラの進行状況を表示しない。		

-Xquit

-Xquit	コンパイル	0..2	0
0	エラーまたは致命的なエラーが出力された場合は、異常終了する (終了ステータス=1)。それ以外の場合は正常に終了します。		
1	警告、エラー、致命的なエラーのいずれかが出力された場合は、異常終了する (終了ステータス=1)。それ以外の場合は正常に終了します。		
2	備考、警告、エラー、致命的なエラーのいずれかが出力された場合は、異常終了する (終了ステータス=1)。それ以外の場合は正常に終了します。		

-Xreg

-Xreg	関数	0..2	0
0	レジスタ候補の変数をレジスタに割り当てない。		

1	レジスタ候補の変数をレジスタに割り当て、グローバルおよびローカルのレジスタ割り当てを実行する。
2	レジスタ候補の変数をレジスタに割り当て、グローバルおよびローカルのレジスタ割り当てを実行する。より積極的なレジスタ最適化を実行。

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xrelaxalias

このスイッチはエイリアス分析ルールを制御します。

メモ:

- -Xrelaxalias=0 は、GCC の -fno-strict-aliasin と同等。
- -Xrelaxalias=2 は、GCC の -fstrict-aliasing とほぼ同等。

少なくとも、-Xrelaxalias=2 (C99 エイリアシング ルール) と動作するようにコードを記述または適合させることをお勧めします。より厳密なエイリアシング ルールにより、コンパイラでは、一部のケースにおいて、より優れたコードを生成することができるようになります。

より厳密なエイリアシング ルールを使用している場合でさえも、__may_alias 属性は、意図的にエイリアシング ポインタをマークするために使用することができます。詳細は、57 ページの「__may_alias__ 属性」を参照してください。

-Xrelaxalias	関数	0..3	0
0	エイリアス チェックを維持する。		
1	タイプ インスタンスが部分的に重複しないとする。		
2	ISO/ANSI C99 規格のセクション 6.5 に応じた、厳密な言語エイリアス規則を使用する。相互に「同様」ではないタイプは、エイリアスとはならない。		
3	厳密な言語エイリアス規則に加え、定数および非定数の変数もエイリアスとはならない。		

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xreorder

-Xreorder	関数	0..1	0
0	基本ブロックの並べ替えを行わない。		
1	最適な実行フローを目的とし、関数内で基本ブロックの並べ替えを試行する。		

-Xreserve

-Xreserve	ファイル	レジスタのリスト	空のリスト
reg{+*reg*...}	SNC では、通常のレジスタの割り当ておよび保存からのレジスタの削除が許可されています。以下の形式でリストを指定することによってこれらのレジスタを指定できます。 *reg1*{+*reg2*...} この reg1 や reg2 などはレジスタの識別子です。たとえば「-Xreserve=r14」では、グローバル レジスタとしてレジスタ r14 が予約されます。関数では、r14 の使用が回避されます。 -Xreserve は、__setreg() イントリンシックと組み合わせて使用することもできます。 メモ：PPU ABI において特定の用途があるレジスタを予約すると、未定義の結果の原因となります。		

-Xrestrict

-Xrestrict	関数	0..2	1
0	__restrict キーワードの影響を受けない。ポインタが他のポインタとエイリアスであると想定する場合。		
1	__restrict で修飾されたポインタが、他の __restrict 修飾ポインタをエイリアスしないと想定する場合。		
2	__restrict で修飾されたポインタが、他のポインタをエイリアスしないと想定する場合。		

-Xretpts

-Xretpts	関数	0..1	1
n	関数内のリターン ポイントの数を制御する。複数の「return」ステートメントのある関数では、すべての return ステートメントに対して関数エピローグ コードがデフォルトで生成される。このスイッチを 1 に設定した場合は、各 return ステートメントが共通の関数エピローグに分岐する別のモードが選択される。インライン化されたエピローグ コードはその実行が高速になるが、コード全体のサイズが大きくなる。		

-Xretstruct

これは、レジスタにおいて 1 つのプリミティブタイプ (int、float vector など) をラップするクラスや構造体を返す関数の結果を戻す、ABI 拡張オプションです。

この最適化では、VMX ベクターをラップする C++ クラスを使用する数値計算ライブラリにおいて、劇的な効果を得ることができます。

-Xretstruct	関数	0..2	0
0	**最適化なし (デフォルト)** ー これが標準 ABI です。		

1	ラップされたベクタータイプをレジスタに戻す。
2	ラップされたすべてのプリミティブタイプをレジスタに戻す。

ラッパーは、仮想関数がない構造体またはクラスで、プリミティブタイプのデータ メンバを 1 つ含みます。以下はその例です。

```
struct MyInt
{
    int mN;
};

class MyVector
{
    vector float mVec;
};
```

メモ:この最適化は、標準的 ABI との互換性がなく、ラッパー構造体を戻すコードを通じ、一貫して使用される必要があります。SDK をコールするコード、またはラッパー構造体を戻すその他ライブラリ関数は、-Xretstruct=0 でコンパイルされる必要があります。

-Xsaverestorefuncs

-Xsaverestorefuncs	関数	0..1	0
0	save/restore ミリコード関数を使用しない。		
1	save/restore ミリコード関数を使用する (GCC の -muse-save-restore-funcs スイッチと同様)。これにより、特定関数の初めにある標準的な save/restore コードが、必要なコード シーケンスを含む標準関数への分岐に置換される。通常、これはパフォーマンスが犠牲になるものの、コードのサイズを削減できる。save/restore ミリコード関数は、-Xsaverestorefuncs の設定に関係なく、__attribute__((cold)) でマークされた関数に対して自動的に使用される。これらが __attribute__((hot)) でマークされた関数に使用されることはなく (-Xsaverestorefuncs の設定には無関係)、__attribute__((inlinesrf)) でマークされた関数に使用されることもない (上記のその他ファクターには無関係)。		

-Xsched

-Xsched	関数	0..2	0
0	スケジューリングを実行しない。		
1	1 つ目のスケジューリングのみを実行する。		
2	両方のスケジューリングを実行する。		

メモ:この制御変数は、最適化制御グループ (-O) の設定による影響を受けます。詳細は、96ページの「最適化グループ (O)」を参照してください。

-Xshow

-Xshow	行	名前リスト	空のリスト
コントロール変数の名前	リストされている各コントロール変数の値を表示する。		

-Xsingleconst

-Xsingleconst	ファイル	0..1	0
0	接尾辞「f」のない浮動小数点の定数を、倍精度タイプ (double) として処理する。		
1	接尾辞「f」のない浮動小数点の定数を、単精度タイプ (float) として処理する。		

-Xsizet

-Xsizet	コンパイル	名前	uint
uint	size_t は、符号なしの int。		
ulong	size_t は、符号なしの long。		
ushort	size_t は、符号なしの short。		

-Xswbr

-Xswbr	コンパイル	0..1	1
0	連続的なケース ラベルを含む switch ステートメントに対し、ジャンプ アドレスのテーブルを生成し、少なくとも 5 ラベルを含む switch ステートメントに対して間接ジャンプを行う (ラベルが十分近いことが条件)。		
1	連続的なケース ラベルを含む switch ステートメントに対し、比較分岐ツリーの生成を強制する。		

-Xswmaxchain

-Xswmaxchain	関数	0..100	8
n	多数のケース ラベルを含む switch ステートメントに対して生成される、決定ツリー (compare/goto 命令) の最大長を決定する。-Xswmaxchain の値が大きいほど、コンパイラでは、より長い compare/goto 命令シーケンスが生成される。		

-Xtrigraphs

-Xtrigraphs	ファイル	0..1	0
0	トライグラフの使用をサポートしない。		

1	コード内のトライグラフ使用をサポートする。

-Xuninitwarn

-Xuninitwarn	ファイル	0..1	1
0	コンパイラのバックエンドから潜在的に未初期化状態にもどすされている変数の使用に対する警告を発行しない。		
1	コンパイラのバックエンドから潜在的に 未初期化状態にもどすされている変数の使用に対する警告を発行する。		

-Xunroll

-Xunroll	ループ	0..max int	0
0	ループをアンロールしない。		
1	自動制御でループをアンロールする。		
n>1	アンロール可能なループを、常に *n* 回アンロールする。		

-Xunrollssa

-Xunrollssa	関数	0..100	0
0	ループを展開しない。		
n	ループを、単独の基本ブロック ループに展開する。*n* 命令は、ループの最終サイズを示す。		
10	非常に小さなループを展開する。		
30	大きなループを展開する。		
100	極度のループ展開。実際のコードで役立つ可能性は低いものの、ベンチマークに対しては役立つ可能性もある。各コールド命令に約 20 サイクルが費やされるため、コードの実行速度は遅くなる。		

-Xuseatexit

-Xuseatexit	関数	0..1	0
0	静的変数のデストラクタ呼び出しに、静的テーブルを使用する。		
1	静的変数のデストラクタ呼び出しに、動的に作成・リンクされたリストを使用する。GNU -fuse-cxa-atexit スイッチと同等。これは、静的変数に対するデストラクタ呼び出しの完全適合逆順に必要とされます。これはコード領域や実行時間という点において若干かさみますが、正しい動作を得るためには (特に、1 つの静的変数のコンストラクタが終了前に別の静的変数を初期化する場合) 必要となります。 -Xuseatexit でコンパイルされた/されないオブジェクト ファイルを混ぜることができ		

	ます。ただしその場合、-Xuseatexit でコンパイルされた全ファイルに対するデストラクタが、-Xuseatexit でコンパイルされないファイルのデストラクタ前にコールされます。特定の SDK とミドルウェア ライブラリは、このオプションではコンパイルされません。このため、-Xuseatexit でコンパイルされたファイルの静的変数に対するデストラクタは、これらライブラリの前にコールされます。

-Xuseintcmp

-Xuseintcmp	関数	0..1	0
0	比較を変換しない。		
1	比較を整数処理に変換する。		

-Xwchart

-Xwchart	コンパイル	名前	ushort
char	wchar_t は、char。		
int	wchar_t は、int。		
long	wchar_t は、long。		
schar	wchar_t は、signed char。		
short	wchar_t は、short。		
unchar	wchar_t は、unsigned char。		
uint	wchar_t は、unsigned int。		
ulong	wchar_t は、unsigned long。		
ushort	wchar_t は、unsigned short。		

-Xwritable_strings

-Xwritable_strings	行	0..2	0
0	文字列を、読み取り専用のデータ セクション (.rdata など) に強制的に配置する。		
1	文字列を、ターゲットと言語に応じたデータ セクションに配置する。		
2	文字列を、書き込み可能のデータ セクション (.data など) に強制的に配置する。		

-Xzeroinit

-Xzeroinit	関数	0..1	0
0	コンストラクタへのコールの前に、コンパイラで生成されたコンストラクタがあるクラスのゼロ初期化をオフにする。		

1	コンストラクタへのコールの前に、コンパイラで生成されたコンストラクタがあるクラスのゼロ初期化をオンにする。

コントロール グループの参照テーブル

以下のサブセクションでは、コントロールについて説明します。

最適化グループ (O)

グループ名 = O、値 = 0.. 3、d 、s、デフォルト O=0。

グループ値	グループ メンバーの名前と値					
	alias	flow	reg	relax alias	sched	
0	0	0	0	0	0	
1	1	0	0	0	0	
2	3	1	2	2	2	
3	3	1	2	2	2	
d	3	1	1	0	0	
s	3	1	2	2	2	

グループ値	グループ メンバーの名前と値					
	autoinline size	constpool	inlinesize	postopt		
0	0	0	0	0		
1	0	0	16	0		
2	64	1	384	6		
3	64	1	384	6		
d	0	0	32	6		
s	32	1	128	6		

これらのコントロール変数の詳細は、35 ページの「最適化のコントロール変数」を参照してください。

9: 組み込み関数リファレンス

JSRE 組み込み関数

メモ 1 — JSRE 組み込み関数に関するサポート

メモ	戻り値	関数	定義される場所
読み込みデータ ストリームのアドレスと方向を指定。アドレスから開始し、キャッシュ ブロックのロードを継続して行います。	void	__dcbt_TH1000(void *address, unsigned direction, unsigned unlimited, unsigned id);	ppu_intrinsics.h
__dcbt_TH1000 で開始されたストリームを制御。ストリームの開始と停止、カウントや他フラグを指定します。	void	__dcbt_TH1010(unsigned go, unsigned stop, unsigned unit_count, unsigned transient, unsigned unlimited, unsigned id);	ppu_intrinsics.h
タイムベースを読み込む。ゼロの値はスキップ、下位 32ビット。	long long	__mftb();	ppu_intrinsics.h
L2 インストラクション キャッシュ ブロックを無効化。	void	__icbi(void *ptr);	ppu_intrinsics.h
L2 データ キャッシュ ブロックを無効化。	void	__dcbi(void *ptr);	ppu_intrinsics.h
L2 データ キャッシュ ブロックをフラッシュ。	void	__dcbf(void *ptr);	ppu_intrinsics.h
L2 データ キャッシュ ブロックをゼロにする。	void	__dcbz(void *ptr);	ppu_intrinsics.h
L2 データ キャッシュ ブロックを出力。	void	__dcbst(void *ptr);	ppu_intrinsics.h
L2 データ キャッシュ ブロックを保存用に読み込む。	void	__dcbtst(void *ptr);	ppu_intrinsics.h
L2 データ キャッシュ ブロックを非ストリーム形式で読み込む。	void	__dcbt(void *ptr);	built in
アトミック値のロードと保存。アトミックオペレーションについては、stwcx と使用。	unsigned	__lwarx(void *base);	ppu_intrinsics.h
アトミック値のロードと保存。アトミックオペレーションについては、stdcx と使用。	unsigned long long	__ldarx(void *base);	ppu_intrinsics.h
別のスレッドでアトミック値が保存されていない場合のみ、保存。	bool	__stwcx(void *base, unsigned value);	ppu_intrinsics.h

別のスレッドでアトミック値が保存されていない場合のみ、保存。	bool	__stdcx(void *base, unsigned long long value);	ppu_intrinsics.h
16 ビット値とリバースバイトをロード。	unsigned int	__lhbrx(void *base);	ppu_intrinsics.h
32 ビット値とリバースバイトをロード。	unsigned int	__lwbrx(void *base);	ppu_intrinsics.h
64 ビット値とリバースバイトをロード。	unsigned long long	__ldbrx(void *base);	ppu_intrinsics.h
16 ビット値とリバースバイトを保存。	void	__sthbrx(void *base, unsigned short value);	ppu_intrinsics.h
32 ビット値とリバースバイトを保存。	void	__stwbrx(void *base, unsigned int value);	ppu_intrinsics.h
64 ビット値とリバースバイトを保存。	void	__stdbrx(void *base, unsigned long long value);	ppu_intrinsics.h
先行ゼロをカウント、64 ビット。	unsigned long long	__cntlzd(long long a);	built in
先行ゼロをカウント、32 ビット。	long long	__cntlzw(long long a);	built in
重量 (Heavyweight) データ同期で、すべての書き込みを確実に完了。	void	__sync();	ppu_intrinsics.h
コードの変更前に使用される同期命令。	void	__isync();	ppu_intrinsics.h
軽量 (Lightweight) メモリ同期。	void	__lwsync();	ppu_intrinsics.h
メモリマップされた I/O に対する、重量 (Heavyweight) 同期。	void	__eieio();	ppu_intrinsics.h
double 値を 64ビット integer に変換。	long long	__fctid(double a);	built in
double 値を 32 ビット integer に変換。	long long	__fctiw(double a);	built in
64ビット値を double に変換。	double	__fcfid(long long a);	built in
浮動小数点ステータスから移行。	double	__mffs();	ppu_intrinsics.h
マスクを使用し、浮動少数点ステータスに移行。	void	__mtfsf(int mask, double value);	ppu_intrinsics.h
直ちに浮動小数点ステータスに移行。	void	__mtfsfi(int bits, int field);	ppu_intrinsics.h
浮動小数点ステータス ビットをクリア。	void	__mtfsb0(int bit);	ppu_intrinsics.h
浮動小数点ステータス ビットを設定。	void	__mtfsb1(int bit);	ppu_intrinsics.h
浮動小数点ステータスを設定、	double	__setflm(double a);	ppu_intrinsics.h

古い値を戻す。			
左に回転して挿入、64 ビット。	long long	__rldimi(long long a, long long b, unsigned char sh, unsigned char mb);	ppu_intrinsics.h
左に回転してクリア、64 ビット。	long long	__rldic(long long a, unsigned char sh, unsigned char mb);	ppu_intrinsics.h
左に回転して左をクリア、64 ビット。	long long	__rldicl(long long a, unsigned char sh, unsigned char mb);	ppu_intrinsics.h
左に回転して右をクリア、64 ビット。	long long	__rldicr(long long a, unsigned char sh, unsigned char me);	ppu_intrinsics.h
左に回転して右をクリア、64 ビットのマイクロコード バージョン。	long long	__rldcr(long long a, long long sh, unsigned char me);	ppu_intrinsics.h
左に回転して左をクリア、64 ビットのマイクロコード バージョン。	long long	__rldcl(long long a, long long sh, unsigned char mb);	ppu_intrinsics.h
左に回転して挿入、32 ビット。	unsigned	__rlwimi(long long a, long long b, unsigned char sh, unsigned char mb, unsigned char me);	ppu_intrinsics.h
左に回転して挿入、32 ビット。	unsigned	__rlwinm(long long a, unsigned char sh, unsigned char mb, unsigned char me);	ppu_intrinsics.h
左に回転して挿入、32 ビットのマイクロコード バージョン。	unsigned	__rlwnm(long long a, long long sh, unsigned char mb, unsigned char me);	ppu_intrinsics.h
メモ 1	void	__cctph( );	built in
メモ 1	void	__cctpl( );	built in
メモ 1	void	__cctpm( );	built in
メモ 1	unsigned long long	__cntlzd(unsigned long long);	built in
メモ 1	unsigned long long	__cntlzw(unsigned long long);	built in
メモ 1	void	__db10cyc( );	built in
メモ 1	void	__db12cyc( );	built in
メモ 1	void	__db16cyc( );	built in
メモ 1	void	__db8cyc( );	built in
メモ 1	void	__dcbt(const void *);	built in
メモ 1	double	__fabs(double);	built in
メモ 1	float	__fabsf(float);	built in
メモ 1	double	__fctid(double);	built in
メモ 1	double	__fctiw(double);	built in
メモ 1	double	__fsel(double, double, double);	built in

メモ 1	float	__fsqrts(float);	built in
メモ 1	unsigned long long	__mfspr(int);	built in
メモ 1	unsigned long long	__mftb( );	built in
メモ 1	void	__nop( );	built in

SNC/GCC 組み込み関数

メモ 2 – asm 変換に関する PPU インストラクション。64ビット PEM を参照。

メモ 3 – altivec.h の実装に GCC で使用される内部組み込み関数。

メモ	戻り値	関数	定義される場所
メモ 2	long long	__addc(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__adde(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__addic(long long a, short b);	ppu_asm_intrinsics.h
メモ 2	long long	__addme(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__addze(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__subfc(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__subfe(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__subfme(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__subfze(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__subfic(long long a, const short b);	ppu_asm_intrinsics.h
メモ 2	long long	__srad(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__sradi(long long a, unsigned char b);	ppu_asm_intrinsics.h
メモ 2	long long	__sraw(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__srawi(long long a, unsigned char b);	ppu_asm_intrinsics.h
メモ 2	long long	__add(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__addi(long long a, short b);	ppu_asm_intrinsics.h
メモ 2	long long	__addis(long long a, short b);	ppu_asm_intrinsics.h
メモ 2	long long	__subf(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__neg(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__divd(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__divdu(unsigned long long a, unsigned long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__divw(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__divwu(long long a, long long b);	ppu_asm_intrinsics.h

メモ 2	long long	__mulhd(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__mulhdu(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__mulhw(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__mulhwu(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__mulld(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__mulli(long long a, short b);	ppu_asm_intrinsics.h
メモ 2	long long	__mullw(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__extsb(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__extsh(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__extsw(long long a);	ppu_asm_intrinsics.h
メモ 2	long long	__and(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__andc(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__eqv(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__nand(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__nor(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__or(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__orc(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__ori(long long a, unsigned short b);	ppu_asm_intrinsics.h
メモ 2	long long	__oris(long long a, unsigned short b);	ppu_asm_intrinsics.h
メモ 2	long long	__xor(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__xori(long long a, const unsigned short b);	ppu_asm_intrinsics.h
メモ 2	long long	__xoris(long long a, const unsigned short b);	ppu_asm_intrinsics.h
メモ 2	double	__fadd(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fadds(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fdiv(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fdivs(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fmadd(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fmadds(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fmr(double b);	ppu_asm_intrinsics.h
メモ 2	double	__fmsubs(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fmsub(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fmul(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fmuls(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fnabs(double a);	ppu_asm_intrinsics.h
メモ 2	double	__fnabsf(double a);	ppu_asm_intrinsics.h

メモ 2	double	__fneg(double a);	ppu_asm_intrinsics.h
メモ 2	double	__fnmadd(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fnmadds(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fnmsub(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	double	__fnmsubs(double a, double b, double c);	ppu_asm_intrinsics.h
メモ 2	float	__fres(float a);	ppu_asm_intrinsics.h
メモ 2	double	__fsqrt(double a);	ppu_asm_intrinsics.h
メモ 2	double	__frsp(double a);	ppu_asm_intrinsics.h
メモ 2	float	__fsels(float a, float b, float c);	ppu_asm_intrinsics.h
メモ 2	double	__frsqrte(double x);	ppu_asm_intrinsics.h
メモ 2	double	__fsub(double a, double b);	ppu_asm_intrinsics.h
メモ 2	double	__fsubs(double a, double b);	ppu_asm_intrinsics.h
メモ 2	long long	__fctiwz(double a);	ppu_asm_intrinsics.h
メモ 2	long long	__lbz(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	long long	__lbzx(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	long long	__ld(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	long long	__ldx(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	double	__lfd(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	double	__lfdx(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	double	__lfs(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	double	__lfsx(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	long long	__lha(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	long long	__lhax(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	long long	__lhz(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	long long	__lhzx(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	long long	__lwa(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	long long	__lwax(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	long long	__lwz(const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	long long	__lwzx(void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	long long	__sld(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__sldi(long long a, unsigned char b);	ppu_asm_intrinsics.h
メモ 2	long long	__slw(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__slwi(long long a, unsigned char b);	ppu_asm_intrinsics.h
メモ 2	long long	__srd(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__srdi(long long a, long long b);	ppu_asm_intrinsics.h

メモ 2	long long	__srw(long long a, long long b);	ppu_asm_intrinsics.h
メモ 2	long long	__srwi(long long a, unsigned char b);	ppu_asm_intrinsics.h
メモ 2	void	__stb(long long a, const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	void	__stbx(long long a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	void	__std(long long a, const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	void	__stdx(long long a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	void	__stfd(double a, const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	void	__stfdx(double a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	void	__stfs(double a, const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	void	__stfsx(double a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	void	__sth(long long a, const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	void	__sthx(long long a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	void	__stw(long long a, const short offset, void *p);	ppu_asm_intrinsics.h
メモ 2	void	__stwx(long long a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	void	__stfiwx(double a, void *p, long long offset);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lbzu(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	unsigned	__ldu(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	double	__lfdu(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	float	__lfsu(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lhau(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lhzu(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lwau(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lwzu(int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	void	__stbu(long long value, int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	void	__stdu(long long value, int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	void	__stfdu(long long value, int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	void	__stfsu(long long value, int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	void	__sthu(long long value, int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	void	__stwu(long long value, int offset, void *&p);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lbzux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	unsigned	__ldux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	double	__lfdux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	float	__lfsux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lhaux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lhzux(void *&p, int offset);	ppu_asm_intrinsics.h

メモ 2	unsigned	__lwaux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	unsigned	__lwzux(void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	void	__stbux(long long value, void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	void	__stdux(long long value, void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	void	__stfdux(long long value, void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	void	__stfsux(long long value, void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	void	__sthux(long long value, void *&p, int offset);	ppu_asm_intrinsics.h
メモ 2	void	__stwux(long long value, void *&p, int offset);	ppu_asm_intrinsics.h
メモ 3	vector signed int	__builtin_altivec_vaddcuw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vaddfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vaddsbs(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vaddshs(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vaddsws(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vaddubm(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vaddubs(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vadduhm(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vadduhs(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vadduwm(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vadduws(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vand(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vandc(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vavgsb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vavgsh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vavgsw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vavgub(vector signed char a, vector signed char b);	ppu_altivec_internals.h

メモ 3	vector signed short	__builtin_altivec_vavguh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vavguw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vcfsx(vector signed int a, const int b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vcfux(vector signed int a, const int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vcmpbfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vcmpeqfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector bool char	__builtin_altivec_vcmpequb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector bool short	__builtin_altivec_vcmpequh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector bool int	__builtin_altivec_vcmpequw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vcmpgefp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vcmpgtfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector bool char	__builtin_altivec_vcmpgtsb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector bool short	__builtin_altivec_vcmpgtsh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector bool int	__builtin_altivec_vcmpgtsw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector bool char	__builtin_altivec_vcmpgtub(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector bool short	__builtin_altivec_vcmpgtuh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector bool int	__builtin_altivec_vcmpgtuw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vctsxs(vector float a, unsigned char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vctuxs(vector float a, unsigned char b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vexptefp(vector float a);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vlogefp(vector float a);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vmaddfp(vector float a, vector float b, vector float c);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vmaxfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vmaxsb(vector signed char a, vector signed char b);	ppu_altivec_internals.h

メモ 3	vector signed short	__builtin_altivec_vmaxsh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmaxsw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vmaxub(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmaxuh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmaxuw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmhaddshs(vector signed short a, vector signed short b, vector signed short c);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmhraddshs(vector signed short a, vector signed short b, vector signed short c);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vminfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vminsb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vminsh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vminsw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vminub(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vminuh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vminuw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmladduhm(vector signed short a, vector signed short b, vector signed short c);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vmrghb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmrghh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmrghw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vmrglb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmrglh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector	__builtin_altivec_vmrglw(vector signed int a,	ppu_altivec_internals.h

	signed int	vector signed int b);	
メモ 3	vector signed int	__builtin_altivec_vmsummbm(vector signed char a, vector signed char b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmsumshm(vector signed short a, vector signed short b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmsumshs(vector signed short a, vector signed short b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmsumubm(vector signed char a, vector signed char b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmsumuhm(vector signed short a, vector signed short b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmsumuhs(vector signed short a, vector signed short b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmulesb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmulesh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmuleub(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmuleuh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmulosb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmulosh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vmuloub(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vmulouh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vnmsubfp(vector float a, vector float b, vector float c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vnor(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vor(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vperm_4si(vector signed int a, vector signed int b, vector signed char c);	ppu_altivec_internals.h
メモ 3	vector pixel	__builtin_altivec_vpkpx(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vpkshss(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vpkshus(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed	__builtin_altivec_vpkswss(vector signed int a,	ppu_altivec_internals.h

	short	vector signed int b);	
メモ 3	vector signed short	__builtin_altivec_vpkswus(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vpkuhum(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vpkuhus(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vpkuwum(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vpkuwus(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vrefp(vector float a);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vrfim(vector float a);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vrfin(vector float a);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vrfip(vector float a);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vrfiz(vector float a);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vrlb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vrlh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vrlw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vrsqrtefp(vector float a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsel_4si(vector signed int a, vector signed int b, vector signed int c);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsl(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vslb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsldoi_4si(vector signed int a, vector signed int b, unsigned char c);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vslh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vslo(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vslw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vspltb(vector signed char a, unsigned char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vsplth(vector signed short a, unsigned char b);	ppu_altivec_internals.h

メモ 3	vector signed char	__builtin_altivec_vspltisb(signed char a);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vspltish(signed char a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vspltisw(signed char a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vspltw(vector signed int a, unsigned char b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsr(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vsrab(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vsrah(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsraw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vsrb(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vsrh(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsro(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsrw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsubcuw(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_vsubfp(vector float a, vector float b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vsubsbs(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vsubshs(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsubsws(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vsububm(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_vsububs(vector signed char a, vector signed char b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vsubuhm(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vsubuhs(vector signed short a, vector signed short b);	ppu_altivec_internals.h
メモ 3	vector	__builtin_altivec_vsubuwm(vector signed int a,	ppu_altivec_internals.h

	signed int	vector signed int b);	
メモ 3	vector signed int	__builtin_altivec_vsubuws(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsum2sws(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsum4sbs(vector signed char a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsum4shs(vector signed short a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsum4ubs(vector signed char a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vsumsws(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vupkhpx(vector signed short a);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vupkhsb(vector signed char a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vupkhsh(vector signed short a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vupklpx(vector signed short a);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_vupklsb(vector signed char a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vupklsh(vector signed short a);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_vxor(vector signed int a, vector signed int b);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_lvebx(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector signed short	__builtin_altivec_lvehx(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector signed int	__builtin_altivec_lvewx(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_lvlx(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_lvlxl(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_lvrx(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector float	__builtin_altivec_lvrxl(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_lvsl(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_lvsr(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	vector signed char	__builtin_altivec_lvx(long long offset, void *p);	ppu_altivec_internals.h

メモ 3	vector signed char	__builtin_altivec_lvxl(long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvebx(vector signed char a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvehx(vector signed short a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvewx(vector signed int a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvlx(vector signed char a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvlxl(vector signed char a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvrx(vector signed char a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvrxl(vector signed char a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvx(vector signed int a, long long offset, void *p);	ppu_altivec_internals.h
メモ 3	void	__builtin_altivec_stvxl(vector signed int a, long long offset, void *p);	ppu_altivec_internals.h

SNC 組み込み関数

メモ 4 – Gcc の __builtin と同等

以下の SNC 組み込み関数はコンパイラにすべて組み込まれています。

メモ	戻り値	関数
メモ 4	unsigned int	__builtin_cellAtomicAdd32(unsigned int *, unsigned int);
メモ 4	unsigned long long	__builtin_cellAtomicAdd64(unsigned long long *, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicAnd32(unsigned int *, unsigned int);
メモ 4	unsigned long long	__builtin_cellAtomicAnd64(unsigned long long *, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicCompareAndSwap32(unsigned int *, unsigned int, unsigned int);
メモ 4	unsigned long long	__builtin_cellAtomicCompareAndSwap64(unsigned long long *, unsigned long long, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicDecr32(unsigned int *);
メモ 4	unsigned long long	__builtin_cellAtomicDecr64(unsigned long long *);
メモ 4	unsigned int	__builtin_cellAtomicIncr32(unsigned int *);
メモ 4	unsigned long long	__builtin_cellAtomicIncr64(unsigned long long *);
メモ 4	unsigned int	__builtin_cellAtomicLockLine32(unsigned int *);

メモ 4	unsigned long long	__builtin_cellAtomicLockLine64(unsigned long long *);
メモ 4	unsigned int	__builtin_cellAtomicNop32(unsigned int *);
メモ 4	unsigned long long	__builtin_cellAtomicNop64(unsigned long long *);
メモ 4	unsigned int	__builtin_cellAtomicOr32(unsigned int *, unsigned int);
メモ 4	unsigned long long	__builtin_cellAtomicOr64(unsigned long long *, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicStore32(unsigned int *, unsigned int);
メモ 4	unsigned long long	__builtin_cellAtomicStore64(unsigned long long *, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicStoreConditional32(unsigned int *, unsigned int);
メモ 4	unsigned int	__builtin_cellAtomicStoreConditional64(unsigned long long *, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicSub32(unsigned int *, unsigned int);
メモ 4	unsigned long long	__builtin_cellAtomicSub64(unsigned long long *, unsigned long long);
メモ 4	unsigned int	__builtin_cellAtomicTestAndDecr32(unsigned int *);
メモ 4	unsigned long long	__builtin_cellAtomicTestAndDecr64(unsigned long long *);
メモ 4	int	__builtin_clz(int);
メモ 4	unsigned long long	__builtin_clzl(unsigned long long);
メモ 4	unsigned long long	__builtin_clzll(unsigned long long);
メモ 4	int	__builtin_constant_p(int);
メモ 4	void	__builtin_dcbf(const void *, int);
メモ 4	void	__builtin_dcbi(void *, int);
メモ 4	void	__builtin_dcbst(const void *, int);
メモ 4	void	__builtin_dcbt(const void *, int);
メモ 4	void	__builtin_dcbt3(unsigned int, int, int);
メモ 4	void	__builtin_dcbtst(void *, long long);
メモ 4	void	__builtin_dcbz(void *, int);
メモ 4	void	__builtin_eieio( );
メモ 4	int	__builtin_expect(int, int);
メモ 4	double	__builtin_fabs(double);
メモ 4	float	__builtin_fabsf(float);
メモ 4	double	__builtin_fcfid(double);
メモ 4	double	__builtin_fctid(double);
メモ 4	double	__builtin_fctidz(double);

メモ 4	double	__builtin_fctiw(double);
メモ 4	double	__builtin_fctiwz(double);
メモ 4	void	__builtin_fence( );
メモ 4	double	__builtin_fmadd(double, double, double);
メモ 4	float	__builtin_fmadds(float, float, float);
メモ 4	double	__builtin_fmsub(double, double, double);
メモ 4	float	__builtin_fmsubs(float, float, float);
メモ 4	double	__builtin_fnabs(double);
メモ 4	float	__builtin_fnabsf(float);
メモ 4	double	__builtin_fnmadd(double, double, double);
メモ 4	float	__builtin_fnmadds(float, float, float);
メモ 4	double	__builtin_fnmsub(double, double, double);
メモ 4	float	__builtin_fnmsubs(float, float, float);
メモ 4	void *	__builtin_frame_address( );
メモ 4	double	__builtin_fre(double);
メモ 4	double	__builtin_frsqrte(double);
メモ 4	float	__builtin_frsqrtes(float);
メモ 4	double	__builtin_fsel(double, double, double);
メモ 4	float	__builtin_fsels(float, float, float);
メモ 4	double	__builtin_fsqrt(double);
メモ 4	float	__builtin_fsqrts(float);
メモ 4	long	__builtin_get_toc( );
メモ 4	void	__builtin_icbi(void *, long long);
メモ 4	void	__builtin_isync( );
メモ 4	long long	__builtin_ldarx(void *, long long);
メモ 4	unsigned int	__builtin_ldbrx(const void *, int);
メモ 4	unsigned int	__builtin_lhbrx(const void *, int);
メモ 4	unsigned int	__builtin_lwarx(void *, long long);
メモ 4	unsigned int	__builtin_lwbrx(const void *, int);
メモ 4	void	__builtin_lwsync( );
メモ 4	void	__builtin_mb( );
メモ 4	volatile double	__builtin_mffs( );
メモ 4	unsigned long long	__builtin_mftb( );
メモ 4	long long	__builtin_mtfsb0(int);

メモ 4	long long	__builtin_mtfsb1(int);
メモ 4	void	__builtin_mtfsf(int, double);
メモ 4	void	__builtin_mtfsfi(int, int);
メモ 4	long long	__builtin_mulhd(long long, long long);
メモ 4	long long	__builtin_mulhdu(long long, long long);
メモ 4	long long	__builtin_mulhw(long long, long long);
メモ 4	long long	__builtin_mulhwu(long long, long long);
タイム ベース レジスタの下位 32 ビットを取得。	unsigned int	__builtin_raw_mftb( );
メモ 4	void *	__builtin_return_address( );
メモ 4	double	__builtin_setflm(double);
メモ 4	void	__builtin_snpause( );
メモ 4	void	__builtin_stdbrx(unsigned int, void *, int);
メモ 4	int	__builtin_stdcx(unsigned long long, void *, long long);
メモ 4	void	__builtin_stfiwx(double, void *, int);
メモ 4	void	__builtin_sthbrx(unsigned short, void *, int);
メモ 4	void	__builtin_stop( );
メモ 4	void	__builtin_stwbrx(unsigned int, void *, int);
メモ 4	int	__builtin_stwcx(unsigned int, void *, long long);
メモ 4	void	__builtin_sync( );
メモ 4	void	__builtin_trap( );
メモ 4	void	__cctph( );
メモ 4	void	__cctpl( );
メモ 4	void	__cctpm( );
メモ 4	unsigned long long	__cntlzd(unsigned long long);
メモ 4	unsigned long long	__cntlzw(unsigned long long);
メモ 4	void	__db10cyc( );
メモ 4	void	__db12cyc( );
メモ 4	void	__db16cyc( );
メモ 4	void	__db8cyc( );
メモ 4	void	__dcbt(const void *);
メモ 4	double	__fabs(double);
メモ 4	float	__fabsf(float);

メモ 4	double	__fctid(double);
メモ 4	double	__fctiw(double);
メモ 4	double	__fsel(double, double, double);
メモ 4	float	__fsqrts(float);
メモ 4	unsigned long long	__mfspr(int);
メモ 4	unsigned long long	__mftb( );
メモ 4	void	__nop( );
コール/システム コール後、直ちにレジスタ値を取得。	unsigned long long	__reg(int);

Altivec 組み込み関数

Altivec Programming Interface Manual (PIM) を参照。以下の Altivec 組み込み関数はコンパイラにすべて組み込まれています。

戻り値	関数
vector float	vec_abs(vector float);
vector signed short	vec_abs(vector signed short);
vector signed int	vec_abs(vector signed int);
vector signed char	vec_abs(vector signed char);
vector signed short	vec_abss(vector signed short);
vector signed int	vec_abss(vector signed int);
vector signed char	vec_abss(vector signed char);
vector float	vec_add(vector float, vector float);
vector signed char	vec_add(vector bool char, vector signed char);
vector unsigned char	vec_add(vector bool char, vector unsigned char);
vector signed char	vec_add(vector signed char, vector bool char);
vector signed char	vec_add(vector signed char, vector signed char);
vector unsigned char	vec_add(vector unsigned char, vector bool char);
vector unsigned char	vec_add(vector unsigned char, vector unsigned char);
vector signed short	vec_add(vector bool short, vector signed short);
vector unsigned short	vec_add(vector bool short, vector unsigned short);
vector signed short	vec_add(vector signed short, vector bool short);
vector signed short	vec_add(vector signed short, vector signed short);

vector unsigned short	vec_add(vector unsigned short, vector bool short);
vector unsigned short	vec_add(vector unsigned short, vector unsigned short);
vector signed int	vec_add(vector bool long, vector signed int);
vector unsigned int	vec_add(vector bool long, vector unsigned int);
vector signed int	vec_add(vector signed int, vector bool long);
vector signed int	vec_add(vector signed int, vector signed int);
vector unsigned int	vec_add(vector unsigned int, vector bool long);
vector unsigned int	vec_add(vector unsigned int, vector unsigned int);
vector unsigned int	vec_addc(vector unsigned int, vector unsigned int);
vector signed char	vec_adds(vector bool char, vector signed char);
vector signed char	vec_adds(vector signed char, vector bool char);
vector signed char	vec_adds(vector signed char, vector signed char);
vector signed short	vec_adds(vector bool short, vector signed short);
vector signed short	vec_adds(vector signed short, vector bool short);
vector signed short	vec_adds(vector signed short, vector signed short);
vector signed int	vec_adds(vector bool long, vector signed int);
vector signed int	vec_adds(vector signed int, vector bool long);
vector signed int	vec_adds(vector signed int, vector signed int);
vector unsigned char	vec_adds(vector bool char, vector unsigned char);
vector unsigned char	vec_adds(vector unsigned char, vector bool char);
vector unsigned char	vec_adds(vector unsigned char, vector unsigned char);
vector unsigned short	vec_adds(vector bool short, vector unsigned short);
vector unsigned short	vec_adds(vector unsigned short, vector bool short);
vector unsigned short	vec_adds(vector unsigned short, vector unsigned short);
vector unsigned int	vec_adds(vector bool long, vector unsigned int);
vector unsigned int	vec_adds(vector unsigned int, vector bool long);
vector unsigned int	vec_adds(vector unsigned int, vector unsigned int);
int	vec_all_eq(vector bool short, vector bool short);
int	vec_all_eq(vector bool short, vector signed short);
int	vec_all_eq(vector bool short, vector unsigned short);
int	vec_all_eq(vector bool long, vector bool long);
int	vec_all_eq(vector bool long, vector signed int);
int	vec_all_eq(vector bool long, vector unsigned int);
int	vec_all_eq(vector bool char, vector bool char);
int	vec_all_eq(vector bool char, vector signed char);
int	vec_all_eq(vector bool char, vector unsigned char);
int	vec_all_eq(vector float, vector float);

int	vec_all_eq(vector pixel, vector pixel);
int	vec_all_eq(vector signed short, vector bool short);
int	vec_all_eq(vector signed short, vector signed short);
int	vec_all_eq(vector signed int, vector bool long);
int	vec_all_eq(vector signed int, vector signed int);
int	vec_all_eq(vector signed char, vector bool char);
int	vec_all_eq(vector signed char, vector signed char);
int	vec_all_eq(vector unsigned short, vector bool short);
int	vec_all_eq(vector unsigned short, vector unsigned short);
int	vec_all_eq(vector unsigned int, vector bool long);
int	vec_all_eq(vector unsigned int, vector unsigned int);
int	vec_all_eq(vector unsigned char, vector bool char);
int	vec_all_eq(vector unsigned char, vector unsigned char);
int	vec_all_ge(vector bool short, vector signed short);
int	vec_all_ge(vector bool short, vector unsigned short);
int	vec_all_ge(vector bool long, vector signed int);
int	vec_all_ge(vector bool long, vector unsigned int);
int	vec_all_ge(vector bool char, vector signed char);
int	vec_all_ge(vector bool char, vector unsigned char);
int	vec_all_ge(vector float, vector float);
int	vec_all_ge(vector signed short, vector bool short);
int	vec_all_ge(vector signed short, vector signed short);
int	vec_all_ge(vector signed int, vector bool long);
int	vec_all_ge(vector signed int, vector signed int);
int	vec_all_ge(vector signed char, vector bool char);
int	vec_all_ge(vector signed char, vector signed char);
int	vec_all_ge(vector unsigned short, vector bool short);
int	vec_all_ge(vector unsigned short, vector unsigned short);
int	vec_all_ge(vector unsigned int, vector bool long);
int	vec_all_ge(vector unsigned int, vector unsigned int);
int	vec_all_ge(vector unsigned char, vector bool char);
int	vec_all_ge(vector unsigned char, vector unsigned char);
int	vec_all_gt(vector bool short, vector signed short);
int	vec_all_gt(vector bool short, vector unsigned short);
int	vec_all_gt(vector bool long, vector signed int);
int	vec_all_gt(vector bool long, vector unsigned int);
int	vec_all_gt(vector bool char, vector signed char);

int	vec_all_gt(vector bool char, vector unsigned char);
int	vec_all_gt(vector float, vector float);
int	vec_all_gt(vector signed short, vector bool short);
int	vec_all_gt(vector signed short, vector signed short);
int	vec_all_gt(vector signed int, vector bool long);
int	vec_all_gt(vector signed int, vector signed int);
int	vec_all_gt(vector signed char, vector bool char);
int	vec_all_gt(vector signed char, vector signed char);
int	vec_all_gt(vector unsigned short, vector bool short);
int	vec_all_gt(vector unsigned short, vector unsigned short);
int	vec_all_gt(vector unsigned int, vector bool long);
int	vec_all_gt(vector unsigned int, vector unsigned int);
int	vec_all_gt(vector unsigned char, vector bool char);
int	vec_all_gt(vector unsigned char, vector unsigned char);
int	vec_all_in(vector float, vector float);
int	vec_all_le(vector bool short, vector signed short);
int	vec_all_le(vector bool short, vector unsigned short);
int	vec_all_le(vector bool long, vector signed int);
int	vec_all_le(vector bool long, vector unsigned int);
int	vec_all_le(vector bool char, vector signed char);
int	vec_all_le(vector bool char, vector unsigned char);
int	vec_all_le(vector float, vector float);
int	vec_all_le(vector signed short, vector bool short);
int	vec_all_le(vector signed short, vector signed short);
int	vec_all_le(vector signed int, vector bool long);
int	vec_all_le(vector signed int, vector signed int);
int	vec_all_le(vector signed char, vector bool char);
int	vec_all_le(vector signed char, vector signed char);
int	vec_all_le(vector unsigned short, vector bool short);
int	vec_all_le(vector unsigned short, vector unsigned short);
int	vec_all_le(vector unsigned int, vector bool long);
int	vec_all_le(vector unsigned int, vector unsigned int);
int	vec_all_le(vector unsigned char, vector bool char);
int	vec_all_le(vector unsigned char, vector unsigned char);
int	vec_all_lt(vector bool short, vector signed short);
int	vec_all_lt(vector bool short, vector unsigned short);
int	vec_all_lt(vector bool long, vector signed int);

int	vec_all_lt(vector bool long, vector unsigned int);
int	vec_all_lt(vector bool char, vector signed char);
int	vec_all_lt(vector bool char, vector unsigned char);
int	vec_all_lt(vector float, vector float);
int	vec_all_lt(vector signed short, vector bool short);
int	vec_all_lt(vector signed short, vector signed short);
int	vec_all_lt(vector signed int, vector bool long);
int	vec_all_lt(vector signed int, vector signed int);
int	vec_all_lt(vector signed char, vector bool char);
int	vec_all_lt(vector signed char, vector signed char);
int	vec_all_lt(vector unsigned short, vector bool short);
int	vec_all_lt(vector unsigned short, vector unsigned short);
int	vec_all_lt(vector unsigned int, vector bool long);
int	vec_all_lt(vector unsigned int, vector unsigned int);
int	vec_all_lt(vector unsigned char, vector bool char);
int	vec_all_lt(vector unsigned char, vector unsigned char);
int	vec_all_nan(vector float);
int	vec_all_ne(vector bool short, vector bool short);
int	vec_all_ne(vector bool short, vector signed short);
int	vec_all_ne(vector bool short, vector unsigned short);
int	vec_all_ne(vector bool long, vector bool long);
int	vec_all_ne(vector bool long, vector signed int);
int	vec_all_ne(vector bool long, vector unsigned int);
int	vec_all_ne(vector bool char, vector bool char);
int	vec_all_ne(vector bool char, vector signed char);
int	vec_all_ne(vector bool char, vector unsigned char);
int	vec_all_ne(vector float, vector float);
int	vec_all_ne(vector pixel, vector pixel);
int	vec_all_ne(vector signed short, vector bool short);
int	vec_all_ne(vector signed short, vector signed short);
int	vec_all_ne(vector signed int, vector bool long);
int	vec_all_ne(vector signed int, vector signed int);
int	vec_all_ne(vector signed char, vector bool char);
int	vec_all_ne(vector signed char, vector signed char);
int	vec_all_ne(vector unsigned short, vector bool short);
int	vec_all_ne(vector unsigned short, vector unsigned short);
int	vec_all_ne(vector unsigned int, vector bool long);

int	vec_all_ne(vector unsigned int, vector unsigned int);
int	vec_all_ne(vector unsigned char, vector bool char);
int	vec_all_ne(vector unsigned char, vector unsigned char);
int	vec_all_nge(vector float, vector float);
int	vec_all_ngt(vector float, vector float);
int	vec_all_nle(vector float, vector float);
int	vec_all_nlt(vector float, vector float);
int	vec_all_numeric(vector float);
vector bool short	vec_and(vector bool short, vector bool short);
vector signed short	vec_and(vector bool short, vector signed short);
vector unsigned short	vec_and(vector bool short, vector unsigned short);
vector bool long	vec_and(vector bool long, vector bool long);
vector float	vec_and(vector bool long, vector float);
vector signed int	vec_and(vector bool long, vector signed int);
vector unsigned int	vec_and(vector bool long, vector unsigned int);
vector bool char	vec_and(vector bool char, vector bool char);
vector signed char	vec_and(vector bool char, vector signed char);
vector unsigned char	vec_and(vector bool char, vector unsigned char);
vector float	vec_and(vector float, vector bool long);
vector float	vec_and(vector float, vector float);
vector signed short	vec_and(vector signed short, vector bool short);
vector signed short	vec_and(vector signed short, vector signed short);
vector signed int	vec_and(vector signed int, vector bool long);
vector signed int	vec_and(vector signed int, vector signed int);
vector signed char	vec_and(vector signed char, vector bool char);
vector signed char	vec_and(vector signed char, vector signed char);
vector unsigned short	vec_and(vector unsigned short, vector bool short);
vector unsigned short	vec_and(vector unsigned short, vector unsigned short);
vector unsigned int	vec_and(vector unsigned int, vector bool long);
vector unsigned int	vec_and(vector unsigned int, vector unsigned int);
vector unsigned char	vec_and(vector unsigned char, vector bool char);
vector unsigned char	vec_and(vector unsigned char, vector unsigned char);
vector bool short	vec_andc(vector bool short, vector bool short);
vector signed short	vec_andc(vector bool short, vector signed short);
vector unsigned short	vec_andc(vector bool short, vector unsigned short);
vector bool long	vec_andc(vector bool long, vector bool long);
vector float	vec_andc(vector bool long, vector float);

vector signed int	vec_andc(vector bool long, vector signed int);
vector unsigned int	vec_andc(vector bool long, vector unsigned int);
vector bool char	vec_andc(vector bool char, vector bool char);
vector signed char	vec_andc(vector bool char, vector signed char);
vector unsigned char	vec_andc(vector bool char, vector unsigned char);
vector float	vec_andc(vector float, vector bool long);
vector float	vec_andc(vector float, vector float);
vector signed short	vec_andc(vector signed short, vector bool short);
vector signed short	vec_andc(vector signed short, vector signed short);
vector signed int	vec_andc(vector signed int, vector bool long);
vector signed int	vec_andc(vector signed int, vector signed int);
vector signed char	vec_andc(vector signed char, vector bool char);
vector signed char	vec_andc(vector signed char, vector signed char);
vector unsigned short	vec_andc(vector unsigned short, vector bool short);
vector unsigned short	vec_andc(vector unsigned short, vector unsigned short);
vector unsigned int	vec_andc(vector unsigned int, vector bool long);
vector unsigned int	vec_andc(vector unsigned int, vector unsigned int);
vector unsigned char	vec_andc(vector unsigned char, vector bool char);
vector unsigned char	vec_andc(vector unsigned char, vector unsigned char);
int	vec_any_eq(vector bool short, vector bool short);
int	vec_any_eq(vector bool short, vector signed short);
int	vec_any_eq(vector bool short, vector unsigned short);
int	vec_any_eq(vector bool long, vector bool long);
int	vec_any_eq(vector bool long, vector signed int);
int	vec_any_eq(vector bool long, vector unsigned int);
int	vec_any_eq(vector bool char, vector bool char);
int	vec_any_eq(vector bool char, vector signed char);
int	vec_any_eq(vector bool char, vector unsigned char);
int	vec_any_eq(vector float, vector float);
int	vec_any_eq(vector pixel, vector pixel);
int	vec_any_eq(vector signed short, vector bool short);
int	vec_any_eq(vector signed short, vector signed short);
int	vec_any_eq(vector signed int, vector bool long);
int	vec_any_eq(vector signed int, vector signed int);
int	vec_any_eq(vector signed char, vector bool char);
int	vec_any_eq(vector signed char, vector signed char);
int	vec_any_eq(vector unsigned short, vector bool short);

int	vec_any_eq(vector unsigned short, vector unsigned short);
int	vec_any_eq(vector unsigned int, vector bool long);
int	vec_any_eq(vector unsigned int, vector unsigned int);
int	vec_any_eq(vector unsigned char, vector bool char);
int	vec_any_eq(vector unsigned char, vector unsigned char);
int	vec_any_ge(vector bool short, vector signed short);
int	vec_any_ge(vector bool short, vector unsigned short);
int	vec_any_ge(vector bool long, vector signed int);
int	vec_any_ge(vector bool long, vector unsigned int);
int	vec_any_ge(vector bool char, vector signed char);
int	vec_any_ge(vector bool char, vector unsigned char);
int	vec_any_ge(vector float, vector float);
int	vec_any_ge(vector signed short, vector bool short);
int	vec_any_ge(vector signed short, vector signed short);
int	vec_any_ge(vector signed int, vector bool long);
int	vec_any_ge(vector signed int, vector signed int);
int	vec_any_ge(vector signed char, vector bool char);
int	vec_any_ge(vector signed char, vector signed char);
int	vec_any_ge(vector unsigned short, vector bool short);
int	vec_any_ge(vector unsigned short, vector unsigned short);
int	vec_any_ge(vector unsigned int, vector bool long);
int	vec_any_ge(vector unsigned int, vector unsigned int);
int	vec_any_ge(vector unsigned char, vector bool char);
int	vec_any_ge(vector unsigned char, vector unsigned char);
int	vec_any_gt(vector bool short, vector signed short);
int	vec_any_gt(vector bool short, vector unsigned short);
int	vec_any_gt(vector bool long, vector signed int);
int	vec_any_gt(vector bool long, vector unsigned int);
int	vec_any_gt(vector bool char, vector signed char);
int	vec_any_gt(vector bool char, vector unsigned char);
int	vec_any_gt(vector float, vector float);
int	vec_any_gt(vector signed short, vector bool short);
int	vec_any_gt(vector signed short, vector signed short);
int	vec_any_gt(vector signed int, vector bool long);
int	vec_any_gt(vector signed int, vector signed int);
int	vec_any_gt(vector signed char, vector bool char);
int	vec_any_gt(vector signed char, vector signed char);

int	vec_any_gt(vector unsigned short, vector bool short);
int	vec_any_gt(vector unsigned short, vector unsigned short);
int	vec_any_gt(vector unsigned int, vector bool long);
int	vec_any_gt(vector unsigned int, vector unsigned int);
int	vec_any_gt(vector unsigned char, vector bool char);
int	vec_any_gt(vector unsigned char, vector unsigned char);
int	vec_any_le(vector bool short, vector signed short);
int	vec_any_le(vector bool short, vector unsigned short);
int	vec_any_le(vector bool long, vector signed int);
int	vec_any_le(vector bool long, vector unsigned int);
int	vec_any_le(vector bool char, vector signed char);
int	vec_any_le(vector bool char, vector unsigned char);
int	vec_any_le(vector float, vector float);
int	vec_any_le(vector signed short, vector bool short);
int	vec_any_le(vector signed short, vector signed short);
int	vec_any_le(vector signed int, vector bool long);
int	vec_any_le(vector signed int, vector signed int);
int	vec_any_le(vector signed char, vector bool char);
int	vec_any_le(vector signed char, vector signed char);
int	vec_any_le(vector unsigned short, vector bool short);
int	vec_any_le(vector unsigned short, vector unsigned short);
int	vec_any_le(vector unsigned int, vector bool long);
int	vec_any_le(vector unsigned int, vector unsigned int);
int	vec_any_le(vector unsigned char, vector bool char);
int	vec_any_le(vector unsigned char, vector unsigned char);
int	vec_any_lt(vector bool short, vector signed short);
int	vec_any_lt(vector bool short, vector unsigned short);
int	vec_any_lt(vector bool long, vector signed int);
int	vec_any_lt(vector bool long, vector unsigned int);
int	vec_any_lt(vector bool char, vector signed char);
int	vec_any_lt(vector bool char, vector unsigned char);
int	vec_any_lt(vector float, vector float);
int	vec_any_lt(vector signed short, vector bool short);
int	vec_any_lt(vector signed short, vector signed short);
int	vec_any_lt(vector signed int, vector bool long);
int	vec_any_lt(vector signed int, vector signed int);
int	vec_any_lt(vector signed char, vector bool char);

int	vec_any_lt(vector signed char, vector signed char);
int	vec_any_lt(vector unsigned short, vector bool short);
int	vec_any_lt(vector unsigned short, vector unsigned short);
int	vec_any_lt(vector unsigned int, vector bool long);
int	vec_any_lt(vector unsigned int, vector unsigned int);
int	vec_any_lt(vector unsigned char, vector bool char);
int	vec_any_lt(vector unsigned char, vector unsigned char);
int	vec_any_nan(vector float);
int	vec_any_ne(vector bool short, vector bool short);
int	vec_any_ne(vector bool short, vector signed short);
int	vec_any_ne(vector bool short, vector unsigned short);
int	vec_any_ne(vector bool long, vector bool long);
int	vec_any_ne(vector bool long, vector signed int);
int	vec_any_ne(vector bool long, vector unsigned int);
int	vec_any_ne(vector bool char, vector bool char);
int	vec_any_ne(vector bool char, vector signed char);
int	vec_any_ne(vector bool char, vector unsigned char);
int	vec_any_ne(vector float, vector float);
int	vec_any_ne(vector pixel, vector pixel);
int	vec_any_ne(vector signed short, vector bool short);
int	vec_any_ne(vector signed short, vector signed short);
int	vec_any_ne(vector signed int, vector bool long);
int	vec_any_ne(vector signed int, vector signed int);
int	vec_any_ne(vector signed char, vector bool char);
int	vec_any_ne(vector signed char, vector signed char);
int	vec_any_ne(vector unsigned short, vector bool short);
int	vec_any_ne(vector unsigned short, vector unsigned short);
int	vec_any_ne(vector unsigned int, vector bool long);
int	vec_any_ne(vector unsigned int, vector unsigned int);
int	vec_any_ne(vector unsigned char, vector bool char);
int	vec_any_ne(vector unsigned char, vector unsigned char);
int	vec_any_nge(vector float, vector float);
int	vec_any_ngt(vector float, vector float);
int	vec_any_nle(vector float, vector float);
int	vec_any_nlt(vector float, vector float);
int	vec_any_numeric(vector float);
int	vec_any_out(vector float, vector float);

vector signed char	vec_avg(vector signed char, vector signed char);
vector signed short	vec_avg(vector signed short, vector signed short);
vector signed int	vec_avg(vector signed int, vector signed int);
vector unsigned char	vec_avg(vector unsigned char, vector unsigned char);
vector unsigned short	vec_avg(vector unsigned short, vector unsigned short);
vector unsigned int	vec_avg(vector unsigned int, vector unsigned int);
vector float	vec_ceil(vector float);
vector signed int	vec_cmpb(vector float, vector float);
vector bool long	vec_cmpeq(vector float, vector float);
vector bool char	vec_cmpeq(vector signed char, vector signed char);
vector bool char	vec_cmpeq(vector unsigned char, vector unsigned char);
vector bool short	vec_cmpeq(vector signed short, vector signed short);
vector bool short	vec_cmpeq(vector unsigned short, vector unsigned short);
vector bool long	vec_cmpeq(vector signed int, vector signed int);
vector bool long	vec_cmpeq(vector unsigned int, vector unsigned int);
vector bool long	vec_cmpge(vector float, vector float);
vector bool long	vec_cmpgt(vector float, vector float);
vector bool char	vec_cmpgt(vector signed char, vector signed char);
vector bool short	vec_cmpgt(vector signed short, vector signed short);
vector bool long	vec_cmpgt(vector signed int, vector signed int);
vector bool char	vec_cmpgt(vector unsigned char, vector unsigned char);
vector bool short	vec_cmpgt(vector unsigned short, vector unsigned short);
vector bool long	vec_cmpgt(vector unsigned int, vector unsigned int);
vector bool long	vec_cmple(vector float, vector float);
vector bool long	vec_cmplt(vector float, vector float);
vector bool short	vec_cmplt(vector signed short, vector signed short);
vector bool long	vec_cmplt(vector signed int, vector signed int);
vector bool char	vec_cmplt(vector signed char, vector signed char);
vector bool short	vec_cmplt(vector unsigned short, vector unsigned short);
vector bool long	vec_cmplt(vector unsigned int, vector unsigned int);
vector bool char	vec_cmplt(vector unsigned char, vector unsigned char);
vector float	vec_ctf(vector signed int, int);
vector float	vec_ctf(vector unsigned int, int);
vector signed int	vec_cts(vector float, int);
vector unsigned int	vec_ctu(vector float, int);
void	vec_dss(int);
void	vec_dssall( );

void	vec_dst(float *, int, int);
void	vec_dst(int *, int, int);
void	vec_dst(long *, int, int);
void	vec_dst(short *, int, int);
void	vec_dst(char *, int, int);
void	vec_dst(unsigned char *, int, int);
void	vec_dst(unsigned int *, int, int);
void	vec_dst(unsigned long *, int, int);
void	vec_dst(unsigned short *, int, int);
void	vec_dst(vector bool short *, int, int);
void	vec_dst(vector bool long *, int, int);
void	vec_dst(vector bool char *, int, int);
void	vec_dst(vector float *, int, int);
void	vec_dst(vector pixel *, int, int);
void	vec_dst(vector signed short *, int, int);
void	vec_dst(vector signd long *, int, int);
void	vec_dst(vector signed char *, int, int);
void	vec_dst(vector unsigned short *, int, int);
void	vec_dst(vector unsigned long *, int, int);
void	vec_dst(vector unsigned char *, int, int);
void	vec_dstst(float *, int, int);
void	vec_dstst(int *, int, int);
void	vec_dstst(long *, int, int);
void	vec_dstst(short *, int, int);
void	vec_dstst(char *, int, int);
void	vec_dstst(unsigned char *, int, int);
void	vec_dstst(unsigned int *, int, int);
void	vec_dstst(unsigned long *, int, int);
void	vec_dstst(unsigned short *, int, int);
void	vec_dstst(vector bool short *, int, int);
void	vec_dstst(vector bool long *, int, int);
void	vec_dstst(vector bool char *, int, int);
void	vec_dstst(vector float *, int, int);
void	vec_dstst(vector pixel *, int, int);
void	vec_dstst(vector signed short *, int, int);
void	vec_dstst(vector signd long *, int, int);
void	vec_dstst(vector signed char *, int, int);

void	vec_dstst(vector unsigned short *, int, int);
void	vec_dstst(vector unsigned long *, int, int);
void	vec_dstst(vector unsigned char *, int, int);
void	vec_dststt(float *, int, int);
void	vec_dststt(int *, int, int);
void	vec_dststt(long *, int, int);
void	vec_dststt(short *, int, int);
void	vec_dststt(char *, int, int);
void	vec_dststt(unsigned char *, int, int);
void	vec_dststt(unsigned int *, int, int);
void	vec_dststt(unsigned long *, int, int);
void	vec_dststt(unsigned short *, int, int);
void	vec_dststt(vector bool short *, int, int);
void	vec_dststt(vector bool long *, int, int);
void	vec_dststt(vector bool char *, int, int);
void	vec_dststt(vector float *, int, int);
void	vec_dststt(vector pixel *, int, int);
void	vec_dststt(vector signed short *, int, int);
void	vec_dststt(vector signd long *, int, int);
void	vec_dststt(vector signed char *, int, int);
void	vec_dststt(vector unsigned short *, int, int);
void	vec_dststt(vector unsigned long *, int, int);
void	vec_dststt(vector unsigned char *, int, int);
void	vec_dstt(float *, int, int);
void	vec_dstt(int *, int, int);
void	vec_dstt(long *, int, int);
void	vec_dstt(short *, int, int);
void	vec_dstt(char *, int, int);
void	vec_dstt(unsigned char *, int, int);
void	vec_dstt(unsigned int *, int, int);
void	vec_dstt(unsigned long *, int, int);
void	vec_dstt(unsigned short *, int, int);
void	vec_dstt(vector bool short *, int, int);
void	vec_dstt(vector bool long *, int, int);
void	vec_dstt(vector bool char *, int, int);
void	vec_dstt(vector float *, int, int);
void	vec_dstt(vector pixel *, int, int);

void	vec_dstt(vector signed short *, int, int);
void	vec_dstt(vector signd long *, int, int);
void	vec_dstt(vector signed char *, int, int);
void	vec_dstt(vector unsigned short *, int, int);
void	vec_dstt(vector unsigned long *, int, int);
void	vec_dstt(vector unsigned char *, int, int);
vector float	vec_expte(vector float);
vector float	vec_floor(vector float);
vector float	vec_ld(int, float *);
vector signed int	vec_ld(int, int *);
vector signed int	vec_ld(int, long *);
vector signed short	vec_ld(int, short *);
vector signed char	vec_ld(int, char *);
vector unsigned char	vec_ld(int, unsigned char *);
vector unsigned int	vec_ld(int, unsigned int *);
vector unsigned int	vec_ld(int, unsigned long *);
vector unsigned short	vec_ld(int, unsigned short *);
vector bool short	vec_ld(int, vector bool short *);
vector bool long	vec_ld(int, vector bool long *);
vector bool char	vec_ld(int, vector bool char *);
vector float	vec_ld(int, vector float *);
vector pixel	vec_ld(int, vector pixel *);
vector signed short	vec_ld(int, vector signed short *);
vector signed int	vec_ld(int, vector signd long *);
vector signed char	vec_ld(int, vector signed char *);
vector unsigned short	vec_ld(int, vector unsigned short *);
vector unsigned int	vec_ld(int, vector unsigned long *);
vector unsigned char	vec_ld(int, vector unsigned char *);
vector signed char	vec_lde(int, char *);
vector unsigned char	vec_lde(int, unsigned char *);
vector signed short	vec_lde(int, short *);
vector unsigned short	vec_lde(int, unsigned short *);
vector float	vec_lde(int, float *);
vector signed int	vec_lde(int, int *);
vector signed int	vec_lde(int, long *);
vector unsigned int	vec_lde(int, unsigned int *);
vector unsigned int	vec_lde(int, unsigned long *);

vector float	vec_ldl(int, float *);
vector signed int	vec_ldl(int, int *);
vector signed int	vec_ldl(int, long *);
vector signed short	vec_ldl(int, short *);
vector signed char	vec_ldl(int, char *);
vector unsigned char	vec_ldl(int, unsigned char *);
vector unsigned int	vec_ldl(int, unsigned int *);
vector unsigned int	vec_ldl(int, unsigned long *);
vector unsigned short	vec_ldl(int, unsigned short *);
vector bool short	vec_ldl(int, vector bool short *);
vector bool long	vec_ldl(int, vector bool long *);
vector bool char	vec_ldl(int, vector bool char *);
vector float	vec_ldl(int, vector float *);
vector pixel	vec_ldl(int, vector pixel *);
vector signed short	vec_ldl(int, vector signed short *);
vector signed int	vec_ldl(int, vector signd long *);
vector signed char	vec_ldl(int, vector signed char *);
vector unsigned short	vec_ldl(int, vector unsigned short *);
vector unsigned int	vec_ldl(int, vector unsigned long *);
vector unsigned char	vec_ldl(int, vector unsigned char *);
vector float	vec_loge(vector float);
vector signed char	vec_lvebx(int, char *);
vector unsigned char	vec_lvebx(int, unsigned char *);
vector signed short	vec_lvehx(int, short *);
vector unsigned short	vec_lvehx(int, unsigned short *);
vector float	vec_lvewx(int, float *);
vector signed int	vec_lvewx(int, int *);
vector signed int	vec_lvewx(int, long *);
vector unsigned int	vec_lvewx(int, unsigned int *);
vector unsigned int	vec_lvewx(int, unsigned long *);
vector float	vec_lvlx(int, float *);
vector signed int	vec_lvlx(int, int *);
vector signed int	vec_lvlx(int, long *);
vector signed short	vec_lvlx(int, short *);
vector signed char	vec_lvlx(int, char *);
vector unsigned char	vec_lvlx(int, unsigned char *);
vector unsigned int	vec_lvlx(int, unsigned int *);

vector unsigned int	vec_lvlx(int, unsigned long *);
vector unsigned short	vec_lvlx(int, unsigned short *);
vector bool short	vec_lvlx(int, vector bool short *);
vector bool long	vec_lvlx(int, vector bool long *);
vector bool char	vec_lvlx(int, vector bool char *);
vector float	vec_lvlx(int, vector float *);
vector pixel	vec_lvlx(int, vector pixel *);
vector signed short	vec_lvlx(int, vector signed short *);
vector signed int	vec_lvlx(int, vector signd long *);
vector signed char	vec_lvlx(int, vector signed char *);
vector unsigned short	vec_lvlx(int, vector unsigned short *);
vector unsigned int	vec_lvlx(int, vector unsigned long *);
vector unsigned char	vec_lvlx(int, vector unsigned char *);
vector float	vec_lvlxl(int, float *);
vector signed int	vec_lvlxl(int, int *);
vector signed int	vec_lvlxl(int, long *);
vector signed short	vec_lvlxl(int, short *);
vector signed char	vec_lvlxl(int, char *);
vector unsigned char	vec_lvlxl(int, unsigned char *);
vector unsigned int	vec_lvlxl(int, unsigned int *);
vector unsigned int	vec_lvlxl(int, unsigned long *);
vector unsigned short	vec_lvlxl(int, unsigned short *);
vector bool short	vec_lvlxl(int, vector bool short *);
vector bool long	vec_lvlxl(int, vector bool long *);
vector bool char	vec_lvlxl(int, vector bool char *);
vector float	vec_lvlxl(int, vector float *);
vector pixel	vec_lvlxl(int, vector pixel *);
vector signed short	vec_lvlxl(int, vector signed short *);
vector signed int	vec_lvlxl(int, vector signd long *);
vector signed char	vec_lvlxl(int, vector signed char *);
vector unsigned short	vec_lvlxl(int, vector unsigned short *);
vector unsigned int	vec_lvlxl(int, vector unsigned long *);
vector unsigned char	vec_lvlxl(int, vector unsigned char *);
vector float	vec_lvrx(int, float *);
vector signed int	vec_lvrx(int, int *);
vector signed int	vec_lvrx(int, long *);
vector signed short	vec_lvrx(int, short *);

vector signed char	vec_lvrx(int, char *);
vector unsigned char	vec_lvrx(int, unsigned char *);
vector unsigned int	vec_lvrx(int, unsigned int *);
vector unsigned int	vec_lvrx(int, unsigned long *);
vector unsigned short	vec_lvrx(int, unsigned short *);
vector bool short	vec_lvrx(int, vector bool short *);
vector bool long	vec_lvrx(int, vector bool long *);
vector bool char	vec_lvrx(int, vector bool char *);
vector float	vec_lvrx(int, vector float *);
vector pixel	vec_lvrx(int, vector pixel *);
vector signed short	vec_lvrx(int, vector signed short *);
vector signed int	vec_lvrx(int, vector signd long *);
vector signed char	vec_lvrx(int, vector signed char *);
vector unsigned short	vec_lvrx(int, vector unsigned short *);
vector unsigned int	vec_lvrx(int, vector unsigned long *);
vector unsigned char	vec_lvrx(int, vector unsigned char *);
vector float	vec_lvrxl(int, float *);
vector signed int	vec_lvrxl(int, int *);
vector signed int	vec_lvrxl(int, long *);
vector signed short	vec_lvrxl(int, short *);
vector signed char	vec_lvrxl(int, char *);
vector unsigned char	vec_lvrxl(int, unsigned char *);
vector unsigned int	vec_lvrxl(int, unsigned int *);
vector unsigned int	vec_lvrxl(int, unsigned long *);
vector unsigned short	vec_lvrxl(int, unsigned short *);
vector bool short	vec_lvrxl(int, vector bool short *);
vector bool long	vec_lvrxl(int, vector bool long *);
vector bool char	vec_lvrxl(int, vector bool char *);
vector float	vec_lvrxl(int, vector float *);
vector pixel	vec_lvrxl(int, vector pixel *);
vector signed short	vec_lvrxl(int, vector signed short *);
vector signed int	vec_lvrxl(int, vector signd long *);
vector signed char	vec_lvrxl(int, vector signed char *);
vector unsigned short	vec_lvrxl(int, vector unsigned short *);
vector unsigned int	vec_lvrxl(int, vector unsigned long *);
vector unsigned char	vec_lvrxl(int, vector unsigned char *);
vector unsigned char	vec_lvsl(int, float *);

vector unsigned char	vec_lvsl(int, int *);
vector unsigned char	vec_lvsl(int, long *);
vector unsigned char	vec_lvsl(int, short *);
vector unsigned char	vec_lvsl(int, char *);
vector unsigned char	vec_lvsl(int, unsigned char *);
vector unsigned char	vec_lvsl(int, unsigned int *);
vector unsigned char	vec_lvsl(int, unsigned long *);
vector unsigned char	vec_lvsl(int, unsigned short *);
vector unsigned char	vec_lvsr(int, float *);
vector unsigned char	vec_lvsr(int, int *);
vector unsigned char	vec_lvsr(int, long *);
vector unsigned char	vec_lvsr(int, short *);
vector unsigned char	vec_lvsr(int, char *);
vector unsigned char	vec_lvsr(int, unsigned char *);
vector unsigned char	vec_lvsr(int, unsigned int *);
vector unsigned char	vec_lvsr(int, unsigned long *);
vector unsigned char	vec_lvsr(int, unsigned short *);
vector float	vec_lvx(int, float *);
vector signed int	vec_lvx(int, int *);
vector signed int	vec_lvx(int, long *);
vector signed short	vec_lvx(int, short *);
vector signed char	vec_lvx(int, char *);
vector unsigned char	vec_lvx(int, unsigned char *);
vector unsigned int	vec_lvx(int, unsigned int *);
vector unsigned int	vec_lvx(int, unsigned long *);
vector unsigned short	vec_lvx(int, unsigned short *);
vector bool short	vec_lvx(int, vector bool short *);
vector bool long	vec_lvx(int, vector bool long *);
vector bool char	vec_lvx(int, vector bool char *);
vector float	vec_lvx(int, vector float *);
vector pixel	vec_lvx(int, vector pixel *);
vector signed short	vec_lvx(int, vector signed short *);
vector signed int	vec_lvx(int, vector signd long *);
vector signed char	vec_lvx(int, vector signed char *);
vector unsigned short	vec_lvx(int, vector unsigned short *);
vector unsigned int	vec_lvx(int, vector unsigned long *);
vector unsigned char	vec_lvx(int, vector unsigned char *);

vector float	vec_lvxl(int, float *);
vector signed int	vec_lvxl(int, int *);
vector signed int	vec_lvxl(int, long *);
vector signed short	vec_lvxl(int, short *);
vector signed char	vec_lvxl(int, char *);
vector unsigned char	vec_lvxl(int, unsigned char *);
vector unsigned int	vec_lvxl(int, unsigned int *);
vector unsigned int	vec_lvxl(int, unsigned long *);
vector unsigned short	vec_lvxl(int, unsigned short *);
vector bool short	vec_lvxl(int, vector bool short *);
vector bool long	vec_lvxl(int, vector bool long *);
vector bool char	vec_lvxl(int, vector bool char *);
vector float	vec_lvxl(int, vector float *);
vector pixel	vec_lvxl(int, vector pixel *);
vector signed short	vec_lvxl(int, vector signed short *);
vector signed int	vec_lvxl(int, vector signd long *);
vector signed char	vec_lvxl(int, vector signed char *);
vector unsigned short	vec_lvxl(int, vector unsigned short *);
vector unsigned int	vec_lvxl(int, vector unsigned long *);
vector unsigned char	vec_lvxl(int, vector unsigned char *);
vector float	vec_madd(vector float, vector float, vector float);
vector signed short	vec_madds(vector signed short, vector signed short, vector signed short);
vector float	vec_max(vector float, vector float);
vector signed char	vec_max(vector bool char, vector signed char);
vector signed char	vec_max(vector signed char, vector bool char);
vector signed char	vec_max(vector signed char, vector signed char);
vector signed short	vec_max(vector bool short, vector signed short);
vector signed short	vec_max(vector signed short, vector bool short);
vector signed short	vec_max(vector signed short, vector signed short);
vector signed int	vec_max(vector bool long, vector signed int);
vector signed int	vec_max(vector signed int, vector bool long);
vector signed int	vec_max(vector signed int, vector signed int);
vector unsigned char	vec_max(vector bool char, vector unsigned char);
vector unsigned char	vec_max(vector unsigned char, vector bool char);
vector unsigned char	vec_max(vector unsigned char, vector unsigned char);
vector unsigned short	vec_max(vector bool short, vector unsigned short);
vector unsigned short	vec_max(vector unsigned short, vector bool short);

vector unsigned short	vec_max(vector unsigned short, vector unsigned short);
vector unsigned int	vec_max(vector bool long, vector unsigned int);
vector unsigned int	vec_max(vector unsigned int, vector bool long);
vector unsigned int	vec_max(vector unsigned int, vector unsigned int);
vector bool char	vec_mergeh(vector bool char, vector bool char);
vector signed char	vec_mergeh(vector signed char, vector signed char);
vector unsigned char	vec_mergeh(vector unsigned char, vector unsigned char);
vector bool short	vec_mergeh(vector bool short, vector bool short);
vector pixel	vec_mergeh(vector pixel, vector pixel);
vector signed short	vec_mergeh(vector signed short, vector signed short);
vector unsigned short	vec_mergeh(vector unsigned short, vector unsigned short);
vector bool long	vec_mergeh(vector bool long, vector bool long);
vector float	vec_mergeh(vector float, vector float);
vector signed int	vec_mergeh(vector signed int, vector signed int);
vector unsigned int	vec_mergeh(vector unsigned int, vector unsigned int);
vector bool char	vec_mergel(vector bool char, vector bool char);
vector signed char	vec_mergel(vector signed char, vector signed char);
vector unsigned char	vec_mergel(vector unsigned char, vector unsigned char);
vector bool short	vec_mergel(vector bool short, vector bool short);
vector pixel	vec_mergel(vector pixel, vector pixel);
vector signed short	vec_mergel(vector signed short, vector signed short);
vector unsigned short	vec_mergel(vector unsigned short, vector unsigned short);
vector bool long	vec_mergel(vector bool long, vector bool long);
vector float	vec_mergel(vector float, vector float);
vector signed int	vec_mergel(vector signed int, vector signed int);
vector unsigned int	vec_mergel(vector unsigned int, vector unsigned int);
volatile vector unsigned short	vec_mfvscr( );
vector float	vec_min(vector float, vector float);
vector signed char	vec_min(vector bool char, vector signed char);
vector signed char	vec_min(vector signed char, vector bool char);
vector signed char	vec_min(vector signed char, vector signed char);
vector signed short	vec_min(vector bool short, vector signed short);
vector signed short	vec_min(vector signed short, vector bool short);
vector signed short	vec_min(vector signed short, vector signed short);
vector signed int	vec_min(vector bool long, vector signed int);
vector signed int	vec_min(vector signed int, vector bool long);
vector signed int	vec_min(vector signed int, vector signed int);

vector unsigned char	vec_min(vector bool char, vector unsigned char);
vector unsigned char	vec_min(vector unsigned char, vector bool char);
vector unsigned char	vec_min(vector unsigned char, vector unsigned char);
vector unsigned short	vec_min(vector bool short, vector unsigned short);
vector unsigned short	vec_min(vector unsigned short, vector bool short);
vector unsigned short	vec_min(vector unsigned short, vector unsigned short);
vector unsigned int	vec_min(vector bool long, vector unsigned int);
vector unsigned int	vec_min(vector unsigned int, vector bool long);
vector unsigned int	vec_min(vector unsigned int, vector unsigned int);
vector signed short	vec_mladd(vector signed short, vector signed short, vector signed short);
vector signed short	vec_mladd(vector signed short, vector unsigned short, vector unsigned short);
vector signed short	vec_mladd(vector unsigned short, vector signed short, vector signed short);
vector unsigned short	vec_mladd(vector unsigned short, vector unsigned short, vector unsigned short);
vector signed short	vec_mradds(vector signed short, vector signed short, vector signed short);
vector signed int	vec_msum(vector signed char, vector unsigned char, vector signed int);
vector signed int	vec_msum(vector signed short, vector signed short, vector signed int);
vector unsigned int	vec_msum(vector unsigned char, vector unsigned char, vector unsigned int);
vector unsigned int	vec_msum(vector unsigned short, vector unsigned short, vector unsigned int);
vector signed int	vec_msums(vector signed short, vector signed short, vector signed int);
vector unsigned int	vec_msums(vector unsigned short, vector unsigned short, vector unsigned int);
void	vec_mtvscr(vector bool short);
void	vec_mtvscr(vector bool long);
void	vec_mtvscr(vector bool char);
void	vec_mtvscr(vector pixel);
void	vec_mtvscr(vector signed short);
void	vec_mtvscr(vector signed int);
void	vec_mtvscr(vector signed char);
void	vec_mtvscr(vector unsigned short);
void	vec_mtvscr(vector unsigned int);
void	vec_mtvscr(vector unsigned char);
vector signed short	vec_mule(vector signed char, vector signed char);
vector signed int	vec_mule(vector signed short, vector signed short);
vector unsigned short	vec_mule(vector unsigned char, vector unsigned char);
vector unsigned int	vec_mule(vector unsigned short, vector unsigned short);

vector signed short	vec_mulo(vector signed char, vector signed char);
vector signed int	vec_mulo(vector signed short, vector signed short);
vector unsigned short	vec_mulo(vector unsigned char, vector unsigned char);
vector unsigned int	vec_mulo(vector unsigned short, vector unsigned short);
vector float	vec_nmsub(vector float, vector float, vector float);
vector bool short	vec_nor(vector bool short, vector bool short);
vector bool long	vec_nor(vector bool long, vector bool long);
vector bool char	vec_nor(vector bool char, vector bool char);
vector float	vec_nor(vector float, vector float);
vector signed short	vec_nor(vector signed short, vector signed short);
vector signed int	vec_nor(vector signed int, vector signed int);
vector signed char	vec_nor(vector signed char, vector signed char);
vector unsigned short	vec_nor(vector unsigned short, vector unsigned short);
vector unsigned int	vec_nor(vector unsigned int, vector unsigned int);
vector unsigned char	vec_nor(vector unsigned char, vector unsigned char);
vector bool short	vec_or(vector bool short, vector bool short);
vector signed short	vec_or(vector bool short, vector signed short);
vector unsigned short	vec_or(vector bool short, vector unsigned short);
vector bool long	vec_or(vector bool long, vector bool long);
vector float	vec_or(vector bool long, vector float);
vector signed int	vec_or(vector bool long, vector signed int);
vector unsigned int	vec_or(vector bool long, vector unsigned int);
vector bool char	vec_or(vector bool char, vector bool char);
vector signed char	vec_or(vector bool char, vector signed char);
vector unsigned char	vec_or(vector bool char, vector unsigned char);
vector float	vec_or(vector float, vector bool long);
vector float	vec_or(vector float, vector float);
vector signed short	vec_or(vector signed short, vector bool short);
vector signed short	vec_or(vector signed short, vector signed short);
vector signed int	vec_or(vector signed int, vector bool long);
vector signed int	vec_or(vector signed int, vector signed int);
vector signed char	vec_or(vector signed char, vector bool char);
vector signed char	vec_or(vector signed char, vector signed char);
vector unsigned short	vec_or(vector unsigned short, vector bool short);
vector unsigned short	vec_or(vector unsigned short, vector unsigned short);
vector unsigned int	vec_or(vector unsigned int, vector bool long);
vector unsigned int	vec_or(vector unsigned int, vector unsigned int);

vector unsigned char	vec_or(vector unsigned char, vector bool char);
vector unsigned char	vec_or(vector unsigned char, vector unsigned char);
vector bool char	vec_pack(vector bool short, vector bool short);
vector signed char	vec_pack(vector signed short, vector signed short);
vector unsigned char	vec_pack(vector unsigned short, vector unsigned short);
vector bool short	vec_pack(vector bool long, vector bool long);
vector signed short	vec_pack(vector signed int, vector signed int);
vector unsigned short	vec_pack(vector unsigned int, vector unsigned int);
vector pixel	vec_packpx(vector unsigned int, vector unsigned int);
vector signed char	vec_packs(vector signed short, vector signed short);
vector signed short	vec_packs(vector signed int, vector signed int);
vector unsigned char	vec_packs(vector unsigned short, vector unsigned short);
vector unsigned short	vec_packs(vector unsigned int, vector unsigned int);
vector unsigned char	vec_packsu(vector signed short, vector signed short);
vector unsigned short	vec_packsu(vector signed int, vector signed int);
vector unsigned char	vec_packsu(vector unsigned short, vector unsigned short);
vector unsigned short	vec_packsu(vector unsigned int, vector unsigned int);
vector bool short	vec_perm(vector bool short, vector bool short, vector unsigned char);
vector bool long	vec_perm(vector bool long, vector bool long, vector unsigned char);
vector bool char	vec_perm(vector bool char, vector bool char, vector unsigned char);
vector float	vec_perm(vector float, vector float, vector unsigned char);
vector pixel	vec_perm(vector pixel, vector pixel, vector unsigned char);
vector signed short	vec_perm(vector signed short, vector signed short, vector unsigned char);
vector signed int	vec_perm(vector signed int, vector signed int, vector unsigned char);
vector signed char	vec_perm(vector signed char, vector signed char, vector unsigned char);
vector unsigned short	vec_perm(vector unsigned short, vector unsigned short, vector unsigned char);
vector unsigned int	vec_perm(vector unsigned int, vector unsigned int, vector unsigned char);
vector unsigned char	vec_perm(vector unsigned char, vector unsigned char, vector unsigned char);
vector float	vec_re(vector float);
vector signed char	vec_rl(vector signed char, vector unsigned char);
vector unsigned char	vec_rl(vector unsigned char, vector unsigned char);
vector signed short	vec_rl(vector signed short, vector unsigned short);
vector unsigned short	vec_rl(vector unsigned short, vector unsigned short);
vector signed int	vec_rl(vector signed int, vector unsigned int);
vector unsigned int	vec_rl(vector unsigned int, vector unsigned int);
vector float	vec_round(vector float);

vector float	vec_rsqrte(vector float);
vector bool short	vec_sel(vector bool short, vector bool short, vector bool short);
vector bool short	vec_sel(vector bool short, vector bool short, vector unsigned short);
vector bool long	vec_sel(vector bool long, vector bool long, vector bool long);
vector bool long	vec_sel(vector bool long, vector bool long, vector unsigned int);
vector bool char	vec_sel(vector bool char, vector bool char, vector bool char);
vector bool char	vec_sel(vector bool char, vector bool char, vector unsigned char);
vector float	vec_sel(vector float, vector float, vector bool long);
vector float	vec_sel(vector float, vector float, vector unsigned int);
vector signed short	vec_sel(vector signed short, vector signed short, vector bool short);
vector signed short	vec_sel(vector signed short, vector signed short, vector unsigned short);
vector signed int	vec_sel(vector signed int, vector signed int, vector bool long);
vector signed int	vec_sel(vector signed int, vector signed int, vector unsigned int);
vector signed char	vec_sel(vector signed char, vector signed char, vector bool char);
vector signed char	vec_sel(vector signed char, vector signed char, vector unsigned char);
vector unsigned short	vec_sel(vector unsigned short, vector unsigned short, vector bool short);
vector unsigned short	vec_sel(vector unsigned short, vector unsigned short, vector unsigned short);
vector unsigned int	vec_sel(vector unsigned int, vector unsigned int, vector bool long);
vector unsigned int	vec_sel(vector unsigned int, vector unsigned int, vector unsigned int);
vector unsigned char	vec_sel(vector unsigned char, vector unsigned char, vector bool char);
vector unsigned char	vec_sel(vector unsigned char, vector unsigned char, vector unsigned char);
vector signed char	vec_sl(vector signed char, vector unsigned char);
vector unsigned char	vec_sl(vector unsigned char, vector unsigned char);
vector signed short	vec_sl(vector signed short, vector unsigned short);
vector unsigned short	vec_sl(vector unsigned short, vector unsigned short);
vector signed int	vec_sl(vector signed int, vector unsigned int);
vector unsigned int	vec_sl(vector unsigned int, vector unsigned int);
vector float	vec_sld(vector float, vector float, int);
vector pixel	vec_sld(vector pixel, vector pixel, int);
vector signed short	vec_sld(vector signed short, vector signed short, int);
vector signed int	vec_sld(vector signed int, vector signed int, int);
vector signed char	vec_sld(vector signed char, vector signed char, int);
vector unsigned short	vec_sld(vector unsigned short, vector unsigned short, int);
vector unsigned int	vec_sld(vector unsigned int, vector unsigned int, int);
vector unsigned char	vec_sld(vector unsigned char, vector unsigned char, int);
vector bool short	vec_sll(vector bool short, vector unsigned short);

vector bool short	vec_sll(vector bool short, vector unsigned int);
vector bool short	vec_sll(vector bool short, vector unsigned char);
vector bool long	vec_sll(vector bool long, vector unsigned short);
vector bool long	vec_sll(vector bool long, vector unsigned int);
vector bool long	vec_sll(vector bool long, vector unsigned char);
vector bool char	vec_sll(vector bool char, vector unsigned short);
vector bool char	vec_sll(vector bool char, vector unsigned int);
vector bool char	vec_sll(vector bool char, vector unsigned char);
vector pixel	vec_sll(vector pixel, vector unsigned short);
vector pixel	vec_sll(vector pixel, vector unsigned int);
vector pixel	vec_sll(vector pixel, vector unsigned char);
vector signed short	vec_sll(vector signed short, vector unsigned short);
vector signed short	vec_sll(vector signed short, vector unsigned int);
vector signed short	vec_sll(vector signed short, vector unsigned char);
vector signed int	vec_sll(vector signed int, vector unsigned short);
vector signed int	vec_sll(vector signed int, vector unsigned int);
vector signed int	vec_sll(vector signed int, vector unsigned char);
vector signed char	vec_sll(vector signed char, vector unsigned short);
vector signed char	vec_sll(vector signed char, vector unsigned int);
vector signed char	vec_sll(vector signed char, vector unsigned char);
vector unsigned short	vec_sll(vector unsigned short, vector unsigned short);
vector unsigned short	vec_sll(vector unsigned short, vector unsigned int);
vector unsigned short	vec_sll(vector unsigned short, vector unsigned char);
vector unsigned int	vec_sll(vector unsigned int, vector unsigned short);
vector unsigned int	vec_sll(vector unsigned int, vector unsigned int);
vector unsigned int	vec_sll(vector unsigned int, vector unsigned char);
vector unsigned char	vec_sll(vector unsigned char, vector unsigned short);
vector unsigned char	vec_sll(vector unsigned char, vector unsigned int);
vector unsigned char	vec_sll(vector unsigned char, vector unsigned char);
vector float	vec_slo(vector float, vector signed char);
vector float	vec_slo(vector float, vector unsigned char);
vector pixel	vec_slo(vector pixel, vector signed char);
vector pixel	vec_slo(vector pixel, vector unsigned char);
vector signed short	vec_slo(vector signed short, vector signed char);
vector signed short	vec_slo(vector signed short, vector unsigned char);
vector signed int	vec_slo(vector signed int, vector signed char);
vector signed int	vec_slo(vector signed int, vector unsigned char);

vector signed char	vec_slo(vector signed char, vector signed char);
vector signed char	vec_slo(vector signed char, vector unsigned char);
vector unsigned short	vec_slo(vector unsigned short, vector signed char);
vector unsigned short	vec_slo(vector unsigned short, vector unsigned char);
vector unsigned int	vec_slo(vector unsigned int, vector signed char);
vector unsigned int	vec_slo(vector unsigned int, vector unsigned char);
vector unsigned char	vec_slo(vector unsigned char, vector signed char);
vector unsigned char	vec_slo(vector unsigned char, vector unsigned char);
vector bool char	vec_splat(vector bool char, int);
vector signed char	vec_splat(vector signed char, int);
vector unsigned char	vec_splat(vector unsigned char, int);
vector bool short	vec_splat(vector bool short, int);
vector pixel	vec_splat(vector pixel, int);
vector signed short	vec_splat(vector signed short, int);
vector unsigned short	vec_splat(vector unsigned short, int);
vector bool long	vec_splat(vector bool long, int);
vector float	vec_splat(vector float, int);
vector signed int	vec_splat(vector signed int, int);
vector unsigned int	vec_splat(vector unsigned int, int);
vector signed short	vec_splat_s16(int);
vector signed int	vec_splat_s32(int);
vector signed char	vec_splat_s8(int);
vector unsigned short	vec_splat_u16(int);
vector unsigned int	vec_splat_u32(int);
vector unsigned char	vec_splat_u8(int);
vector signed char	vec_sr(vector signed char, vector unsigned char);
vector unsigned char	vec_sr(vector unsigned char, vector unsigned char);
vector signed short	vec_sr(vector signed short, vector unsigned short);
vector unsigned short	vec_sr(vector unsigned short, vector unsigned short);
vector signed int	vec_sr(vector signed int, vector unsigned int);
vector unsigned int	vec_sr(vector unsigned int, vector unsigned int);
vector signed char	vec_sra(vector signed char, vector unsigned char);
vector unsigned char	vec_sra(vector unsigned char, vector unsigned char);
vector signed short	vec_sra(vector signed short, vector unsigned short);
vector unsigned short	vec_sra(vector unsigned short, vector unsigned short);
vector signed int	vec_sra(vector signed int, vector unsigned int);
vector unsigned int	vec_sra(vector unsigned int, vector unsigned int);

vector bool short	vec_srl(vector bool short, vector unsigned short);
vector bool short	vec_srl(vector bool short, vector unsigned int);
vector bool short	vec_srl(vector bool short, vector unsigned char);
vector bool long	vec_srl(vector bool long, vector unsigned short);
vector bool long	vec_srl(vector bool long, vector unsigned int);
vector bool long	vec_srl(vector bool long, vector unsigned char);
vector bool char	vec_srl(vector bool char, vector unsigned short);
vector bool char	vec_srl(vector bool char, vector unsigned int);
vector bool char	vec_srl(vector bool char, vector unsigned char);
vector pixel	vec_srl(vector pixel, vector unsigned short);
vector pixel	vec_srl(vector pixel, vector unsigned int);
vector pixel	vec_srl(vector pixel, vector unsigned char);
vector signed short	vec_srl(vector signed short, vector unsigned short);
vector signed short	vec_srl(vector signed short, vector unsigned int);
vector signed short	vec_srl(vector signed short, vector unsigned char);
vector signed int	vec_srl(vector signed int, vector unsigned short);
vector signed int	vec_srl(vector signed int, vector unsigned int);
vector signed int	vec_srl(vector signed int, vector unsigned char);
vector signed char	vec_srl(vector signed char, vector unsigned short);
vector signed char	vec_srl(vector signed char, vector unsigned int);
vector signed char	vec_srl(vector signed char, vector unsigned char);
vector unsigned short	vec_srl(vector unsigned short, vector unsigned short);
vector unsigned short	vec_srl(vector unsigned short, vector unsigned int);
vector unsigned short	vec_srl(vector unsigned short, vector unsigned char);
vector unsigned int	vec_srl(vector unsigned int, vector unsigned short);
vector unsigned int	vec_srl(vector unsigned int, vector unsigned int);
vector unsigned int	vec_srl(vector unsigned int, vector unsigned char);
vector unsigned char	vec_srl(vector unsigned char, vector unsigned short);
vector unsigned char	vec_srl(vector unsigned char, vector unsigned int);
vector unsigned char	vec_srl(vector unsigned char, vector unsigned char);
vector float	vec_sro(vector float, vector signed char);
vector float	vec_sro(vector float, vector unsigned char);
vector pixel	vec_sro(vector pixel, vector signed char);
vector pixel	vec_sro(vector pixel, vector unsigned char);
vector signed short	vec_sro(vector signed short, vector signed char);
vector signed short	vec_sro(vector signed short, vector unsigned char);
vector signed int	vec_sro(vector signed int, vector signed char);

vector signed int	vec_sro(vector signed int, vector unsigned char);
vector signed char	vec_sro(vector signed char, vector signed char);
vector signed char	vec_sro(vector signed char, vector unsigned char);
vector unsigned short	vec_sro(vector unsigned short, vector signed char);
vector unsigned short	vec_sro(vector unsigned short, vector unsigned char);
vector unsigned int	vec_sro(vector unsigned int, vector signed char);
vector unsigned int	vec_sro(vector unsigned int, vector unsigned char);
vector unsigned char	vec_sro(vector unsigned char, vector signed char);
vector unsigned char	vec_sro(vector unsigned char, vector unsigned char);
void	vec_st(vector bool short, int, vector bool short *);
void	vec_st(vector bool long, int, vector bool long *);
void	vec_st(vector bool char, int, vector bool char *);
void	vec_st(vector float, int, float *);
void	vec_st(vector float, int, vector float *);
void	vec_st(vector pixel, int, vector pixel *);
void	vec_st(vector signed short, int, short *);
void	vec_st(vector signed short, int, vector signed short *);
void	vec_st(vector signed int, int, int *);
void	vec_st(vector signed int, int, long *);
void	vec_st(vector signed int, int, vector signd long *);
void	vec_st(vector signed char, int, char *);
void	vec_st(vector signed char, int, vector signed char *);
void	vec_st(vector unsigned short, int, unsigned short *);
void	vec_st(vector unsigned short, int, vector unsigned short *);
void	vec_st(vector unsigned int, int, unsigned int *);
void	vec_st(vector unsigned int, int, unsigned long *);
void	vec_st(vector unsigned int, int, vector unsigned long *);
void	vec_st(vector unsigned char, int, unsigned char *);
void	vec_st(vector unsigned char, int, vector unsigned char *);
void	vec_ste(vector signed char, int, char *);
void	vec_ste(vector unsigned char, int, unsigned char *);
void	vec_ste(vector signed short, int, short *);
void	vec_ste(vector unsigned short, int, unsigned short *);
void	vec_ste(vector float, int, float *);
void	vec_ste(vector signed int, int, int *);
void	vec_ste(vector signed int, int, long *);
void	vec_ste(vector unsigned int, int, unsigned int *);

void	vec_ste(vector unsigned int, int, unsigned long *);
void	vec_stl(vector bool short, int, vector bool short *);
void	vec_stl(vector bool long, int, vector bool long *);
void	vec_stl(vector bool char, int, vector bool char *);
void	vec_stl(vector float, int, float *);
void	vec_stl(vector float, int, vector float *);
void	vec_stl(vector pixel, int, vector pixel *);
void	vec_stl(vector signed short, int, short *);
void	vec_stl(vector signed short, int, vector signed short *);
void	vec_stl(vector signed int, int, int *);
void	vec_stl(vector signed int, int, long *);
void	vec_stl(vector signed int, int, vector signd long *);
void	vec_stl(vector signed char, int, char *);
void	vec_stl(vector signed char, int, vector signed char *);
void	vec_stl(vector unsigned short, int, unsigned short *);
void	vec_stl(vector unsigned short, int, vector unsigned short *);
void	vec_stl(vector unsigned int, int, unsigned int *);
void	vec_stl(vector unsigned int, int, unsigned long *);
void	vec_stl(vector unsigned int, int, vector unsigned long *);
void	vec_stl(vector unsigned char, int, unsigned char *);
void	vec_stl(vector unsigned char, int, vector unsigned char *);
void	vec_stvebx(vector signed char, int, char *);
void	vec_stvebx(vector unsigned char, int, unsigned char *);
void	vec_stvehx(vector signed short, int, short *);
void	vec_stvehx(vector unsigned short, int, unsigned short *);
void	vec_stvewx(vector float, int, float *);
void	vec_stvewx(vector signed int, int, int *);
void	vec_stvewx(vector signed int, int, long *);
void	vec_stvewx(vector unsigned int, int, unsigned int *);
void	vec_stvewx(vector unsigned int, int, unsigned long *);
void	vec_stvlx(vector bool short, int, vector bool short *);
void	vec_stvlx(vector bool long, int, vector bool long *);
void	vec_stvlx(vector bool char, int, vector bool char *);
void	vec_stvlx(vector float, int, float *);
void	vec_stvlx(vector float, int, vector float *);
void	vec_stvlx(vector pixel, int, vector pixel *);
void	vec_stvlx(vector signed short, int, short *);

void	vec_stvlx(vector signed short, int, vector signed short *);
void	vec_stvlx(vector signed int, int, int *);
void	vec_stvlx(vector signed int, int, long *);
void	vec_stvlx(vector signed int, int, vector signd long *);
void	vec_stvlx(vector signed char, int, char *);
void	vec_stvlx(vector signed char, int, vector signed char *);
void	vec_stvlx(vector unsigned short, int, unsigned short *);
void	vec_stvlx(vector unsigned short, int, vector unsigned short *);
void	vec_stvlx(vector unsigned int, int, unsigned int *);
void	vec_stvlx(vector unsigned int, int, unsigned long *);
void	vec_stvlx(vector unsigned int, int, vector unsigned long *);
void	vec_stvlx(vector unsigned char, int, unsigned char *);
void	vec_stvlx(vector unsigned char, int, vector unsigned char *);
void	vec_stvlxl(vector bool short, int, vector bool short *);
void	vec_stvlxl(vector bool long, int, vector bool long *);
void	vec_stvlxl(vector bool char, int, vector bool char *);
void	vec_stvlxl(vector float, int, float *);
void	vec_stvlxl(vector float, int, vector float *);
void	vec_stvlxl(vector pixel, int, vector pixel *);
void	vec_stvlxl(vector signed short, int, short *);
void	vec_stvlxl(vector signed short, int, vector signed short *);
void	vec_stvlxl(vector signed int, int, int *);
void	vec_stvlxl(vector signed int, int, long *);
void	vec_stvlxl(vector signed int, int, vector signd long *);
void	vec_stvlxl(vector signed char, int, char *);
void	vec_stvlxl(vector signed char, int, vector signed char *);
void	vec_stvlxl(vector unsigned short, int, unsigned short *);
void	vec_stvlxl(vector unsigned short, int, vector unsigned short *);
void	vec_stvlxl(vector unsigned int, int, unsigned int *);
void	vec_stvlxl(vector unsigned int, int, unsigned long *);
void	vec_stvlxl(vector unsigned int, int, vector unsigned long *);
void	vec_stvlxl(vector unsigned char, int, unsigned char *);
void	vec_stvlxl(vector unsigned char, int, vector unsigned char *);
void	vec_stvrx(vector bool short, int, vector bool short *);
void	vec_stvrx(vector bool long, int, vector bool long *);
void	vec_stvrx(vector bool char, int, vector bool char *);
void	vec_stvrx(vector float, int, float *);

void	vec_stvrx(vector float, int, vector float *);
void	vec_stvrx(vector pixel, int, vector pixel *);
void	vec_stvrx(vector signed short, int, short *);
void	vec_stvrx(vector signed short, int, vector signed short *);
void	vec_stvrx(vector signed int, int, int *);
void	vec_stvrx(vector signed int, int, long *);
void	vec_stvrx(vector signed int, int, vector signd long *);
void	vec_stvrx(vector signed char, int, char *);
void	vec_stvrx(vector signed char, int, vector signed char *);
void	vec_stvrx(vector unsigned short, int, unsigned short *);
void	vec_stvrx(vector unsigned short, int, vector unsigned short *);
void	vec_stvrx(vector unsigned int, int, unsigned int *);
void	vec_stvrx(vector unsigned int, int, unsigned long *);
void	vec_stvrx(vector unsigned int, int, vector unsigned long *);
void	vec_stvrx(vector unsigned char, int, unsigned char *);
void	vec_stvrx(vector unsigned char, int, vector unsigned char *);
void	vec_stvrxl(vector bool short, int, vector bool short *);
void	vec_stvrxl(vector bool long, int, vector bool long *);
void	vec_stvrxl(vector bool char, int, vector bool char *);
void	vec_stvrxl(vector float, int, float *);
void	vec_stvrxl(vector float, int, vector float *);
void	vec_stvrxl(vector pixel, int, vector pixel *);
void	vec_stvrxl(vector signed short, int, short *);
void	vec_stvrxl(vector signed short, int, vector signed short *);
void	vec_stvrxl(vector signed int, int, int *);
void	vec_stvrxl(vector signed int, int, long *);
void	vec_stvrxl(vector signed int, int, vector signd long *);
void	vec_stvrxl(vector signed char, int, char *);
void	vec_stvrxl(vector signed char, int, vector signed char *);
void	vec_stvrxl(vector unsigned short, int, unsigned short *);
void	vec_stvrxl(vector unsigned short, int, vector unsigned short *);
void	vec_stvrxl(vector unsigned int, int, unsigned int *);
void	vec_stvrxl(vector unsigned int, int, unsigned long *);
void	vec_stvrxl(vector unsigned int, int, vector unsigned long *);
void	vec_stvrxl(vector unsigned char, int, unsigned char *);
void	vec_stvrxl(vector unsigned char, int, vector unsigned char *);
void	vec_stvx(vector bool short, int, vector bool short *);

void	vec_stvx(vector bool long, int, vector bool long *);
void	vec_stvx(vector bool char, int, vector bool char *);
void	vec_stvx(vector float, int, float *);
void	vec_stvx(vector float, int, vector float *);
void	vec_stvx(vector pixel, int, vector pixel *);
void	vec_stvx(vector signed short, int, short *);
void	vec_stvx(vector signed short, int, vector signed short *);
void	vec_stvx(vector signed int, int, int *);
void	vec_stvx(vector signed int, int, long *);
void	vec_stvx(vector signed int, int, vector signd long *);
void	vec_stvx(vector signed char, int, char *);
void	vec_stvx(vector signed char, int, vector signed char *);
void	vec_stvx(vector unsigned short, int, unsigned short *);
void	vec_stvx(vector unsigned short, int, vector unsigned short *);
void	vec_stvx(vector unsigned int, int, unsigned int *);
void	vec_stvx(vector unsigned int, int, unsigned long *);
void	vec_stvx(vector unsigned int, int, vector unsigned long *);
void	vec_stvx(vector unsigned char, int, unsigned char *);
void	vec_stvx(vector unsigned char, int, vector unsigned char *);
void	vec_stvxl(vector bool short, int, vector bool short *);
void	vec_stvxl(vector bool long, int, vector bool long *);
void	vec_stvxl(vector bool char, int, vector bool char *);
void	vec_stvxl(vector float, int, float *);
void	vec_stvxl(vector float, int, vector float *);
void	vec_stvxl(vector pixel, int, vector pixel *);
void	vec_stvxl(vector signed short, int, short *);
void	vec_stvxl(vector signed short, int, vector signed short *);
void	vec_stvxl(vector signed int, int, int *);
void	vec_stvxl(vector signed int, int, long *);
void	vec_stvxl(vector signed int, int, vector signd long *);
void	vec_stvxl(vector signed char, int, char *);
void	vec_stvxl(vector signed char, int, vector signed char *);
void	vec_stvxl(vector unsigned short, int, unsigned short *);
void	vec_stvxl(vector unsigned short, int, vector unsigned short *);
void	vec_stvxl(vector unsigned int, int, unsigned int *);
void	vec_stvxl(vector unsigned int, int, unsigned long *);
void	vec_stvxl(vector unsigned int, int, vector unsigned long *);

void	vec_stvxl(vector unsigned char, int, unsigned char *);
void	vec_stvxl(vector unsigned char, int, vector unsigned char *);
vector float	vec_sub(vector float, vector float);
vector signed char	vec_sub(vector bool char, vector signed char);
vector unsigned char	vec_sub(vector bool char, vector unsigned char);
vector signed char	vec_sub(vector signed char, vector bool char);
vector signed char	vec_sub(vector signed char, vector signed char);
vector unsigned char	vec_sub(vector unsigned char, vector bool char);
vector unsigned char	vec_sub(vector unsigned char, vector unsigned char);
vector signed short	vec_sub(vector bool short, vector signed short);
vector unsigned short	vec_sub(vector bool short, vector unsigned short);
vector signed short	vec_sub(vector signed short, vector bool short);
vector signed short	vec_sub(vector signed short, vector signed short);
vector unsigned short	vec_sub(vector unsigned short, vector bool short);
vector unsigned short	vec_sub(vector unsigned short, vector unsigned short);
vector signed int	vec_sub(vector bool long, vector signed int);
vector unsigned int	vec_sub(vector bool long, vector unsigned int);
vector signed int	vec_sub(vector signed int, vector bool long);
vector signed int	vec_sub(vector signed int, vector signed int);
vector unsigned int	vec_sub(vector unsigned int, vector bool long);
vector unsigned int	vec_sub(vector unsigned int, vector unsigned int);
vector unsigned int	vec_subc(vector unsigned int, vector unsigned int);
vector signed char	vec_subs(vector bool char, vector signed char);
vector signed char	vec_subs(vector signed char, vector bool char);
vector signed char	vec_subs(vector signed char, vector signed char);
vector signed short	vec_subs(vector bool short, vector signed short);
vector signed short	vec_subs(vector signed short, vector bool short);
vector signed short	vec_subs(vector signed short, vector signed short);
vector signed int	vec_subs(vector bool long, vector signed int);
vector signed int	vec_subs(vector signed int, vector bool long);
vector signed int	vec_subs(vector signed int, vector signed int);
vector unsigned char	vec_subs(vector bool char, vector unsigned char);
vector unsigned char	vec_subs(vector unsigned char, vector bool char);
vector unsigned char	vec_subs(vector unsigned char, vector unsigned char);
vector unsigned short	vec_subs(vector bool short, vector unsigned short);
vector unsigned short	vec_subs(vector unsigned short, vector bool short);
vector unsigned short	vec_subs(vector unsigned short, vector unsigned short);

vector unsigned int	vec_subs(vector bool long, vector unsigned int);
vector unsigned int	vec_subs(vector unsigned int, vector bool long);
vector unsigned int	vec_subs(vector unsigned int, vector unsigned int);
vector signed int	vec_sum2s(vector signed int, vector signed int);
vector signed int	vec_sum4s(vector signed char, vector signed int);
vector signed int	vec_sum4s(vector signed short, vector signed int);
vector unsigned int	vec_sum4s(vector unsigned char, vector unsigned int);
vector signed int	vec_sums(vector signed int, vector signed int);
vector float	vec_trunc(vector float);
vector signed int	vec_unpack2sh(vector unsigned short, vector unsigned short);
vector signed short	vec_unpack2sh(vector unsigned char, vector unsigned char);
vector signed int	vec_unpack2sl(vector unsigned short, vector unsigned short);
vector signed short	vec_unpack2sl(vector unsigned char, vector unsigned char);
vector unsigned int	vec_unpack2uh(vector unsigned short, vector unsigned short);
vector unsigned short	vec_unpack2uh(vector unsigned char, vector unsigned char);
vector unsigned int	vec_unpack2ul(vector unsigned short, vector unsigned short);
vector unsigned short	vec_unpack2ul(vector unsigned char, vector unsigned char);
vector unsigned int	vec_unpackh(vector pixel);
vector bool short	vec_unpackh(vector bool char);
vector signed short	vec_unpackh(vector signed char);
vector bool long	vec_unpackh(vector bool short);
vector signed int	vec_unpackh(vector signed short);
vector unsigned int	vec_unpackl(vector pixel);
vector bool short	vec_unpackl(vector bool char);
vector signed short	vec_unpackl(vector signed char);
vector bool long	vec_unpackl(vector bool short);
vector signed int	vec_unpackl(vector signed short);
vector unsigned int	vec_vaddcuw(vector unsigned int, vector unsigned int);
vector float	vec_vaddfp(vector float, vector float);
vector signed char	vec_vaddsbs(vector bool char, vector signed char);
vector signed char	vec_vaddsbs(vector signed char, vector bool char);
vector signed char	vec_vaddsbs(vector signed char, vector signed char);
vector signed short	vec_vaddshs(vector bool short, vector signed short);
vector signed short	vec_vaddshs(vector signed short, vector bool short);
vector signed short	vec_vaddshs(vector signed short, vector signed short);
vector signed int	vec_vaddsws(vector bool long, vector signed int);
vector signed int	vec_vaddsws(vector signed int, vector bool long);

vector signed int	vec_vaddsws(vector signed int, vector signed int);
vector signed char	vec_vaddubm(vector bool char, vector signed char);
vector unsigned char	vec_vaddubm(vector bool char, vector unsigned char);
vector signed char	vec_vaddubm(vector signed char, vector bool char);
vector signed char	vec_vaddubm(vector signed char, vector signed char);
vector unsigned char	vec_vaddubm(vector unsigned char, vector bool char);
vector unsigned char	vec_vaddubm(vector unsigned char, vector unsigned char);
vector unsigned char	vec_vaddubs(vector bool char, vector unsigned char);
vector unsigned char	vec_vaddubs(vector unsigned char, vector bool char);
vector unsigned char	vec_vaddubs(vector unsigned char, vector unsigned char);
vector signed short	vec_vadduhm(vector bool short, vector signed short);
vector unsigned short	vec_vadduhm(vector bool short, vector unsigned short);
vector signed short	vec_vadduhm(vector signed short, vector bool short);
vector signed short	vec_vadduhm(vector signed short, vector signed short);
vector unsigned short	vec_vadduhm(vector unsigned short, vector bool short);
vector unsigned short	vec_vadduhm(vector unsigned short, vector unsigned short);
vector unsigned short	vec_vadduhs(vector bool short, vector unsigned short);
vector unsigned short	vec_vadduhs(vector unsigned short, vector bool short);
vector unsigned short	vec_vadduhs(vector unsigned short, vector unsigned short);
vector signed int	vec_vadduwm(vector bool long, vector signed int);
vector unsigned int	vec_vadduwm(vector bool long, vector unsigned int);
vector signed int	vec_vadduwm(vector signed int, vector bool long);
vector signed int	vec_vadduwm(vector signed int, vector signed int);
vector unsigned int	vec_vadduwm(vector unsigned int, vector bool long);
vector unsigned int	vec_vadduwm(vector unsigned int, vector unsigned int);
vector unsigned int	vec_vadduws(vector bool long, vector unsigned int);
vector unsigned int	vec_vadduws(vector unsigned int, vector bool long);
vector unsigned int	vec_vadduws(vector unsigned int, vector unsigned int);
vector bool short	vec_vand(vector bool short, vector bool short);
vector signed short	vec_vand(vector bool short, vector signed short);
vector unsigned short	vec_vand(vector bool short, vector unsigned short);
vector bool long	vec_vand(vector bool long, vector bool long);
vector float	vec_vand(vector bool long, vector float);
vector signed int	vec_vand(vector bool long, vector signed int);
vector unsigned int	vec_vand(vector bool long, vector unsigned int);
vector bool char	vec_vand(vector bool char, vector bool char);
vector signed char	vec_vand(vector bool char, vector signed char);

vector unsigned char	vec_vand(vector bool char, vector unsigned char);
vector float	vec_vand(vector float, vector bool long);
vector float	vec_vand(vector float, vector float);
vector signed short	vec_vand(vector signed short, vector bool short);
vector signed short	vec_vand(vector signed short, vector signed short);
vector signed int	vec_vand(vector signed int, vector bool long);
vector signed int	vec_vand(vector signed int, vector signed int);
vector signed char	vec_vand(vector signed char, vector bool char);
vector signed char	vec_vand(vector signed char, vector signed char);
vector unsigned short	vec_vand(vector unsigned short, vector bool short);
vector unsigned short	vec_vand(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vand(vector unsigned int, vector bool long);
vector unsigned int	vec_vand(vector unsigned int, vector unsigned int);
vector unsigned char	vec_vand(vector unsigned char, vector bool char);
vector unsigned char	vec_vand(vector unsigned char, vector unsigned char);
vector bool short	vec_vandc(vector bool short, vector bool short);
vector signed short	vec_vandc(vector bool short, vector signed short);
vector unsigned short	vec_vandc(vector bool short, vector unsigned short);
vector bool long	vec_vandc(vector bool long, vector bool long);
vector float	vec_vandc(vector bool long, vector float);
vector signed int	vec_vandc(vector bool long, vector signed int);
vector unsigned int	vec_vandc(vector bool long, vector unsigned int);
vector bool char	vec_vandc(vector bool char, vector bool char);
vector signed char	vec_vandc(vector bool char, vector signed char);
vector unsigned char	vec_vandc(vector bool char, vector unsigned char);
vector float	vec_vandc(vector float, vector bool long);
vector float	vec_vandc(vector float, vector float);
vector signed short	vec_vandc(vector signed short, vector bool short);
vector signed short	vec_vandc(vector signed short, vector signed short);
vector signed int	vec_vandc(vector signed int, vector bool long);
vector signed int	vec_vandc(vector signed int, vector signed int);
vector signed char	vec_vandc(vector signed char, vector bool char);
vector signed char	vec_vandc(vector signed char, vector signed char);
vector unsigned short	vec_vandc(vector unsigned short, vector bool short);
vector unsigned short	vec_vandc(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vandc(vector unsigned int, vector bool long);
vector unsigned int	vec_vandc(vector unsigned int, vector unsigned int);

vector unsigned char	vec_vandc(vector unsigned char, vector bool char);
vector unsigned char	vec_vandc(vector unsigned char, vector unsigned char);
vector signed char	vec_vavgsb(vector signed char, vector signed char);
vector signed short	vec_vavgsh(vector signed short, vector signed short);
vector signed int	vec_vavgsw(vector signed int, vector signed int);
vector unsigned char	vec_vavgub(vector unsigned char, vector unsigned char);
vector unsigned short	vec_vavguh(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vavguw(vector unsigned int, vector unsigned int);
vector float	vec_vcfsx(vector signed int, int);
vector float	vec_vcfux(vector unsigned int, int);
vector signed int	vec_vcmpbfp(vector float, vector float);
vector bool long	vec_vcmpeqfp(vector float, vector float);
vector bool char	vec_vcmpequb(vector signed char, vector signed char);
vector bool char	vec_vcmpequb(vector unsigned char, vector unsigned char);
vector bool short	vec_vcmpequh(vector signed short, vector signed short);
vector bool short	vec_vcmpequh(vector unsigned short, vector unsigned short);
vector bool long	vec_vcmpequw(vector signed int, vector signed int);
vector bool long	vec_vcmpequw(vector unsigned int, vector unsigned int);
vector bool long	vec_vcmpgefp(vector float, vector float);
vector bool long	vec_vcmpgtfp(vector float, vector float);
vector bool char	vec_vcmpgtsb(vector signed char, vector signed char);
vector bool short	vec_vcmpgtsh(vector signed short, vector signed short);
vector bool long	vec_vcmpgtsw(vector signed int, vector signed int);
vector bool char	vec_vcmpgtub(vector unsigned char, vector unsigned char);
vector bool short	vec_vcmpgtuh(vector unsigned short, vector unsigned short);
vector bool long	vec_vcmpgtuw(vector unsigned int, vector unsigned int);
vector signed int	vec_vctsxs(vector float, int);
vector unsigned int	vec_vctuxs(vector float, int);
vector float	vec_vexptefp(vector float);
vector float	vec_vlogefp(vector float);
vector float	vec_vmaddfp(vector float, vector float, vector float);
vector float	vec_vmaxfp(vector float, vector float);
vector signed char	vec_vmaxsb(vector bool char, vector signed char);
vector signed char	vec_vmaxsb(vector signed char, vector bool char);
vector signed char	vec_vmaxsb(vector signed char, vector signed char);
vector signed short	vec_vmaxsh(vector bool short, vector signed short);
vector signed short	vec_vmaxsh(vector signed short, vector bool short);

vector signed short	vec_vmaxsh(vector signed short, vector signed short);
vector signed int	vec_vmaxsw(vector bool long, vector signed int);
vector signed int	vec_vmaxsw(vector signed int, vector bool long);
vector signed int	vec_vmaxsw(vector signed int, vector signed int);
vector unsigned char	vec_vmaxub(vector bool char, vector unsigned char);
vector unsigned char	vec_vmaxub(vector unsigned char, vector bool char);
vector unsigned char	vec_vmaxub(vector unsigned char, vector unsigned char);
vector unsigned short	vec_vmaxuh(vector bool short, vector unsigned short);
vector unsigned short	vec_vmaxuh(vector unsigned short, vector bool short);
vector unsigned short	vec_vmaxuh(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vmaxuw(vector bool long, vector unsigned int);
vector unsigned int	vec_vmaxuw(vector unsigned int, vector bool long);
vector unsigned int	vec_vmaxuw(vector unsigned int, vector unsigned int);
vector signed short	vec_vmhaddshs(vector signed short, vector signed short, vector signed short);
vector signed short	vec_vmhraddshs(vector signed short, vector signed short, vector signed short);
vector float	vec_vminfp(vector float, vector float);
vector signed char	vec_vminsb(vector bool char, vector signed char);
vector signed char	vec_vminsb(vector signed char, vector bool char);
vector signed char	vec_vminsb(vector signed char, vector signed char);
vector signed short	vec_vminsh(vector bool short, vector signed short);
vector signed short	vec_vminsh(vector signed short, vector bool short);
vector signed short	vec_vminsh(vector signed short, vector signed short);
vector signed int	vec_vminsw(vector bool long, vector signed int);
vector signed int	vec_vminsw(vector signed int, vector bool long);
vector signed int	vec_vminsw(vector signed int, vector signed int);
vector unsigned char	vec_vminub(vector bool char, vector unsigned char);
vector unsigned char	vec_vminub(vector unsigned char, vector bool char);
vector unsigned char	vec_vminub(vector unsigned char, vector unsigned char);
vector unsigned short	vec_vminuh(vector bool short, vector unsigned short);
vector unsigned short	vec_vminuh(vector unsigned short, vector bool short);
vector unsigned short	vec_vminuh(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vminuw(vector bool long, vector unsigned int);
vector unsigned int	vec_vminuw(vector unsigned int, vector bool long);
vector unsigned int	vec_vminuw(vector unsigned int, vector unsigned int);
vector signed short	vec_vmladduhm(vector signed short, vector signed short, vector signed short);

vector signed short	vec_vmladduhm(vector signed short, vector unsigned short, vector unsigned short);
vector signed short	vec_vmladduhm(vector unsigned short, vector signed short, vector signed short);
vector unsigned short	vec_vmladduhm(vector unsigned short, vector unsigned short, vector unsigned short);
vector bool char	vec_vmrghb(vector bool char, vector bool char);
vector signed char	vec_vmrghb(vector signed char, vector signed char);
vector unsigned char	vec_vmrghb(vector unsigned char, vector unsigned char);
vector bool short	vec_vmrghh(vector bool short, vector bool short);
vector pixel	vec_vmrghh(vector pixel, vector pixel);
vector signed short	vec_vmrghh(vector signed short, vector signed short);
vector unsigned short	vec_vmrghh(vector unsigned short, vector unsigned short);
vector bool long	vec_vmrghw(vector bool long, vector bool long);
vector float	vec_vmrghw(vector float, vector float);
vector signed int	vec_vmrghw(vector signed int, vector signed int);
vector unsigned int	vec_vmrghw(vector unsigned int, vector unsigned int);
vector bool char	vec_vmrglb(vector bool char, vector bool char);
vector signed char	vec_vmrglb(vector signed char, vector signed char);
vector unsigned char	vec_vmrglb(vector unsigned char, vector unsigned char);
vector bool short	vec_vmrglh(vector bool short, vector bool short);
vector pixel	vec_vmrglh(vector pixel, vector pixel);
vector signed short	vec_vmrglh(vector signed short, vector signed short);
vector unsigned short	vec_vmrglh(vector unsigned short, vector unsigned short);
vector bool long	vec_vmrglw(vector bool long, vector bool long);
vector float	vec_vmrglw(vector float, vector float);
vector signed int	vec_vmrglw(vector signed int, vector signed int);
vector unsigned int	vec_vmrglw(vector unsigned int, vector unsigned int);
vector signed int	vec_vmsummbm(vector signed char, vector unsigned char, vector signed int);
vector signed int	vec_vmsumshm(vector signed short, vector signed short, vector signed int);
vector signed int	vec_vmsumshs(vector signed short, vector signed short, vector signed int);
vector unsigned int	vec_vmsumubm(vector unsigned char, vector unsigned char, vector unsigned int);
vector unsigned int	vec_vmsumuhm(vector unsigned short, vector unsigned short, vector unsigned int);
vector unsigned int	vec_vmsumuhs(vector unsigned short, vector unsigned short, vector unsigned int);
vector signed short	vec_vmulesb(vector signed char, vector signed char);

vector signed int	vec_vmulesh(vector signed short, vector signed short);
vector unsigned short	vec_vmuleub(vector unsigned char, vector unsigned char);
vector unsigned int	vec_vmuleuh(vector unsigned short, vector unsigned short);
vector signed short	vec_vmulosb(vector signed char, vector signed char);
vector signed int	vec_vmulosh(vector signed short, vector signed short);
vector unsigned short	vec_vmuloub(vector unsigned char, vector unsigned char);
vector unsigned int	vec_vmulouh(vector unsigned short, vector unsigned short);
vector float	vec_vnmsubfp(vector float, vector float, vector float);
vector bool short	vec_vnor(vector bool short, vector bool short);
vector bool long	vec_vnor(vector bool long, vector bool long);
vector bool char	vec_vnor(vector bool char, vector bool char);
vector float	vec_vnor(vector float, vector float);
vector signed short	vec_vnor(vector signed short, vector signed short);
vector signed int	vec_vnor(vector signed int, vector signed int);
vector signed char	vec_vnor(vector signed char, vector signed char);
vector unsigned short	vec_vnor(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vnor(vector unsigned int, vector unsigned int);
vector unsigned char	vec_vnor(vector unsigned char, vector unsigned char);
vector bool short	vec_vor(vector bool short, vector bool short);
vector signed short	vec_vor(vector bool short, vector signed short);
vector unsigned short	vec_vor(vector bool short, vector unsigned short);
vector bool long	vec_vor(vector bool long, vector bool long);
vector float	vec_vor(vector bool long, vector float);
vector signed int	vec_vor(vector bool long, vector signed int);
vector unsigned int	vec_vor(vector bool long, vector unsigned int);
vector bool char	vec_vor(vector bool char, vector bool char);
vector signed char	vec_vor(vector bool char, vector signed char);
vector unsigned char	vec_vor(vector bool char, vector unsigned char);
vector float	vec_vor(vector float, vector bool long);
vector float	vec_vor(vector float, vector float);
vector signed short	vec_vor(vector signed short, vector bool short);
vector signed short	vec_vor(vector signed short, vector signed short);
vector signed int	vec_vor(vector signed int, vector bool long);
vector signed int	vec_vor(vector signed int, vector signed int);
vector signed char	vec_vor(vector signed char, vector bool char);
vector signed char	vec_vor(vector signed char, vector signed char);
vector unsigned short	vec_vor(vector unsigned short, vector bool short);

vector unsigned short	vec_vor(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vor(vector unsigned int, vector bool long);
vector unsigned int	vec_vor(vector unsigned int, vector unsigned int);
vector unsigned char	vec_vor(vector unsigned char, vector bool char);
vector unsigned char	vec_vor(vector unsigned char, vector unsigned char);
vector bool short	vec_vperm(vector bool short, vector bool short, vector unsigned char);
vector bool long	vec_vperm(vector bool long, vector bool long, vector unsigned char);
vector bool char	vec_vperm(vector bool char, vector bool char, vector unsigned char);
vector float	vec_vperm(vector float, vector float, vector unsigned char);
vector pixel	vec_vperm(vector pixel, vector pixel, vector unsigned char);
vector signed short	vec_vperm(vector signed short, vector signed short, vector unsigned char);
vector signed int	vec_vperm(vector signed int, vector signed int, vector unsigned char);
vector signed char	vec_vperm(vector signed char, vector signed char, vector unsigned char);
vector unsigned short	vec_vperm(vector unsigned short, vector unsigned short, vector unsigned char);
vector unsigned int	vec_vperm(vector unsigned int, vector unsigned int, vector unsigned char);
vector unsigned char	vec_vperm(vector unsigned char, vector unsigned char, vector unsigned char);
vector pixel	vec_vpkpx(vector unsigned int, vector unsigned int);
vector signed char	vec_vpkshss(vector signed short, vector signed short);
vector unsigned char	vec_vpkshus(vector signed short, vector signed short);
vector signed short	vec_vpkswss(vector signed int, vector signed int);
vector unsigned short	vec_vpkswus(vector signed int, vector signed int);
vector bool char	vec_vpkuhum(vector bool short, vector bool short);
vector signed char	vec_vpkuhum(vector signed short, vector signed short);
vector unsigned char	vec_vpkuhum(vector unsigned short, vector unsigned short);
vector unsigned char	vec_vpkuhus(vector unsigned short, vector unsigned short);
vector bool short	vec_vpkuwum(vector bool long, vector bool long);
vector signed short	vec_vpkuwum(vector signed int, vector signed int);
vector unsigned short	vec_vpkuwum(vector unsigned int, vector unsigned int);
vector unsigned short	vec_vpkuwus(vector unsigned int, vector unsigned int);
vector float	vec_vrefp(vector float);
vector float	vec_vrfim(vector float);
vector float	vec_vrfin(vector float);
vector float	vec_vrfip(vector float);
vector float	vec_vrfiz(vector float);
vector signed char	vec_vrlb(vector signed char, vector unsigned char);

vector unsigned char	vec_vrlb(vector unsigned char, vector unsigned char);
vector signed short	vec_vrlh(vector signed short, vector unsigned short);
vector unsigned short	vec_vrlh(vector unsigned short, vector unsigned short);
vector signed int	vec_vrlw(vector signed int, vector unsigned int);
vector unsigned int	vec_vrlw(vector unsigned int, vector unsigned int);
vector float	vec_vrsqrtefp(vector float);
vector bool short	vec_vsel(vector bool short, vector bool short, vector bool short);
vector bool short	vec_vsel(vector bool short, vector bool short, vector unsigned short);
vector bool long	vec_vsel(vector bool long, vector bool long, vector bool long);
vector bool long	vec_vsel(vector bool long, vector bool long, vector unsigned int);
vector bool char	vec_vsel(vector bool char, vector bool char, vector bool char);
vector bool char	vec_vsel(vector bool char, vector bool char, vector unsigned char);
vector float	vec_vsel(vector float, vector float, vector bool long);
vector float	vec_vsel(vector float, vector float, vector unsigned int);
vector signed short	vec_vsel(vector signed short, vector signed short, vector bool short);
vector signed short	vec_vsel(vector signed short, vector signed short, vector unsigned short);
vector signed int	vec_vsel(vector signed int, vector signed int, vector bool long);
vector signed int	vec_vsel(vector signed int, vector signed int, vector unsigned int);
vector signed char	vec_vsel(vector signed char, vector signed char, vector bool char);
vector signed char	vec_vsel(vector signed char, vector signed char, vector unsigned char);
vector unsigned short	vec_vsel(vector unsigned short, vector unsigned short, vector bool short);
vector unsigned short	vec_vsel(vector unsigned short, vector unsigned short, vector unsigned short);
vector unsigned int	vec_vsel(vector unsigned int, vector unsigned int, vector bool long);
vector unsigned int	vec_vsel(vector unsigned int, vector unsigned int, vector unsigned int);
vector unsigned char	vec_vsel(vector unsigned char, vector unsigned char, vector bool char);
vector unsigned char	vec_vsel(vector unsigned char, vector unsigned char, vector unsigned char);
vector bool short	vec_vsl(vector bool short, vector unsigned short);
vector bool short	vec_vsl(vector bool short, vector unsigned int);
vector bool short	vec_vsl(vector bool short, vector unsigned char);
vector bool long	vec_vsl(vector bool long, vector unsigned short);
vector bool long	vec_vsl(vector bool long, vector unsigned int);
vector bool long	vec_vsl(vector bool long, vector unsigned char);
vector bool char	vec_vsl(vector bool char, vector unsigned short);
vector bool char	vec_vsl(vector bool char, vector unsigned int);
vector bool char	vec_vsl(vector bool char, vector unsigned char);
vector pixel	vec_vsl(vector pixel, vector unsigned short);

vector pixel	vec_vsl(vector pixel, vector unsigned int);
vector pixel	vec_vsl(vector pixel, vector unsigned char);
vector signed short	vec_vsl(vector signed short, vector unsigned short);
vector signed short	vec_vsl(vector signed short, vector unsigned int);
vector signed short	vec_vsl(vector signed short, vector unsigned char);
vector signed int	vec_vsl(vector signed int, vector unsigned short);
vector signed int	vec_vsl(vector signed int, vector unsigned int);
vector signed int	vec_vsl(vector signed int, vector unsigned char);
vector signed char	vec_vsl(vector signed char, vector unsigned short);
vector signed char	vec_vsl(vector signed char, vector unsigned int);
vector signed char	vec_vsl(vector signed char, vector unsigned char);
vector unsigned short	vec_vsl(vector unsigned short, vector unsigned short);
vector unsigned short	vec_vsl(vector unsigned short, vector unsigned int);
vector unsigned short	vec_vsl(vector unsigned short, vector unsigned char);
vector unsigned int	vec_vsl(vector unsigned int, vector unsigned short);
vector unsigned int	vec_vsl(vector unsigned int, vector unsigned int);
vector unsigned int	vec_vsl(vector unsigned int, vector unsigned char);
vector unsigned char	vec_vsl(vector unsigned char, vector unsigned short);
vector unsigned char	vec_vsl(vector unsigned char, vector unsigned int);
vector unsigned char	vec_vsl(vector unsigned char, vector unsigned char);
vector signed char	vec_vslb(vector signed char, vector unsigned char);
vector unsigned char	vec_vslb(vector unsigned char, vector unsigned char);
vector float	vec_vsldoi(vector float, vector float, int);
vector pixel	vec_vsldoi(vector pixel, vector pixel, int);
vector signed short	vec_vsldoi(vector signed short, vector signed short, int);
vector signed int	vec_vsldoi(vector signed int, vector signed int, int);
vector signed char	vec_vsldoi(vector signed char, vector signed char, int);
vector unsigned short	vec_vsldoi(vector unsigned short, vector unsigned short, int);
vector unsigned int	vec_vsldoi(vector unsigned int, vector unsigned int, int);
vector unsigned char	vec_vsldoi(vector unsigned char, vector unsigned char, int);
vector signed short	vec_vslh(vector signed short, vector unsigned short);
vector unsigned short	vec_vslh(vector unsigned short, vector unsigned short);
vector float	vec_vslo(vector float, vector signed char);
vector float	vec_vslo(vector float, vector unsigned char);
vector pixel	vec_vslo(vector pixel, vector signed char);
vector pixel	vec_vslo(vector pixel, vector unsigned char);
vector signed short	vec_vslo(vector signed short, vector signed char);

vector signed short	vec_vslo(vector signed short, vector unsigned char);
vector signed int	vec_vslo(vector signed int, vector signed char);
vector signed int	vec_vslo(vector signed int, vector unsigned char);
vector signed char	vec_vslo(vector signed char, vector signed char);
vector signed char	vec_vslo(vector signed char, vector unsigned char);
vector unsigned short	vec_vslo(vector unsigned short, vector signed char);
vector unsigned short	vec_vslo(vector unsigned short, vector unsigned char);
vector unsigned int	vec_vslo(vector unsigned int, vector signed char);
vector unsigned int	vec_vslo(vector unsigned int, vector unsigned char);
vector unsigned char	vec_vslo(vector unsigned char, vector signed char);
vector unsigned char	vec_vslo(vector unsigned char, vector unsigned char);
vector signed int	vec_vslw(vector signed int, vector unsigned int);
vector unsigned int	vec_vslw(vector unsigned int, vector unsigned int);
vector bool char	vec_vspltb(vector bool char, int);
vector signed char	vec_vspltb(vector signed char, int);
vector unsigned char	vec_vspltb(vector unsigned char, int);
vector bool short	vec_vsplth(vector bool short, int);
vector pixel	vec_vsplth(vector pixel, int);
vector signed short	vec_vsplth(vector signed short, int);
vector unsigned short	vec_vsplth(vector unsigned short, int);
vector signed char	vec_vspltisb(int);
vector signed short	vec_vspltish(int);
vector signed int	vec_vspltisw(int);
vector bool long	vec_vspltw(vector bool long, int);
vector float	vec_vspltw(vector float, int);
vector signed int	vec_vspltw(vector signed int, int);
vector unsigned int	vec_vspltw(vector unsigned int, int);
vector bool short	vec_vsr(vector bool short, vector unsigned short);
vector bool short	vec_vsr(vector bool short, vector unsigned int);
vector bool short	vec_vsr(vector bool short, vector unsigned char);
vector bool long	vec_vsr(vector bool long, vector unsigned short);
vector bool long	vec_vsr(vector bool long, vector unsigned int);
vector bool long	vec_vsr(vector bool long, vector unsigned char);
vector bool char	vec_vsr(vector bool char, vector unsigned short);
vector bool char	vec_vsr(vector bool char, vector unsigned int);
vector bool char	vec_vsr(vector bool char, vector unsigned char);
vector pixel	vec_vsr(vector pixel, vector unsigned short);

vector pixel	vec_vsr(vector pixel, vector unsigned int);
vector pixel	vec_vsr(vector pixel, vector unsigned char);
vector signed short	vec_vsr(vector signed short, vector unsigned short);
vector signed short	vec_vsr(vector signed short, vector unsigned int);
vector signed short	vec_vsr(vector signed short, vector unsigned char);
vector signed int	vec_vsr(vector signed int, vector unsigned short);
vector signed int	vec_vsr(vector signed int, vector unsigned int);
vector signed int	vec_vsr(vector signed int, vector unsigned char);
vector signed char	vec_vsr(vector signed char, vector unsigned short);
vector signed char	vec_vsr(vector signed char, vector unsigned int);
vector signed char	vec_vsr(vector signed char, vector unsigned char);
vector unsigned short	vec_vsr(vector unsigned short, vector unsigned short);
vector unsigned short	vec_vsr(vector unsigned short, vector unsigned int);
vector unsigned short	vec_vsr(vector unsigned short, vector unsigned char);
vector unsigned int	vec_vsr(vector unsigned int, vector unsigned short);
vector unsigned int	vec_vsr(vector unsigned int, vector unsigned int);
vector unsigned int	vec_vsr(vector unsigned int, vector unsigned char);
vector unsigned char	vec_vsr(vector unsigned char, vector unsigned short);
vector unsigned char	vec_vsr(vector unsigned char, vector unsigned int);
vector unsigned char	vec_vsr(vector unsigned char, vector unsigned char);
vector signed char	vec_vsrab(vector signed char, vector unsigned char);
vector unsigned char	vec_vsrab(vector unsigned char, vector unsigned char);
vector signed short	vec_vsrah(vector signed short, vector unsigned short);
vector unsigned short	vec_vsrah(vector unsigned short, vector unsigned short);
vector signed int	vec_vsraw(vector signed int, vector unsigned int);
vector unsigned int	vec_vsraw(vector unsigned int, vector unsigned int);
vector signed char	vec_vsrb(vector signed char, vector unsigned char);
vector unsigned char	vec_vsrb(vector unsigned char, vector unsigned char);
vector signed short	vec_vsrh(vector signed short, vector unsigned short);
vector unsigned short	vec_vsrh(vector unsigned short, vector unsigned short);
vector float	vec_vsro(vector float, vector signed char);
vector float	vec_vsro(vector float, vector unsigned char);
vector pixel	vec_vsro(vector pixel, vector signed char);
vector pixel	vec_vsro(vector pixel, vector unsigned char);
vector signed short	vec_vsro(vector signed short, vector signed char);
vector signed short	vec_vsro(vector signed short, vector unsigned char);
vector signed int	vec_vsro(vector signed int, vector signed char);

vector signed int	vec_vsro(vector signed int, vector unsigned char);
vector signed char	vec_vsro(vector signed char, vector signed char);
vector signed char	vec_vsro(vector signed char, vector unsigned char);
vector unsigned short	vec_vsro(vector unsigned short, vector signed char);
vector unsigned short	vec_vsro(vector unsigned short, vector unsigned char);
vector unsigned int	vec_vsro(vector unsigned int, vector signed char);
vector unsigned int	vec_vsro(vector unsigned int, vector unsigned char);
vector unsigned char	vec_vsro(vector unsigned char, vector signed char);
vector unsigned char	vec_vsro(vector unsigned char, vector unsigned char);
vector signed int	vec_vsrw(vector signed int, vector unsigned int);
vector unsigned int	vec_vsrw(vector unsigned int, vector unsigned int);
vector unsigned int	vec_vsubcuw(vector unsigned int, vector unsigned int);
vector float	vec_vsubfp(vector float, vector float);
vector signed char	vec_vsubsbs(vector bool char, vector signed char);
vector signed char	vec_vsubsbs(vector signed char, vector bool char);
vector signed char	vec_vsubsbs(vector signed char, vector signed char);
vector signed short	vec_vsubshs(vector bool short, vector signed short);
vector signed short	vec_vsubshs(vector signed short, vector bool short);
vector signed short	vec_vsubshs(vector signed short, vector signed short);
vector signed int	vec_vsubsws(vector bool long, vector signed int);
vector signed int	vec_vsubsws(vector signed int, vector bool long);
vector signed int	vec_vsubsws(vector signed int, vector signed int);
vector signed char	vec_vsububm(vector bool char, vector signed char);
vector unsigned char	vec_vsububm(vector bool char, vector unsigned char);
vector signed char	vec_vsububm(vector signed char, vector bool char);
vector signed char	vec_vsububm(vector signed char, vector signed char);
vector unsigned char	vec_vsububm(vector unsigned char, vector bool char);
vector unsigned char	vec_vsububm(vector unsigned char, vector unsigned char);
vector unsigned char	vec_vsububs(vector bool char, vector unsigned char);
vector unsigned char	vec_vsububs(vector unsigned char, vector bool char);
vector unsigned char	vec_vsububs(vector unsigned char, vector unsigned char);
vector signed short	vec_vsubuhm(vector bool short, vector signed short);
vector unsigned short	vec_vsubuhm(vector bool short, vector unsigned short);
vector signed short	vec_vsubuhm(vector signed short, vector bool short);
vector signed short	vec_vsubuhm(vector signed short, vector signed short);
vector unsigned short	vec_vsubuhm(vector unsigned short, vector bool short);
vector unsigned short	vec_vsubuhm(vector unsigned short, vector unsigned short);

vector unsigned short	vec_vsubuhs(vector bool short, vector unsigned short);
vector unsigned short	vec_vsubuhs(vector unsigned short, vector bool short);
vector unsigned short	vec_vsubuhs(vector unsigned short, vector unsigned short);
vector signed int	vec_vsubuwm(vector bool long, vector signed int);
vector unsigned int	vec_vsubuwm(vector bool long, vector unsigned int);
vector signed int	vec_vsubuwm(vector signed int, vector bool long);
vector signed int	vec_vsubuwm(vector signed int, vector signed int);
vector unsigned int	vec_vsubuwm(vector unsigned int, vector bool long);
vector unsigned int	vec_vsubuwm(vector unsigned int, vector unsigned int);
vector unsigned int	vec_vsubuws(vector bool long, vector unsigned int);
vector unsigned int	vec_vsubuws(vector unsigned int, vector bool long);
vector unsigned int	vec_vsubuws(vector unsigned int, vector unsigned int);
vector signed int	vec_vsum2sws(vector signed int, vector signed int);
vector signed int	vec_vsum4sbs(vector signed char, vector signed int);
vector signed int	vec_vsum4shs(vector signed short, vector signed int);
vector unsigned int	vec_vsum4ubs(vector unsigned char, vector unsigned int);
vector signed int	vec_vsumsws(vector signed int, vector signed int);
vector unsigned int	vec_vupkhpx(vector pixel);
vector bool short	vec_vupkhsb(vector bool char);
vector signed short	vec_vupkhsb(vector signed char);
vector bool long	vec_vupkhsh(vector bool short);
vector signed int	vec_vupkhsh(vector signed short);
vector unsigned int	vec_vupklpx(vector pixel);
vector bool short	vec_vupklsb(vector bool char);
vector signed short	vec_vupklsb(vector signed char);
vector bool long	vec_vupklsh(vector bool short);
vector signed int	vec_vupklsh(vector signed short);
vector bool short	vec_vxor(vector bool short, vector bool short);
vector signed short	vec_vxor(vector bool short, vector signed short);
vector unsigned short	vec_vxor(vector bool short, vector unsigned short);
vector bool long	vec_vxor(vector bool long, vector bool long);
vector float	vec_vxor(vector bool long, vector float);
vector signed int	vec_vxor(vector bool long, vector signed int);
vector unsigned int	vec_vxor(vector bool long, vector unsigned int);
vector bool char	vec_vxor(vector bool char, vector bool char);
vector signed char	vec_vxor(vector bool char, vector signed char);
vector unsigned char	vec_vxor(vector bool char, vector unsigned char);

vector float	vec_vxor(vector float, vector bool long);
vector float	vec_vxor(vector float, vector float);
vector signed short	vec_vxor(vector signed short, vector bool short);
vector signed short	vec_vxor(vector signed short, vector signed short);
vector signed int	vec_vxor(vector signed int, vector bool long);
vector signed int	vec_vxor(vector signed int, vector signed int);
vector signed char	vec_vxor(vector signed char, vector bool char);
vector signed char	vec_vxor(vector signed char, vector signed char);
vector unsigned short	vec_vxor(vector unsigned short, vector bool short);
vector unsigned short	vec_vxor(vector unsigned short, vector unsigned short);
vector unsigned int	vec_vxor(vector unsigned int, vector bool long);
vector unsigned int	vec_vxor(vector unsigned int, vector unsigned int);
vector unsigned char	vec_vxor(vector unsigned char, vector bool char);
vector unsigned char	vec_vxor(vector unsigned char, vector unsigned char);
vector bool short	vec_xor(vector bool short, vector bool short);
vector signed short	vec_xor(vector bool short, vector signed short);
vector unsigned short	vec_xor(vector bool short, vector unsigned short);
vector bool long	vec_xor(vector bool long, vector bool long);
vector float	vec_xor(vector bool long, vector float);
vector signed int	vec_xor(vector bool long, vector signed int);
vector unsigned int	vec_xor(vector bool long, vector unsigned int);
vector bool char	vec_xor(vector bool char, vector bool char);
vector signed char	vec_xor(vector bool char, vector signed char);
vector unsigned char	vec_xor(vector bool char, vector unsigned char);
vector float	vec_xor(vector float, vector bool long);
vector float	vec_xor(vector float, vector float);
vector signed short	vec_xor(vector signed short, vector bool short);
vector signed short	vec_xor(vector signed short, vector signed short);
vector signed int	vec_xor(vector signed int, vector bool long);
vector signed int	vec_xor(vector signed int, vector signed int);
vector signed char	vec_xor(vector signed char, vector bool char);
vector signed char	vec_xor(vector signed char, vector signed char);
vector unsigned short	vec_xor(vector unsigned short, vector bool short);
vector unsigned short	vec_xor(vector unsigned short, vector unsigned short);
vector unsigned int	vec_xor(vector unsigned int, vector bool long);
vector unsigned int	vec_xor(vector unsigned int, vector unsigned int);
vector unsigned char	vec_xor(vector unsigned char, vector bool char);

vector unsigned char	vec_xor(vector unsigned char, vector unsigned char);

10: 定義済みのマクロ リファレンス

一般的な定義済みシンボル

名前	デフォルト値	説明
__SNC__	1	常に有効。プログラムがSNC によってコンパイルされていることを示す。
__SN_VER__	各種	バージョン フォーマットにおける、特定ターゲットのSN コンパイラ バージョン。
__DATE__	"Mmm dd yyyy"	「Feb 19 2009」フォーマットでの日付文字列。
__TIME__	"hh:mm:ss"	「15:38:03」フォーマットでの時間文字列。
__EDG__	0	SNC ではリンク互換性を許可する目的で GCC ランタイム ライブラリが使用されるようになったため、__EDG__ は PS3 PPU コンパイラに対し、デフォルトで無効に設定。
__EDG_RUNTIME_NAMESPACES	1	EDG フロント エンドで名前領域を使用することを示す。
__EDG_IA64_ABI	1	コンパイラが IA-64 ABI を使用していることを示すため、1 と定義される。
__EDG_VERSION__	310	EDG バージョン番号。
__VERSION__	"EDG gcc 4.1.1 mode"	EDG バージョン文字列。
__BOOL_IS_KEYWORD	1	bool がキーワードの場合に定義される。
_BOOL_DEFINED	1	bool がキーワードの場合に定義される。
__SIGNED_CHARS__	1	limit.h 内の CHAR_MIN と CHAR_MAX 定義の変更に使用される。
__cplusplus	1	コンパイルがC++ モードの場合に定義される。
__WCHAR_T_IS_KEYWORD	1	wchar_t がキーワードの場合に定義される。
_WCHAR_T_DEFINED	1	wchar_t がキーワードの場合に定義される。
_NO_EX	1	例外処理が無効な場合に定義される。
__EXCEPTIONS	未定義	例外処理が有効な場合に定義される。
__PLACEMENT_DELETE	未定義	例外処理が有効な場合に定義される。
__RTTI	1	RTTI がコンパイラで有効な場合に定義される。
_M_IX86	未定義	Microsoft モードが指定された時に定義される。
_INTEGRAL_MAX_BITS	64	Microsoft モードが指定された時に定義される。
__STDC__	0	ANSI C モードと C++ モードで定義される。C++ モードでは、値の再定義が可能。Microsoft との互換モードでは定義されない。

__STDC_VERSION__	199901L	ANSI C モードでは、値 199409L で定義される。このマクロ名とその値は、ISO C89 標準規格の Normative Addendum 1 で指定される。C99 モードでは、値 199901L で定義される。
__STDC_HOSTED__	1	SNC がホストされた実装であることを示す。

GNU モード シンボル

GNU バージョン シンボル値は、-Xgnuversion 制御変数で管理されます (詳細は、82 ページの「-Xgnuversion」を参照)。デフォルト値は「411」で、コンパイラがデフォルトで GCC 4.1.1 を模倣する事実が反映されています。

名前	デフォルト値	説明
__GNUC__	4	SN コンパイラで受け入れられる主な GNUC バージョン ダイアレクト。
__GNUG__	4	SN コンパイラで受け入れられる主な GNUC バージョン ダイアレクト。(__GNUC__ && __cplusplus) と同等。
__GNUC_MINOR__	1	SN コンパイラで受け入れられるマイナーなGNUC バージョン ダイアレクト。
__GNUC_PATCHLEVEL__	1	バージョン 3.0 から GCC で定義されるパッチ レベルのマクロ。
__ELF__	1	ターゲットで ELF オブジェクト ファイル フォーマットが使用される場合に定義される。

ターゲット特有のシンボル

名前	デフォルト値	説明
__PPU__	1	PPU でアプリケーションが実行される。
__PPC__	1	ターゲット アーキテクチャがPowerPC である。
__PPC64__	1	ターゲット アーキテクチャが PowerPC で、64ビット コンパイル モードが有効である。
__CELLOS_LV2__	1	Havok ライブラリに必須。
_ARCH_PPC64	1	64ビット サポートを含め、アプリケーションがPowerPC で実行される (SCE アトミック ヘッダー ファイルに必須)。
__LP32__	1	ターゲット プラットホームで、int、long int、ポインタ タイプに対して 32ビットを使用。
__STRICT_ALIGNED	1	非標準アラインメントのタイプに対し、アラインメント パラメータを追加する、変異形新演算子が提供される場合、SCE「アラインされた新しい」言語拡張に対して必須となる。
__thread	__declspec (スレッド)	キーワード__thread。
__VEC__	10205	ベクター データ タイプをサポート。

__BIG_ENDIAN__	1	ターゲット プラットホームがビッグ エンディアンである。
__ALTIVEC__	1	ベクター データ タイプをサポート。

特別なマクロ

名前	デフォルト値	説明
__TIMESTAMP__	文字列定数	
__FILE__	文字列定数	コンパイル下にあるファイル名の文字列定数に拡張。
__LINE__	文字列定数	コンパイル下にあるソース ファイルの行番号に拡張。
__COUNTER__	整数定数	0 で始まり、コンパイル中に使用されるたびに 1 ずつ増える整数に拡張。
__BASE_FILE__		コンパイル下にあるプライマリ ソース ファイル名の文字列定数に拡張。
__SN_FILE__	文字列定数	__FILE__ と同じ。
__SN_BASE_FILE__	文字列定数	__BASE_FILE__ と同じ。

役立つリンク

- http://publib.boulder.ibm.com/infocenter/cellcomp/v9v111/index.jsp?topic=/com.ibm.xlcpp9.cell.doc/compiler_ref/platform_related.htm -ターゲット特有の定義済みマクロを多数紹介。
- http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html - GCC 定義済みマクロを説明。

11: インデックス

Q

R

S

T

U

W

X