

# インテル® C++ コンパイラ ユーザーズ・ガイド

© 1996-2002 Intel Corporation

無断での引用、転載を禁じます。

資料番号: CL-700-10

# 目次

目次 .....	2
免責条項 .....	20
インテル(R) C++ コンパイラについて .....	21
インテル® C++ コンパイラへようこそ .....	21
本リリースの新機能 .....	21
特徴と利点 .....	21
製品情報 Web サイトとサポート .....	22
システムの要件 .....	22
IA-32 プロセッサのシステムの要件 .....	22
Itanium® プロセッサのシステムの要件 .....	22
ソフトウェアの要件 .....	22
FLEXlm* 電子ライセンス .....	23
本書について .....	24
本書の使い方 .....	24
組込み関数名の表記 .....	24
クラス・ライブラリ名の表記 .....	25
参考文献 .....	26
コンパイラ・オプションのクイック・リファレンス・ガイド .....	28
オプション・クイック・リファレンス・ガイドの概要 .....	28
オプションのクイック・ガイドの表で使用される表記 .....	28
新しいオプション .....	28
アルファベット順のリスト .....	32
コンパイラ・オプションのクイック・リファレンス・ガイド .....	32
機能グループ別リスト .....	47
コンパイル処理のカスタマイズ・オプション .....	47
代替ツールと代替パス .....	47
前処理オプション .....	47
コンパイラのフローの制御 .....	47
コンパイル出力の制御 .....	48
デバッグ・オプション .....	48
言語適合性オプション .....	50
適合性オプション .....	50
アプリケーション・パフォーマンス最適化オプション .....	51
最適化の種類の指定 .....	51
プロセッサ・ディスパッチのサポート(IA-32 のみ) .....	51
プロシージャ間の最適化 .....	52
プロファイルに基づく最適化 .....	52
高水準言語の最適化 .....	53

最適化レポート .....	53
Windows* と Linux* のコンパイラ・オプションの対応表 .....	55
コンパイラ・オプションの対応表 .....	55
インテル(R) C++ コンパイラの起動 .....	60
コンパイラの起動 .....	60
コマンドラインからのコンパイラの起動 .....	60
環境変数の設定 .....	60
シェル・スクリプトの実行 .....	60
コンパイラの起動 .....	61
コマンド・ラインからの makefile の実行 .....	61
コンパイラの入力ファイル .....	61
コンパイラのデフォルトの動作 .....	62
デフォルトのコンパイラ・オプション .....	62
コンパイラのデフォルトの動作 .....	63
コンパイル・フェーズ .....	64
アプリケーションの開発サイクル .....	65
コンパイル環境のカスタマイズ .....	66
コンパイル環境のカスタマイズ .....	66
環境変数 .....	66
コンパイル環境オプション .....	66
Bash シェル環境 .....	67
設定ファイル .....	67
設定ファイルの使用方法 .....	67
応答ファイル .....	68
インクルード・ファイル .....	68
インクルード・ディレクトリの削除方法 .....	69
コンパイル処理のカスタマイズ .....	70
コンパイル処理のカスタマイズの概要 .....	70
代替ツールと代替パスの指定 .....	70
代替コンポーネントの指定方法 .....	70
他のプログラムへオプションを渡す方法 .....	70
前処理 .....	72
前処理の概要 .....	72
プリプロセッサ・オプション .....	72
前処理のみを行う .....	73
-E の使用 .....	73
-P の使用 .....	73
-EP の使用 .....	74
前処理済みのソース出力の中にコメントを保存する .....	74
前処理ディレクティブと同等の働き .....	74
-A の使用 .....	74

-D の使用 .....	74
-U の使用 .....	75
事前定義済みマクロ .....	75
事前定義済みマクロ .....	75
マクロ定義の抑止 .....	77
コンパイル .....	78
コンパイルの概要 .....	78
コンパイル・オプション .....	79
コンパイルの制御 .....	79
データ設定の監視 .....	79
構造体タグのアライメントの指定 .....	79
Itanium ベースのシステムでデノーマル値をゼロにフラッシュする .....	80
ゼロに初期化される変数を割り当てる .....	80
一部の命令についてのデコード処理の誤りを防ぐ(IA-32 のみ) .....	81
リンク .....	82
リンク .....	82
リンクを行わない設定にする .....	82
デバッグ .....	84
デバッグ・オプションの概要 .....	84
デバッグの準備 .....	84
シンボリック・デバッグのサポート .....	85
構文分析およびセマンティクスのみを行う .....	85
言語適合性 .....	86
C の標準規格との適合性 .....	86
ANSI/ISO標準準拠のC言語について .....	86
コンパイラ付属のマクロ .....	86
C99 サポート .....	87
C++の標準規格との適合性 .....	87
最適化 .....	88
最適化レベル .....	88
最適化レベルの概要 .....	88
最適化レベルの設定 .....	88
Itanium® アーキテクチャ用インテル® C++ コンパイラ .....	88
IA-32 コンパイラ .....	89
IA-32 と Itanium アーキテクチャ用インテル C++ コンパイラ .....	89
最適化の範囲の制限 .....	90
浮動小数点の最適化 .....	92
浮動小数点演算の精度 .....	92
IA-32、Itanium® ベース・システムに使用する各種オプション .....	92
-mp オプション .....	92
-mp1 オプション .....	92

IA-32 のみのオプション .....	93
-long_double オプション .....	93
-prec_div オプション .....	93
-pcn オプション .....	93
-rcd オプション .....	93
-fp_port オプション .....	94
IA-32、Itanium® ベース・システムに使用する浮動小数点演算オプション .....	94
浮動小数点積和/積差演算の縮約 .....	94
FP スペキュレーション .....	94
FP 演算の評価 .....	95
FP 結果の精度制御 .....	95
特定のプロセッサに合わせて最適化を行う .....	96
プロセッサの最適化 .....	96
IA-32 にのみ該当するプロセッサの最適化 .....	96
プロセッサの最適化(Itanium® ベース・システムのみ) .....	97
プロセッサ固有の最適化(IA-32のみ) .....	97
自動CPUディスパッチ(IA-32のみ) .....	98
プロセッサの最適化と自動CPUディスパッチを組み合わせる方法(IA-32のみ) .....	100
プロシージャ間の最適化 .....	102
プロシージャ間の最適化 .....	102
IA-32 および Itanium® ベース・アプリケーション .....	102
IA-32 アプリケーションのみ .....	102
マルチファイルIPO .....	103
マルチファイル IPO の概要 .....	103
コンパイル・フェーズ .....	103
リンケージ・フェーズ .....	103
実際のオブジェクト・ファイルを生成する .....	103
マルチファイル IPO の実行ファイルを作成する .....	104
IA-32の場合の手順 .....	104
Itanium ベース・システムの場合の手順 .....	104
xldl を使用してマルチファイル IPO 実行ファイルを作成する .....	105
使用規則 .....	105
xld オプション .....	106
IPOオブジェクトからライブラリを作成する .....	106
マルチファイル IPO の効果を分析する .....	107
-Qoption 指示子による -ip または -ipo の使用 .....	107
-option 指示子 .....	107
関数のインライン展開 .....	109
ユーザ関数のインライン展開の制御 .....	109
関数のインライン展開の条件 .....	109
プロファイルに基づく最適化 .....	110

プロファイルに基づく最適化の概要 .....	110
インストルメント済みプログラム .....	110
PGO で向上するパフォーマンス .....	110
プロファイルに基づく最適化の方法 .....	110
PGO フェーズ .....	111
基本的なPGOオプション .....	111
インストルメント済みコードの作成 .....	111
プロファイルによって最適化された実行ファイルの生成 .....	112
関数分割の無効(Itanium® コンパイラのみ) .....	112
プロファイルに基づく最適化の例 .....	112
インストルメンテーション・コンパイルとリンク .....	113
IA-32 システム .....	113
Itanium® ベース・システム .....	113
インストルメント済み実行 .....	113
フィードバック・コンパイル .....	113
IA-32のシステム: .....	113
Itanium ベース・システム: .....	113
PGOの環境変数 .....	114
プロファイルに基づく最適化に使う環境変数 .....	114
profmerge を使用して、ソースファイルの再配置 .....	114
ソースの再配置 .....	114
PGO API: プロファイル情報生成サポート .....	116
PGO API サポートの概要 .....	116
プロファイル情報のダンプ .....	116
動的プロファイル・カウンタのリセット .....	117
説明 .....	117
推奨する使用方法 .....	117
プロファイル情報のダンプとリセット .....	117
説明 .....	117
推奨する使用方法 .....	117
インターバル・プロファイル・ダンプ .....	117
説明 .....	117
推奨する使用方法 .....	118
環境変数 .....	118
高水準言語の最適化(HLO) .....	119
高レベル最適化の概要 .....	119
ループ変換 .....	119
ループのアンロール .....	120
ループのアンロールを有効にする方法 .....	120
ループのアンロールを無効にする方法 .....	120
IVDEP ディレクティブによるループ・キャリー・メモリ依存の不在 .....	120

並列化.....	122
並列化オプションの概要.....	122
OpenMP* による並列化.....	123
OpenMP* による並列化の概要.....	123
OpenMPによる並列処理.....	123
パフォーマンス分析.....	123
並列処理スレッドモデル.....	124
実行フロー.....	124
孤立ディレクティブの使用.....	124
データ環境ディレクティブ.....	125
並列処理モデルの疑似コード.....	125
OpenMP、ディレクティブ形式、および診断でのコンパイル.....	127
-openmp オプション.....	127
OpenMP ディレクティブ形式と構造.....	127
OpenMP 診断.....	127
OpenMP* ディレクティブと節.....	128
OpenMP ディレクティブ.....	128
OpenMP の節.....	128
OpenMP* のサポート・ライブラリ.....	129
実行モード.....	129
Serial.....	129
Turnaround.....	129
Throughput.....	130
OpenMP*の環境変数.....	130
標準環境変数.....	130
インテル拡張環境変数.....	130
OpenMP* ランタイム・ライブラリ・ルーチン.....	131
OpenMP* に追加されたインテル拡張機能.....	134
インテル拡張機能.....	134
スタックサイズ.....	134
メモリの割り当て.....	134
スタックサイズ.....	134
メモリの割り当て.....	135
ワークキューイング・モデル.....	136
インテルのワークキューイング・モデルの概要.....	136
ワークキューイング・コンストラクタ.....	136
taskq プラグマ.....	136
制御構造体.....	136
while ループ.....	136
C++ 反復子.....	136
再帰関数.....	137

taskq コンストラクタ .....	137
<b>private</b> .....	138
<b>firstprivate</b> .....	138
<b>lastprivate</b> .....	138
<b>reduction</b> .....	138
<b>ordered</b> .....	138
<b>nowait</b> .....	138
task コンストラクタ .....	139
<b>private</b> .....	139
<b>captureprivate</b> .....	139
parallel と taskq コンストラクタの組み合わせ .....	139
関数例 .....	140
OpenMP* の使用例 .....	140
単純な差分演算子 .....	140
2つの差分演算子 .....	141
自動並列化 .....	142
自動並列化の概要 .....	142
オリジナルの連続コード .....	142
変換された並列コード .....	142
自動並列化のプログラミング .....	143
効率的な自動並列化の使用法のガイドライン .....	143
コーディング・ガイドライン .....	143
自動並列化のデータフロー .....	143
自動並列化: 有効、オプション、および環境変数 .....	144
自動並列化オプション .....	144
自動並列化の環境変数 .....	144
自動並列化のしきい値と診断 .....	145
しきい値制御 .....	145
診断 .....	145
並列化診断レポートの例 .....	145
トラブルシューティングのヒント .....	146
ベクトル化 (IA-32 のみ) .....	147
ベクトル化の概要 .....	147
ベクトライザ・オプション .....	147
使用方法 .....	148
ループの並列化とベクトル化 .....	148
プログラミングの基本となるガイドライン .....	148
ループ本体に関するガイドライン .....	148
ループ本体内では避けたほうがよいもの .....	149
ベクトル化できるコードにする方法 .....	149
制約事項 .....	149



データ依存性.....	150
データ依存性の理論.....	150
ループの構成要素.....	151
ループ出口条件.....	152
ベクトル化ループの種類.....	153
ストリップ・マイニングとクリーンアップ.....	153
ループ本体内の文.....	154
浮動小数点配列の演算.....	154
整数配列の演算.....	154
その他の演算.....	154
言語サポートとディレクティブ.....	155
言語サポート.....	155
マルチ・バージョン・コード.....	155
プログラム・スコープ.....	155
動的依存性のテスト例.....	156
ベクトル化の具体例.....	156
引数のエイリアシング: ベクトルコピー.....	156
データのアライメント.....	157
16バイト境界にまたがる間違ってアライメントされたデータ.....	157
ベクトルおよびスカラのクリーンアップ反復処理.....	158
データのアライメント例.....	158
ループ交換と添字: マトリックス乗算.....	159
最適化サポート機能.....	161
最適化サポート機能の概要.....	161
コンパイラ・ディレクティブ.....	161
コンパイラ・ディレクティブ.....	161
Itanium® ベース・アプリケーションのパイプライン処理.....	161
swpディレクティブの例.....	161
ループカウントとループ分配.....	162
loop count (n)ディレクティブ.....	162
distribute pointディレクティブ.....	162
ループのアンロールのサポート.....	164
unrollディレクティブ.....	164
プリフェッチのサポート.....	164
prefetchディレクティブ.....	164
ベクトル化のサポート(IA-32).....	165
vector alwaysディレクティブ.....	165
ivdepディレクティブ.....	165
vector alignedディレクティブ.....	165
novectorディレクティブ.....	167
アプリケーションの時間測定.....	167

最適化機構レポートの作成.....	168
レポートを作成する最適化の指定.....	169
利用可能なレポート生成.....	170
ライブラリ.....	171
ライブラリの概要.....	171
GNU Cライブラリ.....	171
Dinkumware C++ライブラリ.....	171
デフォルト・ライブラリ.....	171
数値演算ライブラリ.....	172
インテル共用ライブラリ.....	172
この手法の利点.....	172
共用ライブラリ・オプション.....	173
ライブラリの管理.....	173
LD_LIBRARY_PATH の変更.....	173
インテル(R) 数値演算ライブラリ.....	175
インテル数値演算ライブラリの概要.....	175
IA-32 と Itanium® ベース・システムの数値演算ライブラリ.....	175
インテル数値演算ライブラリの使用.....	175
その他の考慮事項.....	177
数値演算関数.....	179
関数リスト.....	179
三角関数.....	181
ACOS.....	181
ACOSD.....	182
ASIN.....	182
ASIND.....	182
ATAN.....	182
ATAN2.....	182
ATAND.....	182
ATAND2.....	182
COS.....	183
COSD.....	183
COT.....	183
COTD.....	183
SIN.....	183
SINCOS.....	183
SINCOSD.....	184
SIND.....	184
TAN.....	184
TAND.....	184
双曲線関数.....	184

ACOSH .....	184
ASINH .....	184
ATANH.....	185
COSH .....	185
SINH .....	185
SINHCOSH.....	185
TANH.....	185
指数関数 .....	185
CBRT.....	185
EXP .....	186
EXP10 .....	186
EXP2.....	186
EXPM1 .....	186
FREXP .....	186
HYPOT.....	186
ILOGB .....	187
LDEXP .....	187
LOG .....	187
LOG10.....	187
LOG1P .....	187
LOG2.....	187
LOGB .....	187
POW.....	188
SCALB .....	188
SCALBN.....	188
SCALBLN .....	188
SQRT.....	188
特殊関数 .....	188
ANNUITY .....	188
COMPOUND.....	189
ERF .....	189
ERFC .....	189
GAMMA .....	189
GAMMA_R.....	189
J0.....	189
J1.....	190
JN.....	190
LGAMMA .....	190
LGAMMA_R.....	190
TGAMMA .....	190
Y0.....	190

Y1.....	190
YN.....	191
丸め関数 .....	191
CEIL .....	191
FLOOR.....	191
LRINT .....	191
LLRINT .....	191
LROUND .....	191
LLROUND.....	192
MODF .....	192
NEARBYINT.....	192
RINT .....	192
ROUND .....	192
TRUNC .....	192
剰余関数 .....	192
FMOD .....	193
REMAINDER.....	193
REMQUO .....	193
その他の関数.....	193
COPYSIGN.....	193
FABS.....	193
FDIM .....	193
FINITE .....	194
FMA .....	194
FMAX.....	194
FMIN .....	194
ISNAN .....	194
NEXTAFTER.....	194
NEXTTOWARD.....	195
複素数関数 .....	195
CABS .....	195
CACOS.....	195
CACOSH.....	195
CARG .....	195
CASIN .....	195
CASINH.....	196
CATAN .....	196
CATANH.....	196
CCOS .....	196
CCOSH.....	196
CEXP.....	196

CIMAG .....	196
CIS .....	196
CLOG .....	197
CONJ .....	197
CPOW.....	197
CPROJ .....	197
CREAL .....	197
CSIN .....	197
CSINH .....	197
CSQRT .....	197
CTAN.....	198
CTANH .....	198
診断およびメッセージ .....	199
診断およびメッセージの概要 .....	199
診断メッセージ .....	199
言語診断 .....	199
lintコメントを使用した警告メッセージの出力停止 .....	200
警告メッセージの出力停止方法とリマーク・メッセージの出力方法 .....	200
エラーの出力個数の制限 .....	201
リマーク・メッセージ .....	201
gcc との互換性 .....	202
gcc との互換性 .....	202
参照情報 .....	205
コンパイラの制限 .....	205
コンパイラの制限 .....	205
主要ファイル .....	207
IA-32 コンパイラの主要なファイルの概要 .....	207
/bin ファイル .....	207
/lib ファイル .....	207
Itanium® コンパイラの主要なファイル概要 .....	207
/bin ファイル .....	207
/lib ファイル .....	208
インテル C++ 組込み関数リファレンス .....	209
概要 .....	209
組込み関数の種類 .....	209
各インテル・プロセッサの組込み関数への対応 .....	209
組込み関数を使用するメリット .....	210
新しいレジスタ .....	210
MMX テクノロジ・レジスタ .....	210
ストリーミングSIMD拡張命令レジスタ .....	210
新しいデータ型 .....	211

新しいデータ型への対応 .....	212
_m64 データ型 .....	212
_m128 データ型 .....	212
新しいデータ型を使用する際のガイドライン .....	212
命名と使用する構文 .....	212
組込み関数の構文 .....	213
すべての IA プロセッサでサポートされる組込み関数 .....	215
すべての IA プロセッサでサポートされる組込み関数 .....	215
整数演算に関連する組込み関数 .....	215
浮動小数点に関連する組込み関数 .....	215
文字列とブロックのコピーに関連する組込み関数 .....	218
その他の組込み関数 .....	218
MMX(R) テクノロジーの組込み関数 .....	220
MMX® テクノロジーのサポート .....	220
EMMS 命令: 必要な理由 .....	220
MMX テクノロジー命令の実行後、EMMS命令でレジスタをリセットする理由 .....	220
EMMSを使用する際のガイドライン .....	221
MMX® テクノロジーの一般的な組込み関数 .....	221
MMX® テクノロジーのパックド算術演算組込み関数 .....	224
MMX® テクノロジーのシフト組込み関数 .....	226
MMX® テクノロジーの論理演算組込み関数 .....	228
MMX® テクノロジーの比較組込み関数 .....	228
MMX® テクノロジーの設定組込み関数 .....	229
Itanium® アーキテクチャ上での MMX® テクノロジー組込み関数の使用 .....	231
データ型 .....	232
ストリーミングSIMD拡張命令 .....	233
ストリーミングSIMD拡張命令のサポート .....	233
ストリーミングSIMD拡張命令を使用する浮動小数点演算組込み関数 .....	233
ストリーミングSIMD拡張命令の算術演算 .....	234
ストリーミングSIMD拡張命令の論理演算 .....	237
ストリーミング SIMD 拡張命令の比較操作 .....	238
ストリーミングSIMD拡張命令の変換操作 .....	244
ストリーミングSIMD拡張命令のロード操作 .....	246
ストリーミングSIMD拡張命令の設定操作 .....	247
ストリーミングSIMD拡張命令のストア操作 .....	248
ストリーミングSIMD拡張命令によるキャッシュ制御 .....	249
ストリーミングSIMD拡張命令を使用する整数演算組込み関数 .....	250
ストリーミングSIMD拡張命令を使用するメモリ操作と初期化操作 .....	252
ストリーミングSIMD拡張命令を使用するその他の組込み関数 .....	257
Itanium® アーキテクチャ上でのストリーミング SIMD 拡張命令の使用 .....	259
データ型 .....	259

互換性とパフォーマンス .....	260
マクロ関数 .....	261
ストリーミングSIMD拡張命令を使用してシャッフルを行うマクロ関数 .....	261
シャッフル関数のマクロ .....	261
シャッフル関数のマクロの元のワードと結果のワード .....	261
コントロール・レジスタを読み書きするマクロ関数 .....	261
_MM_EXCEPT_DIV_ZEROマクロによる例外状態の確認 .....	262
行列の転置を行うマクロ関数 .....	263
_MM_TRANSPOSE4_PSマクロによる行列の転置 .....	263
ストリーミングSIMD拡張命令2 .....	263
ストリーミングSIMD拡張命令2 の組込み関数の概要 .....	263
浮動小数点演算組込み関数 .....	264
ストリーミングSIMD拡張命令2 の浮動小数点算術演算 .....	264
ストリーミングSIMD拡張命令2 の論理演算 .....	266
ストリーミングSIMD拡張命令2 の比較操作 .....	267
ストリーミングSIMD拡張命令2 の変換操作 .....	272
浮動小数点のメモリ操作と初期化操作 .....	276
ストリーミングSIMD拡張命令2 の浮動小数点メモリ操作と初期化操作 .....	276
ストリーミングSIMD拡張命令2 のロード操作 .....	276
ストリーミングSIMD拡張命令2 の設定操作 .....	277
ストリーミングSIMD拡張命令2 のストア操作 .....	278
ストリーミングSIMD拡張命令2 のその他の操作 .....	279
整数演算組込み関数 .....	280
ストリーミングSIMD拡張命令2 の整数算術演算 .....	280
ストリーミングSIMD拡張命令2 の整数論理演算 .....	286
ストリーミングSIMD拡張命令2 の整数シフト操作 .....	286
ストリーミングSIMD拡張命令2 の整数比較操作 .....	290
ストリーミングSIMD拡張命令2 の変換操作 .....	292
シャッフルを行うマクロ関数 .....	293
シャッフル関数のマクロ .....	293
シャッフル関数のマクロの元のワードと結果のワード .....	293
ストリーミングSIMD拡張命令2 のキャッシュ操作 .....	293
PAUSE組込み関数 .....	294
ストリーミングSIMD拡張命令2 のその他の操作 .....	295
整数のメモリ操作と初期化操作 .....	299
ストリーミングSIMD拡張命令2 の整数メモリ操作と初期化操作 .....	299
ストリーミングSIMD拡張命令2 の整数ロード演算 .....	299
ストリーミングSIMD拡張命令2 の整数設定操作 .....	299
ストリーミングSIMD拡張命令2 の整数ストア操作 .....	301
Itanium(R) 命令の組込み関数 .....	303
Itanium® 命令の組込み関数の概要 .....	303

Itanium® 命令のネイティブ組込み関数 .....	303
整数演算 .....	303
FSR演算 .....	304
ロックおよびアトミック操作に関連する組込み関数 .....	306
ロードとストア .....	309
Itanium® ベース・システムのオペレーティング・システムに関連する組込み関数 .....	310
Itanium® ベース・システム用の変換組込み関数 .....	313
getReg() および setReg() のレジスタ名 .....	313
一般的な整数レジスタ .....	314
アプリケーション・レジスタ .....	314
コントロール・レジスタ .....	315
getIndReg() および setIndReg() の間接レジスタ .....	315
Itanium® ベース・システム用のマルチメディア乗算 .....	316
データのアライメント、メモリの割り当て組込み関数、およびインライン・アセンブリ .....	323
データのアライメント、メモリの割り当て組込み関数、およびインライン・アセンブリの概 要 .....	323
アライメントのサポート .....	323
アライメントの合ったメモリブロックの割り当てと解放 .....	324
インライン・アセンブリ .....	325
MASM* スタイルのインライン・アセンブリ .....	325
GNU* 式スタイルのインライン・アセンブリ .....	325
各種のプロセッサでの組込み関数の使用 .....	328
各種のプロセッサでの組込み関数の使用 .....	328
すべての IA プロセッサでサポートされる組込み関数 .....	328
MMX® テクノロジーの組込み関数 .....	332
ストリーミングSIMD拡張命令の組込み関数 .....	335
ストリーミングSIMD拡張命令2 の組込み関数 .....	340
インテル(R) C++ クラス・ライブラリ .....	349
クラス・ライブラリの紹介 .....	349
はじめに .....	349
ハードウェアとソフトウェアの要件 .....	349
クラス・ライブラリを使用するプロセッサの必要条件 .....	349
クラスについて .....	349
技術的な概要 .....	351
ライブラリの詳細 .....	351
インライン・アセンブリ、組込み関数、クラス・ライブラリの比較 .....	351
C++ クラスとSIMD演算 .....	351
ループを使用して複数の要素を加算するときの一般的な方法 .....	352
Ivecクラスを使用して複数の要素を加算するSIMD手法 .....	352
使用できるクラス .....	352
SIMDベクトルクラス .....	352



ヘッダファイルを使用したクラスへのアクセス.....	354
クラスを有効にするためのインクルード・ディレクティブ .....	354
使用上の注意事項 .....	354
MMX テクノロジ・レジスタの消去.....	354
EMMS命令のガイドラインに従う.....	355
機能.....	355
計算 .....	355
水平データのサポート.....	355
分岐の圧縮/削除 .....	356
キャッシング・ヒント.....	356
整数ベクトルクラス.....	357
整数ベクトルクラスの概要 .....	357
Ivecクラスの派生関係図.....	357
用語、規則、および構文 .....	358
Ivecクラスの構文規則.....	358
特殊な用語と規則.....	358
クラス名の表記法.....	359
演算子の規則 .....	359
主要演算子の規則一覧.....	359
データの宣言と初期化 .....	360
Ivecクラスのデータ型の宣言と初期化.....	360
代入演算子 .....	361
代入演算子の例.....	361
論理演算子 .....	361
論理演算子と例外.....	362
Ivec論理演算子の多重定義 .....	362
代入付きIvec論理演算子の多重定義.....	363
加算演算子と減算演算子.....	363
加算演算子および減算演算子の構文の使用 .....	363
加算演算子、減算演算子、およびそれらに対応する組込み関数 .....	364
加算演算子と減算演算子の多重定義.....	364
乗算演算子の記号と対応する組込み関数.....	365
乗算演算子 .....	365
乗算演算子の構文の使用 .....	365
対応する組込み関数と乗算演算子.....	365
乗算演算子の多重定義.....	366
代入付き乗算.....	366
シフト演算子.....	366
シフト演算子の構文の使用例.....	366
シフト演算子と対応する組込み関数.....	367
シフト演算子の多重定義.....	367

比較演算子 .....	367
比較演算子の構文の使用例 .....	368
比較演算子と対応する組込み関数 .....	368
比較演算子の多重定義 .....	369
条件付き選択演算子 .....	369
条件付き選択演算子の構文の使用例 .....	369
条件付き選択演算子と対応する組込み関数 .....	369
条件付き選択演算子の多重定義 .....	370
条件付き選択演算子の戻り値対応表 .....	371
デバッグ .....	371
出力 .....	371
要素アクセス演算子 .....	372
要素代入演算子 .....	373
アンパック演算子 .....	373
パック演算子 .....	378
MMX® テクノロジ・ステート消去演算子 .....	379
ストリーミングSIMD拡張命令用の整数組込み関数 .....	379
変換(Fvec $\longleftrightarrow$ Ivec) .....	381
浮動小数点ベクトルクラス .....	383
浮動小数点ベクトルクラスの概要 .....	383
単精度浮動小数点の諸要素 .....	383
Fvecの表記法 .....	383
Fvecクラスの構文の表記法 .....	383
<b>戻り値の表記法</b> .....	384
戻り値表記の対応表 .....	384
データのアライメント .....	384
変換 .....	384
コンストラクタと初期化 .....	384
Fvecクラスのコンストラクタと初期化 .....	385
算術演算子 .....	386
Fvec算術演算子 .....	386
「標準算術演算子」の使用 .....	386
標準算術演算の戻り値の対応表 .....	387
代入付き算術演算の戻り値の対応表 .....	387
Fvecクラスの標準算術演算 .....	387
「高度な算術演算子」の使用 .....	388
高度な算術演算子の戻り値の対応表 .....	388
Fvecクラス用の高度な算術演算 .....	389
最小値演算子と最大値演算子 .....	390
論理演算子 .....	391
Fvec論理演算子の戻り値の対応表 .....	391

Fvecクラスの論理演算 .....	392
比較演算子 .....	393
比較演算子と対応する組込み関数 .....	393
比較演算子 .....	393
比較演算子の戻り値の対応表 .....	393
Fvecクラスの比較演算 .....	394
Fvecクラスの条件付き選択演算子 .....	396
Fvecクラスの条件付き選択演算子 .....	396
条件付き選択演算子の使用 .....	396
比較演算子の戻り値の対応表 .....	396
Fvecクラスの条件付き選択演算 .....	397
キャッシュ操作 .....	399
デバッグ .....	399
出力演算 .....	399
要素アクセス演算 .....	400
要素代入演算 .....	400
ロード演算子とストア演算子 .....	401
Fvec演算子のアンパック演算子 .....	401
マスク移動演算子 .....	402
各種クラスのクイック・リファレンス .....	402
論理演算子: 対応する組込み関数とクラス .....	402
算術演算子: 対応する組込み関数とクラス .....	402
シフト演算子: 対応する組込み関数とクラス .....	403
比較演算子: 対応する組込み関数とクラス .....	404
条件付き選択演算子: 対応する組込み関数とクラス .....	405
パック演算子とアンパック演算子: 対応する組込み関数とクラス .....	408
変換演算子: 対応する組込み関数とクラス .....	408
プログラミング例 .....	409
浮動小数点演算の精度の制限 .....	410
インクルード・ファイルの検索 .....	411
インクルード・ディレクトリの指定方法 .....	411
インクルード・ディレクトリの削除方法 .....	412
ハイパー・スレッディング・テクノロジー .....	412
索引 .....	414

# 免責条項

## 【輸出規制に関する告知と注意事項】

本資料に掲載されている製品のうち、外国為替および外国為替管理法に定める戦略物資等または役務に該当するものについては、輸出または再輸出する場合、同法に基づく日本政府の輸出許可が必要です。また、米国産品である当社製品は日本からの輸出または再輸出に際し、原則として米国政府の事前許可が必要です。

## 【資料内容に関する注意事項】

- 本ドキュメントの内容を予告なしに変更することがあります。
- インテルでは、この資料に掲載された内容について、市販製品に使用した場合の保証あるいは特別な目的に合うことの保証等は、いかなる場合についてもいたしかねます。また、このドキュメント内の誤りについても責任を負いかねる場合があります。
- インテルでは、インテル製品の内部回路以外の使用にて責任を負いません。また、外部回路の特許についても関知いたしません。
- 本書の情報はインテル製品を使用できるようにする目的でのみ記載されています。

インテルは、製品について「取引条件」で提示されている場合を除き、インテル製品の販売や使用に関して、いかなる特許または著作権の侵害をも含み、あらゆる責任を負わないものとします。

- いかなる形および方法によっても、インテルの文書による許可なく、この資料の一部またはすべてを複写することは禁じられています。

Copyright © Intel Corporation 1996-2002.

Intel、インテル、Celeron、Dialogic、iCOMP、Itanium、MMX、Pentium、i386、i486、Intel386、intel486、Intel740、IntelDX2、IntelDX4、IntelSX2、NetBurst、NetStructure、Xeon、XScale、VTune は、アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

\* その他の名称およびブランド名は、各社の商標および登録商標です。

# インテル(R) C++ コンパイラについて

## インテル® C++ コンパイラへようこそ

インテル® C++ コンパイラへようこそコンパイラを使用する前に、「システムの要件」を参照してください。

ほとんどの Linux ディストリビューションには、GNU\* C ライブラリ、アセンブラ、リンカなどが含まれています。インテル C++ コンパイラには Dinkumware\* C++ ライブラリが含まれています。「ライブラリの概要」を参照してください。

それぞれの内容は、このユーザーズ・ガイドの該当する各項目をご覧ください。最新の情報については、インテル Web サイト <http://www.intel.co.jp/jp/developer/> にアクセスしてください。

## 本リリースの新機能

- 新しいコンパイラのオプション `--tpp1` と `-tpp2` オプションを使用する Itanium® プロセッサおよび Itanium 2 プロセッサのサポート。
- 向上した gcc の互換性
- 拡張最適化ディレクティブ
- OpenMP\* のサポート



注

Itanium アーキテクチャ用インテル C++ ネイティブ・コンパイラは、本リリースに含まれています。Itanium アーキテクチャ用インテル C++ クロス・コンパイラは、本リリースには含まれていません。

## 特徴と利点

インテル® C++ コンパイラを使用すると、インテル・アーキテクチャ・ベースのコンピュータ上でソフトウェアの最高のパフォーマンスを引き出せます。インテル C++ コンパイラは、プログラム全体の最適化やプロファイルに基づく最適化などの新しいコンパイラ最適化、プリフェッチ命令、ストリーミングSIMD拡張命令(SSE)、およびストリーミングSIMD拡張命令 2 (SSE2) のサポートの利用により、高いパフォーマンスを提供します。

特徴	利点
高いパフォーマンス	最適化を使用して、高いパフォーマンスを実現
ストリーミングSIMD拡張命	新しいインテル・マイクロアーキテクチャの利点

令のサポート	
自動ベクトライザ	コード内のSIMD自動並列処理の利点
OpenMP* のサポート	共用メモリ並列プログラミング
浮動小数点の最適化	浮動小数点のパフォーマンスが向上
データ・プリフェッチ機能	データ送信の高速化によりパフォーマンスが向上
プロシージャ間の最適化	大規模アプリケーションのモジュールのパフォーマンスが向上
プロファイルに基づく最適化	頻繁に使用されるプロシージャのプロファイリングに基づくパフォーマンスの向上
プロセッサ・ディスパッチ	最新のインテル・アーキテクチャの機能を利用すると同時に、前世代のインテル Pentium® プロセッサとのオブジェクト・コードの互換性を確保 (IA-32 ベース・システム専用)

## 製品情報 Web サイトとサポート

インテル® C++ コンパイラに関する最新情報は、インテル Web サイト <http://www.intel.co.jp/jp/developer/software/products/> にアクセスしてください。  
 Itanium® アーキテクチャに関する具体的で詳しい内容は、インテル Web サイト [http://www.intel.co.jp/jp/developer/design/itanium/under\\_lnx.htm](http://www.intel.co.jp/jp/developer/design/itanium/under_lnx.htm) にアクセスしてください。

## システムの要件

### IA-32 プロセッサのシステムの要件

- Pentium® プロセッサ、または IA-32 ベースのプロセッサを搭載したコンピュータ (Pentium 4 プロセッサを推奨)。
- 128 MBのRAM (256 MB を推奨)。
- 100 MB のディスク容量。

### Itanium® プロセッサのシステムの要件

- Itanium プロセッサを搭載するコンピュータ
- 256 MB のディスク容量。
- 100 MB のディスク容量。

## ソフトウェアの要件

リリースノートを参照してください。

## FLEXlm\* 電子ライセンス

インテル® C++ コンパイラは、GlobeTrotter 社の FLEXlm\* というライセンス技術を使用しています。コンパイラには、インストール・パスにある `/licenses` ディレクトリ内の有効なライセンス・ファイルが必要です。デフォルトのディレクトリは、`/opt/intel/licenses` です。ライセンス・ファイルは `.lic` のファイル拡張子を持ちます。

ライセンスが必要な場合、『*Using the Intel® License Manager for FLEXlm\**』を参照してください。(flex\_ug.pdf).

## 本書について

### 本書の使い方

このユーザーズ・ガイドでは、インテル® C++ コンパイラの使用方法を説明しています。C++ コンパイラの使用方法を説明します。読者はアプリケーションのパフォーマンスを向上させるための標準的および先進的なコンパイラ最適化手法の使用法を学習できます。

標準的な最適化手法と高度な最適化手法を学べば、アプリケーションのパフォーマンスが最大まで引き出せます。また、ホスト・コンピュータのオペレーティング・システムにも精通している必要があります。



注

本書では、対象となる各アーキテクチャごとに情報や命令がどのように適用されるかを説明しています。どのアーキテクチャかを明示していない場合、説明は両方のアーキテクチャに適用されます。

どのアーキテクチャかを明示していない場合、説明は両方のアーキテクチャに適用されます。

規則

<code>This type style</code>	This type style 構文要素、予約語、キーワード、ファイル名、コンピュータ出力、プログラム例の一部分のいずれかを表します。
<b><code>This type style</code></b>	This type style
<i><code>This type style</code></i>	This type style コマンドラインの引数またはオプションの引数を表します。
<code>[ item ]</code>	角括弧で囲まれたアイテムはオプションです。
<code>{ item1   item2   ... }</code>	中括弧内のitemの中から1つだけを選択します。縦線(   )はアイテムの区切りです。 -ax{ i   M   K   M } のようにオプションによっては、複数のitemを使用できます。
<code>... (省略記号)</code>	省略符号は、直前のアイテムを複数指定できることを表します。

### 組み込み関数名の表記

ほとんどの組み込み関数名は、次の表記規則に従います。

`_mm_<intrin_op>_<suffix>`

<code>&lt;intrin_op&gt;</code>	組み込み関数の基本操作を示します。例えば、加算の場合はadd、減算の場合はsubになります。
<code>&lt;suffix&gt;</code>	命令の操作対象となるデータの型を示します。各サフィックスの最初の1文字または2文字は、



	<p>データがパックドデータ(p)、拡張パックドデータ(ep)、またはスカラデータ(s)であることを示します。その他の文字は、次のとおりデータ型を示します。</p> <ul style="list-style-type: none"> <li>• <code>__s</code> 単精度浮動小数点値</li> <li>• <code>__d</code> 倍精度浮動小数点値</li> <li>• <code>__i128</code> 符号付き128ビット整数</li> <li>• <code>__i64</code> 符号付き64ビット整数</li> <li>• <code>__u64</code> 符号なし64ビット整数</li> <li>• <code>__i32</code> 符号付き32ビット整数</li> <li>• <code>__u32</code> 符号なし32ビット整数</li> <li>• <code>__i16</code> 符号付き16ビット整数</li> <li>• <code>__u16</code> 符号なし16ビット整数</li> <li>• <code>__i8</code> 符号付き8ビット整数</li> <li>• <code>__u8</code> 符号なし8ビット整数</li> </ul>
--	---

変数名を付加した数字は、パックされたオブジェクトの要素を示します。例えば、`r0l`は`r`の最下位ワードです。一部の組込み関数は、2つ以上の命令で実行するため、「複合組込み関数」と呼ばれます。

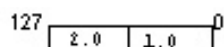
パックされた値は、右から左の順序で表現し、最下位の値がスカラ操作に使用されます。次の操作の例について考えます。

```
double a[2] = {1.0, 2.0}; __m128d t = _mm_load_pd(a);
```

上の操作の結果は、次のそれぞれの操作と同じになります。

```
__m128d t = _mm_set_pd(2.0, 1.0); __m128d t = _mm_setr_pd(1.0, 2.0);
```

つまり、値`t`を保持するxmmレジスタは、次のようになります。



「スカラ」要素は`1.0`です。一部の組込み関数では、命令の性質上、引数として即値(定数整数リテラル)を指定しなければなりません。

## クラス・ライブラリ名の表記

クラス名は、データ型、符号の有無、ビットサイズ、要素数を表現したものです。一般的な形式で表すと次のようになります。

```
<type><signedness><bits>vec<elements>
{ F | I } { s | u } { 64 | 32 | 16 | 8 } vec { 8 | 4 | 2 | 1 }
```

各アイテムの意味は次のとおりです。

<code>&lt;type&gt;</code>	浮動小数点(F)または整数(I)を示します。
---------------------------	------------------------

<signedness>	符号付き(s)または符号なし(u)を示します。Ivecクラスの場合は、このフィールドが空のままだと中間クラスを表します。符号なしのFvecクラスはないため、Fvecクラスの場合、このフィールドは空です。
<bits>	要素あたりのビット数です。
<elements>	要素の個数です。

## 参考文献

インテル® C++ コンパイラの関連情報については、次の資料を参照してください。

- 『ISO/IEC 9989:1990, Programming Languages--C 』
- 『ISO/IEC 14882:1998, Programming Languages--C++ 』
- 『*The Annotated C++ Reference Manual*』第3版。Ellis Margaret、Stroustrup Bjarne著、Addison Wesley刊。1991年。C++ プログラミング言語の解説
- 『*The C++ Programming Language*』第3版。1997年。Addison-Wesley Publishing Company(One Jacob Way, Reading, MA 01867)刊。
- 『*The C Programming Language*』第2版。Kernighan Brian W、Ritchie Dennis W著。Prentice Hall刊。1988年。C 言語のK & R定義の解説。
- 『*C: A Reference Manual*』第3版。Harbison Samuel P、Steele Guy L著。Prentice Hall刊。1991年。C 言語の ANSI 標準と拡張機能の解説。
- 『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、上巻: 基本アーキテクチャ』インテル、資料番号245470J
- 『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻: 命令セット・リファレンス』インテル、資料番号245471J
- 『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、下巻: システム・プログラミング・ガイド』インテル、資料番号245472J
- 『インテル® Itanium® ベース・アセンブラ・ユーザーズ・ガイド』
- 『インテル® Itanium® アーキテクチャ・アセンブリ言語リファレンス・マニュアル』
- 『インテル® Itanium® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、第1巻: アプリケーション・アーキテクチャ』インテル、資料番号245317J-001
- 『インテル® Itanium® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、第2巻: システム・アーキテクチャ』インテル、資料番号245318J-001
- 『インテル® Itanium® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、第3巻: 命令セット・リファレンス』インテル、資料番号245319J-001
- 『インテル® Itanium® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、第4巻: Itanium® プロセッサ・プログラマーズ・ガイド』インテル、資料番号245319J-001
- 『インテル® アーキテクチャ最適化リファレンス・マニュアル』インテル、資料番号245127J
- 『インテル® プロセッサの識別とCUID命令』インテル、資料番号241618J

- 『Intel® Architecture MMX Technology Programmer's Reference Manual 』インテル、資料番号241618
- 『Pentium® Pro Processor Developer's Manual 』(3巻セット)、インテル、資料番号242693
- 『Pentium® II Processor Developer's Manual 』インテル、資料番号243502-001
- 『Pentium® Processor Specification Update 』インテル、資料番号242480
- 『Pentium® Processor Family Developer's Manual 』インテル、資料番号241428-005

インテルのほとんどの資料は、インテル Web サイト(<http://www.intel.co.jp>)から入手できます。

# コンパイラ・オプションのクイック・リファレンス・ガイド

## オプション・クイック・リファレンス・ガイドの概要

### オプションのクイック・ガイドの表で使用する表記

表記	定義
[ - ]	オプションに "[ - ]" が含まれると、このオプションを使用して機能を有効または無効にすることができます。例えば、 <code>-cpp [ - ]</code> オプションは <code>-c99</code> (c99 サポートを有効にする) または <code>-c99-</code> (c99 サポートを無効にする) として使用することができます。
[ n ]	[ ] 内の n の値は省略可能で、任意の値を指定できることを示します。
{ } の縦バーの値	オプションの選択肢に使用されます。例えば、オプション <code>-i { 2   4   8 }</code> には次の選択肢があります: <code>-i2</code> 、 <code>-i4</code> 、 <code>-i8</code>
{ n }	オプションで、n に固定値の1つを指定する必要があることを示します。
オプションに続く"このスタイル"の文字	オプションに必要な引数を表します。複数の引数が必要な場合は、カンマで区切ります。

## 新しいオプション

- IA-32 アーキテクチャ固有のオプション
- Itanium® アーキテクチャ固有のオプション(Itanium ベース・システムのみ)

他のすべてのオプションは、IA-32およびItanium ベース・システムの両方に対応しています。

オプション	説明	デフォルト
<code>-dM</code>	前処理を行った後に有効なマクロ定義を出力します ( <code>-E</code> と使用)。	オフ
<code>-dynamic-linker filename</code>	デフォルト以外の動的リンカ(filename)を選択します。	オフ
<code>-fno-rtti</code>	RTTI のサポートを無効にします。	オフ
<code>-fnsplit [ - ]</code> Itanium ベース・システムのみ	関数分割を有効にします[無効にします]。デフォルトではオンです( <code>-prof_use</code> とともに用いる)。 <code>-prof use</code> の使用中に関数分割を	オフ

	いる)。-prof_useの使用中に関数分割を無効にするには、-fnsplit-も指定してください。	
-fshort-enums	列挙型に必要なバイトを割り当てます。	オフ
-fsyntax-only	-syntax と同じです。	オフ
-funsigned-char	デフォルトの char 型を符号なしに変更します。	オフ
-funsigned-bitfields	デフォルトの bitfield 型を符号なしに変更します。	オフ
-idirafterdir	2番目のインクルード・ファイルの検索先(-lの後)にディレクトリ(dir)を追加します。	オフ
-march=cpu IA-32 のみ	指定した cpu 用に排他的にコードを生成します。cpu の値は以下のようになります。 <ul style="list-style-type: none"> <li>pentiumpro - Pentium® Pro プロセッサ、Pentium II プロセッサの命令</li> <li>pentiumii - MMX® テクノロジーの命令</li> <li>pentiumiii - ストリーミング SIMD拡張命令</li> </ul> pentium4 - Pentium 4 の命令(デフォルト)	オフ
-mcpu=cpu	特定の cpu 用に最適化します。cpu の値は以下のようになります。 <ul style="list-style-type: none"> <li>pentium - Pentium プロセッサに合わせて最適化します。</li> <li>pentiumpro - Pentium Pro プロセッサ、Pentium II プロセッサ、Pentium III プロセッサの各プロセッサに合わせて最適化します。</li> <li>pentium4 - Pentium 4 プロセッサに合わせて最適化します。</li> </ul> Itanium ベース・システムの場合、cpu の値は次のようになります。 <ul style="list-style-type: none"> <li>itanium - Itanium プロセッサに合わせて最適化します。</li> </ul> itanium2 - Itanium 2 プロセッサに合わせて最適化します。(デフォルト)	オン IA-32 の Pentium Itanium ベース・システムの itanium2
-MD	前処理およびコンパイルを行います。依存情	オフ

	報が含まれている出力ファイル(.d 拡張子)を生成します。	
-MF <i>file</i>	makefileの依存情報を <i>file</i> に生成します。 -Mまたは-MMを指定する必要があります。	オフ
-MG	-Mと類似していますが、見つからないヘッダファイルを、生成したファイルとして処理します。	オフ
-MM	-Mと類似していますが、システム・ヘッダ・ファイルをインクルードしません。	オフ
-MMD	-Dと類似していますが、システム・ヘッダ・ファイルをインクルードしません。	オフ
-MX	インテルwbツールで使用される情報を含む依存ファイル(.o.dep拡張子)を生成します。	オフ
-mrelax	リンカに -relax を渡します。	オン
-mno-relax	リンカに -relax を渡しません。	オフ
-mserialize-volatile Itanium ベース・システムのみ	volatileデータ・オブジェクトの参照に対して、制限されたメモリアクセスの順序を適用します。	オフ
-mno-serialize-volatile Itanium ベース・システムのみ	コンパイラは、volatileデータ・オブジェクトの参照に対して、ランタイム時およびコンパイル時のメモリアクセスの順序を抑制する場合があります。特に、.rel/.acqコンプリータはロードおよびストアの参照で実行されません。	オフ
-nodefaultlibs	リンク時に標準ライブラリを使用しません。	オフ
-Obn	コンパイラによるインライン展開を制御します。どの程度までインライン展開が行われるかは、次のようにnの値によって異なります。 <ul style="list-style-type: none"> <li>0: インライン化を無効にします。</li> <li>1: <code>__inline</code> キーワードで宣言された関数のインライン化を有効にします(デフォルト)。C++ 言語に従ったインライン化も有効にします。</li> </ul> 2: どの関数についてもインライン化を有効にします。ただし、どの関数をインライン化するかはコンパイラが決めます。プロシージャ間の最適化が有効になり、-ip と同じ効果が得られます。	オン
-openmp_stubs	シーケンシャル・モードでOpenMP* プログラムのコンパイルを有効にします。openmp ディ	オフ

	レクティブは無視され、スタブOpenMPライブラリがシーケンシャルにリンクされます。	
-std=c99 IA-32 のみ	C プログラムのC99 サポートを有効にします。	オン
-tpp1 Itanium ベース・システムのみ	Itanium プロセッサ向けに最適化します。	オフ
-tpp2 Itanium ベース・システムのみ	Itanium 2 プロセッサ向けに最適化します。生成されたコードは、Itanium プロセッサと互換性があります。	オン
-v	ドライバ・ツール・コマンドを表示し、ツールを実行します。	オフ
-Wall	すべての警告メッセージを表示します。	オフ
-Werror	警告をエラーと見なします。	オフ

## アルファベット順のリスト

### コンパイラ・オプションのクイック・リファレンス・ガイド

ここでは、コンパイルのオプションすべてと、リンカを制御するオプションの一部を紹介します。

- IA-32 アーキテクチャ固有のオプション
- Itanium® アーキテクチャ固有のオプション

他のすべてのオプションは、IA-32 および Itanium ベース・システムの両方に対応しています。

オプション	説明	デフォルト
-Of_check IA-32 のみ	古いプロセッサ向けに記述されたコードに何らかのOf命令が使用されていても、そのデコード処理を誤らないように処置します。 <a href="#">詳細...</a>	オフ
-A-	事前定義済みマクロ <a href="#">詳細...</a>	オフ
-[no]align IA-32 のみ	変数と配列のメモリ・レイアウトを分析し、構成を変更します。	オフ
-Aname[(value)]	nameというシンボルを、指定した一連のvalueに関連付けます。#assert という前処理ディレクティブと同じです。 <a href="#">詳細...</a>	オフ
-ansi	ANSI C/C++規格に厳密に準拠しているダイアレクトを選択します。	オフ
-ansi_alias[-]	-ansi_aliasによってコンパイラは、次の点を前提にします。 <ul style="list-style-type: none"><li>• 配列が外部からアクセスされません。</li><li>• ポインタを非ポインタ型にキャストしません。また、その逆もしません。</li><li>• 2つの異なるスカラ型のオブジェクトへの参照はエイリアス不可です。例えば、int型のオブジェクトはfloat型のオブジェクトと、またはfloat型のオブジェクトはdouble型のオブジェクトとエイリアスの設定はできません。</li></ul> プログラムが上記の条件を満たす場合、-ansi_aliasフラグを設定することでプログラムの最適化が向上します。ただし、プログラ	オフ



	ムが上記の条件を1つでも満たさない場合、コンパイラは-ansi_aliasフラグを設定することで誤ったコードを生成する可能性があります。	
-ax{ i   M   K   W } IA-32 のみ	<p>指定したプロセッサ向けに専用のコード(i、M、K、W)を生成し、かつ汎用性のある IA-32 コードも生成します。</p> <ul style="list-style-type: none"> <li>• i = Pentium® Pro プロセッサ、Pentium II プロセッサの命令</li> <li>• M = MMX® テクノロジーの命令</li> <li>• K = ストリーミング SIMD 拡張命令対応 Pentium III プロセッサ</li> <li>• W = ストリーミング SIMD 拡張命令 2 対応 Pentium 4 プロセッサ</li> </ul> <p><a href="#">詳細...</a></p>	オフ
-C	<p>前処理済みのソース出力の中にコメントを書き込みます。</p> <p><a href="#">詳細...</a></p>	オフ
-c	<p>オブジェクト・ファイルが生成された後、コンパイルの処理を止めます。C/C++ のソースファイルまたは前処理済みのソースファイルのそれぞれについてコンパイラがオブジェクト・ファイルを生成します。同様に、アセンブリ・ファイルについてもアセンブラがオブジェクト・ファイルを生成します。</p> <p><a href="#">詳細...</a></p>	オフ
-c99 [-]	<p>C プログラムの C99 サポートを有効 [無効] にします。</p> <p><a href="#">詳細...</a></p>	オン
-dM	<p>前処理を行った後に有効なマクロ定義を出力します (-Eと使用)。</p>	オフ
-Dname [=value]	<p>マクロ名(name)を定義し、そのマクロ名と指定された値(value)を関連付けます。 #define 前処理ディレクティブと同じ働きをします。</p> <p><a href="#">詳細...</a></p>	オフ
-dryrun	<p>ドライバ・ツール・コマンドを表示し、ツールを実行しません。</p>	オフ
-dynamic-linker filename	<p>デフォルト以外の動的リンカ(filename)を選択します。</p>	オフ

-E	C/C++ のソースファイルの前処理の後、コンパイルの処理を止め、その結果をstdoutに書き込みます。 <a href="#">詳細...</a>	オフ
-EP	#lineディレクティブについては省いて、前処理の結果を標準出力に出力します。 <a href="#">詳細...</a>	オフ
-falias	プログラムでエイリアシングを前提に処理します。	オン
-fcode-asm	オプションのコードのコメント付きアセンブリ・ファイルを生成します。-Sが必要です。	オフ
-fno-alias	プログラムでエイリアシングしないことを前提に処理します。	オフ
-ffnalias	関数内でのエイリアシングを前提に処理します。	オン
-fno-fnalias	関数内でエイリアシングしないことを前提に処理しますが、複数の呼び出しにわたる場合はエイリアシングを前提に処理します。	オフ
-fno-rtti	RTTI サポートを無効にします。	オフ
-f[no]verbose-asm	コンパイラのコメント付きアセンブリ・ファイルを生成します。	オン
-fnsplit[-] Itanium ベース・システムのみ	関数分割を有効にします[無効にします]。デフォルトではオンです(-prof_useとともに用いる)。-prof_useを使用する際に関数分割を無効にするには、-fnsplit-も指定してください。	オフ
-fp IA-32 のみ	汎用レジスタとしての EBP レジスタの使用を無効にします。 <a href="#">詳細...</a>	オフ
-fpic, -fPIC	位置に依存しないコードを生成します。	オフ
-fp_port IA-32 のみ	代入 & 型キャスト時に浮動小数点の結果を丸めます。速度に多少影響します。	オフ
-fr32 Itanium ベース・システムのみ	浮動小数点レジスタのうち下位32個だけを使用します。	オフ
-fshort-enums	列挙型に必要なバイトを割り当てます。	オフ
-fsource-asm	オプションのコードのコメント付きアセンブリ・ファイルを生成します。-Sが必要です。	オフ
-fsyntax-only	-syntax と同じです。	オフ
-ftz[-] Itanium ベース・システム	異常値の結果を0にフラッシュします。 <a href="#">詳細...</a>	オフ

のみ		
-funsigned-char	デフォルトの char 型を unsigned に変更します。	オフ
-funsigned-bitfields	デフォルトのビットフィールド型を unsigned に変更します。	オフ
-g	オブジェクト・コードの中にシンボリック・デバッグ情報を生成します。ソースレベルでのデバグがこの情報を使用します。 <a href="#">詳細...</a>	オフ
-H	インクルード・ファイルの順番を出力して、コンパイルを継続します。	オフ
-help	各種コンパイラ・オプションの一覧を出力します。	オフ
-idirafterdir	2番目のインクルード・ファイルの検索先にディレクトリ( <i>dir</i> )を追加します(-I の後)。	オフ
-Idirectory	インクルード・ファイルを探すときの、標準以外の <i>directory</i> を指定します。 <a href="#">詳細...</a>	オフ
-i_dynamic	インテル提供のライブラリを動的にリンクします。	オフ
-inline_debug_info	コールサイトのソース・ポジションをインライン化コードに割り当てる代わりに、インライン化コードのソース・ポジションを保存します。	オフ
-ip	ファイルを1個コンパイルするときにプロシーダ間の最適化を有効にします。 <a href="#">詳細...</a>	オフ
-IPF_fma[-] Itanium ベース・システムのみ	浮動小数点乗算と加算/減算の組み合わせを有効[無効]にします。 <a href="#">詳細...</a>	オフ
-IPF_fltacc[-] Itanium ベース・システムのみ	浮動小数点の精度に影響する最適化を有効[無効]にします。 <a href="#">詳細...</a>	オフ
-IPF_flt_eval_method0 Itanium ベース・システムのみ	プログラムにより指定された精度で浮動小数点オペランドが評価されます。 <a href="#">詳細...</a>	オフ
-IPF_fp_speculationmode Itanium ベース・システムのみ	次のモード( <i>mode</i> )条件で浮動小数点の推測を有効にします。 <ul style="list-style-type: none"> <li>fast - 浮動小数点演算を推測します</li> <li>safe - 安全な場合のみ推測します</li> </ul>	オフ

	<ul style="list-style-type: none"> <li>• <code>strict - off</code> と同じです</li> <li>• <code>off</code> - 浮動小数点演算の推測を無効にします</li> </ul> <a href="#">詳細...</a>	
<code>-ip_no_inlining</code>	<code>-ip</code> を指定してプロシージャ間の最適化が行われるときに、インライン化については無効にします。しかし、他のプロシージャ間の最適化には何の影響も与えません。 <a href="#">詳細...</a>	オフ
<code>-ip_no_pinlining</code> IA-32 のみ	部分的なインライン展開を無効にします。 <code>-ip</code> 、 <code>-ipo</code> のいずれかが必要です。	オフ
<code>-ipo</code>	複数ファイルにわたるプロシージャ間の最適化を有効にします。 <a href="#">詳細...</a>	オフ
<code>-ipo_c</code>	あとのリンク段階で利用できるマルチファイル・オブジェクト・ファイル( <code>ipo_out.o</code> )を生成します。 <a href="#">詳細...</a>	オフ
<code>-ipo_obj</code>	<code>-ipo</code> とともに指定すると、実際のオブジェクト・ファイルが強制的に生成されます。 <a href="#">詳細...</a>	オフ
<code>-ipo_s</code>	<code>ipo_out.s</code> という名前のマルチファイル・アセンブリ・ファイルを生成します。 <a href="#">詳細...</a>	オフ
<code>-ivdep_parallel</code> Itanium ベース・システムのみ	このオプションは、IVDEP ディレクティブが指定されたループにループ・キャリー・メモリ依存が確実にないことを示します。 <a href="#">詳細...</a>	オフ
<code>-Kc++</code>	ソースファイルも、認識できないファイルも、すべて C のソースファイルとしてコンパイルします。	オン ( <code>icpc/ecpc</code> で使用)
<code>-Knopic, -KNOPIC</code> Itanium ベース・システムのみ	位置に依存しないコードを生成しません。	オフ
<code>-KPIC, -Kpic</code>	位置に依存しないコードを生成します。	オフ(IA-32) オン(Itanium ベース・システム)

-L <i>directory</i>	<i>directory</i> で指定したディレクトリを検索してライブラリを探すようリンクに命令します。 <a href="#">詳細...</a>	オフ
-long_double IA-32 のみ	long double データ型のデフォルト・サイズを64ビットから80ビットに変更します。 <a href="#">詳細...</a>	オフ
-M	ソースファイルに含まれている#include行を基にして、ソースファイルごとにmakefileの依存行を生成します。	オフ
-march= <i>cpu</i> IA-32 のみ	指定された <i>cpu</i> に対して排他的にコードを生成します。 <i>cpu</i> に指定できる値には次のものがあります。 <ul style="list-style-type: none"><li>pentiumpro - Pentium Pro プロセッサ、Pentium II プロセッサの命令</li><li>pentiumii - MMX命令</li><li>pentiumiii - ストリーミング SIMD 拡張命令</li></ul> pentium4 - Pentium 4 命令	オフ
-mcpu= <i>cpu</i>	指定した <i>cpu</i> に合わせて最適化します。 IA-32の場合、 <i>cpu</i> に指定できる値には次のものがあります。 <ul style="list-style-type: none"><li>pentium - Pentium プロセッサに合わせて最適化します。</li><li>pentiumpro - Pentium Pro プロセッサ、Pentium II プロセッサ、Pentium III プロセッサの各プロセッサに合わせて最適化します。</li><li>pentium4 - Pentium 4 プロセッサに合わせて最適化します(デフォルト)。</li></ul> Itanium ベースのシステムの場合、 <i>cpu</i> に指定できる値には次のものがあります。 <ul style="list-style-type: none"><li>itanium - Itanium プロセッサに合わせて最適化します。</li></ul> itanium2 - Itanium 2 プロセッサに合わせて最適化します(デフォルト)。	オン pentium IA-32 の場合 itanium2 Itanium ベース・システムの場合
-MD	前処理およびコンパイルを行います。依存情報が含まれている出力ファイル(.d 拡張子)を生成します。	オフ
-MF <i>file</i>	makefileの依存情報を <i>file</i> に生成します。-M	オフ

	または -MM.を指定する必要があります。	
-MG	-Mと類似していますが、見つからないヘッダファイルを、生成したファイルとして処理します。	オフ
-MM	-Mと類似していますが、システム・ヘッダ・ファイルをインクルードしません。	オフ
-MMD	-MDと類似していますが、システム・ヘッダ・ファイルをインクルードしません。	オフ
-MX	インテルwbツールで使用される情報を含む依存ファイル(.o.dep拡張子)を生成します。	オフ
-mp	<a href="#">詳細...</a>	オフ
-mp1	浮動小数点の精度を上げます(速度に与える影響は-mpより低いです)。 <a href="#">詳細...</a>	オフ
-mrelax	-relax をリンクに渡します。	オン
-mno-relax	-relax をリンクに渡しません。	オフ
-mserialize-volatile Itanium ベース・システムのみ	volatileデータ・オブジェクトの参照に対して、制限されたメモリアクセスの順序を適用します。	オフ
-mno-serialize-volatile Itanium ベース・システムのみ	揮発データ・オブジェクトの参照に対して、ランタイム時およびコンパイル時のメモリアクセスの順序を抑制する場合があります。特に、.rel/.acqコンプリータはロードおよびストアの参照で実行されません。	オフ
-nobss_init	ゼロに初期化される変数を DATA セクションに格納します。ゼロに初期化される変数をBSSに置くのを禁止します(DATAを使用)。 <a href="#">詳細...</a>	オフ
-no_cpprt	C++ ランタイム・ライブラリでリンクしません。	オフ
-nodefaultlibs	リンク時に標準ライブラリを使用しません。	
-nolib_inline	標準ライブラリ関数のインライン展開を禁止します。 <a href="#">詳細...</a>	オフ
-nostartfiles	リンク時に標準起動ファイルを使用しません。	オフ
-nostdlib	リンク時に標準ライブラリと起動ファイルを使用しません。	オフ
-O	IA-32システムの -O1 と同じです。Itanium ベース・システムの -O2 と同じです。	オフ
-O0	最適化を無効にします。 <a href="#">詳細...</a>	オフ

-O1	最適化を行います。速度について最適化します。Itanium コンパイラの場合、-O1 は、コードサイズを小さくするためにソフトウェアによるパイプライン化をオフにします。 <a href="#">詳細...</a>	オン
-O2	IA-32システムの -O1 と同じです。Itanium ベース・システムの -O と同じです。 <a href="#">詳細...</a>	オフ
-O3	-O2の内容に加え、もっと効果の高い最適化を実行しますが、コンパイル時間が長くなる場合があります。パフォーマンスに及ぼす影響はアプリケーションに依存します。パフォーマンスの向上が見られないアプリケーションもあります。 <a href="#">詳細...</a>	オフ
-Obn	コンパイラによるインライン展開を制御します。どの程度までインライン展開が行われるかは、次のように $n$ の値によって異なります。 <ul style="list-style-type: none"> <li>0: インライン化を無効にします。</li> <li>1: <code>__inline</code> キーワードで宣言された関数のインライン化を有効にします(デフォルト)。C++ 言語に従ったインライン化も有効にします。</li> </ul> 2: どの関数についてもインライン化を有効にします。ただし、どの関数をインライン化するかはコンパイラが決めます。プロシージャ間の最適化が有効になり、-ip と同じ効果が得られます。	オン
-ofile	出力ファイルに付ける名前を <i>file</i> に指定します。	オフ
-openmp	OpenMP* ディレクティブに基づいてマルチ・スレッド・コードを生成する処理を、パラライザに許可します。-openmp オプションは、最適化レベル-O2 (デフォルト)またはそれ以上で有効です。 <a href="#">詳細...</a>	オフ
-openmp_report { 0   1   2 }	OpenMP パラライザの診断レベルを制御します。 <a href="#">詳細...</a>	オン  -openmp_report 1
-openmp_stubs	シーケンシャル・モードによるOpenMP プログラ	オフ

	ムのコンパイルを有効にします。openmp ディレクティブは無視され、スタブOpenMPライブラリがリンクされます(シーケンシャル)。	
-opt_report	-opt_report_file が指定されている場合を除き、最適化レポートを作成し、stderr に送ります。	オフ
-opt_report_file filename	最適化レポートを保持するfilename を指定します。このオプションが指定されれば、-opt_report を実行する必要はありません。	オフ
-opt_report_level level	出力の冗長レベル( level )を指定します。有効なレベル引数: <ul style="list-style-type: none"> <li>• min</li> <li>• med</li> <li>• max</li> </ul> レベルが指定されていない場合、minがデフォルトで使用されます。	オフ
-opt_report_phase ename	レポートが生成されるコンパイル・フェーズ名(name)を指定します。複数のフェーズから出力を得るために同じコンパイルで複数回、オプションを使用できます。 有効な name 引数には次のものがあります。 <ul style="list-style-type: none"> <li>• ipo: プロシージャ間最適化</li> <li>• hlo: 高度な最適化</li> <li>• ilo: 中間言語スカラー最適化機構 (Intermediate Language Scalar Optimizer)</li> <li>• ecg: コード・ジェネレータ</li> <li>• omp: OpenMP*</li> </ul> all: すべてのフェーズ	オフ
-opt_report_routines substring	ルーチン部分文字列 (substring )を指定します。名前の一部に substring を含むすべてのルーチンからレポートを生成します。デフォルトでは、すべてのルーチンのレポートが生成されます。	オフ
-opt_report_help	-opt_report_phase の可能なすべての設定を表示します。コンパイルは実行されません。	オフ
-P, -F	C/C++ のソースファイルの前処理が済んだら、	オフ



	コンパイルの処理を止め、その結果をファイルに書き込みます。このファイル名は、コンパイラのデフォルトのファイル命名規則に従って付けられます。 <a href="#">詳細...</a>	
-parallel	安全に並列実行可能な並列ループを検出し、そのループに対するマルチスレッド・コードを自動的に生成します。	オフ
-par_report{0 1 2 3}	自動パラライザの診断レベルを制御します。レベルは、0、1、2、または3で制御されます。 <ul style="list-style-type: none"> <li>• -par_report0 = 診断情報を表示しません。</li> <li>• -par_report1 = 正常に自動並列化されたループを表示します。(デフォルト)</li> <li>• -par_report2 = 正常に自動並列化されたループおよび不正に自動並列化されたループを表示します。</li> </ul> -par_report3 = 2 と同じです。また、自動並列化を妨げる実証された依存および推測された依存についての追加情報を示します。	オフ
-par_threshold[n]	並列でのループの実行が効果的である可能性に基づいてループの自動並列化のしきい値を設定します(n=0 から 100)。このオプションは、コンパイル時に計算量が確定できないループに使用します。 <ul style="list-style-type: none"> <li>• -par_threshold0 - ループは、計算量に関わらず自動並列化を行います。</li> <li>• -par_threshold100 - ループは並列化実行が有効であることが確実な場合にのみ自動並列化されます。</li> </ul> <a href="#">詳細...</a>	オフ
-pc32 IA-32 のみ	内部FPU精度を24ビットの仮数に設定します。	オフ
-pc64 IA-32 のみ	内部FPU精度を53ビットの仮数に設定します。	オン
-pc80 IA-32 のみ	内部FPU精度を64ビットの仮数に設定します。	オフ
-prec_div	浮動小数点除算から乗算へ変換する最適化処	オフ

IA-32 のみ	理を無効にします。浮動小数点除算の精度を上げます。 <a href="#">詳細...</a>	
-prof_dir <i>dirname</i>	プロファイル情報(*.dyn、*.dpi)を格納するディレクトリ( <i>dirname</i> )を指定します。 <a href="#">詳細...</a>	オフ
-prof_file <i>filename</i>	プロファイル・サマリ・ファイルに付けるファイル名( <i>filename</i> )を指定します。 <a href="#">詳細...</a>	オフ
-prof_gen[x]	プログラムをインストールしてインストール実行に備え、さらに静的プロファイル情報ファイル(.spi)も新たに生成します。x修飾子を付けると、補足情報が収集されます。 <a href="#">詳細...</a>	オフ
-prof_use	動的フィードバック情報を使用します。 <a href="#">詳細...</a>	オフ
-Qinstall <i>dir</i>	コンパイラのインストール先のルートとして <i>dir</i> を設定します。	オフ
-Qlocation, <i>tool</i> , <i>path</i>	<i>tool</i> で指定したツールの所在として <i>path</i> を指定します。 <a href="#">詳細...</a>	オフ
-Qoption, <i>tool</i> , <i>list</i>	コンパイルの一連の処理の中で、引数 <i>list</i> を、アセンブラやリンカなど別の <i>tool</i> に渡します。 <a href="#">詳細...</a>	オフ
-qp, -p, -pg	UNIX* prof toolを使って関数のプロファイルリングができるようにコンパイルとリンクを行います。	オフ
-rcd IA-32 のみ	FPU の丸め制御の変更を無効にします。浮動小数点から整数への高速変換を行います。 <a href="#">詳細...</a>	オフ
-[no] restrict	restrict指示子とともに指定すると、ポインタの一義化が有効 [無効] になります。	オフ
-S	.s拡張子の付いたアセンブリ・ファイルを生成し、コンパイルを停止します。 <a href="#">詳細...</a>	オフ
-shared	共用オブジェクトを生成します。	オフ
-size_lp64 Itanium ベース・システム	long型、ポインタ型のビットサイズは、64ビットを前提に処理します。	オフ

のみ		
-sox [-] IA-32 のみ	コンパイラのオプションとバージョン情報を実行ファイルに保存します[保存しません]。注: このオプションの目的は、Itanium ベース・システムでの互換性を得ることのみです。	オフ
-static	共用ライブラリとリンクしないようにします。	オフ
-std=c99	C プログラムのC99 サポートを有効にします。	オン
-syntax	C/C++ のソースファイルおよび前処理済みのソースファイルの構文解析が済んだらコンパイラの処理を止めます。つまり、プログラムの構文のチェックだけを行います。コードも出力ファイルも生成しません。警告とメッセージはstderrに出力されます。 <a href="#">詳細...</a>	オフ
-tpp1 Itanium ベース・システムのみ	Itanium プロセッサ向けに最適化します。	オフ
-tpp2 Itanium ベース・システムのみ	Itanium 2 プロセッサ向けに最適化します。生成されたコードは、Itanium プロセッサと互換性があります。	オン
-tpp5 IA-32 のみ	インテル Pentium プロセッサ向けに最適化します。 <a href="#">詳細...</a>	オフ
-tpp6 IA-32 のみ	インテル Pentium Pro プロセッサ、Pentium II プロセッサ、および Pentium III プロセッサ向けに最適化します。 <a href="#">詳細...</a>	オフ
-tpp7 IA-32 のみ	Pentium 4 およびインテル® Xeon™ プロセッサに有利に働くようにコードを調整します。 <a href="#">詳細...</a>	オン
-Uname	どのマクロnameについても定義できないようにします。これは、#undef という前処理ディレクティブと同じ働きをします。 <a href="#">詳細...</a>	オフ
-unroll0 Itanium ベース・システムのみ	ループのアンロールを有効にします。 <a href="#">詳細...</a>	オフ
-unroll [n] IA-32 のみ	ループをアンロールする最大回数を設定します。nを省略すると、アンロールの適否をコンパイラが判断します。ループのアンロールを禁止	オフ

	<p>するときは、<math>n = 0</math>にしてください。</p> <p>詳細...</p>	
-use_asm	アセンブラからオブジェクト・ファイルを生成します。	オフ
-use_msasm IA-32 のみ	GNUスタイルではなく、Microsoft* MASMスタイルのインライン化アセンブリ・フォーマットを使用します。	オフ
-u <i>symbol</i>	<i>symbol</i> が未定義のようにみせます。	オフ
-V	コンパイラのバージョン情報を表示します。	オフ
-v	ドライバ・ツール・コマンドおよび実行ツールを表示します。	
-vec_report [n] IA-32 のみ	<p>ベクトライザの診断情報の分量を調整します。</p> <ul style="list-style-type: none"> <li>• <math>n = 0</math> 診断情報なし</li> <li>• <math>n = 1</math> ベクトル化されたループを示す(デフォルト)</li> <li>• <math>n = 2</math> ベクトル化ループ/非ベクトル化ループを示す</li> <li>• <math>n = 3</math> ベクトル化ループ/非ベクトル化ループと、データ依存性情報の禁止とについて示す</li> <li>• <math>n = 4</math> 非ベクトル化ループを示す</li> <li>• <math>n = 5</math> 非ベクトル化ループと、データの禁止とについて示す</li> </ul> <p>詳細...</p>	<p>オン</p> <p>-vec_report1</p>
-w	警告メッセージをまったく表示しません。	オフ
-Wall	すべての警告を有効にします。	オフ
-wn	<p>診断機能の動作を調整します。</p> <ul style="list-style-type: none"> <li>• <math>n = 0</math> エラーを表示する(-wと同じ)</li> <li>• <math>n = 1</math> 警告とエラーとを表示する(デフォルト)</li> <li>• <math>n = 2</math> リマーク、警告、エラーを表示する</li> </ul> <p>詳細...</p>	<p>オン</p> <p>-w1</p>
-wdL1[, L2, ...]	<p>L1からLNまでの診断を禁止します。</p> <p>詳細...</p>	オフ
-weL1[, L2, ...]	<p>L1からLNまでの診断結果の重要度をエラーに変更します。</p> <p>詳細...</p>	オフ
-Werror	警告をエラーと見なします。	オフ

-wnn	何個エラーが出力されたらコンパイルを止めるかを指定します。個数は $n$ で指定します。 <a href="#">詳細...</a>	オン -wn100
-wr $L1$ [, $L2$ , ...]	$L1$ から $LN$ までの診断結果の重要度を警告に変更します。 <a href="#">詳細...</a>	オフ
-ww $L1$ [, $L2$ , ...]	$L1$ から $LN$ までの診断結果の重要度を警告に変更します。 <a href="#">詳細...</a>	オフ
-W $l$ , o $1$ [, o $2$ , ...]	リンカに渡されるo $1$ 、o $2$ 、その他のオプションです。	オフ
-xtype	-xtypeに続くすべてのソースファイルは次のいずれであると認識されます。 <ul style="list-style-type: none"><li>• c - C ソースファイル</li><li>• c++ - C++ ソースファイル</li><li>• c-header - C ヘッダファイル</li><li>• cpp-output - C 前処理済みファイル</li><li>• assembler - アセンブリ・ファイル</li><li>• assembler-with-cpp - 前処理の必要なアセンブリ・ファイル。</li></ul> none - 認識を無効にし、ファイル拡張子へ戻します。	オフ
-X	インクルード・ファイルの検索先ディレクトリ・リストから標準ディレクトリを削除します。 <a href="#">詳細...</a>	オフ
-Xa	拡張ANSI Cダイアレクトを選択します。	オン
-Xc	ANSI規格に厳密に準拠しているダイアレクトを選択します。	オフ
-x { i   M   K   W } IA-32 のみ	指定したプロセッサでのみ動作するコードを生成します。i、M、K、Wのプロセッサ別コードで指定します。 <ul style="list-style-type: none"><li>• i = Pentium Pro プロセッサ、Pentium II プロセッサの命令</li><li>• M = MMX 命令</li><li>• K = ストリーミング SIMD 拡張命令</li><li>• W = Pentium 4 プロセッサの命令</li></ul> <a href="#">詳細...</a>	オフ

-Xlinker val	リンカに直接渡されるvalです。	オフ
-Zp{1 2 4 8 16}	構造体および共用体のアライメント境界を設定します。1バイト、2バイト、4バイト、8バイト、16バイトのいずれかになります。 <a href="#">詳細...</a>	オン -Zp16

## 機能グループ別リスト

### コンパイル処理のカスタマイズ・オプション

#### 代替ツールと代替パス

オプション	説明
<code>-Qlocation, tool, path</code>	アセンブラ、リンカ、プリプロセッサ、コンパイラなどのツールの所在を示すパスが指定できます。
<code>-Qoption, tool, optlist</code>	<code>optlist</code> で指定したオプションを <code>tool</code> に渡します。 <code>optlist</code> は、複数のオプションをカンマで区切って並べたリストです。

#### 前処理オプション

オプション	説明
<code>-Aname [(values,...)]</code>	<code>name</code> というシンボルを、指定した一連の <code>values</code> に関連付けます。 <code>#assert</code> という前処理ディレクティブと同じです。
<code>-A-</code>	事前定義済みマクロとアサーションをすべて無効にします。
<code>-C</code>	前処理済みのソース出力の中にコメントを保存します。
<code>-Dname [(value)]</code>	<code>name</code> という名前のマクロを定義し、そのマクロを <code>value</code> で指定した値に関連付けます。デフォルト ( <code>-Dname</code> ) では、 <code>value</code> が1のマクロを定義します。
<code>-E</code>	ソース・モジュールを展開してその結果を標準出力に出力するようにプリプロセッサに指示します。
<code>-EP</code>	ソース・モジュールを展開してその結果を標準出力に出力するようにプリプロセッサに指示します。 <code>#line</code> ディレクティブは出力に含まれません。
<code>-P</code>	ソース・モジュールを展開してその結果をカレント・ディレクトリ内の <code>.i</code> ファイルに保存するようにプリプロセッサに指示します。
<code>-Uname</code>	指定したマクロ <code>name</code> に対する自動定義を行わない設定にします。

#### コンパイラのフローの制御

オプション	説明
<code>-C</code>	オブジェクト・ファイルが生成された後、コンパイルの処理を止めます。C/C++ のソースファイルまたは前処理済みのソースファイルのそれぞれについてコンパイラがオブジェクト・ファイ

	ルを生成します。同様に、アセンブリ・ファイルについてもアセンブラがオブジェクト・ファイルを生成します。
-Kpic, -KPIC	位置に依存しないコードを生成します。
-lname	<i>name</i> で指定したライブラリにリンクします。
-nobss_init	ゼロに初期化される変数を DATA セクションに格納します。
-P, -F	C/C++ のソースファイルの前処理が済んだら、コンパイルの処理を止め、その結果をファイルに書き込みます。このファイル名は、コンパイラのデフォルトのファイル命名規則に従って付けられます。
-S	.s 拡張子の付いたアセンブリ・ファイルを生成し、コンパイルを停止します。
-sox [-] (IA-32のみ)	コンパイラのオプションとバージョン情報を実行ファイルに保存します[保存しません]。
-Zp{1 2 4 8 16}	構造体および共用体のアライメント境界を設定します。1バイト、2バイト、4バイト、8バイト、16バイトのいずれかになります。
-Of_check (IA-32のみ)	古いプロセッサ向けに記述されたコードに何らかの Of 命令が使用されていても、そのデコード処理を誤らないように処置します。

## コンパイル出力の制御

オプション	説明
-L <i>directory</i>	<i>directory</i> で指定したディレクトリを検索してライブラリを探すようリンクに命令します。
-o <i>name</i>	<i>name</i> で指定したファイル名の付いた実行可能出力ファイルを生成します。 <i>name</i> を指定しない場合は、デフォルトのファイル名が付きます。
-S	.s 拡張子の付いたアセンブリ・ファイルを生成し、コンパイルを停止します。

## デバッグ・オプション

Option(s)	結果
-g	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O0が有効になり、-f_pが有効になります。
-g -O1	デバッグ情報が生成され、-O1による最適化が有効になります。
-g -O2	デバッグ情報が生成され、-O2による最適化が有効になります。
-g -O3	デバッグ情報が生成され、-O3による最適化が有効になります。



	す。
-g -O3 -fp	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O3による最適化が有効になり、-fpが有効になります。

## 言語適合性オプション

### 適合性オプション

オプション	説明
-ansi	ANSI 準拠のプログラムであることを明確に示します[示しません]。
-ansi_alias [-]	<p>-ansi_aliasによってコンパイラは、次の点を前提にします。</p> <ul style="list-style-type: none"><li>• 配列が外部からアクセスされません。</li><li>• ポインタを非ポインタ型にキャストしません。また、その逆もしません。</li><li>• 2つの異なるスカラ型のオブジェクトへの参照はエイリアス不可です。例えば、int型のオブジェクトはfloat型のオブジェクトと、またはfloat型のオブジェクトはdouble型のオブジェクトとエイリアスの設定はできません。</li></ul> <p>プログラムが上記の条件を満たす場合、-ansi_aliasフラグを設定することでプログラムの最適化が向上します。ただし、プログラムが上記の条件を1つでも満たさない場合、コンパイラは-ansi_aliasフラグを設定することで誤ったコードを生成する可能性があります。</p>
-mp	浮動小数点演算について、できる限り ANSI C 標準と IEEE 754 標準に適合するようにします。NaN比較の動作は準拠しません。

# アプリケーション・パフォーマンス最適化オプション

## 最適化の種類の設定

オプション	説明
-O0	最適化を無効にします。
-O1	最適化を有効にします。速度について最適化します。-O1を指定すると、ライブラリ関数のインライン展開が禁止されます。Itanium®コンパイラの場合、-O1 は、コードサイズを小さくするためにソフトウェアによるパイプライン化をオフにします。
-O2	-O1オプションと同じ働きです。
-O3	-O1、-O2の両オプションに高度な最適化処理を追加します。ループとメモリアクセスの変換が行われない限り、パフォーマンスが高くなることは保証されません。-axKか-xKと組み合わせてこのオプションを使用すると、-O2のときよりも詳細にデータ依存性解析が実行されます。そのためコンパイル時間が長くなる場合があります。

## プロセッサ・ディスパッチのサポート(IA-32 のみ)

オプション	説明
-tpp5	インテル® Pentium® プロセッサを対象として最適化します。Pentium プロセッサで最適の性能が得られます。
-tpp6	Pentium Pro プロセッサ、Pentium II プロセッサ、Pentium III プロセッサの各プロセッサを対象として最適化します。この3種類のプロセッサで最適の性能が得られます。
-tpp7	Pentium 4 プロセッサを対象として最適化します。RedHat* Linux* 7.1 または 7.2、およびストリーミング SIMD 拡張命令 2 をサポートしている必要があります。Pentium 4 プロセッサで最適の性能が得られます。
-ax{i M K W}	以下の各記号で指定したプロセッサ専用のコードを、1個のバイナリ・ファイル上に生成します。 <ul style="list-style-type: none"> <li>• i Pentium Pro プロセッサ、Pentium II プロセッサ</li> <li>• M MMX® テクノロジ Pentium プロセッサ</li> <li>• K Pentium III プロセッサ</li> <li>• W Pentium 4 プロセッサ</li> </ul> また、-axでは汎用のIA-32 コードも生成されます。通常は、汎用コードのほうが処理速度は遅くなります。
-x{i M K W}	以下の各記号で指定した拡張命令の利用できるプロセッサでしか動作しないコードを生成します。 <ul style="list-style-type: none"> <li>• i Pentium Pro プロセッサ、Pentium II プロセッサ</li> </ul>

	<ul style="list-style-type: none"> <li>• M MMX テクノロジ Pentium プロセッサ</li> <li>• K Pentium III プロセッサ</li> </ul> W Pentium 4 プロセッサ
--	--

## プロシージャ間の最適化

オプション	説明
-ip	ファイルを1個コンパイルするときにプロシージャ間の最適化を有効にします。
-ip_no_inlining	-ipを指定してプロシージャ間の最適化が行われるときに、インライン化については無効にします。しかし、他のプロシージャ間の最適化には何の影響も与えません。
-ipo	マルチファイルにわたるプロシージャ間の最適化を有効にします。
-ipo_c	あとのリンク段階で利用できるマルチファイル・オブジェクト・ファイルを生成します。
-ipo_obj	-ipoとともに指定すると、実際のオブジェクト・ファイルが強制的に生成されます。
-ipo_s	ipo_out.asmという名前のマルチファイル・アセンブリ・ファイルを生成します。このファイルは後のリンク段階で使うことができます。
-inline_debug_info	コールサイトのソース・ポジションをインライン化コードに割り当てる代わりに、インライン化コードのソース・ポジションを保存します。
-nolib_inline	標準ライブラリ関数のインライン展開を禁止します。

## プロファイルに基づく最適化

オプション	説明
-prof_gen[x]	インストルメント済み実行の準備として、インストルメント済みコードをオブジェクト・ファイル内に生成するようにコンパイラに命令します。注: 動的情報ファイルは、インストルメント済み実行ファイルを実行する第2フェーズで生成されます。
-prof_use	プロファイルによって最適化された実行ファイルを生成し、利用可能な動的情報(.dyn)ファイルをいくつかマージしてpgopti.dpi ファイルを1個作成するようにコンパイラに命令します。インストルメント済みプログラムを何回か実行する場合は、-prof_use を指定すると、実行するたびに同じ動的情報ファイル同士がマージされ、その前のpgopti.dpi ファイルは上書きされます。

-prof_dirdir	プロファイリングの出力ファイル、*.dyn と *.dpi で プロファイル情報(*.dyn、*.dpi)を格納するディレクトリ (dir) を指定します。
-prof_filefile	サマリファイルのプロファイリングに使うファイル名を指定 します。

## 高水準言語の最適化

オプション	説明
-openmp	OpenMP* ディレクティブに基づいてマルチ・スレッド・ コードを生成する処理を、パラライザに許可します。 単一プロセッサ・システムとマルチプロセッサ・システ ムのいずれでも並列実行が可能になります。
-openmp_report {0   1   2}	OpenMP*パラライザの診断レベルを制御します。 <ul style="list-style-type: none"> <li>0 - 情報なし</li> <li>1 - ループ、領域、およびセクション(デフォル ト)</li> </ul> 2 - 1の診断に加えて、主コンストラクタ、単一コンス トラクタなど。
-unroll [n]	ループをアンロールする最大回数(n)を設定します。n を省略すると、アンロールの適否をコンパイラが判断し ます。n = 0にすると、ループのアンロールが禁止され ます。Itanium® ベースのアプリケーションの場合、 -unroll [0] は互換性を確保する目的にのみ使用 します。

## 最適化レポート

オプション	説明
-opt_report	最適化レポートを生成して、stderr に送ります。
-opt_report_filefilename	最適化レポートを保持するfilename を指定しま す。
-opt_report_level {min /med/max}	最適化レポートの詳細レベルを指定します。 デフォルト: -opt_report_level min
-opt_report_phasephase	レポートを生成する最適化を指定します。複数の最適 化をコマンドライン上で複数回指定できます。
-opt_report_help	利用可能なすべてのフェーズを画面に出力します。 -opt_report phase.
-opt_report_routinesubstring	名前の一部に部分文字列(substring)を含むすべ てのルーチンからレポートを生成します。指定されない

	場合、すべてのルーチンからのレポートが生成されます。
--	----------------------------

## Windows\* と Linux\* のコンパイラ・オプションの対応表

### コンパイラ・オプションの対応表

Linux*	Windows*	説明	デフォルト
-Of	-QIOf	Pentium® プロセッサの Of エラッタに対応したパッチを適用するかしないかを指定します。	オフ
-A-	-QA-	すべての事前定義マクロを削除します。	オフ
-Aname [ (val) ]	-QAname [ (val) ]	val という値を持つアサーション名を作成します。	オフ
-ansi	-Za	ANSI に準拠していることをコンパイラに通知するかしないかを指定します。	オン
-ax { i   K   M   W }	-Qax { i   K   M   W }	プロセッサの拡張機能に対応した専用のコードを生成します。プロセッサの種類は i、K、M、W で指定します。同時に汎用の IA-32 コードも生成します。 <ul style="list-style-type: none"> <li>i = Pentium Pro プロセッサ、Pentium II プロセッサの命令</li> <li>K = ストリーミング SIMD 拡張命令</li> <li>M = MMX® テクノロジー</li> <li>W = ストリーミング SIMD 拡張命令2</li> </ul>	オフ
-C	-C	コメントを削除しません。	オフ
-c	-c	オブジェクト・ファイル(.o)までコンパイルします。リンクはしません。	オフ
-Dname [=value]	-Dname [=value]	マクロを定義します。	オフ
-E	-E	前処理を行い、その結果を標準出力に出力します。	オフ
-fP	-Oy-	すべての関数についてEBPベースのスタックフレームを使用しま	オフ

		す。	
-g	-Zi	シンボリック・デバッグ情報をオブジェクト・ファイル内に生成します。	オフ
-H	-QH	インクルード・ファイルの順番を出力します。	オフ
-help	-help	ヘルプ・メッセージ一覧を出力します。	オフ
-Idirectory	-Idirectory	インクルード・ファイルの検索先にディレクトリを追加します。	オフ
-inline_debug_info	-Qinline_debug_info	コールサイトのソース・ポジションをインライン化コードに割り当てる代わりに、インライン化コードのソース・ポジションを保存します。	オフ
-ip	-Qip	シングル・ファイルIPOを有効にします(複数ファイル内)。	オフ
-ip_no_inlining	-Qip_no_inlining	IP の動作を最適化します。全体のインライン化も、部分的なインライン化も禁止されます(-ip または -ipo のいずれかが必要です)。	オフ
-ipo	-Qipo	マルチファイルIPOを有効にします(複数ファイル間)。	オフ
-ipo_obj	-Qipo_obj	IPの動作を最適化します。実際のオブジェクト・ファイルを強制的に生成します。-ipoが必要です。	オフ
-KPIC	NA	位置に依存しないコードを生成します。-Kpicと同じです。	オフ
-Kpic	NA	位置に依存しないコードを生成します。-KPICと同じです。	オフ
-long_double	-Qlong_double	80ビット長のlong double型を有効にします。	オフ
-m	NA	マップファイルを生成するようにリンクに命令します。	オフ
-M	-QM	makefileの依存情報を生成します。	オフ
-mp	-Op [-]	浮動小数点の精度を維持します。一部の最適化項目が無効になります。	オフ



-mp1	-Qprec	浮動小数点の精度を上げます(速度に与える影響は-mpより低いです)。	オフ
-nobss_init	-Qnobss_init	ゼロに初期化された変数をBSSに配置するのを禁止します(DATAを使用)。	オフ
-nolib_inline	-Oi[-]	組み込み関数のインライン展開を禁止します。	オフ
-O	-O2		オフ
-ofile	-Fefile または -Fofile	出力ファイルに付ける名前をfileに指定します。	オフ
-O0	-Od	最適化を禁止します。	オフ
-O1	-O1	速度について最適化します。	オフ
-O2	-O2		オン
-P	-EP	ファイルまで前処理を行います。	オフ
-pc32	-Qpc 32	内部FPU精度を24ビットの仮数に設定します。	オフ
-pc64	-Qpc 64	内部FPU精度を53ビットの仮数に設定します。	オン
-pc80	-Qpc 80	内部FPU精度を64ビットの仮数に設定します。	オフ
-prec_div	-Qprec_div	浮動小数点除算の精度を上げます。速度に多少影響します。	オフ
-prof_dir directory	-Qprof_dir directory	出力ファイル(*.dyn、*.dpi)のプロファイリングに使うディレクトリを指定します。	オフ
-prof_file filename	-Qprof_file filename	サマリファイルのプロファイリングに使うファイル名を指定します。	オフ
-prof_gen[x]	-Qprof_genx	プロファイリングができるようにプログラムをインストールします。x修飾子を付けると、補足情報が収集されます。	オフ
-prof_use	-Qprof_use	最適化中にプロファイリング情報が使えるようにします。	オフ
-Qinstall dir	NA	コンパイラのインストール先のルートとしてdirを設定します。	オフ
-Qlocation, str, dir	-Qlocation, tool, path	strで指定したツールの位置としてdirを設定します。	オフ
-Qoption, str, opts	-Qoption, tool, list	strで指定したツールにoptsオプションを渡します。	オフ

-qp, -p	NA	UNIX* gprof toolを使って関数のプロファイリングができるようにコンパイルとリンクを行います。	オフ
-rcd	-Qrcd	浮動小数点から整数への高速変換機能を有効にします。	オフ
-restrict	-Qrestrict	ポインタの一義化ができるように restrict キーワードを有効にします。	オフ
-S	-S	.s 拡張子の付いたアセンブリ・ファイルを生成し、コンパイルを停止します。	オフ
-sox [-]	-Qsox	実行可能ファイル内にコンパイラ・オプションとバージョンとを保存するかどうかを指定します。特に指定しなければ保存されます。	オン
-syntax	-Zs	構文チェックのみを行います。	オフ
-tpp5	-G5	Pentium プロセッサに合わせて最適化します。	オフ
-tpp6	-G6	Pentium Pro プロセッサ、Pentium II プロセッサ、Pentium III プロセッサの各プロセッサに合わせて最適化します。	オフ
-tpp7	-G7	Pentium 4 プロセッサに合わせて最適化します。	オフ
-Uname	-Uname	事前定義済みマクロを無効にします。	オフ
-unroll [n]	-Qunrolln	ループをアンロールする最大回数を設定します。n を省略すると、アンロールの適否をコンパイラが判断します。n=0 にすると、ループをアンロールする機能が働きません。	オフ
-V	-QV	コンパイラのバージョン情報を表示します。	オフ
-w	-w	エラーを表示します。	オフ
-w2	-W4	リマーク、警告、エラーの各メッセージを表示します。	
-wn	-Wn	診断機能の動作を調整します。エラーを表示するときは、n=0 にします。警告とエラーとを表示すると	オフ

		きは、n=1にします。リマーク、警告、エラーを表示するときは、n=2にします。	
-wdL1 [, L2,...]	-Qwd [tag]	L1からLNまで、診断機能を無効にします。	オフ
-weL1 [, L2,...]	-Qwe [tag]	L1からLNまでの診断結果の重要度をエラーに変更します。	オフ
-wnn	-Qwn [tag]	エラーの最大数nを出力します。	オフ
-wrL1 [, L2,...]	-Qwr [tag]	L1からLNまでの診断結果の重要度をリマークに変更します。	オフ
-wwL1 [, L2,...]	-Qww [tag]	L1からLNまでの診断結果の重要度を警告に変更します。	オフ
-X	-X	インクルード・ファイルの検索先から、標準のディレクトリを外します。	オフ
-x { i   K   M   W }	-Qx [ i   M   K   W ]	プロセッサの拡張機能に対応した専用のコードを生成します。プロセッサの種類は、i、K、M、Wで指定します。同時に汎用のIA-32コードも生成します。 <ul style="list-style-type: none"> <li>• i = Pentium Pro プロセッサ、Pentium II プロセッサの命令</li> <li>• K = ストリーミングSIMD 拡張命令</li> <li>• M = MMX テクノロジ</li> </ul> W = ストリーミングSIMD拡張命令2	オフ
-Xa	-Ze	拡張ANSI Cダイアレクトを選択します。	オフ
-Xc	-Za	ANSI規格に厳密に準拠しているダイアレクトを選択します。	オフ
-Zp { 1   2   4   8   16 }	-Zp [n]	構造体のアライメント境界をバイト単位で指定します。n = 1、2、4、8、16。特に指定しなければn = 8。このオプションのほうがコードのデフォルトのアライメントより優先されます。	オフ

# インテル(R) C++ コンパイラの起動

## コンパイラの起動

インテル® C++ コンパイラの起動方法は次のとおりです。

- 直接起動する: コマンドラインからのコンパイラの起動
- システムmakefileを使う: コマンドラインからのmakefileの実行

## コマンドラインからのコンパイラの起動

コマンド・ラインからインテル® C++ コンパイラを起動するには、次の2つの手順が必要です。

1. 環境変数を設定
2. `icc`または`ecc`でコンパイラを起動



注

IA-32 と Itanium® ベースのシステムでそれぞれ、C++ ソース・ファイルに対して`icpc`および`ecpc`でコンパイラを起動することもできます。本書の`icc` および`ecc` コンパイラの例は、C と C++ ソース・ファイルに適用されるものです。

## 環境変数の設定

コンパイラを実行する前に、環境変数を設定して、各種コンポーネントの場所を指定する必要があります。インテル C++ コンパイラには、環境変数の設定に使えるシェル・スクリプトを含んでいます。コマンド・ラインから、各自に合ったシェル・スクリプトを実行してください。デフォルトでコンパイラをインストールすると、これらのスクリプトは次の場所にあります。

- **IA-32 システム:** `/opt/intel/compiler70/ia32/bin/iccvars.sh`
- **Itanium ベース・システム:**  
`/opt/intel/compiler70/ia64/bin/eccvars.sh`

## シェル・スクリプトの実行

IA-32の`iccvars.sh`スクリプトを実行するには、次のコマンドをコマンド・ラインで入力してください。

```
prompt>source /opt/intel/compiler70/ia32/bin/iccvars.sh
```

Linux\* を起動したときに、`iccvars.sh` を自動的に実行する場合は、起動ファイルを編集します。ファイルの最後に同じ行を追加してください。

```
# set up environment for Intel compiler icc
source /opt/intel/compiler70/ia32/bin/iccvars.sh
```

手順は、Itanium ベースのシステム上で`eccvars.sh`シェル・スクリプトを実行するのと似

ています。

## コンパイラの起動

一旦、環境変数を設定すると、下記のようにコンパイラを起動できます。

- **IA-32 システム:** `prompt>icc [options] file1 [file2 . . .]`
- **Itanium ベース・システム:** `prompt>ecc [options] file1 [file2 . . .]`

構文	説明
options	1つ以上のコマンドライン・オプションを示します。コンパイラは、ハイフン(-)が先頭にある1文字以上の文字をオプションとして認識します。
file1, file2 . . .	コンパイル・システムによって処理される1つ以上のファイルを示します。複数のファイルを指定することもできます。複数のファイルの区切りにはスペースを使用してください。
linker_options	リンカに渡されるオプションです。

## コマンド・ラインからの makefile の実行

インテル® C++ コンパイラを使ってコマンド・ラインから実行するときは、`/usr/bin`をパスに付けてください。Cシェルを使う場合は、`.cshrc`ファイルを編集して、次の行を追加できます。

```
setenv PATH /usr/bin:<full path to Intel compiler>
```



インテル・コンパイラを使用する場合、`CC=icc` 設定を `makefile` に含める必要があります。コマンドラインから`makefile`でインテル・コンパイラを使用する場合、同じ設定をコマンドラインで使用してください。`makefile`でGNU\* C コンパイラ(`gcc`)を使用する場合、インテル・コンパイラで認識されないコマンド・ライン・オプションを変更する必要があります。

その後、次のようにコンパイルできます。

```
prompt>make -f my_makefile
```

## コンパイラの入力ファイル

デフォルトでは、コンパイラは `.cpp` および `.cxx` ファイルを C++ ファイルとして、`.c` ファイルを C 言語のソースファイルとして認識します。本書で示す例では `.c` 拡張子を使用し

ます。インテル® C++ コンパイラでは、`icpc` および `ecpc` を使用し、C++ ファイルとして `.c` ファイルをコンパイルします。また、インテル C++ コンパイラは下の表のデフォルトのファイル名拡張子も認識できます。

ファイル名	ファイルの種類	処理
<code>filename.a</code>	オブジェクト・ライブラリ	リンカに渡されます。
<code>filename.i</code>	C++ プリプロセッサで前処理し展開された C/C++ ソース	コンパイラに渡されます。
<code>filename.o</code>	コンパイル済みのオブジェクト・モジュール	リンカに渡されます。
<code>filename.s</code>	アセンブリ・ファイル	
<code>filename.so</code>	共有オブジェクト・ファイル	
<code>filename.S</code>	前処理が必要なアセンブリ・ファイル	

## コンパイラのデフォルトの動作

### デフォルトのコンパイラ・オプション

- IA-32 アーキテクチャ固有のオプション
- Itanium® アーキテクチャ固有のオプション

他のすべてのオプションは、IA-32およびItanium ベース・システムの両方に対応しています。

オプション	説明
<code>-c99</code>	C プログラムのC99 サポートを有効にします。
<code>-falias</code>	プログラムでエイリアシングを前提に処理します。
<code>-ffnalias</code>	関数内でのエイリアシングを前提に処理します。
<code>-fverbose-asm</code>	コンパイラ・コンポーネント付きアセンブリ・ファイルを生成します。
<code>-KPIC, -Kpic</code>	位置に依存しないコードを生成します。
<code>-mcpu=pentium4</code>	Pentium® 4 プロセッサを対象として最適化します(IA-32 システムのみ)。
<code>-mcpu=itanium2</code>	Itanium 2 プロセッサを対象として最適化します(Itanium ベース・システムのみ)。
<code>-O1</code>	最適化を行います。速度について最適化します。Itanium コンパイラの場合、 <code>-O1</code> は、コードサイズを小さくするためにソフトウェアによるパイプライン化をオフにします。 <a href="#">詳細...</a>
<code>-O2</code>	IA-32システムの <code>-O1</code> と同じです。Itanium ベース・システムの <code>-O</code> と同じです。 <a href="#">詳細...</a>

-Ob1	<code>__inline</code> キーワードで宣言された関数のインライン化を有効にします。C++ 言語に従ったインライン化も有効にします。
-openmp_report1	OpenMP パラライザの診断レベルを制御します。 <a href="#">詳細...</a>
-pc64 IA-32 のみ	内部FPU精度を53ビットの仮数に設定します。
-std=c99	C プログラムの C99 サポートを有効にします。
-tpp2 Itanium ベース・システムのみ	Itanium 2 プロセッサ向けに最適化します。生成されたコードは、Itanium プロセッサと互換性があります。
-tpp7 IA-32 のみ	Pentium 4 プロセッサおよびインテル® Xeon™ プロセッサに有利に働くようにコードを調整します。 <a href="#">詳細...</a>
-vec_report1 IA-32 のみ	ベクトル化されたループを示すために、ベクトライザの診断情報の量を調整します。 <a href="#">詳細...</a>
-w1	診断機能の動作を調整します。警告メッセージとエラー・メッセージを表示します。 <a href="#">詳細...</a>
-Zp16	構造体および共用体の16バイト・アライメント境界を設定します。 <a href="#">詳細...</a>

## コンパイラのデフォルトの動作

オプションを何も指定せずにインテル® C++ コンパイラを起動すると、次のデフォルト設定が使用されます。

- ファイル名 `a.o` の実行可能出力ファイルを生成します。
- 設定ファイルで指定されているオプションを先に起動します。「設定ファイル」を参照してください。
- 現在のディレクトリ内でライブラリ・ファイルが見つからない場合は、`LD_LIBRARY_PATH` 変数で 指定されたディレクトリ内でライブラリ・ファイルを検索します。
- 構造体のアライメントの最も厳密な制約条件を8バイトに設定します。
- エラー・メッセージと警告メッセージを表示します。
- デフォルトの `-O2` オプションを使用して標準的な最適化を実行します。「最適化レベルの設定」を参照してください。

コンパイラがコマンドライン・オプションを認識しない場合、そのオプションは無視され、警告を

表示します。システム・メッセージの詳細については、「診断メッセージ」を参照してください。

## コンパイル・フェーズ

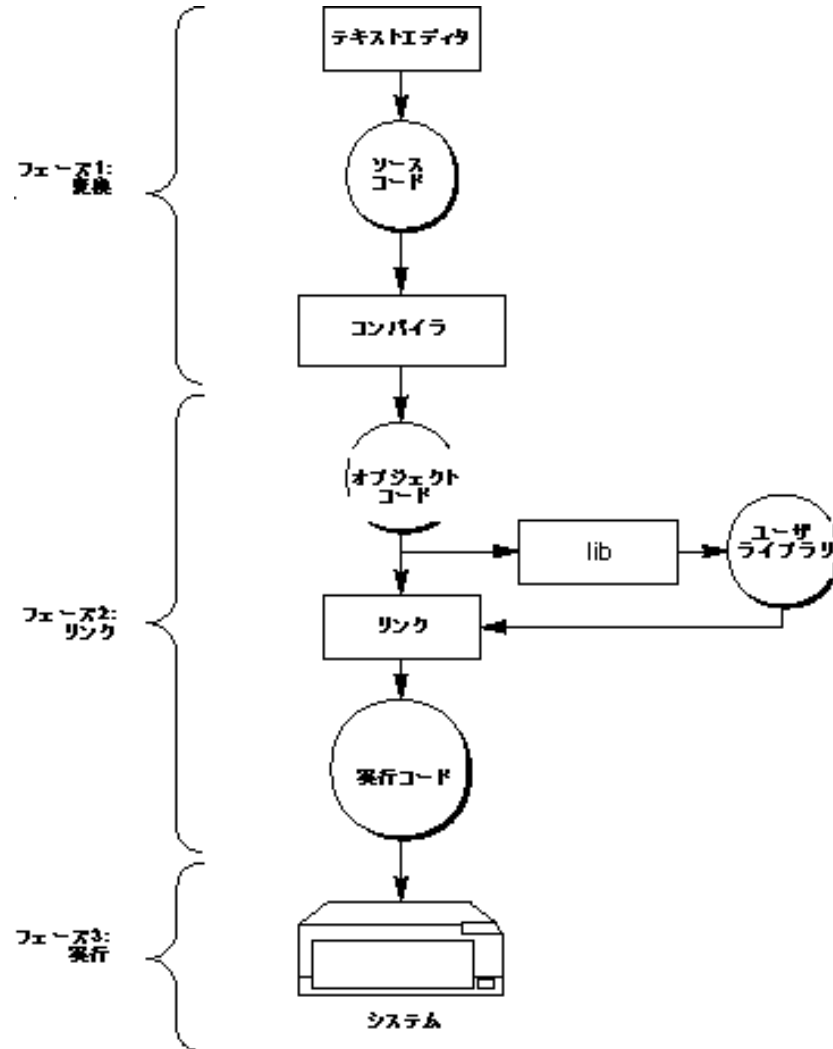
実行ファイルを生成する場合、コンパイラはデフォルトで、コンパイル・フェーズとリンクフェーズを実行します。コンパイラを起動すると、ファイル名の拡張子とコマンドラインで指定したコンパイル・オプションを基にして、どのコンパイル・フェーズを実行するかをコンパイラ・ドライバが判断します。

コンパイラは、オブジェクト・ファイルと、認識できないファイル名とをリンカに渡します。次に、リンカは、そのファイルがオブジェクト・ファイル(.o)なのかライブラリ(.a)なのかを判断します。コンパイル・ドライバは、どの種類の入力ファイルも正しく処理できるため、どのコンパイル・フェーズを起動するときにも使用できます。

コンパイラとシステム専用のプログラミング支援ツールの関係を下図に示します。



## アプリケーションの開発サイクル



OM09714

# コンパイル環境のカスタマイズ

## コンパイル環境のカスタマイズ

IA-32 アーキテクチャおよびインテル® Itanium® アーキテクチャでは、コンパイル環境を設定する必要があります。コンパイル時の環境は、次の変数、オプション、ファイルを指定してカスタマイズができます。

- 環境変数 -- コンパイラとその他のツールが特定のファイルを検索するときのパス
- 設定ファイル -- 各コンパイル時に使用するオプション
- 応答ファイル -- 各プロジェクトに使用するオプションとファイル
- インクルード・ファイル -- ソース・ヘッダ・ファイルの名前と場所

## 環境変数

コンパイラがライブラリやインクルード・ファイルなど特殊なファイルを検索する際のパス(検索先)を指定して、環境をカスタマイズできます。

- `LD_LIBRARY_PATH` は、インテルが提供する全ライブラリの場所を指定します。
- `PATH` は、システムがバイナリ実行ファイルを検索するディレクトリを指定します。
- `ICCCFG` は、`icc` コンパイラでコンパイルをカスタマイズする設定ファイルを指定します。
- `ICPCCFG` は、`icpc` コンパイラでコンパイルをカスタマイズする設定ファイルを指定します。
- `ECCCFG` は、`ecc` コンパイラでコンパイルをカスタマイズする設定ファイルを指定します。
- `ECPCCFG` は、`ecpc` コンパイラでコンパイルをカスタマイズする設定ファイルを指定します。
- `TMP` は一時ファイルの格納ディレクトリを指定します。`TMP`で指定されたディレクトリが存在しない場合、コンパイラは現在のディレクトリに一時ファイルを置きます。
- `IA32ROOT`(IA32 ベース・システム)は、`bin`、`lib`、`include` の各ディレクトリおよびヘッダを置換するディレクトリの入っているディレクトリを選択します。
- `IA64ROOT`(Itanium® ベース・システム)は、`bin`、`lib`、`include` の各ディレクトリおよびヘッダを置換するディレクトリの入っているディレクトリを選択します。

## コンパイル環境オプション

インテル® C++ コンパイラには、環境変数の設定に使えるシェル・スクリプトを含んでいます。コマンドラインから、各自に合ったシェル・スクリプトを実行してください。このスクリプトは次の

場所にありますが(標準のディレクトリにインストールした場合)。

## Bash シェル環境

- IA-32 システム: `/opt/intel/compiler70/ia32/bin/iccvars.sh`
- Itanium ベース・システム:  
`/opt/intel/compiler70/ia64/bin/eccvars.sh`

`iccvars.sh` スクリプトを実行するには、次のコマンドをコマンドラインで入力してください。

```
prompt>source /opt/intel/compiler70/ia32/bin/iccvars.sh
```

Linux を起動したときに、`iccvars.sh` を自動的に実行する場合は、起動スクリプト (bash シェル用の `.bash_profile`) を編集します。ファイルの最後に同じ行を追加してください。

```
# set up environment for icc
source /opt/intel/compiler70/ia32/bin/iccvars.sh
```

## 設定ファイル

コマンドラインへ入力する項目が頻繁にある場合は、設定ファイルを使用してその入力作業を自動化できます。そうすれば、コマンドライン・オプションの入力時間が短くなるだけでなく、統一を図れるようになります。有効なコマンドライン・オプションであれば何でも設定ファイルに書き込めます。起動されたコンパイラは、設定ファイルに記述している各コマンドライン・オプションを上から順に処理します。



注

コンパイラを実行するたびに、設定ファイルに書かれたオプションが実行されます。プロジェクトごとにオプションを変える必要がある場合は、「応答ファイル」を参照してください。

## 設定ファイルの使用方法

基本的な設定ファイルの書き方を次の例に示します。`.cfg` ファイルの作成が済んだら、あとはコンパイラを実行するときにその設定ファイルをコンパイラの実行ファイルと同じディレクトリに格納するだけです。シャープ(#)文字の後ろのテキストはコメントとして認識します。IA-32 用にコンパイルするときの設定ファイルは `icc.cfg` です。Itanium® ベース・システムを対象にコンパイルするときの設定ファイルは `ecc.cfg` です。

```
## Sample configuration file.
## Define preprocessor macro MY_PROJECT.
-DMY_PROJECT
## Additional directories to be searched
## for INCLUDE files, before the default.
-I /project/include
```

## 応答ファイル

特定のコンパイル処理に使用するオプションを指定し、さらにその情報をそれぞれ別のファイルに保存するときは、応答ファイルを使用します。応答ファイルはコマンドラインのオプションとして呼び出します。応答ファイルに含まれている各種オプションは、その応答ファイルを呼び出した場所のコマンドラインに挿入します。

応答ファイルを使用すれば、コマンドラインでの入力作業が自動化されるため、コマンドライン・オプションの入力時間が短くなるだけでなく、統一が図れます。プロジェクトが変更される際に設定ファイルの編集を回避するため、特定のプロジェクトに使用するオプションを維持するには、各応答ファイルを使用します。

オプションやファイル名も、応答ファイルの1行の中に何個でも書き込めます。同じコマンドラインの中で複数の応答ファイルを参照できます。シャープ記号(#)の後ろに入力したテキストはコメントとして認識されます。

応答ファイルを使用する際は次の構文を使用します。

- IA-32 システム: `prompt>icc @response_file filenames`
- Itanium® ベース・システム: `prompt>ecc @response_file filenames`



注

コマンドライン上では、応答ファイル名の前にアットマーク (@) を付けることに注意してください。

## インクルード・ファイル

インクルード・ディレクトリは、デフォルトのシステムエリアと `-Idirectory` オプションで指定された場所が検索されます。検索先ディレクトリを複数指定するときは、`-Idirectory` コマンドを複数指定する必要があります。コンパイラはインクルード・ファイルを検索するとき、次の順番で各ディレクトリを見ます。

- `include` が含まれるソースファイル・ディレクトリ
- `-I` オプションで指定されるディレクトリ

## インクルード・ディレクトリの削除方法

デフォルトのシステムエリアの検索をコンパイラに禁止するには、`-X` オプションを使用します。  
`-X` オプションと `-I` オプションを併用すると、インクルード・ファイルを検索する際のデフォルトのパスではなく別のパスを検索するようにコンパイラに命令できます。

例えば、デフォルトのパスの代わりに `/alt/include` のパスを検索するようにコンパイラに命令するときは次のようにします。

- **IA-32 システム:** `prompt>icc -X -I/alt/include prog.c`
- **Itanium® ベース・システム:** `prompt>ecc -X -I/alt/include prog.c`

# コンパイル処理のカスタマイズ

## コンパイル処理のカスタマイズの概要

ここでは、コンパイル処理をカスタマイズするオプションについて説明します。

- 前処理
- コンパイル
- リンク
- デバッグ

## 代替ツールと代替パスの指定

前処理、コンパイル、アセンブリ、およびリンクに使用される、代替ツールを指定するようにコンパイラに指示できます。また、選択したツール専用の各種オプションをコマンドラインで呼び出せます。こうした操作は、`-Qlocation`および`-Qoption`を使用して行います。次に、それぞれについて説明します。

### 代替コンポーネントの指定方法

ツールへの代替パスを指定するには、`-Qlocation` を使用します。このオプションには引数を2個指定します。構文は次のとおりです。

`-Qlocation, tool, path`

ツール	説明
cpp	コンパイラのフロントエンド・プリプロセッサを指定します。
c	C++ コンパイラを指定します。
asm	アセンブラを指定します。
ld	リンカを指定します。

`path`は、ツールの所在を示す完全パスです。

## 他のプログラムへオプションを渡す方法

`-Qoption`を使用すると、`optlist`で指定したオプションが`tool`に渡されます。`optlist`は、複数のオプションをカンマで区切って並べたリストです。このコマンドの構文は次のとおりです。

`-Qoption, tool, optlist`

ツール	説明
cpp	コンパイラのフロントエンド・プリプロセッサを指定します。

c	C++ コンパイラを指定します。
asm	アセンブラを指定します。
ld	リンカを指定します。

optlist は、指定したプログラムに有効な引数文字列です。1個でも複数個でも指定できます。指定する引数がコマンドライン・オプションの場合は、ハイフンを含めてください。スペースかタブ文字を含む引数を指定するときは、その引数全体を二重引用符(")で囲んでください。引数を複数指定するときは、それぞれをカンマで区切ってください。例えば、コンパイラを使用してソースから実行ファイルを作成し、そのときにリンカを使用してメモリマップを作成するときは、次のようにします。

- **IA-32 システム:** prompt>icc -Qoption,link,-map,proto.map proto.c
- **Itanium® ベース・システム:** prompt>ecc -Qoption,link,-map,proto.map proto.c

上の例の -Qoption, link というオプションによって -map オプションがリンカに渡されます。これは、コンパイルを行うときに各種ツールに各種引数を明示的に渡す方法の1つです。また、-Xlinker val を使用して、リンカに値(val)を渡すこともできます。

# 前処理

## 前処理の概要

この節では、プリプロセッサの動作の指定に使用できるオプションについて説明します。前処理は、マクロ置換、条件付きコンパイル、ファイルのインクルードといった処理を行います。

### プリプロセッサ・オプション

オプション	説明
-Aname [ (value)]	<i>name</i> というシンボルを、指定する <i>value</i> に関連付けます。 <code>#assert</code> という前処理ディレクティブと同じです。
-A-	事前定義済みマクロとアサーションをすべて無視します。
-C	前処理済みのソース出力の中にコメントを保存します。
-Dname [=text]	<i>name</i> という名前のマクロを定義し、そのマクロを指定した <i>text</i> に関連付けます。 <code>-Dname</code> を指定しないと、 <i>text</i> を1としてマクロが定義されます。
-E	ソース・モジュールを展開してその結果を標準出力に出力するようにプリプロセッサに指示します。 <code>#line</code> ディレクティブについて出力します。
-EP	ソース・モジュールを展開してその結果を標準出力に出力するようにプリプロセッサに指示します。 <code>#line</code> ディレクティブについて出力しません。
-P	ソース・モジュールを展開してその結果をカレント・ディレクトリ内の <code>.i</code> ファイルに保存するようにプリプロセッサに指示します。 <code>#line</code> ディレクティブについて出力しません。
-Uname	指定したマクロ <i>name</i> に対する自動定義を行わない設定にします。
-X	インクルード・ファイルの検索先から、標準のディレクトリを外します。
-H	インクルードされたファイルのフルパス名を順番に標準出力に出力します。 <code>#include</code> の依存性を表すのに、インデントが使用されます。
-M	makefileの依存情報を生成します。
-MD	前処理およびコンパイルを行います。依存情報が含まれている出力ファイル( <code>.d</code> 拡張子)を生成します。
-MFfile	makefileの依存情報を <i>file</i> に生成します。 <code>-M</code> または <code>-MM</code> を指定する必要があります。
-MG	<code>-M</code> と類似していますが、見つからないヘッダファイルを、生成したファイルとして処理します。



-MM	-Mと類似していますが、システム・ヘッダ・ファイルをインクルードしません。
-MMD	-MDと類似していますが、システム・ヘッダ・ファイルをインクルードしません。
-MX	インテルwbツールで使用される情報を含む依存ファイル(.o.dep拡張子)を生成します。
-dM	前処理を行った後に有効なマクロ定義を出力します (-Eと使用)。
-MD	前処理およびコンパイルを行います。依存情報が含まれている出力ファイル(.d 拡張子)を生成します。
-Idirectory	インクルード・ファイルを探すときの、標準以外のdirectoryを指定します。

## 前処理のみを行う

ソースファイルをコンパイルせず、前処理のみ行うには、-E オプション、-P オプションまたは -EP オプションを使用します。これらのオプションを使用する場合、コンパイルの前処理フェーズのみが起動されます。

### -E の使用

-E オプションを指定すると、プリプロセッサによってソース・モジュールが展開され、その結果を標準出力に出力します。前処理されたソースには#line ディレクティブが含まれます。このディレクティブは、コンパイラがソースファイルと行番号を判断するために使用します。例えば、2つのソースファイルを前処理し、結果を標準出力に出力するには、次のコマンドを使用します。

- IA-32 システム: `prompt>icc -E prog1.c prog2.c`
- Itanium® ベース・システム: `prompt>ecc -E prog1.c prog2.c`

### -P の使用

-P オプションを指定した場合、プリプロセッサはソース・モジュールを展開し、標準出力の代わりに .i ファイルに出力するように指示します。-E オプションとは異なり、-P オプションの出力には #line 番号ディレクティブは含まれません。デフォルトでは、プリプロセッサは、.i 拡張子の付いたソースファイル名のプリフィックスを使用して、出力ファイルの名前を作成します。-ofile オプションを使用してこれを変更できます。例えば、次のコマンドを実行すると、prog1.iとprog2.iという名前の2つのファイルが作成され、これらのファイルを別のコンパイルの入力として使用できます。

- IA-32 システム: `prompt>icc -P prog1.c prog2.c`
- Itanium ベース・システム: `prompt>ecc -P prog1.c prog2.c`



**注意**

-P オプションを指定した場合は、名前も拡張子も同じファイルは上書きされます。

## -EP の使用

-EP オプションを使用して、出力に `#line` ディレクティブを書き込まないようにプリプロセッサに指示します。-EP オプションは、-E -P オプションと同じ働きをします。

- IA-32 システム: `prompt>icc -EP prog1.c prog2.c`
- Itanium ベース・システム: `prompt>ecc -EP prog1.c prog2.c`

## 前処理済みのソース出力の中にコメントを保存する

前処理済みのソース出力の中にコメントを保存するには、-Cオプションを使用します。次の前処理ディレクティブをコメントにしますが、保存されません。

## 前処理ディレクティブと同等の働き

-A オプション、-D オプションおよび -U オプションを使用して、前処理ディレクティブと同等の働きをするようにできます。

- -A は、`#assert` という前処理ディレクティブと同等の働きをします。
- -D は、`#define` という前処理ディレクティブと同等の働きをします。
- -U は、`#undef` という前処理ディレクティブと同等の働きをします。

## -A の使用

アサーションを作成するには、-Aオプションを使用します。構文: `-Aname[(value)]`。

引数 -	説明
<code>name</code>	アサーションの識別子(名前)を指定します。
<code>value</code>	アサーションの <code>value</code> (値)を指定します。 <code>value</code> を指定する場合は、それを区切る括弧とともに引用符で囲む必要があります。

例えば、識別子を `fruit` とし、値を `orange` と `banana` としてアサーションを作成するには、次のコマンドを実行します。

- IA-32 システム: `prompt>icc -A"fruit(orange,banana)" prog1.c`
- Itanium® ベース・システム: `prompt>ecc -A"fruit(orange,banana)" prog1.c`

## -D の使用

マクロの定義には-Dオプションを使用します。構文: `-Dname[=value]`。

引数 -	説明
<code>name</code>	定義するマクロの名前です。
<code>value</code>	名前として入力する値です。 <code>value</code> を入力しなかった場合、 <code>name</code> は1に設定されます。英数字以外を含んでいる値は引用符

	で囲んでください。
--	-----------

例えば、SIZEという名前のマクロを定義し、値を100にするには、次のコマンドを実行します。

- IA-32 システム: `prompt>icc -DSIZE=100 prog1.c`
- Itanium ベース・システム: `prompt>ecc -DSIZE=100 prog1.c`

-D オプションは、関数を定義するときにも使用されます。例: `icc -D"f(x)=x" prog1.c`

## -U の使用

事前定義済みマクロを削除(取消し)するには、-U オプションを使用します。構文: `-Uname`.

引数 -	説明
<code>name</code>	取り消すマクロの名前



### 注

同じコンパイルで、-D オプションと -U オプションを使用する場合、コンパイラは、コマンドラインで入力した順番ではなく、-U オプションより先に -D オプションを処理します。

## 事前定義済みマクロ

下記の表に、インテル固有の事前定義済みマクロを説明します。「デフォルト」欄では、デフォルトでそのマクロが有効(オン)になるのか、あるいは無効(オフ)になるのかを示します。「アーキテクチャ」欄では、事前定義済みマクロをサポートするインテル・アーキテクチャを示します。ANSI/ISO標準準拠で指定される事前定義済みマクロは、表にはありません。有効な全マクロ定義のリストについては、-E -dM を使用してください。次に例を示します。

- IA-32 システム: `prompt>icc -E -dM prog1.c`
- Itanium® ベース・システム: `prompt>ecc -E -dM prog1.c`

### 事前定義済みマクロ

マクロ名	デフォルト	アーキテクチャ	説明と用途
<code>__ECC=n</code>	n=700	Itanium アーキテクチャのみ	インテル® C++ コンパイラを有効にします。代入した値はコンパイラのバージョンを指します(例えば、700は7.00)。このマクロは、既存システムとの互換性を保つために残っているため、代わりに <code>__INTEL_COMPILER</code> を使用してください。
<code>__EDG__</code>	オン	両方	値 1 を持つよう定義します。
<code>__ELF__</code>	オン	両方	

<code>GXX_ABI_VERSION=100</code>		両方	
<code>__i386</code>	オン	IA-32	
<code>__i386__</code>	オン	IA-32	
<code>i386</code>	オン	IA-32	
<code>__ia64</code>	オン	Itanium アーキテクチャのみ	
<code>__ia64__</code>	オン	Itanium アーキテクチャのみ	
<code>ia64</code>	オン	Itanium アーキテクチャのみ	
<code>__ICC=n</code>	オン n=700	IA-32 のみ	インテル C++ コンパイラを有効にします。代入した値はコンパイラのバージョンを指します(例えば、700は7.00)。このマクロは、既存システムとの互換性を保つために残っているなので、代わりに <code>__INTEL_COMPILER</code> を使用してください。
<code>__INTEL_COMPILER=n</code>	オン n=700	両方	コンパイラのバージョンを定義します。インテル C++ コンパイラ V7.0 に対しては 700 を定義します。
<code>__INTEGRAL_MAX_BITS=n</code>	n=64	Itanium アーキテクチャのみ	<code>__int64</code> 型に対応していることを示します。
<code>__linux</code>	オン	両方	
<code>__linux__</code>	オン	両方	
<code>linux</code>	オン	両方	
<code>__LONG_MAX=n</code>	n=9223372036854775807L	Itanium アーキテクチャのみ	
<code>__LP64</code>	オン	Itanium アーキテクチャのみ	
<code>__lp64</code>	オン	Itanium アーキテクチャのみ	
<code>__LP64__</code>	オン	Itanium アーキテクチャのみ	
<code>__M_IA64=n</code>	n=64100	Itanium アーキテクチャのみ	Itanium アーキテクチャであることを表すため、プリプロセッサ識別子の値を指定します。

<code>__OPTIMIZE__</code>	オン	両方	すべての最適化をオフにすると、無効になります。
<code>_PGO_INSTRUMENT</code>	オフ	両方	<code>-prof_gen</code> または <code>-prof_genx</code> でコンパイルするときに定義されます。
<code>__PTRDIFF_TYPE__</code>	オン	両方	IA-32: <code>__PTRDIFF_TYPE__=int</code> Itanium アーキテクチャ: <code>__PTRDIFF_TYPE__=long</code>
<code>__SIZE_TYPE__</code>	オン	両方	IA-32: <code>__SIZE_TYPE__=unsigned</code> Itanium アーキテクチャ: <code>__SIZE_TYPE__=unsigned long</code>
<code>__unix</code>	オン	両方	
<code>__unix__</code>	オン	両方	
<code>unix</code>	オン	両方	
<code>__USER_LABEL_PREFIX__</code>	オン	両方	

## マクロ定義の抑止

`-Uname` オプションは、指定された`name`について現在有効になっているマクロ定義を抑止するために使用します。`-U` オプションは、`#undef` プリプロセッサ・ディレクティブと同じ機能を果たします。

# コンパイル

## コンパイルの概要

ここでは、コンパイル処理とそれらによって作成される出力を指定するインテル® C++ コンパイラ・オプションについて説明します。デフォルトでは、ソースコードはコンパイラによって実行ファイルに直接変換されます。適切なオプションを指定して、コンパイラに次の出力ファイルを作成させることができます。

- -Pオプションを使用した場合、前処理済みのファイル(.i)が作成されます。
- -Sオプションを使用した場合、アセンブリ・ファイル(.s)が作成されます。
- -cオプションを使用した場合、オブジェクト・ファイル(.o)が作成されます。
- デフォルトでは、実行ファイル(.out)が作成されます。

また、出力ファイルに名前を付けたり、リンクに渡すオプションのセットを指定できます。フェーズ限定オプションを指定すると、コンパイラは、各主要入力ファイルに対して、完了した最後のフェーズの出力を示す別の出力ファイルを作成します。

## コンパイル・オプション

### コンパイルの制御

エラーなしで生成された出力ファイルは、その後コンパイラへの入力ファイルとして使用できます。下の表は、出力を制御する各種オプションについて説明したものです。

1つ前に実行した処理	オプション	コンパイラ入力	コンパイラ出力
前処理	-E、-P、または -EP	ソースファイル	前処理済みファイル (.i ファイル)
コンパイルのみ	-C	• ソースファイル 前処理済みのファイル	オブジェクト・ファイル (.o)までコンパイル します。リンクはしません。
	-S	• ソースファイル 前処理済みのファイル	.s 拡張子の付いた アセンブリ・ファイル を生成し、コンパイル 処理を停止します。
構文チェック	-syntax	• ソースファイル 前処理済みのファイル	診断リスト
リンク	(デフォルト)	• ソースファイル • 前処理済みのファイル • アセンブリ・ファイル • オブジェクト・ファイル ライブラリ	実行ファイル(.out ファイル)

### データ設定の監視

ここでは、インテル・コンパイラから生成されたコードを監視するオプションについて説明します。

### 構造体タグのアライメントの指定

構造体および共用体のアライメント境界を指定する方法は2つあります。

- ソースファイル内にパックプラグマを配置する

- コマンドラインにアライメント・オプションを入力する

どちらでも構造体タグのアライメント境界を変更できます。

構造体宣言に関するアライメント境界を指定するには、`-Zp`オプションを使用します。一般に、条件を狭くするとデータ・セクションが小さくなり、条件を広くすると実行が速くなります。

-Zp オプションの形式は `-Zpn` です。

アライメント境界は、次の値で指定します。

n=1	1バイト
n=2	2バイト
n=4	4バイト
n=8	8バイト
n=16	16バイト

例えば、`prog.c`ファイルのすべての構造体と共用体に対するアライメント条件として、2バイトを指定するには、次のコマンドを使用します。

- IA-32 システム: `prompt>icc -Zp2 prog.c`
- Itanium® ベース・システム: `prompt>ecc -Zp2 prog.c`



注

デフォルトのアライメントでコンパイルされたシステム・ライブラリを使用する場合、アライメントを変更すると問題が生じる可能性があります。

## Itanium ベースのシステムでデノーマル値をゼロにフラッシュする

`-ftz`オプションは、アプリケーションが段階的アンダーフロー・モードの場合に、デノーマル結果をゼロにフラッシュします。このオプションは、デノーマル値がアプリケーション動作に影響を与えない場合に使用します。`-ftz`オプションでデノーマル値をゼロにフラッシュすると、アプリケーションのパフォーマンスが向上する可能性があります。`-ftz`オプションのデフォルト状態はオフです。デフォルトでは、コンパイラは結果を段階的アンダーフローにします。FTZモードをオンにするには、`-ftz`オプションを`main()`関数が含まれているソースに対してのみ使用する必要があります。初期スレッドおよびそのプロセスによってその後に作成されるあらゆるスレッドは、FTZモードで動作します。

`-ftz` スイッチは、`main()` 関数が含まれているソースでのみ使用する必要があります。  
`-ftz` スイッチは、`main()` から起動された処理に対して FTZ モードをオンにします。初期スレッドおよびそのプロセスによってその後に作成されるあらゆるスレッドは、FTZモードで動作します。

## ゼロに初期化される変数を割り当てる

デフォルトにより、明示的にゼロに初期化される変数はBSSセクションに配置されます。しかし



-nobss\_initオプションを使用すると、ゼロで明示的に初期化されるいずれの変数も、必要ならDATAセクションに配置できます。

## 一部の命令についてのデコード処理の誤りを防ぐ(IA-32のみ)

一部の命令は、先頭バイトに0fを含む2バイトのオペコードを持っています。ごくまれに、Pentium® プロセッサは、そのような命令のデコード処理を誤る場合があります。この種の命令のデコード処理の誤りを防ぐためには、-0f\_checkオプションを指定してください。

## リンク

## リンク

ここでは、ツールやライブラリとリンクする方法を設定/変更するのに使うオプションについて説明します。また、ldリンクの出力を定義するオプションについても説明します。リンクの詳細は ld の man page を参照してください。

オプション	説明
-Ldirectory	directoryで指定したディレクトリを検索してライブラリを探すようリンクに指示します。
-Qoption,tool,list	一連のコンパイル処理の中で、アセンブラやリンクなど別のプログラムに引数リストを渡します。
-shared	実行ファイルの代わりに、動的共用オブジェクト(DSO)をビルドするようコンパイラに指示します。
-i_dynamic	インテルが提供するライブラリすべてを動的にリンクするよう指定します。
-static	「動的」とは対照的に、実行ファイルをすべてのライブラリへ静的にリンクします。
-Bstatic	<p>このオプションは、ユーザのコマンドラインの場所に対応するリンク・コマンドラインに配置されます。コマンドラインで渡されるライブラリのリンクの動作を制御するのに、このオプションを使用します。</p> <p>-Bstatic の未使用時:</p> <ul style="list-style-type: none"><li>• /lib/ld-linux.so.2 がリンクされます。</li><li>• libm、libcxa、およびlibc は動的にリンクされます。</li><li>• その他すべての lib は静的にリンクされます。</li></ul> <p>-Bstatic の使用時:</p> <ul style="list-style-type: none"><li>• /lib/ld-linux.so.2 is not linked in</li></ul> <p>その他すべての lib は静的にリンクされます。</p>
-Bdynamic	<p>このオプションは、ユーザのコマンドラインの場所に対応したリンク・コマンドラインに配置されます。コマンドラインで渡されるライブラリのリンクの動作を制御するのに、このオプションを使用します。</p>

## リンクを行わない設定にする

リンクを行わない設定にするには、-cオプションを使用します。例えば、次のコマンドを実行す

ると、file1.oとfile2.oという2つのオブジェクト・ファイルを生成します。

- IA-32 システム: `prompt>icc -c file1.c file2.c`
- Itanium® ベース・システム: `prompt>ecc -c file1.c file2.c`



上のコマンドを実行しても、これらファイルがリンクして1個の実行ファイルにはなりません。

# デバッグ

## デバッグ・オプションの概要

-g オプションを使用して、デバッグ情報を生成します。-g オプションを指定すると、コンパイラは、-O0 を起動して最適化を無効にします。-g または -O0 オプションを指定すると、自動的に -fp オプションが有効になります(IA-32 のみ)。-fp オプションは、汎用レジスタとしての EBP レジスタの使用を無効にします。

-O1、-O2 または -O3 とともに、-g オプションを指定すると、-fp は無効になり、最適化に汎用レジスタとして EBP レジスタを使用できます。ただし、ほとんどのデバッガは、スタック・フレーム・ポインタとして EBP を使用するため、必要な操作をしない限りスタック・バックトレースを生成できません。-fp オプションを使用すると、生成するコードの効率が少し低下する場合があります。

オプション	結果
-g	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O0が有効になり、-fpが有効になります。
-g -O1	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O1による最適化が有効になり、-fpが無効になります。
-g -O2	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O2による最適化が有効になり、-fpが無効になります。
-g -O3	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O3による最適化が有効になり、-fpが無効になります。
-g -O3 -fp	IA-32を対象にコンパイルする場合は、デバッグ情報が生成され、-O3による最適化が有効になり、-fpが有効になります。
-ip	デバッグ用に生成したシンボルと行番号
-ipo	デバッグ用に生成したシンボルと行番号

## デバッグの準備

シンボリック・デバッグ対応のコードの生成をコンパイラに指示するには、-gオプションを使用します。次に例を示します。

- IA-32 システム: `prompt>icc -g prog.c`
- Itanium® ベース・システム: `prompt>ecc -g prog.c`

このコンパイラでは、アセンブリ・ファイルの中にデバッグ情報を生成できません。-gオプションを指定すれば、アセンブリ・ファイルには含まれないデバッグ情報が、生成されるオブジェクト

ト・ファイルには含まれることになります。

## シンボリック・デバッグのサポート

コマンドライン上で `-g` とともに、最適化オプションである `-O1`、`-O2`、または `-O3` を指定すれば、シンボリック・デバッグ対応のコードをコンパイラに生成できます。ただし、次のような予期しない結果になる場合があります。

- `-g` オプションとともに `-O1` オプション、`-O2`、または `-O3` オプションを指定すると、返されるデバッグ情報の一部が最適化の副作用として不正確になることがあります。
- `-O1`、`-O2`または`-O3`オプションを指定すると、`-fp(IA-32のみ)`は無効になります。

## 構文分析およびセマンティクスのみを行う

C++ 言語のエラーについてソースファイルの構文解析が終了して処理を止めるには、`-syntax`オプションを使用します。これは、ソースファイルが構文的または意味的に正しいかどうかをすばやく確認するための方法です。コンパイラから出力ファイルは生成されません。次の例では、コンパイラは `prog.c` という名前のファイルをチェックし、標準エラー出力に診断情報を表示します。

- **IA-32 システム:** `prompt>icc -syntax prog.c`
- **Itanium® ベース・システム:** `prompt>ecc -syntax prog.c`

# 言語適合性

## C の標準規格との適合性

インテル® C++ コンパイラは、次のどちらの C コードにも対応できるよう設定できます。

- -Xc または -ansi オプションを使用する、ANSI 規格に厳密に準拠しているダイアレクト、または
- -Xa オプションを使用する機能拡張型 ANSI C 言語。

このコンパイラは、デフォルトでは、ANSI/ISO 標準に制限されずに拡張を受け入れるように設定します。

## ANSI/ISO標準準拠のC言語について

インテル C++ コンパイラは、C 言語のコンパイルについて定めた ANSI/ISO 標準 (ISO/IEC 9899:1990) に準拠しています。ANSI/ISO 標準に準拠するためには、翻訳処理に要求する最低要件を満たしている必要があります。インテル C++ コンパイラは、翻訳処理に対して ANSI/ISO の要求している最低要件をすべて満たしています。

## コンパイラ付属のマクロ

C 言語の ANSI/ISO 標準に準拠するためには、コンパイラに所定の事前定義済みマクロが付属していなければなりません。次の表は、同標準に従ってインテル C++ コンパイラに組み込まれたマクロの一覧です。

コンパイラには、標準から要求される事前定義マクロのほかにも事前定義マクロをいくつか用意しています。

マクロ	説明
__cplusplus	C++ 変換単位をコンパイルするときに、名前 __cplusplus が定義されます。
__DATE__	Mmm dd yyyy という形式の文字列リテラルとしてコンパイルの日付を表現します。
__FILE__	コンパイルされるファイルの名前を表す文字列リテラル。
__LINE__	現在の行番号。10進数の定数で表現します。
__STDC__	C 変換単位をコンパイルするときに、名前 __STDC__ が定義されます。
__TIME__	コンパイル時刻。hh:mm:ss という形式の文字列リテラルとして表現します。

## C99 サポート

-c99 [-] オプションを使用することによって、このバージョンのインテル C++ コンパイラは、次のC99機能をサポートします。

- 制限付きポインタ(restrict キーワード。-restrictで利用可能)。下記の注を参照してください。
- 可変長配列
- 柔軟な配列メンバ
- 複素数のサポート(\_Complex キーワード)
- 16進浮動小数点定数
- コンパウンド・リテラル
- 指定初期化子
- 混在する宣言とコード
- 可変長引数を持つマクロ
- インライン関数(inline キーワード)
- ブール型(\_Bool キーワード)



注

-restrictオプションで、ANSI 標準規格で定義されたrestrict キーワードを認識します。restrict キーワードでポインタを限定することによって、ポインタ経由でアクセスされるオブジェクトが、与えられたスコープ内では、当該ポインタのみによってアクセスされることを宣言できます。この宣言が正しい場合のみこの restrict キーワードを使用することが重要です。この場合、プログラムの正当性には影響を与えませんが、より良い最適化が可能になる場合があります。

これらの機能はサポートされていません。

- #pragma STDC FP\_CONTRACT
- #pragma STDC FENV\_ACCESS
- #pragma STDC CX\_LIMITED\_RANGE
- long double (128ビット表記)

## C++の標準規格との適合性

インテル® C++ コンパイラは、C++ 言語の ANSI/ISO 標準(ISO/IEC 14882:1998)に準拠していますが、テンプレート用 export キーワードは実装していません。

# 最適化

## 最適化レベル

### 最適化レベルの概要

下の表は、オプション -O1、-O2、または -O3 を使用した際にインテル® C++ コンパイラが適用する最適化項目です。

最適化項目

定数伝播

コピー伝播

不要コード排除

グローバル・レジスタの確保

命令スケジューリング

ループのアンロール(-O2、-O3 オプションのみ)

ループ不変コードの移動

部分冗長性の排除

ストレングスのレダクション／誘導変数の簡略化

変数の名前変更

例外処理の最適化

末端再帰

ピープホールの最適化

構造体代入の低下および最適化

不要ストアの排除

### 最適化レベルの設定

インテル・アーキテクチャによって、最適化は異なる効果が得られます。目的のアーキテクチャに合った最適化を指定するには、以下の表を参照してください。

#### Itanium® アーキテクチャ用インテル® C++ コンパイラ

オプション	効果
-O1	ソフトウェアによるパイプライン化をオフにすることによって、コードのサイズを最適化します。ループのアンロールおよびソフトウェアによるパイプライン化を除く -O と同じ最適化を有効にします。-O および -O2 は、ソフトウェアによるパイプライン化を有効にします。ほとんどの場合、-O1 よりも -O2 または -O1 のほうをお勧めします。



## IA-32 コンパイラ

オプション	効果
-O, -O1, -O2	速度について最適化します。-fp オプションを無効にします。-O2 オプションはデフォルトではオンです。組込み関数の認識は無効です。
-O3	-O2オプションに、最適化項目を加えて最適化を実行します。ループとメモリアクセスの変換が行われない限り、パフォーマンスが高くなることは保証されません。-axK と -xK オプション(IA-32 のみ)を組み合わせてこのオプションを使用すると、-O2 のときよりも詳細にデータ依存性解析が実行されます。そのためコンパイル時間が長くなる場合があります。

## IA-32 と Itanium アーキテクチャ用インテル C++ コンパイラ

オプション	効果
-O2	<p>デフォルトでは、このオプションがオンになっています。-O2組込み関数のインライン化がオンになります。次の例では、ゼロ除算例外が発生したかどうかをテストします。</p> <ul style="list-style-type: none"> <li>• 定数伝播</li> <li>• コピー伝播</li> <li>• 不要コード排除</li> <li>• グローバル・レジスタの確保</li> <li>• 命令スケジューリング</li> <li>• ループのアンロール</li> <li>• コード選択の最適化</li> <li>• 部分冗長性の排除</li> <li>• ストレングスのレダクション／誘導変数の簡略化</li> <li>• 変数の名前変更</li> <li>• 例外処理の最適化</li> <li>• Tail recursions</li> <li>• ピープホールの最適化</li> <li>• 構造体代入の低下および最適化</li> </ul> <p>不要ストアの排除</p>
-O3	-O2オプションに、例えば次の最適化項目を加えて最適化を実行します。プリフェッチ、スカラ置換、ループ変換。ループとメモリアクセスの変換が行われない限り、パフォーマンスが高くなることは保証され

	ません。アプリケーションの時間を計るには「アプリケーションの時間測定」を参照してください。
--	---

## 最適化の範囲の制限

最適化の範囲を絞ったり、最適化を禁止したりする場合は、以下の各オプションを使います。

オプション	説明
-O0	最適化を一切禁止します。
-mp1	浮動小数点の精度を高くします。速度に与える影響は -mp より低くなります。
-fp IA-32 のみ	汎用レジスタとしての EBP レジスタの使用を無効にします。
-prec_div IA-32 のみ	浮動小数点除算から乗算へ変換する最適化処理を無効にします。
-fp_port IA-32 のみ	代入 & 型キャスト時に浮動小数点の結果を丸めます(実行速度に少し影響があります)。
-ftz [-] Itanium® ベース・システムのみ	異常値の結果を0にフラッシュを有効 [無効] にします。
-IPF_fma [-] Itanium ベース・システムのみ	浮動小数点乗算と加算/減算の組み合わせを有効[無効]にします。
-IPF_fltacc [-] Itanium ベース・システムのみ	浮動小数点の精度に影響する最適化を有効 [無効] にします。
-IPF_flt_eval_method0 Itanium ベース・システムのみ	プログラムにより指定された精度で浮動小数点オペランド評価が働きます。
-IPF_fp_speculation<mode> Itanium ベース・システムのみ	次のモード条件で浮動小数点の推測を有効にします。 <ul style="list-style-type: none"> <li>• fast - 浮動小数点演算を推測します</li> <li>• safe - 安全な場合のみ推測します</li> <li>• strict-off と同じです</li> </ul> off - 浮動小数点演算の推測を無効にします



注

#pragma optimizeを使用して、特定の関数のすべての最適化をオフにできます。次の例では、関数foo()ですべての最適化がオフになります。

```
#pragma optimize("", off)
foo() {
```

```
...  
}
```

#pragma optimize の有効な2番目の引数は"on"または"off."です。"on" の引数を使用すると、foo() 関数は、残りのプログラム同じように最適化してコンパイルされます。コンパイラは、最初の引数の値を無視します。

# 浮動小数点の最適化

## 浮動小数点演算の精度

### IA-32、Itanium® ベース・システムに使用する各種オプション

#### -mp オプション

-mp オプションを指定すると、精度は宣言された水準が保たれます。また、浮動小数点演算の処理は、ANSI および IEEE のそれぞれ当該規格にほぼ準拠する結果となります。このオプションは、ほとんどのプログラムのパフォーマンスに不利に働きます。目的のアプリケーションにこのオプションが必要かどうかよくわからない場合は、このオプションを指定した場合と指定しない場合で実際にプログラムをコンパイルし、実行して、パフォーマンスと精度の両方に対する効果を評価してみてください。-mp オプションを指定すると、プログラムのコンパイルに次の影響が生じます。

- 浮動小数点型として定義されたユーザ変数は、レジスタに割り当てられません。
- 式がスピル(レジスタからメモリへ移動)する場合、64ビット(倍精度)ではなく、80ビット(拡張精度)としてスピルされます。
- 浮動小数点演算の比較は、NaN (非数)の動作以外は IEEE 754 の仕様に準拠します。
- 演算は、コード内で指定したとおりに実行します。例えば、除算が「逆数の乗算」に変更されることはありません。
- コンパイラは関連付けを変更せずに、指定した順序で浮動小数点演算を実行します。
- 浮動小数点値に対しては、定数の畳込みによる最適化は実行しません。定数の畳込みとは、1を掛けたり、1で割ったり、0の足し引きをしたりといった計算を省くことです。定数の畳込みを実行しないので、例えば、0.0の足し算についても記述したとおりに実行します。また、コンパイル時の浮動小数点演算も実行しません。これは、浮動小数点例外についてもその状態を変えないようにするためです。
- 浮動小数点演算はANSI Cに準拠します。float型およびdouble型への代入を行うと、その精度は、80ビット(拡張型)から32ビット(float)か64ビット(double)に丸められます。-mpを指定しなければ、精度が丸められずに変数が再使用される場合もあります。
- -nolib\_inline オプションが使用されます。これは、関数のインライン展開を無効にするオプションです。

**注:** -nolib\_inline および -mp オプションは、-Xc (ANSI C 完全準拠) オプションを選択するとデフォルトで有効になります。

#### -mp1 オプション

浮動小数点の精度を高くするには、-mp1 オプションを使用します。-mp1 は -mp に比べると、

禁止される最適化処理が少なく、またパフォーマンスに与える影響も小さくなります。

## IA-32 のみのオプション



### 注意

デフォルトの精度制御方式または丸めモードを変更すると(例えば、`-pc32` フラグの使用やユーザの介入によって)、いくつかの算術関数で返された結果に影響する場合があります。

## `-long_double` オプション

`long double`型のサイズを80ビットに変更するには、`-long_double`を使用します。インテル・コンパイラのデフォルトの`long double`型は、サイズが`double`型と同じ64ビットになっています。このオプションを使用すると、このオプションを使用しないでコンパイルした他のファイルと互換性のない部分やライブラリ・ルーチンへの呼び出し命令に整合しない部分が発生します。したがって、このオプションを指定してコンパイルをするときは、`long double`型の変数はファイル単体の中だけのローカル変数にするのを勧めます

## `-prec_div` オプション

`-xK`や`-xW`などを指定して最適化する場合は、浮動小数点の除算が、分母の逆数による乗算に変更されます。計算速度を上げるため、例えば $A \div B$ を $A \times (1/B)$ として計算します。ただし、 $B$ が $2^{126}$ より大きい場合は $1/B$ の値が0になります。 $1/B$ の値を維持しなければならない場合は、`-prec_div`を使って、浮動小数点の除算を乗算に変換する最適化処理を禁止してください。`-prec_div`を指定すると、精度は上がりますが、若干パフォーマンスが落ちます。

## `-pcn` オプション

浮動小数点の仮数の精度を制御するには、`-pcn`オプションを使用します。浮動小数点アルゴリズムの中には、浮動小数点値の仮数部または小数部の精度に影響を受けやすいものがあります。例えば、除算や平方根の計算のように反復処理が多いものは、`-pcn`オプションを使用して精度を下げると計算が速くなる場合があります。 $n$  は次のいずれかの値に設定すると、仮数はそれぞれ示したビット数に丸められます。

- `-pc32`: 24 ビット (単精度) -- 上記の注意を参照してください。
- `-pc64`: 53 ビット (単精度)
- `-pc80`: 64 ビット (単精度) -- デフォルト

$n$ のデフォルト値は倍精度を示す「80」です。このオプションを指定すると、全面的な最適化を実行できます。このオプションで影響されるのは浮動小数点値の小数部だけなので、`-Op`オプションとは異なり、パフォーマンスに悪影響は与えません。指数の部分は影響を受けません。指数の部分は影響を受けません。`-pcn`オプションを指定すると、`main()`関数がコンパイルされるときに浮動小数点の精度の制御方式が変わります。`-pcn`を使用するプログラムは入口点として`main()`を使用しなければならず、また`main()`を含むファイルは`-pcn`を指定してコンパイルしなければなりません。

## `-rcd` オプション

浮動小数点から整数への変換を必要とするコードのパフォーマンスを改善するには、`-rcd` オプションを使用します。これは、丸めモードの変更を制御して最適化を行います。デフォルトでは、浮動小数点の丸めモードは「最近値への丸め」となっています。つまり、各値は浮動小数点の計算中に丸められます。ただし、C言語の場合は、整数への変換時に浮動小数点値を切り捨てる必要があります。これを行うには、コンパイラは、浮動小数点から整数へ変換する前に丸めモードを「切り捨て」方式に変更し、変換が終わったらまた元に戻すという処理をしなければなりません。`-rcd` オプションを指定すると、浮動小数点から整数への変換をはじめとして浮動小数点の計算すべてについて、丸めモードは「切り捨て」方式に変更できなくなります。このオプションを指定するとパフォーマンスは改善されるかもしれませんが、浮動小数点から整数への変換処理についてはC言語のセマンティクスに適合しなくなります。

## **-fp\_port オプション**

`-fp_port` オプションは、代入と型キャスト時に浮動小数点の結果を丸めます。これは、速度に影響する場合があります。

## **IA-32、Itanium® ベース・システムに使用する浮動小数点演算オプション**

次のオプションは、Itanium® ベースのシステム上で浮動小数点計算用にコンパイラの最適化の制御をできるようにします。

- `-ftz [-]`
- `-IPF_fma [-]`
- `-IPF_fp_speculationmode`
- `-IPF_flt_eval_method0`
- `-IPF_fltacc [-]` (Default: `-IPF_fltacc -`)

### **浮動小数点積和/積差演算の縮約**

`-IPF_fma [-]` は、浮動小数点積和/積差演算の1つの演算への縮約記を有効[無効]にします。`-mp` が指定されない限りは、コンパイラは可能な限りこれらの演算を縮約します。`-mp` オプションは、縮約を無効にします。`-IPF_fma` および `-IPF_fma-` は、デフォルトのコンパイラ動作を無効にするのに使用されます。例えば、`-mp` および `-IPF_fma` は、コンパイラの縮約演算を有効にします。

```
prompt>ecc -mp -IPF_fma prog.c
```

### **FP スペキュレーション**

`-IPF_fp_speculationmode` は、次のいずれかの`mode`で、コンパイラの浮動小数点演算のスペキュレーションを設定します。

- `fast`: コンパイラの浮動小数点演算のスペキュレーションを設定します。
- `safe`: 安全な場合のみコンパイラの浮動小数点演算のスペキュレーションを有効にします。

- `strict`: 浮動小数点演算のスペキュレーションを無効にします。
- `off`: 浮動小数点演算のスペキュレーションを無効にします。

## FP 演算の評価

`-IPF_flt_eval_method0` は、プログラムで宣言された変数型により指定される精度での浮動小数点演算を含む式を評価するようにコンパイラに指示します。

## FP 結果の精度制御

`-IPF_fltacc [-]` は、浮動小数点の精度に影響を与える最適化を有効 [無効] にします。デフォルト(`-IPF_fltacc-`)では、コンパイラは浮動小数点の精度を低下する最適化を適用する場合があります。浮動小数点の精度を高めるのに `-IPF_fltacc` または `-mp` を使用できますが、いくつかの最適化を無効にしなくてはなりません。

# 特定のプロセッサに合わせて最適化を行う

## プロセッサの最適化

### IA-32 にのみ該当するプロセッサの最適化

-tpp{5|6|7} オプションは、特定のインテル・プロセッサに合わせてアプリケーションのパフォーマンスを最適化します。最適化を行ったアプリケーションは、下記の表で記述されている他のプロセッサ上でも実行することができます。インテル® C++ コンパイラには、gcc\* と互換性のある -tpp オプションが含まれています。これらのオプションは、「gcc のバージョン」項目に記述されています。

オプション	gcc* バージョン	最適化の対象
-tpp5	-mcpu=pentium	Pentium® プロセッサ
-tpp6	-mcpu=pentiumpro	Pentium Pro プロセッサ、 Pentium II プロセッサ、 および Pentium III プロセッサ
-tpp7	-mcpu=pentium4	Pentium 4 およびインテル ® Xeon™ プロセッサ



#### 注

iccまたはicpcを実行した際、-tpp7オプションはデフォルトでオンに設定されます。

#### 例

次のコンパイルによる結果はすべて Pentium 4 およびインテル Xeon プロセッサ向けに最適化されます。これらの結果は Pentium、Pentium Pro、Pentium II および Pentium III プロセッサ上でも実行することができます。

```
prompt>icc prog.c
```

```
prompt>icc -tpp7 prog.c
```

```
prompt>icc -mcpu=pentium4 prog.c
```



## プロセッサの最適化(Itanium® ベース・システムのみ)

-tpp{1|2} オプションは、アプリケーションのパフォーマンスを指定したインテル Itanium プロセッサ向けに最適化します。最適化を行ったアプリケーションは、下記の表で記述されているプロセッサ上でも実行することができます。インテル C++ コンパイラには、gcc\* と互換性のある -tpp オプションが含まれています。これらのオプションは、「gcc のバージョン」項目に記述されています。

オプション	gcc* バージョン	最適化の対象
-tpp1	-mcpu=itanium	Itanium プロセッサ
-tpp2	-mcpu=itanium2	Itanium 2 プロセッサ



### 注

eccまたはecpcを実行した際、-tpp2オプションはデフォルトでオンに設定されます。

### 例

次のコンパイルによる結果はすべて Itanium 2 プロセッサ向けに最適化されます。この結果は Itanium プロセッサ上でも実行することができます。

```
prompt>ecc prog.c
```

```
prompt>ecc -tpp2 prog.c
```

```
prompt>ecc -mcpu=itanium2 prog.c
```

## プロセッサ固有の最適化(IA-32のみ)

-x{i|M|K|W} オプションは、プログラムを実行するプロセッサ用の必要最小限のプロセッサ命令セットを指定することにより、プログラムを特定のIA-32プロセッサ上で動作するようにします。生成するコードには、指定したプロセッサ命令の無条件の使用を取り込みます。インテル® C++ コンパイラには、-x{i|M|K|W} オプションの gcc\* 互換バージョンが含まれています。"gcc バージョン"の欄に、これらのオプションの一覧を示します。

オプション	gcc* バージョン	最適化の対象
-xi	-march=pentiumpro	インテル® Pentium® Pro プロセッサと Pentium II プロセッサ(CMOV、FCMOV および FCOMI 命令を使用)
-xM	-march=pentiumii	MMX® テクノロジ Pentium プロセッサ(iの命令セットは含まない)
-xK	-march=pentiumiii	ストリーミングSIMD拡張命令(i とMの命令セットを含む)対応 Pentium III プロセッサ
-xW	-march=pentium4	ストリーミングSIMD拡張命令2(i、M、および Kの命令セットを含む)対応 Pentium 4 プロセッサおよびインテル® Xeon™ プロセッサ

### 例

下記は、Kの命令セットをサポートするプロセッサ用に prog.c をコンパイルします。最適化されたバイナリ・ファイルを正常に実行させるには、インテル Pentium III プロセッサ、Pentium 4 プロセッサまたはインテル Xeon プロセッサ上で行う必要があります。Pentium プロセッサ、Pentium Pro プロセッサ、Pentium II プロセッサまたは MMX テクノロジ Pentium プロセッサでは、正常に実行されない可能性があります。

```
prompt> icc -xK prog.c
```



#### 注意

-x{i|M|K|W}を使用してコンパイルされたプログラムを、指定された命令セットをサポートしていないプロセッサ上で実行すると、不正命令例外か、他の予期しない動作が発生する場合があります。

## 自動CPUディスパッチ(IA-32のみ)

-ax{i|M|K|W} オプションは、コンパイラに対して特定のプロセッサごとに命令を使い分ける別々の関数を生成する機会があるかどうかの確認を求めます。(次の表を参照してください。)そしてコンパイラはこのような機会を見つけると、プロセッサ固有の関数バージョンを生成することがパフォーマンスの向上につながるかをチェックします。パフォーマンスが向上すると判明した場合、コンパイラはプロセッサ固有の関数バージョンと汎用バージョンの両方を生成します。汎用バージョンはどのIA-32プロセッサ上でも実行できます。

プログラムの実行時に、プログラムが現在実行されているプロセッサに応じて、この 2つのバージョンのどちらを実行するか選択されます。このため、プログラムはそれ以前のプロセッサでも正常に動作しながら、新しいプロセッサにおいてパフォーマンスの大幅な向上を実現できます。

ただし、-ax{i|M|K|W}を使用した場合には、次のような欠点があります。

- プロセッサ固有のコード・バージョンと汎用のコード・バージョンの両方が含まれるため、コンパイルされたバイナリのサイズが大きくなります。
- 実行するコードを決定するランタイム・チェックによって、パフォーマンスに影響があります。



#### 注

このオプションでコンパイルするアプリケーションは、どのIA-32プロセッサ上でも実行できます。ただし、このようなコンパイル処理は、-xオプションを使用したコンパイル処理時に加えられる専用コードと同様の制約を受けます。

オプション	最適化の対象
-axi	(CMOV命令、FCMOV命令、およびFCOMI命令を使用する)インテル® Pentium® Pro プロセッサと Pentium II プロセッサ
-axM	MMX® テクノロジ Pentium プロセッサ
-axK	ストリーミングSIMD拡張命令(i とMの命令セットを含む)対応 Pentium III プロセッサ
-axW	ストリーミングSIMD拡張命令2(i、M、および K の命令セットを含

	む)対応 Pentium 4 およびインテル® Xeon™ プロセッサ
--	-------------------------------------

例

下記のコンパイルは、次のような1つの実行ファイルを生成します。

- あらゆるIA-32プロセッサ上で使用できる汎用バージョン。
- パフォーマンスが向上する場合、Pentium III プロセッサ用に最適化されたバージョン。
- パフォーマンスが向上する場合、Pentium 4 プロセッサおよびインテル Xeon プロセッサ用に最適化されたバージョン。

```
prompt>icc -axKW prog.c
```

## プロセッサの最適化と自動CPUディスパッチを組み合わせる方法(IA-32のみ)

次の表は、各種のオプションを組み合わせでコンパイルしたアプリケーションによって、最適化の対象となるプロセッサと使用可能なプロセッサのさまざまな組み合わせに対応する方法を示しています。

最適化の 対象となる プロセッサ	使用可能なプロセッサ					
	インテル® Pentium® プロセッサ	MMX® テ クノロジ Pentium プロセッサ	Pentium Pro プロセ ッサ	Pentium II プロセッサ	Pentium III プロセッサ	Pentium 4 プロ セッサおよびイン テル® Xeon™ プロセッサ
Pentium プロセッサ	-tpp5	-tpp5	-tpp5	-tpp5	-tpp5	-tpp5
MMX テク ノロジ Pentium プロセッサ	N/A	-tpp5, -xM	-tpp5	-tpp5, -xM	-tpp5, -xM	-tpp5, -xM
Pentium Pro プロセ ッサ	N/A	N/A	-tpp6, - xi	-tpp6, - xi	-tpp6, - xi	-tpp6, -xi
Pentium II プロセッサ	N/A	N/A	N/A	-tpp6, - xiM	-tpp6, - xiM	-tpp6, -xiM
Pentium III プロセッサ	N/A	N/A	N/A	N/A	-tpp6, - xK	-tpp6, -xK
Pentium 4 プロセッサ およびイン テル Xeon プロセッサ	N/A	N/A	N/A	N/A	N/A	-tpp7, -xW

例

次のようなアプリケーションが必要であるとします。

- MMX テクノロジの拡張命令を常に必要とする
- アプリケーションを実行するプロセッサが Pentium Pro プロセッサをサポートしている場合は、Pentium Pro プロセッサを使用する。
- プロセッサがPentium Pro プロセッサをサポートしていない場合は、Pentium Pro プロセッサを使用しない。

次のコマンドラインを使用して、このようなアプリケーションを生成します。

```
prompt>icc -O2 -xM -axi prog.c
```

上の -xMはアプリケーションをMMX テクノロジ Pentium プロセッサ以上で動作するよう制限します。これ以前の世代のインテル 32ビット・プロセッサ上でもアプリケーションを実行する

には、次のコマンドラインを使用します。

```
prompt>icc -O2 -axiM prog.c
```

このコンパイルは、拡張 `i` と `M` の両方に対応したプロセッサ用に最適化されたコードを生成します。しかし、コンパイルされたプログラムはどの IA-32 プロセッサ上でも動作します。

## プロシージャ間の最適化

### プロシージャ間の最適化

-ip と -ipo を使用してプロシージャ間の最適化(IPO)を有効にすれば、コンパイラにコードを分析させて、次の表に示す最適化がどの部分に適用するかを調べられます。

#### IA-32 および Itanium® ベース・アプリケーション

最適化項目	影響を受ける部分
関数のインライン展開	呼び出し、ジャンプ、分岐、ループ
プロシージャ間での定数伝播	引数、グローバル変数、戻り値
モジュール・レベルでの静的変数の監視	現状以上の最適化、ループ不変コード
不要コードの排除	コードのサイズ
関数特性の伝播	呼び出し命令の削除と呼び出し命令の移動
マルチファイルの最適化	-ip と同じ部分に影響を与えますが、複数のファイル全体にわたって最適化を行います。

#### IA-32 アプリケーションのみ

最適化項目	影響を受ける部分
レジスタ内での引数の受け渡し	呼び出し、レジスタの使用
ループ不変コードの移動	現状以上の最適化、ループ不変コード

関数のインライン展開は、プロシージャ間の最適化機構によって実行する主な最適化機能のうちの1つです。コンパイラは、頻繁に実行する関数呼び出しが存在していると判断した場合、その呼び出し命令を、当該関数自体のコードに置き換えるがあります。(呼出しのインライン展開)

-ip オプションを指定すると、現在のソースファイル内で定義している関数内での呼び出し命令について関数のインライン展開を実行します。ただし、-ipo オプションを使用してマルチファイルIPOを指定すると、別々のファイル内で定義している関数内での呼び出し命令について関数のインライン展開が実行されます。このため、ipo を指定する場合、アプリケーション全体または複数の関連するソースファイルを一緒にコンパイルすることは重要です。

IPOによる最適化は、デフォルトでは実行されません。

# マルチファイルIPO

## マルチファイル IPO の概要

マルチファイル IPO とは、最適化の余地がないかどうかについての情報を、マルチファイル・プログラムの個々のプログラム・モジュールから得るものです。コンパイラはこの情報を使用して、複数のモジュール全体にわたって最適化を行います。

プログラムを構築する工程はコンパイルとリンケージに分かれています。マルチファイル IPO の処理内容は、実行するのがコンパイルかリンケージかによって違ってきます。

### コンパイル・フェーズ

各ソースファイルがコンパイルされるたびに、そのソースコードの中間表現( IR )が、仮のオブジェクト・ファイルに格納します。このオブジェクト・ファイルには、最適化に使うサマリ情報が含まれています。

特に指定しない限り、コンパイラは、マルチファイルIPOのコンパイル・フェーズの最中に、仮のオブジェクト・ファイルをいくつか生成します。実際のオブジェクト・ファイルの代わりに仮のオブジェクト・ファイルを生成すると、マルチファイル IPO のコンパイル・フェーズに要する時間が短くなります。各オブジェクト・ファイルには、それと対応するソースファイルの IR を含んでいますが、実際のコードもデータも含んでいません。この仮のオブジェクトは、`-ipo` オプションと`icc` を使用するか`xild`ツールを使用してコンパイラにリンクする必要があります。



注

`icc`、`-ipo`、`xild`のいずれかを使用して仮のオブジェクト・ファイルをリンクしないとリンケージ・エラーが発生します。場合によっては、仮のオブジェクト・ファイルを使用できない場合があります。詳細については、「実際のオブジェクト・ファイルを生成する」を参照してください。

### リンケージ・フェーズ

`-ipo`を指定すると、コンパイラはリンカの直前に起動します。IRを含んでいるオブジェクト・ファイルすべてを対象にしてマルチファイルIPOが実行されます。



注

スタティック・ライブラリ( `.a` ファイル)についてはマルチファイル IPO は利用できません。詳細については、「実際のオブジェクト・ファイルを生成する」を参照してください。

`-ipo`を指定すると、ドライバとコンパイラとで自動的にプログラム全体を検出できるようになります。プログラム全体を検出すれば、プロシージャ間の定数伝播、スタックフレームのアライメント、データ・レイアウト、共通ブロックのパディングといった最適化をもっと効率よく実行しますが、削除される不要な関数の数も増えます。このオプションは安全です。

## 実際のオブジェクト・ファイルを生成する

状況によっては、`-ipo` を使用して実際のオブジェクト・ファイルを生成する必要があります。IPOを行う場合、仮のオブジェクト・ファイルではなく実際のオブジェクト・ファイルを強制的に生成するのに、`ipo_obj` オプションと `-ipo` オプションを組み合わせで使用します。

次の場合は、`-ipo_obj` を使用する必要があります。

- `-ipo` を指定したコンパイル・フェーズで生成したオブジェクトを、`xild` ツールか `xild -lib` ツールでスタティック・ライブラリに格納する場合。スタティック・ライブラリについてはマルチファイル IPO を利用できないため、スタティック・ライブラリはすべてリンクに渡されます。仮のオブジェクト・ファイルを含んでいるスタティック・ライブラリにリンクするとリンケージ・エラーが発生します。`-ipo_obj` を指定すると、スタティック・ライブラリの中で使用できるオブジェクト・ファイルが生成されます。
- 一方、`xild`か`xild -lib`を使って作成したスタティック・ライブラリは、普通のライブラリとして機能します。
- `-ipo`を指定したコンパイル・フェーズで生成したオブジェクトを、`-ipo` オプションと `xild` を使用しないでリンクする場合
- `-ipo`を指定してコンパイルを行っている最中に、`-S` を用いてソースファイルごとにアセンブリ・リストを生成したい場合。`-ipo_obj`とではなく`-S`と一緒に`-ipo`を使用すると、警告メッセージが出て、コンパイルしたソースファイルごとに空のアセンブリ・ファイルが生成されます。

## マルチファイル IPO の実行ファイルを作成する

ここでは、IA-32 と Itanium® ベース・システムを対象とするコンパイルのマルチファイル IPO の実行ファイルを作成する方法を説明します。

### IA-32の場合の手順

`-ipo` でソース・モジュールのコンパイルとリンクを別々にする場合：

1. 次のように`-ipo`を使用してモジュールをコンパイルします。  

```
prompt>icc -ipo -c a.c b.c c.c
```
2. `.o` ファイルの生成後にコンパイルを中止するときは `-c` オプションを使用してください。どのオブジェクト・ファイルにも、対応するソースファイルの IR を含んでいます。上記手順の実行結果を使用して次のようにすると、プロシージャ間の最適化が実行できます。  

```
prompt>icc -ipo a.o b.o c.o
```

IR を含むモジュールにだけマルチファイル IPO が適用され、該当するものがない場合は、オブジェクト・ファイルはリンク段階に渡されます。効率を上げたいときは、次のように手順1と2を組み合わせてください。

```
prompt>icc -ipo a.c b.c c.c
```

### Itanium ベース・システムの場合の手順

`-ipo` でソース・モジュールを別々にコンパイルとリンクをする場合

1. 次のように`-ipo`を使用してモジュールをコンパイルします。  

```
prompt>ecc -ipo -c a.c b.c c.c
```
2. `.o`ファイルの生成後にコンパイルを中止するときは`-c`を使用してください。どのオブジェクト・ファイルにも、対応するソースファイルの IR を含んでいます。上記手順の実行



結果を使用して次のようにすると、プロシージャ間の最適化が実行できます。

```
prompt>ecc -ipo a.o b.o c.o
```

IR を含むモジュールにだけマルチファイル IPO が適用され、該当するものがない場合は、オブジェクト・ファイルはリンク段階に渡されます。効率を上げたいときは、次のように手順1と2を組み合わせてください。

```
prompt>ecc -ipo a.c b.c c.c
```

プロファイル情報を用いたマルチファイルIPOを行うと、さらに最適化を図ることができます。この方法は、この節の後半にある「プロファイルに基づく最適化の例」を参照してください。

## xild を使用してマルチファイル IPO 実行ファイルを作成する

インテルのリンカ xild は次のように動作します。

- IRを含むオブジェクトが見つかり、インテル・コンパイラが起動し、マルチファイル IPOを実行します。
- GNU リンカ ld を起動して、アプリケーションをリンクします。

xild のコマンドラインの構文は次のとおりです。

```
prompt>xild [<options>] <LINK_commandline>
```

各アイテムの意味は次のとおりです。

- [<options>] (オプション)には、あらゆる gcc リンカ・オプション、または xild でのみサポートされるオプションを含めることができます。
- <LINK\_commandline> は、ld リンカへの有効な引数のセットを含むリンカ・コマンドラインです。

マルチファイル IPO の実行ファイルを ipo\_file に格納するには、-ofilename オプションを使用します。例：

```
prompt>xild oipo_file a.o b.o c.o
```

xild は、IR を含むオブジェクトの IPO を実行するためにインテル・コンパイラを呼び出し、リンクされるオブジェクトの新しい一覧を生成します。そして、xild は ld を呼び出して、新しいリストで指定されたオブジェクト・ファイルにリンクして、-ofilename オプションで指定された ipo\_file 実行ファイルを生成します。



-ipoオプションを使用した場合、コマンドラインにオブジェクト・ファイルとリンカ引数を複数入力すると、その並び順が変わってしまうことがあります。したがって、コマンドラインに引数を複数入力する場合は、その順番を崩してはならないプログラムに対して -ipo を使用すると、プログラムが誤動作することがあります。

## 使用規則

次のような場合、インテルリンカ xild を使用して、アプリケーションをリンクする必要があります。

- 有効なマルチファイル IPO でソースファイルをコンパイルしました。コマンドライン・オ

プシオン `-ipo` を指定して、マルチファイル IPO を有効にします。

- 通常、アプリケーションをリンクするには、`ld` を起動します。

## xild オプション

`xild` がサポートしている追加オプションは、マルチファイルIPOの結果を検証するのに使用できます。次の表で、これらのオプションを説明します。

オプション	説明
<code>-ipo_o[file.s]</code>	マルチファイルIPOコンパイルのアセンブリ・ファイル を生成します。リストファイルの名前(オプション)また はファイルを置くディレクトリ(バックスラッシュを付け る)を指定できます。デフォルトのリスト名は、 <code>ipo_out.s</code> です。
<code>-ipo_o[file.o]</code>	マルチファイルIPOコンパイルのオブジェクト・ファイル を生成します。オブジェクト・ファイルの名前(オプショ ン)またはファイルを置くディレクトリ(バックスラッシュ を付ける)を指定できます。デフォルトのオブジェクト・ ファイル名は、 <code>ipo_out.o</code> です。
<code>-ipo_fcode-asm</code>	アセンブリ・ファイルにコードバイトを追加します。
<code>-ipo_fsource-asm</code>	アセンブリ・ファイルに高レベル・ソース・コードを追加 します。
<code>-ipo_fverbose-asm,</code> <code>-ipo_fnoverbose-asm</code>	<code>xild</code> のアセンブリ・ファイルに使用されたバージョ ンおよびオプションを含むコメントの挿入を有効、無効 にします。

## IPOオブジェクトからライブラリを作成する

通常、ライブラリの作成には`ar`などのライブラリ・マネージャを使います。ライブラリ・マネージャは、オブジェクト・リストを読み取り、そのオブジェクトを、次のリンク段階で使用するライブラリに挿入します。

```
prompt>xiar cru user.a a.o b.o
```

上記にすると、`a.o`と`b.o`とを含んだライブラリ`user.a`が作成されます。

ただし、`-ipo -c`を使って作成したオブジェクトには、有効なオブジェクトが含まれず、そのオブジェクト・ファイルの中間表現(IR)だけを含みます。次に例を示します。

```
prompt>icc -ipo -c a.c b.c
```

上記にすると、リンク時のコンパイルに使われるIRだけを含んだ`a.o`と`b.o`が生成されます。ライブラリ・マネージャでは、このオブジェクトはライブラリには挿入できません。

この場合はライブラリ・ドライバ`xild -ar`を使用しなければなりません。このドライバは、オブジェクト・ファイルに保存しているIR上にコンパイラを呼び出し、ライブラリに挿入できる有効なオブジェクトを生成します。

```
prompt>xild -lib cru user.a a.o b.o
```

「xldl を使用してマルチファイル IPO 実行ファイルを作成する」を参照してください。

## マルチファイル IPO の効果を分析する

-ipo\_cおよび-ipo\_Sというオプションは、マルチファイルIPOの効果を分析する場合や、完成したプログラムになる前の複数のモジュール間でマルチファイルIPOの実験をする場合に便利です。

複数のファイル全体にわたって最適化を行い、オブジェクト・ファイルを1個生成するには、-ipo\_cオプションを使用します。このオプションを使用すると、-ipoの解説で述べた最適化処理を行います。最後のリンク段階に進む前にその処理は停止し、最適化されたオブジェクト・ファイルはそのまま残ります。このファイルのデフォルト名はipo\_out.oです。

複数のファイル全体にわたって最適化を行い、アセンブリ・ファイルを1個生成するには、-ipo\_Sオプションを使用します。このオプションを使用すると、-ipoの解説で述べた最適化処理を行います。最後のリンク段階に進む前にその処理は停止し、最適化されたアセンブリ・ファイルはそのまま残ります。このファイルのデフォルト名はipo\_out.sです。

「関数のインライン展開」を参照してください。

## -Qoption 指示子による -ip または -ipo の使用

特定のインライン展開およびループ最適化を選択するには、適切なキーワードを用いて-Qoptionを使用します。このオプションを入力する場合は、次の例に示すように、-ipまたは-ipoを指定しなければなりません。

```
prompt>icc -ip -Qoption,tool,opts
toolはC++ (c)で、optsは-Qoption 指示子です(下記参照)。
```

### -option 指示子

-Qoption修飾なしで-ipoまたは-ipoを指定すると、コンパイラは次のことを行います。

- 関数のインライン展開
- 定数引数の伝播
- レジスタ内での引数の受け渡し
- モジュール・レベルでのスタティック変数の監視

次の-Qoption指示子を使用して、プロシージャ間の最適化を設定できます。これを有効にするには、例に示すように、-Qoption オプションは-ipまたは-ipoも同時に入力しなければなりません。

```
prompt>icc -ip -Qoption,c,ip_specifier
ip_specifierは次の表で説明される指定子のいずれかです。
```

指示子	説明
-ip_args_in_reg	レジスタ内での引数の受け渡しを無効にするオプションです。デフ

s=0	オルトでは、ローカルに呼び出された外部関数はレジスタ内で引数の受け渡しができます。通常、スタティック関数のみはレジスタ内での引数の受け渡しができますが、それには条件が2つあり、1つはその関数のアドレスが取得されないこと、もう1つは個数の定まらない引数をその関数が使用しないことです。
-ip_ninl_max_stats=n	インライン展開する関数1個について、中間言語の文を最大何個まで許容させるかを設定するオプションです。n は正の整数です。通常は、中間言語の文の個数は、ソース言語の文の実際の個数を超えます。n のデフォルト値は「230」です。コンパイラは、ユーザのインライン関数用に、制限をより大きくします。
-ip_ninl_min_stats=n	インライン展開する関数1個について、中間言語の文を最小何個まで許容させるかを設定するオプションです。n は正の整数です。 ip_ninl_min_statsのデフォルト値: <ul style="list-style-type: none"> <li>IA-32 コンパイラ: ip_ninl_min_stats = 7</li> <li>Itanium® コンパイラ: ip_ninl_min_stats = 15</li> </ul>
-ip_ninl_max_total_stats=n	インライン化のために、中間言語文で、関数のサイズに展開できる最大数を設定します。n はデフォルト値2000の正の整数です。

次のコマンドは、source.cにおいてプロシージャおよびプロシージャ間の最適化をアクティブにし、各関数について中間言語文の最大数を5に設定します。

```
prompt>icc -ip -Qoption,c,-ip_ninl_max_stats=5 source.c
```

## 関数のインライン展開

### ユーザ関数のインライン展開の制御

コンパイラでは、下の表に示すオプションを使用して、関数のインライン展開を制御できます。

<code>-ip_no_inlining</code>	このオプションは、 <code>-ip</code> オプションも指定されている場合にのみ使用できます。この場合、 <code>-ip_no_inlining</code> オプションは、 <code>-ip</code> によるプロシージャ間の最適化の結果から生じるインライン展開を無効にします。ただし、これ以外のプロシージャ間の最適化に対しては何の効果もありません。
<code>ip_no_pinlineing</code>	部分的なインライン化を無効にします。 <code>-ip</code> または <code>-ipo</code> を指定しても使用できます。

### 関数のインライン展開の条件

以上の諸条件を満たすと、コンパイラはどのルーチンをインライン展開すればプログラムのパフォーマンスに最も寄与するかを調べて選び出します。プロファイルに基づく最適化 (`-prof_use`)を使用するかどうかによって、コンパイラが使用するヒューリスティックは異なります。`-ip` または `-ipo`でプロファイルに基づく最適化を使用する場合、コンパイラは次の手法を使用します。

- プログラムに対して収集されたプロファイル情報を基に、デフォルトの手法では、最も頻繁に実行される呼び出しサイトに重点がおかれます。
- デフォルトでは、コンパイラは230以上の中間文では、関数のインライン化は行いません。この値は次のオプション、  
`-Qoption,c,-ip_ninl_max_stats=new_value`を使用して変更できます。注: `inline` または `__inline`のようにユーザに宣言される関数には、より高い制限があります。
- デフォルト・ヒューリスティックは、直接再帰( `direct recursion` )を検出すると停止します。
- デフォルト・ヒューリスティックを使用すると、インライン化の最低条件を満たしている非常に小さな関数は必ずインライン化します。
  - Itanium® ベース・アプリケーションのデフォルト値:  
`ip_ninl_min_stats=15`.
  - IA-32 アプリケーションのデフォルト値: `ip_ninl_min_stats = 7`この上限は、`-Qoption,c,-ip_ninl_min_stats=new_value`オプションを使用して変更できます。

`-ip` または `-ipo` でプロファイルに基づく最適化を使用しない場合、コンパイラは次のような効果の低いインライン手法を使用します。

- インライン展開で最終のプログラムのサイズが増えない場合、関数をインライン化。
- `inline` または `__inline` キーワードで宣言された場合、関数をインライン化。

# プロファイルに基づく最適化

## プロファイルに基づく最適化の概要

プロファイルに基づく最適化(PGO)を行うと、アプリケーションのどの領域が最も頻繁に実行するかがコンパイラに伝えられます。コンパイラはこの領域を知ると、以前のコンパイルのフィードバックを使用して、より慎重にアプリケーションの最適化を行います。例えば、PGO を使用するとコンパイラは関数のインライン化についての的確な判断が下せる場合が多くなり、その結果、プロシージャ間の最適化の効率が向上します。

### インストルメント済みプログラム

プロファイルに基づく最適化は、ソース・コードとコンパイラの特殊コードからインストルメント済みプログラムを作成します。インストルメント済みコードを実行するたびにインストルメント済みプログラムは動的情報ファイルを生成します。2回目にコンパイルすると、動的情報ファイルはサマリ・ファイルにマージされます。このファイルのプロファイル情報を使用して、コンパイラは、プログラムで最もよく使用されるパスの実行の最適化を試みます。

サイズや速度に厳密に使用されるような他の最適化とは異なり、IPO および PGO の結果は様々です。これは、各プログラムは異なるプロファイルと異なる最適化の機会を持つためです。このガイドラインは、IPO と PGO を使用する利点があるかどうかを決定するのに役立ちます。

### PGO で向上するパフォーマンス

インテル® C++ コンパイラのバージョン6.0以降、次のように PGO が向上しました。

- レジスタの確保は、プロファイル情報を使用して、スピル・コードの場所を最適化します。
- 直接関数呼び出しでは、最も可能性の高い対象を識別することにより、分岐予測が向上されます。Pentium® 4 プロセッサのより長いパイプラインで、強化された分岐予測は大幅なパフォーマンスの向上を実現します。
- コンパイラは、実行の繰返し回数が少ないループを検出しベクトル化せず、ベクトル化により追加される可能性のあるランタイム・オーバーヘッドを減少します。

## プロファイルに基づく最適化の方法

PGO は、コンパイルの時点での予測が困難な、繰返し実行される分岐を含むコードの最適化として最も有効に働きます。例えば、エラーチェック処理を多く含んでいるのに、実際にエラー判定を下してみるとほとんどの場合エラー条件に合致しないコードがこれに相当します。これにより分岐予測をほとんど誤ることのない、いわゆる「コールド」(cold)なエラー処理コードを配置できます。「ホット」コードと「コールド」コードが交互に配置する状況をなくせば、命令キャッシュの動作が改善されます。例えば、PGO を使用するとコンパイラは関数のインライン化に



についての確な判断が下せる場合が多くなり、その結果、プロシージャ間の最適化の効率が向上します。

## PGO フェーズ

PGO を行うには次の3つのフェーズが必要です。

- 第 1 フェーズ: `-prof_gen[x]` によるインストルメンテーション・コンパイルとリンク
- 第 2 フェーズ: 実行ファイルの実行によるインストルメント済み実行
- 第 3 フェーズ: `-prof_use`によるフィードバック・コンパイル

PGO の適否を決める手掛かりの1つは、コードのどの部分に処理が集中しているかを知ることです。プログラムに与えられるデータセットがいつもほとんど変わらず、何度実行しても同じような動作にしかならない場合は、PGO によってプログラム実行が最適化されます。一方、プログラムに与えられるデータセットが毎回異なり、そのたびに種々のアルゴリズムが呼び出されることもあります。このような場合、プログラムは実行のたびに違った動作になるときがあります。

実行するたびに毎回動作が大きく異なるようなコードに対しては、PGO はそれほど有効ではありません。プロファイルを最新状態に保つ作業をするだけの価値が、そのプロファイル情報から得られるかどうかを常に考える必要があります。

## 基本的なPGOオプション

オプション	説明
<code>-prof_gen[x]</code>	インストルメント済み実行の準備として、インストルメント済みコードをオブジェクト・ファイル内に生成するようにコンパイラに命令するオプションです。
<code>-prof_use</code>	プロファイルによって最適化された実行ファイルを生成し、利用可能な動的情報( <code>.dyn</code> )ファイルをいくつかマージして <code>pgopti.dpi</code> ファイルを1個作成するようにコンパイラに命令するオプションです。

コードの動作が実行ごとに大きく異なる場合は、プロファイル情報によって得られるメリットが、最新のプロファイルを維持する作業に見合うものであるかどうか検討する必要があります。基本となるプロファイルに基づく最適化には、次のオプションが PGO フェーズで使用されます。

## インストルメント済みコードの作成

`-prof_gen[x]` オプションは、各基本ブロックの実行カウントを取得するために、プロファイル用プログラムをインストルメントします。インストルメント済みのプログラムを実行する準備として、PGO の第1フェーズ で、インストルメント済みコードをオブジェクト・ファイルに生成するようにコンパイラに指示します。`-prof_genx` コンパイルでは、並列化 `make` を自動的にサポートします。

## プロファイルによって最適化された実行ファイルの生成

`-prof_use` オプションは、PGO の第3フェーズ で使用され、プロファイルによって最適化された実行ファイルを生成し、利用可能な動的情報(`.dyn`) ファイルをいくつかマージして `pgopti.dpi` ファイルを1個作成するようにコンパイラに指示します。



注

動的情報ファイルは、インストルメント済み実行ファイルを実行する第2フェーズで生成されます。

インストルメント済みプログラムを何回か実行する場合は、`-prof_use` を指定すると、実行するたびに同じ動的情報ファイル同士がマージされ、その前の `pgopti.dpi` ファイルは上書きされます。

## 関数分割の無効(Itanium® コンパイラのみ)

`-fnsplit-`は関数分割を無効にします。フェーズ3で`-prof_use`により関数分割は有効になります。これは、ルーチンを異なるセクションに分割することによって、コードの局所性を向上させるためです。異なるセクションとは、コールドまたは、あまり実行されないコードを含むセクションと、残りのコード(ホットコード)を含むセクションです。

次のような理由により、`-fnsplit-`を使用して、関数分割を無効にできます。

- 最も重要なことは、デバッグの機能を向上させることです。デバッグのシンボルテーブルでは、分割ルーチン、すなわちホット・コード・セクションとコールド・コード・セクションを持つルーチンを表示するのは困難です。
- `-fnsplit-`オプションは、ルーチン内の分割を無効にしますが、関数のグループ化を有効にします。これは、コールド・コード・セクションまたはホット・コード・セクションのいずれかにルーチン全体を配置する最適化機能です。関数のグループ化は、デバッグ機能を低下させません。
- 別の理由としては、プロファイル・データが実際のプログラム動作をしなかった場合、つまり、ルーチンが実際は稀ではなく頻繁に使用される場合です。



注

Itanium ベース・アプリケーションの場合、`-O3` レベルの最適化で `-prof_use` オプションを使用する場合、`-O3` オプションが必要です。`-O2` またはより低いレベルの最適化で `-prof_use` オプションを使用する場合、デフォルトのオプションでプロファイル・データを生成できます。

## プロファイルに基づく最適化の例

PGO には 3 つの基本フェーズがあります。

- インストルメンテーション・コンパイルとリンク
- インストルメント済み実行



- フィードバック・コンパイル

## インストルメンテーション・コンパイルとリンク

-prof\_genを使用して、インストルメント済み情報付きの実行ファイルを生成します。また、多くのプログラムに適した -prof\_dir オプションを使用してください。特に、複数のディレクトリに渡るソース・ファイルを持つアプリケーションには使用してください。-prof\_dir は、プロファイル情報が同じ場所で生成されることを保証します。次に例を示します。

### IA-32 システム

```
prompt>icc -prof_gen -prof_dir:¥profdata -c a1.c a2.c a3.c
prompt>icc a1.o a2.o a3.o
```

### Itanium® ベース・システム

```
prompt>ecc -prof_gen -prof_dir:¥profdata -c a1.c a2.c a3.c
prompt>ecc a1.o a2.o a3.o
```

2番目のコマンドの代わりにリンクを直接使用して、インストルメント済みプログラムを生成できます。

## インストルメント済み実行

代替の何らかのデータセットを使用してインストルメント済みプログラムを実行して、動的情報ファイルを作成します。

```
prompt>./a.o
```

作成される動的情報ファイルは、a.o を実行するたびに毎回別の名前と .dyn というサフィックスが付きます。また、このインストルメント済みファイルは、特定のデータセットでこのプログラムを実行したときの振舞いを予測するのに役立ちます。このプログラムは、入力データを変えて何度でも実行できます。

## フィードバック・コンパイル

-prof\_use を指定してソースファイルのコンパイルとリンクを行い、動的情報を使用して、そのプロファイルに従ってプログラムを最適化します。

### IA-32のシステム:

```
prompt>icc -prof_use -ipo a1.c a2.c a3.c
```

### Itanium ベース・システム:

```
prompt>ecc -prof_use -ipo a1.c a2.c a3.c
```

最適化のほかに、コンパイラは pgopti.dpi ファイルを生成します。一般に、第1フェーズではデフォルトの最適化(-O2)を行い、第3フェーズでは、さらに高度な最適化(-ipo)を指定します。上の例では第1フェーズで -O2 を使用し、第3フェーズで -O2 -ipo を使用しました。



注

-prof\_gen[x] オプションを使用すると、-prof\_gen[x] オプションが無視されます。  
x修飾子を付けると、補足情報が収集されます。

## PGOの環境変数

下の表は、動的情報ファイルを格納するディレクトリを指定するとき、または pgopti.dpi を上書きするかどうかを指定するときに使用する環境変数を示します。環境変数の設定方法については、ご使用のオペレーティング・システムのマニュアルを参照してください。

### プロファイルに基づく最適化に使う環境変数

変数	説明
PROF_DIR	動的情報ファイルの作成先ディレクトリを指定する変数です。この変数は、プロファイル処理の3つのフェーズすべてに適用されます。
PROF_NO_CLOBBER	フィードバック・コンパイル・フェーズでの処理を少し変更する変数です。デフォルトでは、フィードバック・コンパイル・フェーズでは、すべての動的情報ファイルから得たデータがマージされます。そして .dyn ファイルが既存の pgopti.dpi ファイルよりも新しい場合は、pgopti.dpi ファイルを新たに作成します。この変数を設定すると、コンパイラは既存の pgopti.dpi ファイルを上書きせずに、警告を発行します。その代わりに、コンパイラから警告メッセージが発行されます。他の動的情報ファイルを使用したい場合は、その既存の pgopti.dpi ファイルを削除してください。

## profmerge を使用して、ソースファイルの再配置

コンパイラはソースファイルのフルパスを使用して、プロファイル・サマリ情報を検索します。デフォルトでは、次のことが行えません。

- アプリケーション・ソースを移動する場合、プロファイル・サマリ・ファイル(.dpi)を使用する
- 異なるディレクトリにある同一アプリケーションのソースをビルドする別のユーザとプロファイル・サマリ・ファイルを共有する

### ソースの再配置

プロファイル・サマリ・ファイルの共有とアプリケーション・ソースの移動を有効にするには、-src\_old オプションおよび -src\_new オプションとともに profmerge を使用します。次に例を示します。

**IA-32 システム:** `prompt>profmerge -prof_dir <p1> -src_old <p2> -src_new <p3>`

**Itanium® ベース・システム:** `prompt>profmerge -em -p64 -prof_dir <p1> -src_old <p2> -src_new <p3>`

各アイテムの意味は次のとおりです。

- <p1> は動的情報ファイル(.dpi)のフルパス。
- <p2> はソースファイルの古いフルパス。
- <p3> はソースファイルの新しいフルパス。

上記のコマンドで `pgopti.dpi` ファイルが読み込まれます。`pgopti.dpi` ファイルのソースパスは <p2> プリフィックスで始まり、ファイルに記述されている各関数用に、`profmerge` は そのプリフィックスを <p3> に置換します。`pgopti.dpi` ファイルは新しいソースパス情報に更新されます。



#### 注意事項

- 1 つの `pgopti.dpi` ファイルで 1 回以上 `profmerge` を実行できます。複数のディレクトリにソースファイルがある場合、複数実行する必要があるかもしれません。次に例を示します。

```
profmerge -prof_dir -src_old /src/prog_1 -src_new  
/src/prog_2
```

```
profmerge -prof_dir -src_old /proj_1 -src_new /proj_2
```

- `-src_old` と `-src_new` に指定される値では、大文字と小文字の区別はありません。同様に、フォワード・スラッシュ(/)とバック・スラッシュ(¥)は、同じ文字とみなされます。
- `profmerge` のソース再配置機能が `pgopti.dpi` ファイルを変更するため、ソースの再配置を実行する前に、必要に応じて、ファイルのバックアップをとってください。

## PGO API: プロファイル情報生成サポート

### PGO API サポートの概要

プロファイル情報生成サポート(Profile Information Generation Support) で、プロファイルに基づいた最適化のインストルメント済みの実行フェーズ中にプロファイル情報の生成を制御できます。通常、プロファイル情報は、インストルメント済みアプリケーションが、標準的な `exit()` 関数を呼び出してアプリケーションを終了したときに生成されます。この節で説明される関数は、プロファイル情報が次の状況で確実に生成されなければならない場合があります。

- 非標準の終了ルーチンでインストルメント済みアプリケーションが終了する場合。
- インストルメント済みアプリケーションが、`exit()` を呼び出すことのない非終了アプリケーションの場合。
- プロファイル情報の生成時点を制御する必要がある場合。

この節では、プロファイル情報生成サポートを構成する関数および環境変数について説明します。関数は、関数を使用されるソース・ファイルの上に `#include <pgouser.h>` を挿入することにより使用することができます。

`-prof_gen` または `-prof_genx` のいずれかを使用してコンパイルする場合、コンパイラは `_PGO_INSTRUMENT` の `define` を設定をします。

### プロファイル情報のダンプ

```
void _PGOPTI_Prof_Dump(void);
```

#### 説明

この関数は、インストルメントされたアプリケーションにより収集されたプロファイル情報をダンプします。プロファイル情報は `.dyn` ファイルに記録されます。

#### 推奨する使用方法

アプリケーションを終了する関数本体にこの関数への呼び出しを挿入します。通常、`_PGOPTI_Prof_Dump` の呼び出しは、一度だけでなければなりません。この関数を `_PGOPTI_Prof_Reset()` とともに使用して、複数の `.dyn` ファイル (入力データの複数のセットから) を作成することができます。

<p>例</p> <pre>// Selectively collect profile information for the // portion // of the application involved in processing input data.  input_data = get_input_data();  while(input_data) {</pre>
---

```
_PGOPTI_Prof_Reset();
process_data(input_data);
_PGOPTI_Prof_Dump();
input_data = get_input_data();
}
```

## 動的プロファイル・カウンタのリセット

```
void _PGOPTI_Prof_Reset(void);
```

### 説明

この関数は、動的プロファイル・カウンタをリセットします。

### 推奨する使用方法

この関数を使用し、インストール済みのアプリケーションのセクションでプロファイル情報を収集する前にプロファイル・カウンタをクリアにします。PGOPTI\_Prof\_Dump() の例を参照してください。

## プロファイル情報のダンプとリセット

```
void _PGOPTI_Prof_Dump_And_Reset(void);
```

### 説明

この関数は、1回以上呼び出される場合があります。各呼び出しでは、プロファイル情報を新しい .dyn ファイルにダンプします。そして、動的プロファル・カウンタはリセットされ、その後、インストール済みのアプリケーションの実行が続行します。

### 推奨する使用方法

この関数の定期的な呼び出しにより、非終了アプリケーションは、1 つまたは複数のプロファイル情報ファイルを作成することができます。これらのファイルは、プロファイルに基づく最適化のフィードバック・フェーズ中に結合されます。この関数の直接的な使用により、プロファイル情報の生成時にアプリケーションが正確に制御することができます。

## インターバル・プロファイル・ダンプ

```
void _PGOPTI_Set_Interval_Prof_Dump(int interval);
```

### 説明

この関数は、インターバル・プロファイル・ダンプをアクティブにし、ダンプ発生時のおおよその頻度を設定します。インターバル・パラメータは、プロファイルのダンプが発生する時間の間隔 (ミリ秒単位) で指定します。例えば、インターバルが 5000 として設定された場合、プロファイル・ダンプとリセットは約 5 秒ごとに行われます。ダンプとリセットの時間設定は、アプリケーションのインストールされた関数へのエントリに対して行われるため、インターバルは概算となります。



注

- インターバルを0または負の数に設定すると、インターバル・プロファイルのダンプを無効にします。
- インターバルが非常に少ない値で設定されると、インストルメント済みのアプリケーションがプロファイル情報のダンプにほとんどの時間を費やしてしまう場合があります。インターバルには、できるだけ大きな値を設定し、アプリケーションが本来の作業を行え、十分なプロファイル情報が収集されるようにします。

### 推奨する使用方法

この関数は、非終了 アプリケーションの開始時に呼び出され、インターバル・プロファイル・ダンプを開始します。環境変数の `PROF_DUMP_INTERVAL` をアプリケーションが開始する前に必要なインターバルの値を設定しても、インターバル・プロファイル・ダンプを開始することができます。インターバル・プロファイル・ダンプの目的は、プロファイルするのに非終了アプリケーションのソースコードの変更を最小限にすることです。

### 環境変数

`PROF_DUMP_INTERVAL`

この環境変数は、インストルメント済みのアプリケーションでインターバル・プロファイル・ダンプに使用されることがあります。詳細は、`_PGOPTI_Set_Interval_Prof_Dump` の「推奨する使用方法」を参照してください。

# 高水準言語の最適化(HLO)

## 高レベル最適化の概要

高レベル最適化(HLO)とは、C++ といった高水準プログラミング言語で開発されたアプリケーションに含まれているループや配列など、ソースコード上の特性を利用した最適化手法です。これには、ループ交換、ループ融合、ループ展開、ループ分配、アンロール・ジャム、ブロッキング、データ・プリフェッチ、スカラ置換、データ・レイアウトの最適化といった手法があります。高レベルな最適化をオンにするオプションは-O3です。

IA-32 および Itanium® ベース・アプリケーション	
-O3	-O2オプションに加えて、さらに強力な最適化(例えば、ループ変換やプリフェッチ)を有効にします。 -O3は、最大速度について最適化を行いますが、パフォーマンスが向上しないプログラムもあります。
IA-32 アプリケーション	
-O3	ベクトル化オプション-ax{M K W}、 -x{M K W}に組み合わせて-O3を指定すると、-O2の場合より詳細にデータ依存性の解析が実行されます。そのためコンパイル時間が長くなる場合があります。

## ループ変換

ループ変換ができるかどうかは、データの依存関係によって異なります。誘導変数の排除、定数伝播、コピー伝播、先行代入、不要コードの排除といった手法もあります。ループ変換の手法には次のものがあります。

- ループ正規化
- ループ逆転
- ループ交換/並べ替え
- ループ傾斜
- ループ分配
- ループ融合
- スカラ置換

以上は、Itanium® アーキテクチャのいずれでも使用できるループ変換ですが、これ以外にも Itanium アーキテクチャでは折り畳み手法が利用できます。

## ループのアンロール

ループはアンロールができます。また、コンパイラに最大何回までアンロールさせるかの指定もできます。

### ループのアンロールを有効にする方法

ループをアンロールするには、`-unroll [n]` オプションを使用します。変数  $n$  は、アンロールを実行する最大回数です。これを適用するのは、アンロールしたほうがよいとコンパイラが判断したループに対してのみです。アンロールするかしないかをコンパイラに判断させるには、 $n$  変数を省略してください。

ループのアンロール回数を最大4回にするには、次のようにします。

**IA-32 システム:** `prompt>icc -unroll4 a.c`

ループをアンロール回数に大きな値を指定する場合、アプリケーションがレジスタなどの特定のリソースを消耗することがあるので注意してください。これは、プログラムのパフォーマンスの低下を引き起こします。ループをアンロール回数に大きな値を指定する場合は、アプリケーションを時間測定することを推奨します。([「アプリケーションの時間測定」](#)を参照してください)

### ループのアンロールを無効にする方法

ループのアンロールを無効にするには、変数  $n$  を0に設定します。ループのアンロールを無効にするには、次のようにします。

**IA-32 システム:** `prompt>icc -unroll0 a.c`

## IVDEP ディレクティブによるループ・キャリー・メモリ依存の不在

ただし、Itanium® ベースのアプリケーションでは、`-ivdep_parallel` オプションは、IVDEPディレクティブが指定されたループにループ・キャリー・メモリ依存が絶対がないことを示します。この手法は、スパース・マトリックス・アプリケーションに役立ちます。例えば、次のループは ループ・キャリー依存がないことを示す IVDEP ディレクティブの他に `-ivdep_parallel` を必要とします。

例
<pre>#pragma ivdep  for(i=1; i&lt;n; i++) { e[ix[2][i]]=e[ix[2][i]]+1.0; e[ix[3][i]]=e[ix[3][i]]+2.0; }</pre>

次の例では、このオプションとIVDEPディレクティブを使用すると、`a()` への格納でループキャリー依存が確実にないことを示します。



例

```
#pragma ivdep
```

```
for(j=0; j<n; j++)  
{  
  a[b[j]]=a[b[j]]+1;  
}
```

# 並列化

## 並列化オプションの概要

並列プログラミング用に、インテル® C++ コンパイラは、OpenMP\* 2.0 API と自動並列化機能をサポートしています。次の表は、OpenMPおよび自動並列化を行うオプションを列挙したものです。

オプション	説明
-openmp	OpenMP ディレクティブに基づいてマルチスレッド・コードを生成する処理を、パラライザに許可します。デフォルト: OFF
-openmp_report {0   1   2}	OpenMP パラライザの診断レベルを制御します。デフォルト: -openmp_report1
-openmp_stubs	シーケンシャル・モードでOpenMP プログラムのコンパイルを有効にします。OpenMP ディレクティブは無視され、スタブ OpenMP ライブラリがリンクされます。デフォルト: OFF
-parallel	自動並列化を有効にして、並列で安全に実行できるループのマルチスレッド・コードを生成します。デフォルト: OFF
-par_threshold {n}	並列でのループの実行が効果的である可能性に基づいてループの自動並列化のしきい値を設定します (n=0 から 100)。n=0 は "常に" を意味します。デフォルト: -par_threshold75.
-par_report {0   1   2   3}	自動パラライザの診断レベルを制御します。デフォルト: -par_report1

### 注

-openmp と -parallel の両方がコマンドラインで指定されると、-parallel オプションは、OpenMPディレクティブを含まないルーチンでのみ有効となります。OpenMPディレクティブを含むルーチンでは、-openmpオプションのみが有効です。

# OpenMP\* による並列化

## OpenMP\* による並列化の概要

インテル® C++ コンパイラは、OpenMP\* C++ バージョン2.0 API 仕様をサポートしています。OpenMP は、次の主な機能を持った対称型マルチプロセッシング (SMP) を提供します。

- OpenMP は、反復のパーティショニング、データの共用、スレッドのスケジューリング、および同期化に関する下位の詳細レベルを処理して、ユーザの負担を軽減します。
- 共有メモリ、マルチプロセッサ・システムの場合に得られるパフォーマンスを提供します。

インテル C++ コンパイラは、ソース・プログラムのユーザの OpenMP ディレクティブの指定に従って、コード変換を実行し、マルチ・スレッド・コードを生成して、既存のソフトウェアヘスレッドを追加しやすくします。インテル・コンパイラは、現在の業界標準のOpenMPディレクティブのすべてに対応しています。ただし、WORKSHAREおよびOpenMPディレクティブの注釈のある並列実行プログラムのコンパイラは除きます。さらに、インテル C++ コンパイラは、ランタイム・ライブラリ・ルーチンおよび環境変数を含むOpenMP C++ バージョン 2.0 仕様にインテル独自の拡張機能を提供します。



注

コンパイラの他の高度機能と同じように、OpenMP ディレクティブを効果的に使用し、プログラムの予期しない動作を避けるためには、OpenMP ディレクティブの機能を正しく理解する必要があります。

インテルC++コンパイラのOpenMP機能の全オプションについては、「並列化オプションの概要」を参照してください。

OpenMP 標準の詳細については、Web サイト <http://www.openmp.org> をご覧ください。OpenMP\* C++ バージョン 2.0 API 仕様については、<http://www.openmp.org/specs/> を参照してください。

## OpenMPによる並列処理

OpenMPでコンパイルするには、OpenMPディレクティブでコードを注釈するプログラムを準備する必要があります。インテル C++ コンパイラは、はじめにアプリケーションを処理して、コードのマルチスレッド・バージョンを生成してからコードをコンパイルします。その出力は、並列領域またはコンストラクタを実行するスレッドによって実装される並列処理の実行プログラムです。

## パフォーマンス分析

プログラムのパフォーマンス分析には、パフォーマンス情報を表示する「インテル® VTune™ パフォーマンス・アナライザ」を使用してください。コードのどの部分が最も多く実行時間を必要としているか、また並列パフォーマンスの問題個所の特定に関する詳細情報が得られます。

## 並列処理スレッドモデル

ここでは、並列化されたプログラム処理の説明と、並列プログラミングで使用する用語の定義を追加説明します。

### 実行フロー

前述したように、OpenMP\* の C++ API コンパイラ・ディレクティブを持つプログラムは、単一のプロセスとして実行を開始します。これは、**マスタスレッド**の実行と呼ばれます。最初の**並列コンストラクタ**が検出されるまで、マスタスレッドは、シーケンシャルに実行します。

OpenMP C++ APIでは、`#pragma omp parallel` ディレクティブは、並列コンストラクタを定義します。マスタスレッドが並列コンストラクタを検出すると、そのマスタスレッドがチームのマスタとなるように、スレッドの**チーム**を作成します。並列コンストラクタに囲まれたプログラム文は、チーム内の各スレッドごとに並列して実行されます。これらの文は、囲まれた文の中から呼び出されるルーチンを含みます。

コンストラクタ内で字句的に囲まれた文は、コンストラクタの**静的範囲**を定義します。**動的範囲**は、コンストラクタ内から呼び出されるルーチンと同様に静的範囲を含みます。`#pragma omp parallel` ディレクティブが終了すると、チーム内のスレッドは同期化され、そのチームは消滅して、マスタスレッドのみが実行し続けます。チーム内の他のスレッドは、待機状態になります。単一プログラム内で並列コンストラクタは何回でも指定できます。結果として、プログラム実行中に、スレッドのチームは何度も生成され消滅します。

### 孤立ディレクティブの使用

並列コンストラクタ内で呼び出されたルーチンで、ディレクティブを使用することができます。並列コンストラクタの字句範囲ではなく、動的範囲のディレクティブは、**孤立ディレクティブ**と呼ばれます。孤立ディレクティブは、プログラムのシーケンシャル・バージョンに最小限の変更を行うだけで、プログラムの主要部分を並列に実行できます。この機能を使用すると、プログラムの最上位レベルで並列コンストラクタをコーディングでき、ディレクティブを使用して呼び出されるすべてのルーチンの実行を制御することができます。次に例を示します。

```
int main(void)
{
    ...
    #pragma omp parallel
    {
        phase1();
    }
}

void phase1(void)
{
    ...

    #pragma omp for private(i) shared(n)
    for(i=0; i < n; i++)
```

```

    {
        some_work(i);
    }
}

```

並列範囲が字句的に指定されていないので、これが孤立ディレクティブとなります。

### データ環境ディレクティブ

データ環境ディレクティブは、並列コンストラクタの実行中にデータ環境を制御します。並列およびワークシェアリング構造内でデータ環境を制御できます。ディレクティブとディレクティブのデータ環境節を使用して次のことが可能です。

- THREADPRIVATE ディレクティブを使用して、スコープ変数をプライベート化します。
- THREADPRIVATE ディレクティブの節を使用して、データスコープ属性を制御します。データスコープ属性節:
  - COPYIN
  - DEFAULT
  - PRIVATE
  - FIRSTPRIVATE
  - LASTPRIVATE
  - REDUCTION
  - SHARED

複数のディレクティブ節を使用して、変数のデータスコープ属性を指定したコンストラクタの継続期間中にその属性を制御することができます。ディレクティブでデータスコープ属性節を指定しない場合、ディレクティブに影響を受ける変数のデフォルトは SHARED になります。

### 並列処理モデルの疑似コード

一般的な OpenMP ディレクティブをいくつか使用した疑似プログラムのサンプルには、次のものがあります。この例ではまた、シリアル領域と並列領域の相違も示しています。

```

main() {
    ...
    #pragma omp parallel
    {
        ...
        ...
    }
    #pragma omp sections
    {
        #pragma omp section
        {...}
    }
}

```

// Begin serial execution  
// Only the master thread  
executes  
// Begin a Parallel Construct,  
form  
// a team. This is Replicated  
Code  
// (each team member executes  
// the same code)  
//  
// Begin a Worksharing  
Construct  
//  
// One unit of work  
//

```

#pragma omp section      // Another unit of work
{...}                  //
}                        // Wait until both units of work
                        // complete
...                     // More Replicated Code
                        //
#pragma omp for nowait   // Begin a Worksharing
for(...) {              Construct;
                        // each iteration is unit of
                        // work
                        //
...                     // Work is distributed among
                        // the team members
                        //
}                        // End of Worksharing
                        Construct;
                        // nowait was specified, so
                        // threads proceed
                        //
#pragma omp critical     // Begin a Critical Section
{                        //
...                     // Replicated Code, but only
                        // one
                        // thread can execute it at a
}                        // given time
...                     // More Replicated Code
                        //
#pragma omp barrier      // Wait for all team members to
                        // arrive
...                     // More Replicated Code
                        //
}                        // End of Parallel Construct;
                        // disband team and continue
                        // serial execution
                        //
...                     // Possibly more Parallel
                        // constructs
                        //
}                        // End serial execution

```

## OpenMP、ディレクティブ形式、および診断でのコンパイル

OpenMP\* モードでインテル® C++ コンパイラを起動するには、`-openmp` オプションでコンパイラを起動します。

**IA-32 アプリケーション:** `icc -openmp input_file`

**Itanium® ベース・アプリケーション:** `ecc -openmp input_file`

マルチスレッド・コードを起動する前に、OpenMP 環境変数 `OMP_NUM_THREADS` で使用するスレッドの数を設定できます。詳しい情報は、「OpenMP の環境変数」を参照してください。

## -openmp オプション

`-openmp` は、パラライザが OpenMP ディレクティブに基づいてマルチ・スレッド・コードを生成できるようにします。このコードは、単一プロセッサ・システムとマルチプロセッサ・システムのいずれでも並列実行が可能です。`-openmp` オプションは、2つの `-O0` (最適化なし) と `-O1`、`-O2` (デフォルト) および `-O3` の最適化レベルで動作します。`-openmp` と `-O0` を指定すると、OpenMP アプリケーションのデバッグに有効です。

## OpenMP ディレクティブ形式と構造

OpenMP ディレクティブの形式は次のとおりです:

```
#pragma omp directive-name [clause, ...] newline
```

各アイテムの意味は次のとおりです。

- `#pragma omp` -- すべてのOpenMPディレクティブに必須
- `directive-name` -- 有効なOpenMPディレクティブpragma の後および節の前に出力する必要があります。
- `clause` -- オプションclauseは順番に関係なく指定でき、制限されていない限り必要に応じて繰り返すことができます。
- `newline` -- 必須。このディレクティブに囲まれた構造ブロックを続行します。

## OpenMP 診断


`-openmp_report {0 | 1 | 2}` オプションは、OpenMPパラライザの診断レベル 0、1、2を次のように制御します。

- `-openmp_report0` = 診断情報を表示しません。
- `-openmp_report1` = 正常に並列化されたループ、領域、およびセクションを示す診断を表示します。
- `-openmp_report2` = `-openmp_report1`の診断に加えて、正常に処理された主コンストラクタ、単一コンストラクタ、重大なセクション、オーダコンストラクタ、アトミック・ディレクティブなどを示す診断を表示します。

デフォルトは`-openmp_report1`です。

## OpenMP\* ディレクティブと節

### OpenMP ディレクティブ

ディレクティブ名	説明
parallel	並列実行領域を定義します。
for	関連付けられたループの反復が並列実行される領域を指定する、反復的なワークシェアリングのコンストラクタを識別します。
sections	チーム内のスレッド間で分割される一連のコンストラクタを指定する、非反復的なワークシェアリングのコンストラクタを識別します。
single	対応する構造化ブロックがチーム内の1つのスレッドだけで実行されるように指定するコンストラクタを識別します。
parallel for	1つのforディレクティブを含む並列領域のショートカット。  <b>注</b> OpenMPディレクティブparallelまたはforの直後には、for文を続けなければなりません。parallelまたはforディレクティブとfor文の間に、他の文またはOpenMPディレクティブがあると、インテル® C++ コンパイラは構文エラーを生成します。
parallel sections	1つのsectionsディレクティブを含む並列領域を指定するショートカット形式。
master	チームのmasterスレッドで実行する構造化ブロックを指定する構成体を示します。
critical [lock]	関連する構造化ブロックの実行を一度に1スレッドだけに制限する構成体を示します。
barrier	チーム内のすべてのスレッドを同期化します。
atomic	特定のメモリ・ロケーションをアトミックに更新します。
flush	「クロススレッド」シーケンス・ポイントを指定します。このポイントでは、チーム内のすべてのスレッドから見たメモリ内の特定のオブジェクトの状態の整合性が保たれるように、プログラム上で保証する必要があります。
ordered	orderedディレクティブに続く構造化ブロックを、シーケンシャル・ループ内で反復が実行される順序で実行します。
threadprivate	指定された名前付きのファイル有効範囲変数または名前空間の有効範囲変数を特定のスレッドに対してプライベートにし、そのスレッド内ではファイル有効範囲を参照可能にします。

## OpenMP の節

節	説明
---	----



private	チーム内の各スレッドに対してprivateになるように変数を宣言します。
firstprivate	private節で指定される機能のスーパーセットを指定します。
lastprivate	private節で指定される機能のスーパーセットを指定します。
shared	チーム内のすべてのスレッドで変数を共有します。
default	変数のデータ有効範囲属性を設定できます。
reduction	スカラー変数の削減を実行します。
ordered	orderedディレクティブに続く構造ブロックを、シーケンシャル・ループ内で反復が実行される順序で実行します。
if	IF(scalar_logical_expression)節を指定していると、scalar_logical_expressionが.TRUE.と評価した場合に限り、囲まれているコードブロックが並列に実行されます。それ以外の場合は、コードブロックは直列に実行されます。
schedule	forループの反復がチームのスレッド間でどのように分割するかを指定します。
copyin	並列領域を実行しているチーム内の各スレッドのthreadprivate変数に、同じ名前を割り当てます。

## OpenMP\* のサポート・ライブラリ

OpenMP\* をサポートするインテル® C++ コンパイラは、製品サポート・ライブラリ libguide.a を提供します。このライブラリを使用して、アプリケーションを異なる実行モードで実行することができます。これは、既にチューニングされているアプリケーションによる通常実行またはパフォーマンスが重要な実行において使用します。

## 実行モード

OpenMP をサポートするインテル・コンパイラは、ランタイム時に指定した実行モードでアプリケーションを実行することができます。このライブラリはserial、turnaroundおよびthroughputモードをサポートします。これらのモードは、実行時にKMP\_LIBRARY 環境変数を使用することによって選択されます。

### Serial

Serialモードは、並列アプリケーションをシングル・プロセッサ上で強制的に実行します。

### Turnaround

すべてのプロセッサが、プログラムの全実行に対し排他的に割り当てられる専用（バッチまたはシングルユーザ）並列環境では、常にすべてのプロセッサを効果的に利用することが最も重要です。turnaroundモードは、並列計算を行うすべてのプロセッサをアクティブな状態で維持して、単一ジョブの実行時間を最小限に抑えるよう設計されています。作業スレッドは、追加の並列作業を他のスレッドにわたすことなく、アクティブな状態で待機します。



注

過剰なシステムリソースの割り当てを避けてください。過剰なシステムリソースの割り当ては、

スレッドが多すぎるか、または実行時に利用可能なプロセッサが少なすぎる場合に発生します。システムリソースが過剰に割り当てられると、このモードはパフォーマンスの低下を起こします。この問題が発生した場合、throughputモードを使用してください。

## Throughput

並列マシン上のロードが一定ではない、またはジョブ・ストリームが予測できないマルチユーザ環境下では、throughput用にデザインおよびチューニングする方が良い場合もあります。これにより、複数のジョブを同時に実行した際の合計時間を最小限に抑えることができます。このモードでは、作業スレッドは追加の並行作業の待機中、他のスレッドへ作業を渡します。

turnaroundモードは、プログラムにその実行環境を認知させ（つまりシステムの読み込み）、リソースの使用を調整することで、動的環境における効率の良い実行を行えるよう設計されています。Throughputモードはデフォルトです。

## OpenMP\*の環境変数

このトピックでは、OpenMP\* の環境変数（OMP\_ プリフィックス付き）およびインテル独自の環境変数（KMP\_ プリフィックス付き）を説明します。

### 標準環境変数

変数	説明	デフォルト
OMP_SCHEDULE	ランタイム・スケジュールの型とブロックサイズを設定します。	STATIC ( ブロックサイズの指定なし)
OMP_NUM_THREADS	実行時に使用するスレッド数を設定します。	プロセッサの数
OMP_DYNAMIC	スレッド数の動的な調整を有効(TRUE)または無効(FALSE)にします。	FALSE
OMP_NESTED	ネストされた並列処理を有効(TRUE)または無効(FALSE)にします。	FALSE

## インテル拡張環境変数

環境変数	説明	デフォルト
KMP_LIBRARY	OpenMP ランタイム・ライブラリ・スループットを選択します。この変数の値には、実行モードを示す serial、turnaround または throughput があります。この変数が指定されなかった場合、デフォルト値の throughput が使用されます。	throughput (実行モード)
KMP_STACKSIZE	各並行スレッドがプライベート・スタックとして使用するバイト数を設定します。オプションの b、k、m、g または t	IA-32: 2m Itanium® コンパイラ: 4m

	サフィックスを使用して、確保するバイト数をバイト、キロバイト、メガバイト、ギガバイトまたはテラバイトで指定してください。	
--	--	--

## OpenMP\* ランタイム・ライブラリ・ルーチン

OpenMP\* では、並列モードでプログラムを管理するためのランタイム・ライブラリ・ルーチンをいくつか用意しています。これらの関数の多くは、デフォルトとして設定できる環境変数を持っています。これらの環境変数はランタイム・ライブラリ関数によって、動的に変更できます。いかなる場合も、ランタイム・ライブラリ・ルーチンが呼び出されると、それに対応する環境変数は無効になります。

次の表は、これらランタイム・ライブラリ・ルーチンとのインターフェイスを明記したものです。ルーチン名はユーザ名前空間に保存しています。コンパイラをインストールしたINCLUDEディレクトリにomp.hおよびomp\_lib.hヘッダファイルがあります。

この表中の関数に使用する2つの異なるロック( lock )である omp\_lock\_kind と omp\_nest\_lock\_kind には、定義がいくつかあります。

関数	説明
<b>実行環境ルーチン</b>	
omp_set_num_threads( <i>nthreads</i> )	後続の並列領域に使用するスレッド数を設定します。
omp_get_num_threads()	現在の並列領域に使用されているスレッドの数を返します。
omp_get_max_threads()	並列実行に利用可能なスレッドの最大数を返します。
omp_get_thread_num()	コードのこのセクションを現在実行しているスレッドはどれか、その固有番号を返します。
omp_get_num_procs()	プログラムに利用可能なプロセッサの数を返します。
omp_in_parallel()	並列で実行する並列領域の動的範囲内で呼び出された場合は TRUE を返します。そうでない場合は FALSEを返します。
omp_set_dynamic( <i>dynamic_threads</i> )	並列領域の実行に使用するスレッド数の動的な調整を有効または無効にします。 <i>dynamic_threads</i> がTRUEの場合は、ダイナミック・スレッドは有効です。 <i>dynamic_threads</i> がFALSEの場合は、ダイナミック・スレッドは無効ですダイナミック・スレッドはデフォルトでは無効です。
omp_get_dynamic()	動的なスレッド調整が有効の場合は、TRUEを返します。そうでない場合、FALSEを返します。

<code>omp_set_nested(<i>nested</i>)</code>	ネストされた並列処理を有効または無効にします。 <i>nested</i> が TRUE の場合は、ネストされた並列処理は有効です。 <i>nested</i> が FALSE の場合は、ネストされた並列処理は無効です。ネストされた並列処理はデフォルトでは無効です。
<code>omp_get_nested()</code>	ネストされた並列処理が有効な場合 TRUE を返します。そうでない場合、FALSE を返します。
<b>ロックルーチン</b>	
<code>omp_init_lock(<i>lock</i>)</code>	後続の呼び出しに使用する <i>lock</i> に関連付けられたロックを初期化します。
<code>omp_destroy_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられたロックを未定義にします。
<code>omp_set_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられているロックが使用可能な状態になるまで実行中のスレッドを強制的に待機させます。ロックが使用可能になると、スレッドにはそのロックの所有権が与えられます。
<code>omp_unset_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられているロックの所有権から実行スレッドを解放します。 <i>lock</i> に関連付けられたロックを実行中のスレッドが所有していない場合の動作は不定です。
<code>omp_test_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられているロックを設定しようと試みます。成功した場合、TRUE を返します。そうでない場合、FALSE を返します。
<code>omp_init_nest_lock(<i>lock</i>)</code>	後続の呼び出しに使用する <i>lock</i> に関連付けられたネストされたロックを初期化します。
<code>omp_destroy_nest_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられたネストされたロックを未定義にします。
<code>omp_set_nest_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられているネストされたロックが使用可能な状態になるまで実行中のスレッドを強制的に待機させます。ロックが使用可能になると、スレッドにはそのネストされたロックの所有権が与えられます。
<code>omp_unset_nest_lock(<i>lock</i>)</code>	ネストしているカウント数がゼロの場合は、 <i>lock</i> に関連付けられたネストされたロックの所有権から実行中のスレッドを解放します。 <i>lock</i> に関連付けられたネストされたロックを実行中のスレッドが所有していない場合の動作は不定です。
<code>omp_test_nest_lock(<i>lock</i>)</code>	<i>lock</i> に関連付けられているネストされたロック

	を設定しようと試みます。成功した場合はネスト数を返し、失敗した場合は0を返します。
<b>タイミング・ルーチン</b>	
omp_get_wtime()	任意の参照時間から経過したウォールクロック時間(秒)に等しい倍精度値を返します。参照時間は、プログラム実行中には変更されません。
omp_get_wtick()	連続するクロック刻みの間隔の秒数に等しい倍精度値を返します。

## OpenMP\* に追加されたインテル拡張機能

### インテル拡張機能

インテル® C++ コンパイラは、OpenMP\* ランタイム・ライブラリへの拡張機能として、次の関数グループをサポートしています。

- 並列スレッドのスタックサイズの取得と設定
- メモリの割り当て

ここで説明するインテル拡張機能は、ライブラリ・コードとアプリケーションが目的どおりに機能することを確認する低レベルのデバッグに使用できます。これらの関数を使用するには、プログラムをシーケンシャルに実行する `-Qopenmp_stubs` コマンドライン・オプションを使用しなければならないため、充分注意して使用してください。これらの関数はまた、一般的に他のベンダの OpenMP 互換コンパイラには認識されません。これらのコンパイラでは、リンクの段階で失敗します。



注

以下の関数は、プリプロセッサ・ディレクティブ `#include <omp.h>` を必要とします。

#### スタックサイズ

多くの場合、ディレクティブは拡張命令の代わりに使用されます。例えば、並列スレッドのスタックサイズは、`kmp_set_stacksize_s()` 関数ではなく、`KMP_STACKSIZE` 環境変数を使用して設定します。



注

インテル拡張機能へのランタイムの呼び出しは、対応する環境変数の設定よりも優先します。下の「スタックサイズ」表でスタックサイズ関数の定義を参照してください。

#### メモリの割り当て

インテル C++ コンパイラでは、OpenMP ランタイム・ライブラリに対する拡張機能として、メモリ割り当て関数を実装しています。そのため、スレッドは各スレッドにローカルなヒープからメモリを割り当てることが可能です。これらの関数は、`kmp_malloc()`、`kmp_calloc()` および `kmp_realloc()` です。これらの関数によって割り当てられたメモリは、`kmp_free()` 関数によって解放する必要があります。あるスレッドによってメモリを割り当て、別のスレッドでメモリを `kmp_free()` を呼び出しても不正な処理ではありませんが、このような処理によってパフォーマンスが多少低下します。下の「メモリの割り当て」表でこれらの関数の定義を参照してください。

#### スタックサイズ

関数	説明
<code>kmp_get_stacksize_s()</code>	各並列スレッドがプライベート・スタックとして使用するバイト数を返します。この値は、最初の並列領域の前に <code>kmp_set_stacksize_s()</code> で変更するか、または <code>KMP_STACKSIZE</code> 環境変数で変更できます。

<code>kmp_get_stacksize()</code>	この関数は、下位互換性のみ提供します。異なるインテル・プロセッサとの互換性には <code>kmp_get_stacksize_s()</code> を使用します。
<code>kmp_set_stacksize_s(size)</code>	各並列スレッドがプライベート・スタックとして使用するバイト数を <i>size</i> に設定します。この値は、 <code>KMP_STACKSIZE</code> 環境変数を経由しても設定できます。 <code>kmp_set_stacksize_s()</code> を有効にするには、プログラムの最初の(動的に実行された)並列領域の先頭の前に呼び出す必要があります。
<code>kmp_set_stacksize(size)</code>	この関数は、下位互換性のみ提供します。異なるインテル・プロセッサとの互換性には <code>kmp_set_stacksize_s()</code> を使用します。

## メモリの割り当て

関数	説明
<code>kmp_malloc(size)</code>	スレッド・ローカル・ヒープから <i>size</i> バイトのメモリブロックを割り当てます。
<code>kmp_calloc(nelem, elsize)</code>	スレッド・ローカル・ヒープからサイズ <i>elsize</i> の <i>nelem</i> 要素の配列を割り当てます。
<code>kmp_realloc(ptr, size)</code>	スレッド・ローカル・ヒープからアドレス <i>ptr</i> および <i>size</i> バイトにメモリブロックを再割り当てします。
<code>kmp_free(ptr)</code>	スレッド・ローカル・ヒープからアドレス <i>ptr</i> のメモリブロックを解放します。メモリは、以前に <code>kmp_malloc()</code> 、 <code>kmp_calloc()</code> 、または <code>kmp_realloc()</code> に割り当てられている必要があります。



# ワークキューイング・モデル

## インテルのワークキューイング・モデルの概要

ワークキューイング・モデルでは、OpenMP\*モデルにサポートされる制御構造の範囲を超えた制御構造を並列化できます。同時にOpenMPで定義されるフレームワークに合うようにします。特に、ワークキューイング・モデルは、ワークシェアリング・コンストラクタの開始時に作業単位が事前計算されないように指定する柔軟性のあるメカニズムです。single、forおよびsectionsコンストラクタでは、コンストラクタが実行を開始する時点で、実行可能なすべての作業単位が判明しています。ワークキューイング・プラグマtaskqとtaskは、環境 (taskq) と作業単位 (tasks) を別々に指定することによって、この制限を緩和します。

## ワークキューイング・コンストラクタ

### taskq プラグマ

taskq プラグマは、囲まれた作業(タスク)単位を実行する環境を指定します。最初に、taskq プラグマを実行するすべてのスレッドの中から、ひとつのスレッドが選択されます。概念的には、taskq プラグマは、選択されたスレッドを実行する空のキューを生成され、その中で taskq ブロックの内側のコードシングルスレッドで実行されます。他のすべてのスレッドは、この概念キューに作業がキューイングされるのを待機します。task プラグマは、潜在的に異なるスレッドで実行される作業単位を指定します。taskq ブロック内に task プラグマが存在すると、task ブロックの内側のコードは、taskq に関連付けられている概念キューにキューイングされます。キューイングされたすべての作業が終了し、そして taskq ブロックの最後に達すると、概念キューはなくなります。

### 制御構造体

多くの制御構造体は、異なる作業反復と作業生成のパターンを表しているため、ワークキューイング・モデルで並列化可能です。一般的なケースは次のものです。

- while ループ
- C++ 反復子
- 再帰関数

### while ループ

while ループの各反復における計算が独立している場合、ループ全体が taskq プラグマに対する環境になります。while ループ本体内の文が、task プラグマで指定される作業単位になります。while ループの条件と制御変数への修正は、task ブロックの外側に置かれ、データを制御変数に依存させるようシーケンシャルに実行されます。

### C++ 反復子

C++ STL(標準テンプレート・ライブラリ) 反復子は、前述したwhile ループに非常に類似しています。STLにストアされるデータの演算は、すべてのデータに対する反復動作から独立しているからです。データが独立している演算の場合、その演算は、作業の反復がシーケンシャルであれば、並列に処理されます。このタイプの while ループ並列処理は、ループの標準



OpenMP\* のforループワークシェアリングを一般化したものです。forループのワークシェアリングでは、ループの増分演算が反復子で、ループの本体が作業単位です。しかし、for ループ反復子の変数は、大抵クローズされたフォームを持つため、その変数は並列に処理することができ、シーケンシャルなステップを回避できます。

## 再帰関数

再帰関数もまた、並列に反復処理を指定するのに使用できます。そのメカニズムは、sections プラグマを使用する並列処理を指定するのに類似していますが、より柔軟性があります。taskq と task プラグマ間に任意のコードを置くことができ、関数の再帰的ネストで、taskq キューの概念ツリーをビルドできるためです。taskq プラグマの再帰的ネストは、ネストされた OpenMP 並列領域のように動作する OpenMP のワークシェアリング・コンストラクタの概念を拡張したものです。ネストされた並列領域のように、ネストされた各ワークキューイング・コンストラクタは新しいインスタンスで、1つのスレッドに検出されます。主な相違点は、ネストされたワークキューイング・コンストラクタは新しいスレッドやチームを生成しないことです。むしろチームからスレッドを再利用します。これにより、動的環境で、マルチ・アルゴリズム並列処理を容易にし、並列処理の各レベルでスレッドの数を確保する必要がなくなり、トップレベルだけで済ますことができます。その時点から、大量の作業量が内側のレベルで発生すると、外側のレベルからアイドルスレッドは、その作業の終了を支援することができます。例えば、1つのスレッドに各インカミング・リクエストを処理させ、多くのスレッドがインカミング・リクエストを待機している状態は、サーバ環境では非常に一般的です。特定のリクエストにおいて、スレッドが処理を開始する時点では、そのサイズが不明な場合があります。スレッドが、ネストされたワークキューイング・コンストラクタを使用し、内側のコンストラクタが開始された後にリクエストのスコープが大きくなると、外側のコンストラクタのスレッドは、リクエストを終了するために内側のコンストラクタに簡単に移動できます。

ワークキューイング・モデルはシーケンシャルなセマンティクスを保持するように設計されているため、同期化は taskq ブロックのセマンティクスでは固有のものです。taskq コンストラクタを検出したスレッドの taskq ブロックの完了時に暗黙的なチームバリアがあるため、taskq ブロック内で指定されたすべてのタスクの実行が終了したことを確認できます。この taskq バリアは、オリジナル・プログラムのシーケンシャル・セマンティクスを実行します。OpenMP ワークシェアリング・コンストラクタのように、依存性が存在しないこと、またはタスクブロック間、あるいはタスクブロックのコードと、タスクブロック外の taskq ブロックのコード間で依存性が適切に同期されることを確認する必要があります。

文法、セマンティクスおよび構文は、OpenMP\* ワークシェアリング・コンストラクタと同じように設計されています。OpenMP ワークシェアリング・コンストラクタで有効な構文のほとんどは、ワークキューイング・プラグマでも、適切な意味を持ちます。

## taskq コンストラクタ

```
#pragma intel omp taskq [clause[,]clause]...]  
    structured-block
```

clauseは次のいずれかです。

- `private (variable-list)`
- `firstprivate (variable-list)`
- `lastprivate (variable-list)`
- `reduction (operator : variable-list)`
- `ordered`
- `nowait`

### **private**

`private` 構文は、`taskq` の `variable-list` にある各オブジェクトの `default-constructed` バージョンの `private` を生成します。囲まれた各タスクで、`captureprivate` も含みます。各変数に参照されるオリジナルのオブジェクトは、コンストラクタに入る時点で中間値を持ちます。コンストラクタの動的範囲内では、オブジェクトは変更してはいけません。そして、コンストラクタから出る時点で中間値を持ちます。

### **firstprivate**

`firstprivate` 構文は、`taskq` の `variable-list` に各オブジェクトにある `copy-constructed` バージョンの `private` を生成します。囲まれた各タスクで、`captureprivate` も含みます。各変数に参照されるオリジナルのオブジェクトは、コンストラクタの動的範囲内では、変更してはいけません。そして、コンストラクタから出る時点で中間値を持ちます。

### **lastprivate**

`lastprivate` 構文は、`taskq` の `variable-list` にある各オブジェクトの `default-constructed` バージョンの `private` を生成します。囲まれた各タスクで、`captureprivate` も含みます。各変数に参照されるオリジナルのオブジェクトは、コンストラクタに入る時点で中間値を持ちます。コンストラクタの動的範囲内では、オブジェクトは変更してはいけません。そして、タスクの実行が終了した後に、オブジェクトは囲まれた最後のタスクからのオブジェクトの値をコピー割り当てされます。

### **reduction**

`reduction` 構文は、`variable-list` にある各オブジェクトの囲まれたタスク・コンストラクタで与えられた演算子を持つリダクション演算を実行します。`operator` と `variable-list` は、OpenMP 仕様と同じように定義されます。

### **ordered**

`ordered` 構文は、オリジナルのシーケンシャル実行順序で、囲まれた `task` コンストラクタで、`ordered` コンストラクタを実行します。`ordered` と結合される `taskq` ディレクティブには `ordered` 構文がなければなりません。

### **nowait**

`nowait` 構文は、`taskq` の最後にある暗黙的なバリアを削除します。`taskq` コンストラクタにキューされたすべての `task` コンストラクタが処理される前に、スレッドは `taskq` コンストラクタを終了できます。

## task コンストラクタ

```
#pragma intel omp task [clause[,]clause]...]  
structured-block
```

clause は次のいずれかです。

- private( variable-list )
- captureprivate( variable-list )

### private

private 構文は、task の variable-list にある各オブジェクトの default-constructed バージョンの private を生成します。変数に参照されるオリジナルのオブジェクトは、コンストラクタに入る時点で中間値を持ちます。コンストラクタの動的範囲内では、オブジェクトは変更してはいけません。そして、コンストラクタから出る時点で中間値を持ちます。

### captureprivate

captureprivate 構文は、task がエンキューされた時点で、task の variable-list に各オブジェクトにある copy-constructed バージョンの private を生成します。各変数に参照されるオリジナルのオブジェクトは、その値を保持しますが、task コンストラクタの動的範囲内では、変更してはいけません。

## parallel と taskq コンストラクタの組み合わせ

```
#pragma intel omp parallel taskq ¥  
[clause[,]clause]...]  
structured-block
```

clause は次のいずれかです。

- if(scalar-expression)
- num\_threads(integer-expression)
- copyin(variable-list)
- default(shared | none)
- shared(variable-list)
- private(variable-list)
- firstprivate(variable-list)
- lastprivate(variable-list)
- reduction(operator : variable-list)
- ordered

Clause は、上記の OpenMP の parallel コンストラクタまたは taskq コンストラクタと同様です。

## 関数例

下記のtest1関数は、ワークキューイング・モデルを使用する並列化処理を示します。parallel taskq プラグマを持つループとtaskプラグマを持つループ本体の作業を付けることによって並列処理を記述できます。parallel taskq プラグマは、囲まれたtaskプラグマに指定された作業単位をエンキューするwhileループの環境を指定します。従って、ループの制御構造とエンキューは、シングルスレッドで実行され、チーム内の他のスレッドはtaskqキューからの作業をデキューし、実行します。captureprivate構文は、各タスクがエンキューされる時点、つまり、シーケンシャル・セマンティクスが保存される時点で、リンクポイントのprivateコピーが確実に取り込まれるようにします。

```
void test1 (LIST p)
{
    #pragma intel omp parallel taskq shared(p)
    {
        while (p != NULL)
        {
            #pragma intel omp task captureprivate(p)
            {
                do_work1(p);
            }
            p = p->next;
        }
    }
}
```

## OpenMP\* の使用例

次の例に、OpenMP 機能の使用方法を示します。

### 単純な差分演算子

この例は、各繰返しごとにワーク量が異なる単純な並列ループを示したものです。負荷のバランスをとるために、ダイナミック・スケジューリングを使用しています。並列領域の最後に暗黙的なバリアがあるため、forにnowaitが含まれています。

```
void for_1 (float a[], float b[], int n)
{
    int i, j;
    #pragma omp parallel shared(a,b,n) private(i,j)
    {
        #pragma omp for schedule(dynamic,1) nowait
        for(i = 1; i < n; i++)
        {
```

```

        for(j = 0; j <= i; j++)
            b[j + n*i] = (a[j + n*i] + a[j + n*(i-1)]) / 2.0;
    }
}

```

## 2つの差分演算子

下記の例では、fork/joinのオーバーヘッドを減らすために融合される2つの並列ループを使用します。2番目のループで使用するすべてのデータは最初のループで使用されるすべてのデータと異なるため、最初のforにはnowaitを含んでいます。

```

void for_2 (float a[], float b[], float c[], ¥
float d[], int n, int m)
{
    int i, j;
    #pragma omp parallel shared(a,b,c,d,n,m) private(i,j)
    {
        #pragma omp for schedule(dynamic,1) nowait
        for(i = 1; i < n; i++)
        {
            for(j = 0; j <= i; j++)
                b[j + n*i] = ( a[j + n*i] + a[j + n*(i-1)] ) / 2.0;
        }

        #pragma omp for schedule(dynamic,1) nowait
        for(i = 1; i < m; i++)
        {
            for(j = 0; j <= i; j++)
                d[j + m*i] = ( c[j + m*i] + c[j + m*(i-1)] ) / 2.0;
        }
    }
}

```

# 自動並列化

## 自動並列化の概要

インテル® C++ コンパイラの自動並列化機能は、入力プログラムのシーケンシャル部分を同等のマルチ・スレッド・コードに自動的に変換します。自動パラライザは、プログラムのループのデータフローを分析して、安全かつ効率的に並列実行可能なループに対するマルチスレッド・コードを生成します。これにより、SMP(対称型マルチプロセッサ)システムの並列アーキテクチャを活用できます。

自動並列化は次のようなユーザの負担を軽減します。

- ワークシェアリング候補であるループを見つけなければならない。
- 正しい並列実行を確認するためにデータフロー分析を行う。
- OpenMP ディレクティブのプログラミングに必要な場合、スレッドコード生成のデータをパーティショニングする。

並列ランタイム・サポートは、ループの反復修正、スレッド・スケジューリング、および同期化の詳細を処理するような、OpenMP\*と同じランタイム機能を提供します。

OpenMPディレクティブはシリアル・アプリケーションを素早く並列アプリケーションに変換できる一方、並列処理を含み、適切なコンパイラ・ディレクティブを追加するアプリケーション・コードの特定部分を、プログラマは明示的に識別する必要があります。-parallelオプションで起動された自動並列化は、並列処理を含むループ構造を自動的に識別します。コンパイル中、コンパイラは、並列処理のためにコード・シーケンスを別々のスレッドに自動的に分解しようと試みます。他にプログラマにかかる負荷はありません。

次の例は、2つのスレッドで同時に実行できるようにループの反復を分割する方法を示します。

### オリジナルの連続コード

```
for (i=1; i<100; i++)  
{  
    a[i] = a[i] + b[i] * c[i];  
}
```

### 変換された並列コード

```
Thread 1  
for (i=1; i<50; i++)  
{  
    a[i] = a[i] + b[i] * c[i];  
}  
Thread 2  
for (i=50; i<100; i++)  
{  
    a[i] = a[i] + b[i] * c[i];  
}
```

## 自動並列化のプログラミング

自動並列化機能は、ワークシェアリング・コンストラクタ(ディレクティブの並列化 )などの OpenMP\*のいくつかのコンセプトを取り入れています。ここでは、自動並列化について説明します。

## 効率的な自動並列化の使用法のガイドライン

次の場合、ループは並列化が可能です。

- ループがコンパイル時にカウント可能な場合、つまり、ループの実行回数(ループ繰返し回数)を表す式が、ループに入る直前に生成されることを意味しています。
- FLOW (WRITEの後のREAD)、OUTPUT (READの後のWRITE)、または ANTI (READの後のWRITE) ループ・キャリー・データ依存性がない場合。同じメモリの場所が、ループの異なる反復で参照されるときに、ループ・キャリー・データ依存が起こります。コンパイラの判断で、推測されたループキャリー依存性がランタイム依存性のテストで解決されると、ループは並列化されることがあります。

コンパイラは、コンパイル時に定数ではないループ・パラメータを持つ `parallel for` ループで、実行するメリットを検証するためにランタイム・テストを生成します。

## コーディング・ガイドライン

次のコーディング・ガイドラインにより自動並列化の威力と効率を強化することができます。

- 可能な限りループの繰返し回数を明確化してください。特に繰返し回数が既知の場合は定数を使用し、ループ・パラメータはローカル変数に保存してください。
- コンパイラがキャリー依存データと判断する可能性のある構造(関数呼び出し、不明瞭な間接参照、グローバル参照など)をループ本体内に配置しないでください。

## 自動並列化のデータフロー

自動並列化の処理では、コンパイラは次のステップを実行します。

1. データフローの解析
2. ループの分類
3. 依存性の解析
4. 高度な並列化
5. データのパーティショニング
6. マルチスレッド・コードの生成

これらのステップには次のものが含まれます。

- データフローの解析: プログラムを通してデータのフローを計算します。
- ループの分類: しきい値解析で示されるように 正確さと効率に基づいて並列化のループの候補を決定します。
- 依存性の解析: 各ループのネストで参照における依存性の解析を計算します。
- 高度な並列化

- 依存性のグラフを解析し、並列で実行できるループを決定します。
- ランタイムの依存性を計算します。
- データのパーティショニング: shared、private、およびfirstprivateのアクセスのタイプに基づいて、データ参照とパーティションを検査します。
- マルチスレッド・コードの生成
  - ループのパラメータを修正します。
  - スレッドタスクごとに入口/出口を生成します。
  - スレッド生成と同期化の並列ランタイム・ルーチンへの呼出しを生成します。

## 自動並列化: 有効、オプション、および環境変数

自動パラライザを有効にするには、`-parallel` オプションを使用します。

`-parallel` オプションは、並列で安全に実行できる並列ループを検出して自動的にこれらのループのマルチスレッド・コードを生成します。次に、自動並列化を使用するコマンドの例を示します。

**IA-32 システム:** `prompt>icc -c -parallel prog.c`

**Itanium® ベース・システム:** `prompt>ecc -c -parallel prog.c`

## 自動並列化オプション

`-O2` (または`-O3`)最適化オプションがオン(デフォルトは `-O2`)の場合、`-parallel`オプションは自動並列化を有効にします。`-parallel` オプションは、並列で安全に実行できる並列ループを検出して自動的にこれらのループのマルチスレッド・コードを生成します。

オプション	説明
<code>-parallel</code>	自動パラライザを有効にします。
<code>-parallel_threshold{1-100}</code>	自動並列化に必要な作業しきい値を制御します。詳細は、後のサブセクションを参照してください。
<code>-par_report{1 2 3}</code>	自動並列化の診断メッセージを制御します。詳細は、後のサブセクションを参照してください。

## 自動並列化の環境変数

変数	説明	デフォルト
<code>OMP_NUM_THREADS</code>	使用されるスレッド数を制御します。	実行ファイルを生成する際にシステムに現在搭載されているプロセッサ数
<code>OMP_SCHEDULE</code>	ランタイム・スケジューリングのタイプを指定します。	<code>static</code>



## 自動並列化のしきい値と診断

### しきい値制御

`-par_threshold{n}` オプションは、並列ループ実行の有効性に基づいて、ループの自動並列化のしきい値を設定します。nの値は0から100までを設定できます。デフォルト値は75 です。このオプションは、コンパイル時に計算量が確定できないループに使用します。しきい値は、通常、ループ繰返し回数がコンパイル時に不明なときに関係します。

`-par_threshold{n}` オプションには次のバージョンと機能があります。

- デフォルト: `-par_threshold` はコマンドラインでは、指定されません。これは、`-par_threshold0` が指定されるときと同様です。ループは、計算量に関わらず自動並列化を行います。つまり、常に並列化します。
- `-par_threshold100` - ループは並列化実行が有効であることが確実な場合にのみ自動並列化されます。
- 1 から 99 の値は、有効な速度の向上の可能性の比率を表します。例えば、n=50 の場合、並列実行された場合にコードの速度が向上する可能性が 50パーセントの場合に並列化を行います。
- nのデフォルト値は、n=75 (または`-par_threshold75`)です。  
`-par_threshold` を数字の指定なしで、コマンドラインで使用すると、デフォルト値は75になります。

コンパイラは、作成された複数のスレッドのオーバーヘッドとスレッド間を共有できるワーク量のバランスをとろうとするヒューリスティクスを適用します。

### 診断

`-par_report{0|1|2|3}` オプションは、自動パラライザの診断レベル 0、1、2、3 を次のように制御します。

- `-par_report0` = 診断情報を表示しません。
- `-par_report1` = ループが正常に自動並列化されたことを示します(デフォルト)。並列ループに対して"LOOP AUTO-PARALLELIZED"メッセージを出力します。
- `-par_report2` = 正常に自動並列化されたループおよび不正に自動並列化されたループを表示します。
- `-par_report3` = 2 と同じです。また、自動並列化を妨げる実証された依存および推測された依存についての追加情報を示します(並列化されない理由)。

### 並列化診断レポートの例

次の例は、`-par_report3`で生成された出力結果です。

**IA-32 システム:** `prompt>icc -c -parallel -par_report3 prog.c`

Sample Output
program prog procedure: prog serial loop: line 5: not a parallel candidate due to statement at line 6

```
serial loop: line 9  
flow data dependence from line 10 to line 10, due to  
"a"  
12 Lines Compiled
```

prog.c は次のとおりです。

Sample prog.c

```
/* Assumed side effects */  
  
for (i=1; i<10000; i++)  
{  
    a[i] = foo(i);  
}  
  
/* Actual dependence */  
  
for (i=1; i<10000; i++)  
{  
    a[i] = a[i-1] + i;  
}
```

## トラブルシューティングのヒント

- -par\_threshold0 を使用して、コンパイラが、十分な計算量がないことを推定したかどうかを確認します。
- -par\_report3 を使用して、診断結果を表示します。
- -ipo を使用して、推定される関数の呼出しへの副作用を回避します。

## ベクトル化 (IA-32 のみ)

### ベクトル化の概要

ベクトライザはインテル® C++ コンパイラのコンポーネントです。C++ コンパイラは、MMX®、SSE、SSE2 命令セットにある SIMD 命令を自動的に使用します。ベクトライザは並列に実行できるプログラムの演算子を検出します。そのあと、シーケンシャル・プログラムを変換し、データ型によって1回の演算で 2、4、8、16 要素のいずれかを処理します。

ここには、インテル C++ コンパイラによるベクトル化のガイドライン、オプション解説、および具体例を収録しました(IA-32のみ)。内容は次のとおりです。

- ベクトル化の機能および特徴の早見表(クイック・リファレンス)
- ベクトル化制御用コンパイラ・スイッチの解説
- ベクトル化制御用C++ 言語機能の解説
- Dベクトル化の各段階についての解説と一般的なガイドライン
  - 自動ベクトル化
  - ユーザの介入によるベクトル化
- ベクトル化を行うときの問題点とその解決方法の具体例

### ベクトライザ・オプション

オプション	説明
<code>-ax{i M K W}</code>	ベクトライザを有効にし、専用 IA-32 コードと汎用 IA-32 コードを生成します。通常は、汎用コードのほうが専用コードよりも処理速度は遅くなります。
<code>-x{i M K W}</code>	ベクトライザをオンにし、プロセッサ専用に特化されたコードを生成します。
<code>-vec_reportn</code>	次のようにベクトライザの診断メッセージのレベルを制御します。 <ul style="list-style-type: none"><li>• <math>n=0</math> 診断情報を表示しない。</li><li>• <math>n=1</math> ループのベクトル化が成功したことを示す診断メッセージを表示する(デフォルト)。</li><li>• <math>n=2</math> <math>n=1</math>のメッセージに加えて、ループのベクトル化が失敗したことを示す診断メッセージも表示する。</li></ul> $n=3$ $n=2$ のメッセージに加えて、判明した依存関係または想定される依存関係についての補足情報も表示する。

## 使用方法

`-vec_report {n}` オプションと一緒に `-c`、`-ipo` を、または `-vec_report {n}` オプションと一緒に `-c`、`-x {M|K|W}` または `-ax {M|K|W}` を使用する場合、コンパイラは警告メッセージを表示し、レポートは生成しません。

前述のオプションを使用してレポートを生成するには、`-ipo_obj` オプションを追加する必要があります。 `-c` および `-ipo_obj` の組み合わせで1つのファイルをコンパイルし、それが、オブジェクト・コードを生成し、最終的にレポートが生成されます。

次のコマンドで、ベクトル化のレポートが生成されます。

- `prompt>icc -x {M|K|W} -vec_report3 file.c`
- `prompt>icc -x {M|K|W} -ipo -ipo_obj -vec_report3 file.c`
- `prompt>icc -c -x {M|K|W} -ipo -ipo_obj -vec_report3 file.c`

次のコマンドでは、ベクトル化のレポートは生成されません。

- `prompt>icc -c -x {M|K|W} -vec_report3 file.c`
- `prompt>icc -x {M|K|W} -ipo -vec_report3 file.c`
- `prompt>icc -c -x {M|K|W} -ipo -vec_report3 file.c`

## ループの並列化とベクトル化

`-parallel` オプションと `-x {M|K|W}` オプションを組み合わせると、同一のコンパイルで、自動ループ並列化と自動ループベクトル化の両方を試みるようにコンパイラに指示します。多くの場合、コンパイラは、並列化には最も外側のループ、ベクトル化には最も内側のループを認識します。しかし、有効であると判断された場合、コンパイラは、同じループに並列化とベクトル化を適用します。

ループ並列化(自動またはOpenMP\*ディレクティブのいずれか)の成功は、コンパイラにレポートされるループベクトル化におけるメッセージに影響する場合があるので注意してください。例えば、`-vec_report2` オプションでは、ループのベクトル化が成功しなかったことが示されます。

## プログラミングの基本となるガイドライン

ベクトル化コンパイラの目標は、自動的にSIMD (single-instruction multiple data) 処理を活用することです。次に示したガイドラインと制約条件をよく確認し、コンパイラが最適なベクトル化を行うのを阻害する曖昧な部分を除去するために、あなたのコードと先の項に示すコードの具体例を照合してください。

### ループ本体に関するガイドライン

- 直列型コード(単一基本ブロック)を使用する。
- ベクトルデータだけを使用する。つまり、代入式の右辺には配列と不変式を使用して

ださい。代入式の左辺には配列参照を使用してもかまいません。

- 代入文だけを使用する。

## ループ本体内では避けたほうがよいもの

- 関数呼び出し
- ベクトル化できない演算
- ベクトル化できる複数の型を同じループの中に混在させること
- データ依存性を持つループ出口条件

## ベクトル化できるコードにする方法

ベクトル化可能なコードに変えるには、ループを変更しなければならない場合がよくあります。ただし、変更する部分は、ベクトル化に最低必要なところだけとし、他の部分まで変更しないようにしてください。特に、次の項目に注意してください。

- ループのアンロールはしないこと。これはコンパイラが自動的に行います。
- ループの本体内に文がいくつか含まれている場合は、そのループを複数の単文ループに分解しないこと。

## 制約事項

**ハードウェア:** コンパイラは、それを動かすハードウェアからいくつか制約を受けます。ストリーミング SIMD 拡張命令を実行する場合、ベクトルメモリ演算は、16バイトでアライメントしたメモリ参照が優先するため、ストライドが1のアクセスに制限されます。つまり、コンパイラは、例えばループをベクトル化可能と抽象的に認めたとしても、別のターゲット・アーキテクチャに対してはベクトル化をしない可能性があります。

**コードの書きかた:** ソースコードの書きかたによっては最適化の妨げになるときもあります。例えば、グローバル・ポインタを使用する場合に、2つのメモリ参照が別々の場所で行われるかどうかをコンパイラが判定できないという問題がよく起こります。そうすると、一部の変換処理は順序の並べ替えができなくなります。

コンパイラによる自動ベクトル化を阻害する多くの要因はループ構造の書きかたにあります。ループ本体にはキーワード、演算子、データ参照、およびメモリ演算が含まれており、それらが互いに作用し合うためループの動作がよく見えなくなるのです。

しかし、これらの制約を理解し、診断メッセージの読みかたを知ることにより、そうした既知の制約条件を克服して効率よくベクトル化できるようにプログラムを修正することが可能です。以降の各項では、ループ構造についてベクトライザの機能と制約条件を簡単に述べます。

## データ依存性

データ依存性とは、連続したいくつかのループに含まれている各演算の実行順序を制限する関係のことです。ベクトル化を行うと各演算の実行順序が変わるため、自動ベクトライザはデータ依存性の解析が自由にできる何らかの仕組みを持っていなければなりません。例として、データ依存性を持つコードを下に示します。この例に示した配列の各要素の値は、その要素自体とその要素の前後の要素によって決まります。

データ依存性を持つループ
<pre>float data[N]; int i;  for (i=1; i&lt;N-1; i++) {     data[i]=data[i-1]*0.25+data[i]*0.5+data[i+1]*0.25; }</pre>

上の例に示したループはベクトル化しません。なぜなら、現在の要素`data[i]`に書き込まれる内容は、その前の反復のときに1つ前の要素`data[i-1]`にどんなデータが書き込まれたかによって決まるからです。この例は、わかりやすくするために最初の2回の反復で配列がどのようにアクセスするかを表したものです。

データ依存性を持つループをベクトル化したもの
<pre>for(i=0; i&lt;100; i++) a[i]=b[i]; アクセス パターン read b[0] write a[0] read b[1] write a[1] i=1: READ data[0] READ data[1] READ data[1] WRITE data[1] i=2: READ data[1] READ data[1] READ data[3] WRITE data[2]</pre>

上に示したループを通常どおり逐次実行する場合、2回目の反復のときに読み取られる`data[1]`の値は1回目の反復のときに書き込まれたものになります。ベクトル化を行うためには、元のループのセマンティクスを変えることなく、対象となるすべての反復を並列に実行しなければなりません。

## データ依存性の理論

データ依存性の解析とは、2つのメモリアクセスの重なり合う条件を見つけることです。その条

件は、1つのプログラムの中で参照を2回行くと仮定した場合は、次の2つの事項によって規定されます。

- 参照するいくつかの変数が、メモリ内の同じ領域のエイリアスであるかどうか(つまり、互いに重複しているかどうか)。
- 配列参照の場合は、添字同士の関連性

配列参照を対称に、インテル® C++ コンパイラのデータ依存性解析機構で解析をするときは、使用する時間とメモリ領域を次第に増やしながら判定する一連の検定(判定法)を実行します。どれか1つの次元の中にでも独立性が認められれば、それによって依存関係が排除できるため、最初は1次元ずつ単純な検定をいくつか実行します。宣言済みの次元境界にまたがる可能性のある多次元配列参照については、線形化された形に変換してから各検定を適用できます。この単純な検定の中には、高速 GCD 検定と拡張限界検定があります。高速 GCD 検定は、ループ添字のすべての係数の最大公約数を求め、その最大公約数で定数項を割り切れない場合を「独立性あり」と判定します。拡張限界検定は、添字式の極値同士の重なり合っていないのを判定します(GCD = greatest common divisor、最大公約数)。

どの単純な検定でも独立性を証明できなかった場合は、最終的に Fourier-Motzkin 法の消去を用いた強力な階層型依存性解法を使用して、すべての次元におけるデータ依存性問題を解きます。

## ループの構成要素

ループの構成要素としては、ごく一般的なforやwhile-doのほかにrepeat-untilなどもあります。またgotoやラベルを使ってループを構成する方法もあります。ただし、ループは、入口が1つだけでかつ出口が1つだけでないと、ベクトル化できません。

正しい使用方法
<pre>while (i&lt;n) {     // If branch is inside body of loop      a[i]=b[i]*c[i];     if(a[i]&lt;0.0)     {         a[i]=0.0;     }     i++; }</pre>

誤った使用方法
<pre>while (i&lt;n) {</pre>



```

    if (condition) break;
    // 2nd exit.
    ++j;
}

```

## ループ出口条件

ループ出口条件とは、1つのループの繰返し回数を決める条件のことです。for ループの場合は、固定インデックスによって繰返し回数が決まっています。ループの繰返し回数は数えられるものでなければなりません。つまり、繰返し回数を指定するときは次のいずれかを使用しなければなりません。

- 定数
- ループ不変項
- 最も外側のループ添字の線形関数

ループの出口が計算結果によって左右されるようなループは、その繰返し回数を数えられません。次の例は、可算/不可算ループの構成要素を表しています。

正しい使用方法(可算ループ)
<pre> // Exit condition specified by "N-1b+1" count=N;  ...  while (count!=1b) {     // 1b is not affected within loop     a[i]=b[i]*x;     b[i]=[i]+sqrt(d[i]);     --count; } </pre>

正しい使用方法(可算ループ)
<pre> // Exit condition is "(n-m+2)/2" i=0; for(l=m; l&lt;n; l+=2) {     a[i]=b[i]*x;     b[i]=c[i]+sqrt(d[i]);     ++i; } </pre>



#### 間違った使用方法(不可算ループ)

```
i=0;

// Iterations dependent on a[i]
while(a[i]>0.0)
{
    a[i]=b[i]*c[i];
    ++i;
}
```

## ベクトル化ループの種類

整数ループの場合、Itanium® アーキテクチャ用 MMX® テクノロジ命令およびストリーミング SIMD 拡張命令は、32ビット、16ビット、および 8ビットの整数データ型での算術演算と論理演算のほとんどに各種 SIMD 命令を提供しています。整数丸め演算の最終的な精度を保持するのであればベクトル化できることがあります。例えば、最終的に保存された値が 16ビット整数の場合は、32ビット右シフト演算子はベクトル化されません。また、MMX 命令セットとストリーミング SIMD 拡張命令セットは完全には直交していないため(例えば、バイトシフトに対応していない)、実際にはすべての整数演算がベクトル化できるわけではありません。

32ビット単精度および 64ビット倍精度の浮動小数点数を使用して演算を行うループの場合、ストリーミング SIMD 拡張命令は、+、-、\*、/ という算術演算子に SIMD 命令を提供しています。また、ストリーミング SIMD 拡張命令は、MIN、MAX という二項演算子、および SQRT という単項演算子に SIMD 命令を提供しています。その他いくつかの算術演算子(三角関数 SIN、COS、TAN など)の SIMD 版は、インテル® C++ コンパイラに付属しているベクトル算術演算用ランタイム・ライブラリ収録のソフトウェアで対応しています。

## ストリップ・マイニングとクリーンアップ

このコンパイラは、自動的にループをストリップマイニングし、クリーンアップ・ループを生成します。つまり、ユーザはループをアンロールする必要がありません。これにより、ほとんどの場合はベクトル化の度合いも高まります。

#### ベクトル化前

```
i=0;
while(i<n)
{
    // Original loop code
    a[i]=b[i]+c[i];
    ++i;
}
```

#### ベクトル化後

```
// The vectorizer generates the following two loops
i=0;

while (i < (n-n%4))
{
    // Vector strip-mined loop
    // Subscript [i:i+3] denotes SIMD execution
    a[i:i+3]=b[i:i+3]+c[i:i+3];
    i=i+4;
}

while (i < n)
{
    // Scalar clean-up loop
    a[i]=b[i]+c[i];
}
```

## ループ本体内の文

ベクトル化可能な演算は、浮動小数点データと整数データで異なります。

### 浮動小数点配列の演算

ループ本体内の文には float 型の演算(通常は配列)を使用できます。可能な算術演算は、加算、減算、乗算、除算、否定、平方根、最大値、および最小値です。-xW または -axW コンパイラ・オプションを使用して、Pentium® 4 プロセッサ・システムに合わせて最適化しない限り、倍精度型の演算はできません。

### 整数配列の演算

ループ本体内の文には、char、unsigned char、short、unsigned short、int、および unsigned int という各型を使用できます。sqrt や fabs といった関数を呼び出せます。算術演算に使用できるのは、加算、減算、ビット単位 AND、ビット単位 OR、ビット単位 XOR、除算(16ビットのみ)、乗算(16ビットのみ)、最小値、および最大値だけです。データ型を複数混在させられるのは、変換しても精度が失われない場合だけです。例えば、乗算演算子、シフト演算子、単項演算子は混在させられます。

### その他の演算

上記の浮動小数点演算と整数演算以外の文は使用できません。特に、特殊なデータ型である `__m64` と `__m128` はベクトル化できないので注意してください。ループ本体に関数呼び出しを含めることはできません。ストリーミングSIMD拡張命令の組込み関数(`_mm_add_ps`)は

使用できません。

## 言語サポートとディレクティブ

ここでは、コードをベクトル化するのに役立つ言語機能について説明します。

`__declspec (align (n))` 宣言は、ハードウェアのアライメント境界を克服します。  
`restrict` 指示子とプラグマは、語彙スコープ、データ依存性、両義性の解決のために書きかたを処理します。

### 言語サポート

オプション	説明
<code>__declspec (align (n))</code>	変数をnバイト境界にアライメントするようコンパイラに命令します。変数のアドレスは <code>address mod n=0</code> です。
<code>__declspec (align (n, off))</code>	変数をnバイト境界にアラインするようコンパイラに命令します。各nバイト境界内でのオフセット値は、nで指定します。変数のアドレスは <code>address mod n = off</code> です。
<code>restrict</code>	エイリアス仮定を行うとき一義化機構に自由度を持たせます。これによってベクトル化の度合いがより高くなります。
<code>__assume_aligned (a, n)</code>	配列aがnバイト境界にアラインしていると思なすようコンパイラに指示します。アライン情報が取得できなかった場合に使用します。
<code>#pragma ivdep</code>	ベクトル依存性が存在していると推定されてもそれを無視するようコンパイラに命令します。
<code>#pragma vector {aligned   unaligned   always}</code>	ループをベクトル化するときの方法を指定し、効率の高いヒューリスティックについては無視したほうがよいことを示します。
<code>#pragma novector</code>	ループをベクトル化しないよう指定します。

## マルチ・バージョン・コード

不明な値のポインタにより、データ依存性の解析が、ループの独立性を証明できなかった場合に備えて、マルチ・バージョン・コードはコンパイラによって生成されます。この機能は、動的依存性のテストと呼ばれます。

## プログラム・スコープ

「ベクトル化のサポート」を参照してください。

## 動的依存性のテスト例

### サンプルコード

```
float *p, *q;

for(i=L; i<=U; i++)
{
    p[i]=q[i];
}

...

pL=p*4*L;
pH=p+4*U;
qL=q*4*L;
qH=q+4*U;

if (pH<qL || pL>qH)
{

    // Loop without data dependence
    for(i=L; i<=U; i++)
    {
        p[i]=q[i];
        {aligned | unaligned | always}

        for(i=L; i<=U; i++)
        {
            p[i]=q[i];
        }
    }
}
```

## ベクトル化の具体例

この節では、ベクトル・プログラミングでよく起こる問題について簡単な例をいくつか挙げて説明します。

### 引数のエイリアシング:ベクトルコピー

下の例に示したループはベクトルコピーを行うものですが、これはベクトル化されます。その理由は、コンパイラが`dest[i]`と`src[i]`との区別が付くのを証明できるからです。

ベクトル化可能なコピー操作(区別が付くのを証明されないため)

```
void vec_copy(float *dest, float *src, int len)
{
    int i;
    for(i=0; i<len; i++;)
    {
        dest[i]=src[i];
    }
}
```

下の例に示したrestrictキーワードは、各ポインタが別々のオブジェクトを参照しているのを示します。したがって、マルチ・バージョン・コードを生成しなくてもベクトル化できます。

ベクトル化できるだけの違いがあることを証明するために restrict を使用する

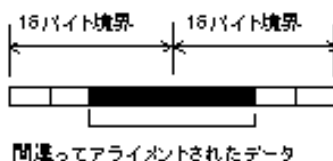
```
void vec copy(float *restrict dest, float *restrict
src, int len)
{
    int i;
    for(i=0; i<len; i++)
    {
        dest[i]=src[i];
    }
}
```

## データのアライメント

16バイト以上のデータ構造体または配列は、その基底アドレスが16の倍数になる方法で各構造体や配列要素の先頭をアライメントしてください。

下の図は、アライメントの合っていないデータが原因でデータ・キャッシュ・ユニット( DCU )を分割した場合の影響を示したものです。アライメントの合っていないデータをロードすると16バイト境界にまたがるため、メモリアクセスが1回余分に発生し、その結果、6～12サイクルのストールが発生します。データのアライメントの合っているのがわかっている場合や、アライメントされているものとして指定した場合は、このストールを避けられます。

## 16バイト境界にまたがる間違っアライメントされたデータ



例えば、要素a[0]とb[0]が16バイト境界にアライメントされているのがわかっている場合は、アライメント・オプション(#pragma vector aligned)をオンにすると、次のループはベクトル化できます。

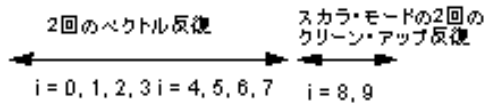
ポインタのアライメントが確認済みの場合

```
float *a, *b;
int i;

for(int i=0; i<10; i++)
{
    a[i]=b[i];
}
```

ベクトル化すると、ループはこのように実行します。

## ベクトルおよびスカラのクリーンアップ反復処理



ベクトル反復処理である  $a[0:3] = b[0:3]$  と  $a[4:7] = b[4:7]$  は、要素  $a[0]$  と  $b[0]$  (同様に  $a[4]$  と  $b[4]$ ) が16バイトでアライメントしている場合には、アライメントの合った移動命令を使用して両方とも実装できます。



**注意**

不正なアライメント・オプションでベクトライザを指定すると、コンパイラの動作は予測がつかなくなります。特に、アライメントの合っていないデータを、アライメントの合っている移動命令で処理すると、不正命令例外が発生します。

## データのアライメント例

下の例は、アライメントの合っていないメモリ命令でしかベクトル化しないループを示したものです。コンパイラはこのローカル配列をアライメントできますが、コンパイル時には `lb` の値がわからないため、アライメントが正しいかどうかの判断はできません。

コンパイル時に変数の値がわからないためにアライメントされないループ

```
void f(int lb)
{
    float z2[N], a2[N], y2[N], x2;
    for(i=lb; i<N; i++)
    {
        a2[i]=a2[i]*x2+y2[i];
    }
}
```

`lb` が4の倍数であることがわかっているときは、下の例に示したように `#pragma vector aligned` を使用してループをアライメントできます。

変数の値が4の倍数であるという前提でアライメントを行う

```
void f(int lb)
{
```

```

float z2[N], a2[N], y2[N], x2;
assert(lb%4==0);

#pragma vector aligned

for(i=lb; i<N; i++)
{
    a2[i]=a2[i]*x2+y2[i];
}

```

## ループ交換と添字: マトリックス乗算

一般に、マトリックス乗算(行列積)は下の例のように記述します。

一般的なマトリックス乗算
<pre> for (i=0; i&lt;N; i++) {     for(j=0; j&lt;n; j++)     {         for(k=0; k&lt;n; k++)         {             c[i][j]=c[i][j]+a[i][k]*b[k][j];         }     } } </pre>

$b[k][j]$  の用法は、ストライドが1ではないため、通常はベクトル化できません。ただし、下の例に示すように、そのループを交換すると、すべてのストライドが1になります。

### 注意

依存関係のあるときは交換できないこともあり、交換してしまうと実行結果が異なる場合があります。

ストライドが1のマトリックス乗算
<pre> for (i = 0; i&lt;N; i++) {     for(k=0; k&lt;n; k++)     {         for(j=0; j&lt;n; j++)         {             c[i][j]=c[i][j]+a[i][k]*b[k][j];         }     } } </pre>

```
}  
}
```



# 最適化サポート機能

## 最適化サポート機能の概要

このセクションでは、ソースコードを直接最適化できるインテル® C++ コンパイラの言語拡張機能について説明します。パフォーマンスの強化や分析に活用できるインテルの拡張ディレクティブおよびライブラリ・ルーチンによる最適化の例を示します。

## コンパイラ・ディレクティブ

### コンパイラ・ディレクティブ

このセクションでは、次の場合に使用する言語拡張ディレクティブを説明します。

- ソフトウェアによるパイプライン処理
- ループカウントとループ分配
- ループのアンロール
- プリフェッチ
- ベクトル化

### Itanium® ベース・アプリケーションのパイプライン処理

swpおよびnoswp ディレクティブは、ループにソフトウェアのパイプライン処理を行うかどうかを指示します。swp ディレクティブは、データ依存性を使用しません。しかし、プロファイル・カウントまたはlop-sidedコントロール・フローに基づいた手法を上書きします。このディレクティブの構文は次のとおりです。

```
#pragma swp
#pragma noswp
```

#### swpディレクティブの例

```
#pragma swp
for (i=0; i<m ; i++)
{
    if (a[i]==0)
    {
        b[i]=a[i]+1;
    }
}
```

```

else
{
    b[i]=a[i]*2;
}
}

```

swp ディレクティブで起動されるソフトウェアによるパイプライン処理の最適化は、最も内側のループをスケジューリングする命令を適用します。また、ループ内で命令が異なるステージに分割され、増大した命令レベルの並列処理を許可します。これで、長い待ち時間の演算による影響を減らし、結果として、より速いループを実行します。ソフトウェアのパイプライン処理に選択されるループは、インライン化されないプロシージャ呼出しを含まない、常に最も内側のループです。最適化は、完全にアンロールされたループを最も内側のループとしてもはや認識しないため、完全にアンロールするループは、追加のループを最も内側のループにすることができません。最適化のレポートをリクエスト、および表示して、ソフトウェアのパイプライン処理が適用されたかどうかを確認できます。(「最適化機構レポートの作成」を参照してください。)

## ループカウントとループ分配

### loop count (n)ディレクティブ

loop count (n) ディレクティブはループカウントがnになるように指定します。このディレクティブの構文は次のとおりです。

```
#pragma loop count (n)
```

n は整数定数です。loop count の値はソフトウェアのパイプライン処理、ベクトル化およびループ変換に使用される手法に影響を与えます。

loop count (n)ディレクティブの例
<pre> #pragma loop count (10000)  for(i=0; i&lt;m; i++) {     //swp likely to occur in this loop     a[i]=b[i]+1.2; } </pre>

### distribute pointディレクティブ

distribute pointディレクティブはループ分配の実行優先度をコンパイラに指示します。このディレクティブの構文は次のとおりです。

```
#pragma distribute point
```

ループ分配は、大きなループを小さなループに分配することがあります。これは、より多くのループにおいてソフトウェアのパイプライン処理を有効にします。ディレクティブをループの内側に置く場合、分配はディレクティブの後で行われ、あらゆるループキャリーの依存性が無視されます。ディレクティブをループの前に置く場合、コンパイラは分配する場所を決定し、データ依存性を監視します。1つのdistributeディレクティブのみが、ループの内側に置かれる際にサ

ポートされます。

#### distribute pointディレクティブの例

```
#pragma distribute point

for(i=1; i<m; i++)
{
    b[i]=a[i]+1;

    ...

    //Compiler will automatically
    //decide where to distribute.
    //Data dependency is observed.

    c[i]=a[i]+b[i];

    ...

    d[i]=c[i]+1;
}

for(i=1; i<m; i++)
{
    b[i]=a[i]+1;

    ...

    #pragma distribute point

    //Distribution will start here,
    //ignoring all loop-carried dependency.

    sub(a,n);
    c[i]=a[i]+b[i];

    ...

    d[i]=c[i]+1;
}
```

## ループのアンロールのサポート

### unrollディレクティブ

unrollディレクティブ(unroll(n) | nounroll)は、コンパイラに、カウントされたループをアンロールする回数を伝えます。このディレクティブの構文は次のとおりです。

```
#pragma unroll
#pragma unroll(n)
#pragma nounroll
```

n は0から255までの整数定数です。unrollディレクティブは、各forループが動作するfor文の前になければなりません。nを指定する場合、最適化機構はループをn回アンロールします。n を省略する場合、またはnが有効範囲外である場合、最適化機構はループをアンロールする数数を割り当てます。unrollディレクティブは、コマンドラインから行われるループ・アンロールの設定を変更します。ディレクティブは、最も内側のネストされたループにのみ適用されます。外側のループに適用された場合、無視されます。コンパイラは、nとループカウンタを比較することによって、正しいコードを生成します。

#### unrollディレクティブの例

```
#pragma unroll(4)

for(i=1; i<m; i++)
{
    b[i]=a[i]+1;
    d[i]=c[i]+1;
}
```

## プリフェッチのサポート

### prefetchディレクティブ

prefetchおよびnoprefetchディレクティブは、データ・プリフェッチがメモリ参照に生成されるか、されないかを指定します。これは、コンパイラが使用する手法に影響を与えます。このディレクティブの構文は次のとおりです。

```
#pragma noprefetch
#pragma prefetch
#pragma prefetch a,b
```

ループの前にprefetch aを置いて、ループ内で式a[j]を使用する場合、コンパイラはループ内のa[j+d] のプリフェッチを挿入します。dはコンパイラによって決定されます。-O3 オプションがオンのとき、このディレクティブはサポートされます。

#### prefetchディレクティブの例

```
#pragma noprefetch b
#pragma prefetch a
```

```
for(i=0; i<m; i++)
{
    a[i]=b[i]+1;
}
```

## ベクトル化のサポート(IA-32)

vectorディレクティブは、プログラム中の後に続くループのベクトル化を制御しますが、コンパイラはネストされたループには適用しません。ネストされたそれぞれのループは、その前に、固有のディレクティブが必要です。vectorディレクティブは、loop 文の前に配置してください。

### vector alwaysディレクティブ

vector always ディレクティブは、ベクトル化するかどうかを決定する際、効率的なヒューリスティックを無効にするようコンパイラに命令します。そして1以外のストライドまたはほとんどアライメントの合っていないメモリアクセスをベクトル化します。

vector alwaysディレクティブの例

```
#pragma vector always

for(i=0; i<=N; i++)
{
    a[32*i]=b[99*i];
}
```

### ivdepディレクティブ

ivdep は、ベクトル依存性が存在していると推定されてもそれを無視するようコンパイラに命令します。正しいコードにするため、コンパイラは、想定される依存性を証明された依存性として扱います。これは、ベクトル化を行わないようにします。このディレクティブは、その決定を無視します。推定されたループの依存性が安全で、無視できる場合にのみivdepを使用してください。下記の例のループは、kの値が不明なため、ivdepではベクトル化をしません（ベクトル化は $k < 0$  の場合、不正です）。

ivdepディレクティブの例

```
#pragma ivdep

for(i=0; i<m; i++)
{
    a[i]=a[i+k]*c;
}
```

### vector alignedディレクティブ

vector alignedディレクティブは、ループのベクトル化が可能である限り、ベクトル化に

よるメリットの存否についての通常のヒューリスティックな判断を無視してベクトル化を行います。alignedまたはunaligned指示子が使用されると、ループは、alignedまたはunaligned演算を使用して、ベクトル化されます。alignedまたはunalignedのいずれか1つを指定します。



#### 注意

アライメントされているものを引数に指定する場合、ループはこの命令を使用してベクトル化可能であることが確実でなければなりません。それ以外の場合、コンパイラは誤ったコードを生成します。次の例に示したループは、アライメントされた指示子を使用して、アライメントされた命令でループがベクトル化されるように要求を出します。コンパイラは通常そうすることが安全であると証明できないように、配列が宣言されているためです。

#### vector alignedディレクティブの例

```
#void foo(float *a)
{
    #pragma vector aligned
    for(i=0; i<m; i++)
    {
        a[i]=a[i]*c;
    }
}
```

コンパイラは、コンパイル時にデータ構造のアライメントがわからない場合に備えて、いくつかのアライメント手法を持っています。以下に簡単な例を次に示します(但し、他の方法もサポートされています)。次の例のループにおいて、aのアライメントが不明な場合、コンパイラはプレリウドのループを生成し、ほとんどの場合に発生する配列参照がアライメントされたアドレスにヒットするまでループを繰り返します。これにより、aのアライメント・プロパティが判明し、そのプロパティに応じてベクトル・ループが最適化されます。

#### アライメント手法の例

```
float *a;

//Alignment unknown
for(i=0; i<100; i++)
{
    a[i]=a[i]+1.0f;
}

//Dynamic loop peeling
p=a & 0x0f;
if (p!=0)
{
    p=(16-p)/4;
    for(i=0; i<p; i++)
```

```

{
    a[i]=a[i]+1.0f;
}
}

//Loop with a aligned.
//Will be vectorized accordingly.
for(i=p; i<100; i++)
{
    a[i]=a[i]+1.0f;
}

```

## novectorディレクティブ

novector ディレクティブは、ループのベクトル化が有効であっても、ベクトル化を行わないよう指定します。この例では、繰返し回数 (ub - lb) が低すぎて、ベクトル化が無駄になるとわかっています。novector を使用して、ループがベクトル化可能であると認識されても、コンパイラにベクトル化しないように指示できます。

### novectorディレクティブの例

```

void foo(int lb, int ub)
{
    #pragma novector
    for(j=lb; j<ub; j++)
    {
        a[j]=a[j]+b[j];
    }
}

```

## アプリケーションの時間測定

アプリケーションの実行速度は、パフォーマンスを測る1つの目安になります。アプリケーションの速度を測るときは、次の環境を考慮してください。

- ユーザが誰もアクティブではないときに、プログラムの時間測定を行います。時間測定中、1つまたは複数のCPU集中型プロセスが起動していると、結果に影響します。
- 最も正確な結果を得るには、毎回、同じ条件でプログラムを実行するようにしてください。特に、同一プログラムの以前のバージョンと比較する際は注意してください。可能ならば、同じシステム(プロセッサ・モデル、メモリ、オペレーティング・システムのバージョンなど)で実行してください。
- システムを変更する必要がある場合、両方のシステムでプログラムの同一バージョンの速度を測ります。こうすることによって、システムによる速度の違いを把握できます。

- 数秒以下で実行されるプログラムの場合は、数回測定し、正しい結果であることを確認してください。外部プログラムをロードするような一部のオーバーヘッドのあるプログラムは、短い時間に大きな影響を与えます。
- プログラムが多くのテキストを表示する場合、プログラムの出力をリダイレクトすることを考慮してください。プログラムの出力をリダイレクトすると、画面I/Oが減少するため、レポートされる時間が変わります。

次に、プログラムの時間測定のモデルを示します。

時間測定のサンプル
<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;time.h&gt; int main(void) {     clock_t start, finish;     long loop;     double duration, loop_calc;     start = clock();     for(loop=0; loop &lt;= 2000; loop++)     {         loop_calc = 123.456 * 789;          //printf() included to facilitate example         printf("¥nThe value of loop is: %d", loop);     }     finish = clock();     duration = (double)(finish - start)/CLOCKS_PER_SEC;     printf("¥n%2.3f seconds¥n", duration); }</pre>

## 最適化機構レポートの作成

インテル® C++ コンパイラ最適化レポートを生成および管理するオプションを提供します。

- `-opt_report` は最適化レポートを生成して、`stderr` に送ります。デフォルトでは、コンパイラは最適化レポートを生成しません。
- `-opt_report_filefilename` は、最適化レポートを作成し *filename* で指定されたファイルに送ります。
- `-opt_report_level{min/med/max}` は、最適化レポートの詳細レベルを指定します。*min* 引数は概要を、*max* 引数は、完全なレポートを作成します。デフォルトは `-opt_report_levelmin` です。



- `-opt_report_routinefileroutine_substring` は、名前の一部に `substring` を含むすべてのルーチンからレポートを作成します。指定されない場合、すべてのルーチンからのレポートが生成されます。デフォルトでは、コンパイラはすべてのルーチンのレポートを生成します。

## レポートを作成する最適化の指定

コンパイラは、`-opt_report_phasephase` オプションの `phase` 引数で指定される最適化機構のレポートを作成できます。複数の最適化機構のレポートを作成するために同じコマンドライン上に複数回、オプションを使用できます。現在、次の最適化機構レポートがサポートされています。

最適化機構論理名	最適化機構のフルネーム
<code>ipo</code>	Interprocedural Optimizer(プロシージャ間の最適化)
<code>hlo</code>	High Level Optimizer(高レベル最適化)
<code>ilo</code>	Intermediate Language Scalar Optimizer(中間言語スカラー最適化機構)
<code>ecg</code>	Code Generator (コード・ジェネレータ)
<code>omp</code>	Open MP
<code>all</code>	すべてのフェーズ

上記の最適化機構の論理名の1つが指定されるとその最適化機構からのすべてのレポートが作成されます。

例えば、`-opt_report_phaseipo -opt_report_phaseecg` は、プロシージャ間の最適化機構とコード・ジェネレータからレポートを生成します。

各最適化機構は、特定の最適化を行うことができます。これらの各最適化は、論理名の 1 つがプリフィックスになります。次に例を示します。

Optimizer_optimization	フルネーム
<code>ipo_inline</code>	Interprocedural Optimizer (プロシージャ間の最適化)、inline expansion of functions (関数のインライン展開)
<code>ipo_constant_propagation</code>	Interprocedural Optimizer(プロシージャ間の最適化)、 constant propagation (定数伝播)
<code>ipo_function_reorder</code>	Interprocedural Optimizer(プロシージャ間の最適化)、 function reorder (関数の再順序化)
<code>ilo_constant_propagation</code>	Intermediate Language Scalar Optimizer(中間言語スカラー最適化機構)、constant propagation (定数の伝播)
<code>ilo_copy_propagation</code>	Intermediate Language Scalar Optimizer(中間言語スカラー最適化機構)、copy propagation(コピー伝播)
<code>ecg_software_pipelining</code>	コード・ジェネレータ、ソフトウェア・パイプライン処理

指定された最適化機構のプリフィックスと一致するすべての最適化レポートが作成されます。例えば、`-opt_report_phase ilo_co` が指定された場合、定数伝播およびコピー伝播の両方からのレポートが作成されます。

## 利用可能なレポート生成

`-opt_report_help` オプションは、レポートの作成に利用可能な最適化機構の論理名をリストします。

# ライブラリ

## ライブラリの概要

インテル® C++ コンパイラはGNU\* CライブラリとDinkumware\* C++ ライブラリを使用します。各ライブラリに関する資料は以下のURLをご覧ください。

### GNU Cライブラリ

[http://www.gnu.org/manual/glibc-2.2.3/html\\_chapter/libc\\_toc.html](http://www.gnu.org/manual/glibc-2.2.3/html_chapter/libc_toc.html) (英語)

### Dinkumware C++ライブラリ

[http://www.dinkumware.com/htm\\_cpl/lib\\_cpp.html](http://www.dinkumware.com/htm_cpl/lib_cpp.html) (英語)

## デフォルト・ライブラリ

インテル® C++ コンパイラには、以下のライブラリがあります。

ライブラリ	説明
libguide.a libguide.so	OpenMP*用
libsvml.a	SVML (Short Vector Mathematical Library)
libirc.a	PGOおよびCPUディスパッチ用にインテルの用意したライブラリ
libimf.a	インテル数値演算ライブラリ
libimf.so	インテル数値演算ライブラリ
libcprts.a libcprts.so	Dinkumware C++ライブラリ
libunwind.a libunwind.so	Unwinder ライブラリ
libcxa.a libcxa.so	インテル・ランタイムは C++ 機能をサポートしています。

別のライブラリや補助的なライブラリにプログラムをリンクするときは、コマンドラインの最後に目的のライブラリを指定してください。例えば、prog.c をコンパイルして mylib.a にリンクするときは、次のコマンドを実行します。

- IA-32 システム: `prompt>icc -oprog prog.c mylib.a`
  - Itanium® ベース・システム: `prompt>ecc -oprog prog.c mylib.a`
- ldリンクの場合、mylib.aライブラリは、コマンドライン上でlibimf.aライブラリよりも前

に指定してください。



#### 注意

Linux\*のシステム・ライブラリとコンパイラ・ライブラリは、`-align` オプションではビルドされません。このため、`-align` オプションでコンパイルし、配布されたコンパイラまたはシステム・ライブラリへ呼び出し、インターフェイスに `long long`、`double`、または `long double` があると、アライメントにおける違いのため、間違った結果が得られます。`-align` でビルドされたコードは、`-align` でビルドされない限り(`-align` なしでは動作しない場合)、インターフェイスでこれらの型を使用するライブラリへの呼び出しはできません。

## 数値演算ライブラリ

インテル C++ コンパイラには、数値演算ライブラリ、`libimf.a` が含まれています。この数値演算ライブラリには、C の標準ランタイム・ライブラリの数値演算関数を最適化したものがあります。`libimf.a` の関数は、Pentium® III プロセッサおよび Pentium 4 プロセッサでのプログラム実行速度を最適化しています。Itanium コンパイラにはまた、Itanium ベース・システムでの実行速度を最適化するために設計された `libimf.a` が含まれています。



#### 注

`-lm` スイッチはリンクに使用されます。`-limf` を最後に指定してください。システム `libm.a` の前に `libimf.a` がリンクされます。

例: `prompt>icc prog.c -limf`

「ライブラリの管理」を参照してください。

## インテル共用ライブラリ

インテル® C++ コンパイラは、ライブラリをリンク時には静的にリンクし、実行時には動的にリンクします。実行時には、ライブラリはDSO (Dynamic Shared Object)としてリンクされます。デフォルトでは、ライブラリは次のようにリンクされます。

- C++ ライブラリ、数値演算ライブラリ、`libcprts.a`ライブラリはリンク時、つまり静的にリンクされます。
- `libcxa.so` は、動的にリンクされます。
- GNUライブラリとLinuxシステム・ライブラリは、動的にリンクされます。

## この手法の利点

この手法には、次の利点があります。

- IA-32 コンパイラ用と Itanium® コンパイラ用の同じモデルを維持できる。

- Linuxモデルとの整合性のあるモデルを使用できる(Linuxモデルでは、システム・ライブラリは動的にリンクされ、アプリケーション・ライブラリは静的にリンクされる)。
- ユーザは、必要に応じてダイナミック版のインテル・ライブラリを使用して、バイナリのサイズを縮小できる。
- ユーザには、インテルが提供するライブラリを配布するライセンスが付与される。

## 共用ライブラリ・オプション

共用ライブラリに関する主なオプションは、`-i_dynamic`と`-shared`です。

`-i_dynamic`オプションを使用して、インテルが提供するライブラリをすべて動的にリンクするように指定できます。このオプションの影響を、次のコマンドの比較で示します。

1. `prompt>icc prog.c`

このコマンドによって、次の結果が生じます(デフォルト)。

- C++ ライブラリ、数値演算ライブラリ、`libirc.a`ライブラリ、および`libcprts.a`ライブラリは、リンク時に静的にリンクされます。
- `libcxa.so`のダイナミック版は、実行時にリンクされます。

静的にリンクされたライブラリを使用すると、アプリケーション・バイナリのサイズが大きくなります。しかし、静的にリンクされるライブラリは、アプリケーションが実行されるシステム上にインストールされている必要はありません。

2. `prompt>icc -i_dynamic prog.c`

このコマンドは、上記のライブラリをすべて動的にリンクします。これには、アプリケーション・バイナリのサイズが小さくなる利点があります。しかし、この場合、すべてのダイナミック版ライブラリが、アプリケーションが実行されるシステム上にインストールされていなければなりません。

`-shared`オプションは、実行ファイルの代わりにDSO (Dynamic Shared Object)を作成するように、コンパイラに指示します。詳細については、マニュアルの`ld`のページを参照してください。

## ライブラリの管理

環境変数`LD_LIBRARY_PATH`には、コロンで区切られたディレクトリ・リストが含まれています。リンクは、ライブラリ(`.a`)ファイルを探すときに、このディレクトリの中を検索します。他のライブラリをリンクに検索させたいときは、その名前を `LD_LIBRARY_PATH`、コマンドライン、応答ファイル、または設定ファイルのいずれかに追加できます。どの場合も、追加したライブラリ名のほうが先にリンクに渡され、そのあと、当該ドライバがいつも指定するインテルのライブラリがリンクに渡されます。

### `LD_LIBRARY_PATH` の変更

例えば、`LD_LIBRARY_PATH` に `/libs` ディレクトリを追加する場合、次のいずれの方

法で可能です。

- コマンドライン: `prompt>export  
LD_LIBRARY_PATH=/libs:$LD_LIBRARY_PATH`
- 起動ファイル `export  
LD_LIBRARY_PATH=/libs:$LD_LIBRARY_PATH`

次のコマンドでは、`file.c` がコンパイルされ `mylib.a` ライブラリにリンクされます。

- IA-32 システム: `prompt>icc file.c mylib.a`
- Itanium® ベース・システム: `prompt>ecc file.c mylib.a`

ファイル名は、次の順番でリンクに渡されます。

1. オブジェクト・ファイル
2. コマンドライン、応答ファイル、または設定ファイルのいずれかに指定されたオブジェクトまたはライブラリ
3. `libimf.a`ライブラリ

# インテル(R) 数値演算ライブラリ

## インテル数値演算ライブラリの概要

インテル® C++ コンパイラには、高度に最適化された正確な数学関数を含む数値演算ソフトウェア・ライブラリが含まれています。これらの関数は、科学やグラフィック・アプリケーションに広く使用されます。同様に、浮動小数点演算を多用するプログラムにも使用されます。C99 `_Complex`データ型のサポートは、`-c99` コンパイラ・オプションを使用することにより含まれます。`mathimf.h`ヘッダファイルには、ライブラリ関数のプロトタイプが含まれています。「インテル数値演算ライブラリの使用」を参照してください。利用可能な関数の一覧は、このセクションの「関数リスト」を参照してください。

## IA-32 と Itanium® ベース・システムの数値演算ライブラリ

アプリケーションにリンクされる数値演算ライブラリは、指定されたコンパイルまたはリンクージ・オプションに依存します。下の表を参照してください。

ライブラリ	説明
<code>libimf.a</code>	デフォルトの静的数値演算ライブラリ
<code>libimf.so</code>	デフォルトの共用数値演算ライブラリ

## インテル数値演算ライブラリの使用

インテル数値演算ライブラリを使用するには、プログラムに、ヘッダファイル`mathimf.h`をインクルードしてください。下記に、数値演算ライブラリを使用した2つのプログラム例を示します。

実関数の使用例
<pre>// real_math.c  #include &lt;stdio.h&gt; #include &lt;mathimf.h&gt;  int main() {  float fp32bits; double fp64bits; long double fp80bits; long double pi_by_four = 3.141592653589793238/4.0;  // pi/4 radians is about 45 degrees.  fp32bits = (float) pi_by_four; // float approximation to pi/4</pre>

```

fp64bits = (double) pi_by_four; // double approximation to pi/4
fp80bits = pi_by_four;         // long double (extended) approximation
to pi/4

// The sin(pi/4) is known to be 1/sqrt(2) or
approximately .7071067

printf("When x = %8.8f, sinf(x) = %8.8f ¥n", fp32bits,
sinf(fp32bits));
printf("When x = %16.16f, sin(x) = %16.16f ¥n", fp64bits,
sin(fp64bits));
printf("When x = %20.20Lf, sinl(x) = %20.20f ¥n",
fp80bits, sinl(fp80bits));

return 0;
}

```

上記のプログラム例では、long doubleデータ型を含むため、-long\_doubleコンパイラ・オプションを必ず含めるようにしてください。

**IA-32 システム:** `icc -long_double real_math.c`

**Itanium® ベース・システム:** `ecc -long_double real_math.c`

a.out の出力は下記のようになります。

```

When x = 0.78539816, sinf(x) = 0.70710678
When x = 0.7853981633974483, sin(x) = 0.7071067811865475
When x = 0.78539816339744827900, sinl(x) =
0.70710678118654750275

```

#### 複素数関数の使用例

```

// complex_math.c

#include <stdio.h>
#include <mathimf.h>

int main()
{

float _Complex c32in,c32out;
double _Complex c64in,c64out;
double pi_by_four= 3.141592653589793238/4.0;

c64in = 1.0 + __I__* pi_by_four;

// Create the double precision complex number 1 + pi/4

```



```

I
// where __I__ is the imaginary unit.

c32in = (float _Complex) c64in;

// Create the float complex value from the double complex
value.

c64out = cexp(c64in);
c32out = cexpf(c32in);

// Call the complex exponential,
// cexp(z) = cexp(x+Iy) = e^(x + i y) = e^x * (cos(y)
+ i sin(y))

printf("When z = %7.7f + %7.7f I, cexpf(z) = %7.7f + %7.7f
I
¥n", crealf(c32in), cimagf(c32in), crealf(c32out), cima
gf(c32out));
printf("When z = %12.12f + %12.12f I, cexp(z) = %12.12f
+ %12.12f I
¥n", creal(c64in), cimag(c64in), creal(c64out), cimagf(
c64out));

return 0;
}

```

**IA-32 システム:** `icc complex_math.c`

**Itanium ベース・システム:** `ecc complex_math.c`

`a.out` の出力は下記ようになります。

When z = 1.0000000 + 0.7853982 I, cexpf(z) = 1.9221154 +  
1.9221156 I

When z = 1.0000000000000 + 0.785398163397 I, cexp(z) =  
1.922115514080 + 1.922115514080 I



**注**

`_Complex`データ型は、Cプログラムではサポートされていますが、C++プログラムにはサポートされていません。

## その他の考慮事項

コンパイラによって、自動的にインライン化される数値演算関数もあります。実際にインライン化される関数は、使用するベクトル化またはプロセッサ固有のコンパイラ・オプションに依存し、異なります。詳細は、「関数のインライン展開の条件」を参照してください。

デフォルトの精度制御または丸めモードを変更すると、いくつかの算術関数で返される結果に影響する場合があります。「浮動小数点演算の精度」を参照してください。

使用するデータ型によるいくつかの重要なコンパイラ・オプションは次のとおりです。

- `-long_double`: `long double`データ型(80ビット浮動小数点)のサポートが必要なプログラムをコンパイルする場合は、このオプションを使用します。このオプションがなくても、コンパイルは成功しますが、`long double`データ型は、`double`データ型にマップされます。
- `-c99`: `_Complex`データ型のサポートが必要なプログラムをコンパイルする場合は、このオプションを使用します

# 数値演算関数

## 関数リスト

インテル数値演算ライブラリ関数の一覧を下記に示します。

ACOS  
ACOSD  
ACOSH  
ANNUITY  
ASIN  
ASIND  
ASINH  
ATAN  
ATAN2  
ATAND  
ATAND2  
ATANH  
CABS  
CACOS  
CACOSH  
CARG  
CASIN  
CASINH  
CATAN  
CATANH  
CBRT  
CCOS  
CCOSH  
CEIL  
CEXP  
CIMAG  
CLOG  
COMPOUND  
CONJ  
COPYSIGN  
COS  
COSD  
COSH  
COT  
COTD

CPOW  
CPROJ  
CREAL  
CSIN  
CSINH  
CSQRT  
CTAN  
CTANH  
ERF  
ERFC  
EXP  
EXP10  
EXP2  
EXPM1  
FABS  
FDIM  
FINITE  
FLOOR  
FMA  
FMAX  
FMIN  
FMOD  
FREXP  
GAMMA  
HYPOT  
ILOGB  
ISNAN  
J0  
J1  
JN  
LDEXP  
LGAMMA  
LGAMMA\_R  
LLRINT  
LLROUND  
LOG  
LOG10  
LOG1P  
LOG2  
LOGB

LRINT  
LROUND  
MODF  
NEARBYINT  
NEXTAFTER  
NEXTTOWARD  
POW  
REMAINDER  
REMQUO  
RINT  
ROUND  
SCALB  
SCALBLN  
SCALBN  
SIN  
SINCOS  
SINCOSD  
SIND  
SINH  
SINHCOSH  
SQRT  
TAN  
TAND  
TANH  
TGAMMA  
TRUNC  
Y0  
Y1  
YN

## 三角関数

インテル数値演算ライブラリは、次の三角関数をサポートします。

### ACOS

**説明:** `acos`関数は、範囲 $[-1,1]$ の $x$ の弧度 $[0,\pi]$ 範囲内で、 $x$ の逆余弦の主値を返します。

**呼出しインターフェイス:**

```
long double acosl(long double x);  
double acos(double x);  
float acosf(float x);
```

## ACOSD

**説明:** acosd関数は、範囲 $[-1,1]$ の $x$ の角度 $[0,180]$ の範囲で、 $x$ の逆余弦の主値を返します。

**呼出しインターフェイス:**

```
long double acosdl(long double x);
double acosd(double x);
float acosdf(float x);
```

## ASIN

**説明:** asin関数は、範囲 $[-1,1]$ の $x$ の弧度 $[-\pi/2, +\pi/2]$ 範囲内で、 $x$ の逆正弦の主値を返します。

**呼出しインターフェイス:**

```
long double asinl(long double x);
double asin(double x);
float asinf(float x);
```

## ASIND

**説明:** asind関数は、範囲 $[-1,1]$ の $x$ の角度 $[-90,90]$ の範囲で、 $x$ の逆正弦の主値を返します。

**呼出しインターフェイス:**

```
long double asindl(long double x);
double asind(double x);
float asindf(float x);
```

## ATAN

**説明:** atan関数は、弧度 $[-\pi/2, +\pi/2]$ の範囲内で、 $x$ の逆正接の主値を返します。

**呼出しインターフェイス:**

```
long double atanl(long double x);
double atan(double x);
float atanf(float x);
```

## ATAN2

**説明:** atan2関数は、弧度 $[-\pi, +\pi]$ の範囲内で、 $y/x$ の逆正接の主値を返します。

**呼出しインターフェイス:**

```
long double atan2l(long double x, long double y);
double atan2(double x, double y);
float atan2f(float x, float y);
```

## ATAND

**説明:** atand関数は、角度 $[-90,90]$ の範囲で、 $x$ の逆正接の主値を返します。

**呼出しインターフェイス:**

```
long double atandl(long double x);
double atand(double x);
float atandf(float x);
```

## ATAND2

**説明:** atand2関数は、角度 $[-180, +180]$ の範囲内で、 $y/x$ の逆正接の主値を返しま

す。

**呼出しインターフェイス:**

```
long double atand2l(long double x, long double y); /* IA-32
only */
double atand2(double x, double y);
float atand2f(float x, float y);
```

## COS

**説明:** cos関数は、弧度法でxの余弦を返します。

**呼出しインターフェイス:**

```
long double cosl(long double x);
double cos(double x);
float cosf(float x);
```

## COSD

**説明:** cosd関数は、角度法でxの余弦を返します。

**呼出しインターフェイス:**

```
long double cosdl(long double x);
double cosd(double x);
float cosdf(float x);
```

## COT

**説明:** cot関数は、弧度法でxの余接を返します。

**呼出しインターフェイス:**

```
long double cotl(long double x);
double cot(double x);
float cotf(float x);
```

## COTD

**説明:** cotd関数は、角度法でxの余接を返します。

**呼出しインターフェイス:**

```
long double cotdl(long double x);
double cotd(double x);
float cotdf(float x);
```

## SIN

**説明:** sin関数は、弧度法でxの正弦を返します。

**呼出しインターフェイス:**

```
long double sinl(long double x);
double sin(double x);
float sinf(float x);
```

## SINCOS

**説明:** sincos関数は、弧度法でxの正弦と余弦の両方を返します。

**呼出しインターフェイス:**

```
void sincosl(long double x, long double *sinval, long
double *cosval);
```

```
void sincos(double x, double *sinval, double *cosval);
void sincosf(float x, float *sinval, float *cosval);
```

## SINCOSD

**説明:** sincosd関数は、角度法でxの正弦と余弦の両方を返します。

**呼出しインターフェイス:**

```
void sincosdl(long double x, long double *sinval, long
double *cosval);
void sincosd(double x, double *sinval, double *cosval);
void sincosdf(float x, float *sinval, float *cosval);
```

## SIND

**説明:** sind関数は、角度法でxの正弦を計算します。

**呼出しインターフェイス:**

```
long double sindl(long double x);
double sind(double x);
float sindf(float x);
```

## TAN

**説明:** tan関数は、弧度法でxの正接を返します。

**呼出しインターフェイス:**

```
long double tanl(long double x);
double tan(double x);
float tanf(float x);
```

## TAND

**説明:** tand関数は、角度法でxの正接を返します。

**呼出しインターフェイス:**

```
long double tandl(long double x);
double tand(double x);
float tandf(float x);
```

## 双曲線関数

インテル数値演算ライブラリは、次の双曲線関数をサポートします。

### ACOSH

**説明:** acosh関数は、xの逆双曲線余弦を返します。

**呼出しインターフェイス:**

```
long double acoshl(long double x);
double acosh(double x);
float acoshf(float x);
```

### ASINH

**説明:** asinh関数は、xの逆双曲線正弦を返します。

**呼出しインターフェイス:**



```
long double asinhl(long double x);
double asinh(double x);
float asinhf(float x);
```

## ATANH

**説明:** atanh関数は、 $x$ の逆双曲線正接を返します。

**呼出しインターフェイス:**

```
long double atanh1(long double x);
double atanh(double x);
float atanhf(float x);
```

## COSH

**説明:** cosh関数は、 $x$ の双曲線余弦、 $(e^x + e^{-x})/2$ を返します。

**呼出しインターフェイス:**

```
long double coshl(long double x);
double cosh(double x);
float coshf(float x);
```

## SINH

**説明:** sinh関数は、 $x$ の双曲線正弦、 $(e^x - e^{-x})/2$ を返します。

**呼出しインターフェイス:**

```
long double sinhl(long double x);
double sinh(double x);
float sinh1f(float x);
```

## SINHCOSH

**説明:** sinhcosh関数は、 $x$ の双曲線正弦と双曲線余弦の両方を返します。

**呼出しインターフェイス:**

```
void sinhcoshl(long double x, long double *sinval, long
double *cosval);
void sinhcosh(double x, float *sinval, float *cosval);
void sinhcoshf(float x, float *sinval, float *cosval);
```

## TANH

**説明:** tanh関数は、 $x$ の双曲線正接、 $(e^x - e^{-x}) / (e^x + e^{-x})$ を返します。

**呼出しインターフェイス:**

```
long double tanhl(long double x);
double tanh(double x);
float tanhf(float x);
```

## 指数関数

インテル数値演算ライブラリは、次の指数関数をサポートします。

### CBRT

**説明:** cbrt関数は、 $x$ の立方根を返します。

**呼出しインターフェイス:**

```
long double cbrt1(long double x);  
double cbrt(double x);  
float cbrtf(float x);
```

## EXP

**説明:** exp関数は、 $e$ を $x$ 乗した値、 $e^x$ を返します。

**呼出しインターフェイス:**

```
long double expl(long double x);  
double exp(double x);  
float expf(float x);
```

## EXP10

**説明:** exp10関数は、10を $x$ 乗した値、 $10^x$ を返します。

**呼出しインターフェイス:**

```
long double exp10l(long double x);  
double exp10(double x);  
float exp10f(float x);
```

## EXP2

**説明:** exp2関数は、2を $x$ 乗した値、 $2^x$ を返します。

**呼出しインターフェイス:**

```
long double exp2l(long double x);  
double exp2(double x);  
float exp2f(float x);
```

## EXPM1

**説明:** expm1関数は、 $e$ を $x$ 乗した値から1をマイナスした値、 $e^x - 1$ を返します。

**呼出しインターフェイス:**

```
long double expm1l(long double x);  
double expm1(double x);  
float expm1f(float x);
```

## FREXP

**説明:** frexp関数は、 $x$ の浮動小数点数を、2のべき乗を乗じた $[1/2, 1)$  範囲で符号付き正規化分数に変換します。符号付き正規化分数が返され、その整数の指数部分をexpに格納します。

**呼出しインターフェイス:**

```
long double frexp(long double x, int *exp);  
double frexp(double x, int *exp);  
float frexpf(float x, int *exp);
```

## HYPOT

**説明:** hypot関数は、平方和の平方根の値を返します。

**呼出しインターフェイス:**

```
long double hypotl(long double x, long double y);  
double hypot(double x, double y);  
float hypotf(float x, float y);
```

## ILOGB

**説明:** ilogb関数は、 $x$ のべき指数を符号付int値として返します。

**呼出しインターフェイス:**

```
int ilogbl(long double x);
int ilogb(double x);
int ilogbf(float x);
```

## LDEXP

**説明:** ldexp関数は、 $x$ と2のべき乗 $exp$ を乗じた値、 $x * 2^{exp}$ を返します。

**呼出しインターフェイス:**

```
long double ldexpl(long double x, int exp);
double ldexp(double x, int exp);
float ldexpf(float x, int exp);
```

## LOG

**説明:** log関数は、 $x$ の自然対数、 $\ln(x)$ を返します。

**呼出しインターフェイス:**

```
long double logl(long double x);
double log(double x);
float logf(float x);
```

## LOG10

**説明:** log10関数は、基数10の $x$ の対数、 $\log_{10}(x)$ を返します。

**呼出しインターフェイス:**

```
long double log10l(long double x);
double log10(double x);
float log10f(float x);
```

## LOG1P

**説明:** log1p関数は、 $(x+1)$ の自然対数、 $\ln(x + 1)$ を返します。

**呼出しインターフェイス:**

```
long double log1pl(long double x);
double log1p(double x);
float log1pf(float x);
```

## LOG2

**説明:** log2関数は、基数2の $x$ の対数、 $\log_2(x)$ を返します。

**呼出しインターフェイス:**

```
long double log2l(long double x);
double log2(double x);
float log2f(float x);
```

## LOGB

**説明:** logb関数は、 $x$ の符号付き指数を返します。

**呼出しインターフェイス:**

```
long double logbl(long double x);
double logb(double x);
```

```
float logbf(float x);
```

## POW

説明: pow関数は、 $x$ を $y$ 乗した値、 $x^y$ を返します。

呼出しインターフェイス:

```
long double powl(double x, double y); /* Itanium®-based  
systems only*/  
double pow(double x, double y);  
float powf(float x, float y);
```

## SCALB

説明: scalb関数は、 $x \cdot 2^y$ を返します。 $y$ は浮動小数点値です。

呼出しインターフェイス:

```
long double scalbl(long double x, long double y);  
double scalb(double x, double y);  
float scalbf(float x, float y);
```

## SCALBN

説明: scalbn関数は、 $x \cdot 2^y$ を返します。 $y$ は整数値です。

呼出しインターフェイス:

```
long double scalbnl(long double x, int y);  
double scalbn(double x, int y);  
float scalbnf(float x, int y);
```

## SCALBLN

説明: scalbln関数は、 $x \cdot 2^n$ を返します。

呼出しインターフェイス:

```
long double scalblnl(long double x, long int n);  
double scalbln(double x, long int n);  
float scalblnf(float x, long int n);
```

## SQRT

説明: sqrt関数は、正確な丸め平方根を返します。

呼出しインターフェイス:

```
long double sqrtl(long double x);  
double sqrt(double x);  
float sqrtf(float x);
```

## 特殊関数

インテル数値演算ライブラリは、次の特殊関数をサポートしています。

### ANNUITY

説明: annuity関数は、一定期間による一定額の支払い(年金)の原価係数を計算し

ます。 $(1 - (1+x)^{-y}) / x$ の  $x$ は利率、 $y$ は期間です。

**呼出しインターフェイス:**

```
double annuity(double x, double y);
float annuityf(float x, double y);
/* All annuity functions: IA-32 only */
```

## COMPOUND

**説明:** compound関数は、複利係数を計算します。 $(1+x)^y$ で $x$ は利率、 $y$ は期間です。

**呼出しインターフェイス:**

```
double compound(double x, double y);
float compoundf(float x, double y);
/* All compound functions: IA-32 only */
```

## ERF

**説明:** erf関数は、誤差関数の値を返します。

**呼出しインターフェイス:**

```
long double erfl(long double x);
double erf(double x);
float erff(float x);
```

## ERFC

**説明:** erfc関数は、相補誤差関数の値を返します。

**呼出しインターフェイス:**

```
long double erfc1(long double x);
double erfc(double x);
float erfcf(float x);
```

## GAMMA

**説明:** gamma関数は、gammaの絶対値の対数値を返します。

**呼出しインターフェイス:**

```
double gamma(double x);
float gammaf(float x);
```

## GAMMA\_R

**説明:** gamma\_r関数は、gammaの絶対値の対数値を返します。gamma関数の符号は、外部整数 signgamで返されます。

**呼出しインターフェイス:**

```
double gamma_r(double x, int *signgam);
float gammaf_r(float x, int *signgam);
```

## J0

**説明:** 0次の $x$ のベッセル関数(第1種)を計算します。

**呼出しインターフェイス:**

```
double j0(double x);
float j0f(float x);
```

## J1

**説明:** 1次のxのベッセル関数(第1種)を計算します。

**呼出しインターフェイス:**

```
double j1(double x);  
float j1f(float x);
```

## JN

**説明:** n次のxのベッセル関数(第1種)を計算します。

**呼出しインターフェイス:**

```
double jn(int n, double x);  
float jnf(int n, float x);
```

## LGAMMA

**説明:** lgamma関数は、gammaの絶対値の対数値を返します。

**呼出しインターフェイス:**

```
long double lgammal(long double x); /* Itanium®-based  
systems only */  
double lgamma(double x);  
float lgammaf(float x);
```

## LGAMMA\_R

**説明:** lgamma\_r関数は、gammaの絶対値の対数値を返します。gamma関数の符号は、外部整数 signgamで返されます。

**呼出しインターフェイス:**

```
double lgamma_r(double x, int *signgam);  
float lgammaf_r(float x, int *signgam);
```

## TGAMMA

**説明:** tgamma関数は、xのgamma関数を計算します。

**呼出しインターフェイス:**

```
long double tgammal(long double x); /* Itanium-based  
systems only */  
double tgamma(double x);  
float tgammaf(float x);
```

## Y0

**説明:** 0次のxのベッセル関数(第2種)を計算します。

**呼出しインターフェイス:**

```
double y0(double x);  
float y0f(float x);
```

## Y1

**説明:** 1次のxのベッセル関数(第2種)を計算します。

**呼出しインターフェイス:**

```
double y1(double x);  
float y1f(float x);
```

## YN

**説明:**  $n$ 次の $x$ のベッセル関数(第2種)を計算します。

**呼出しインターフェイス:**

```
double yn(int n, double x);
float ynf(int n, float x);
```

## 丸め関数

インテル数値演算ライブラリは、次の丸め関数をサポートします。

### CEIL

**説明:** `ceil`関数は、 $x$ より小さくない値で、最も小さい整数値を浮動小数点数として返します。

**呼出しインターフェイス:**

```
long double ceill(long double x);
double ceil(double x);
float ceilf(float x);
```

### FLOOR

**説明:** `floor`関数は、 $x$ より大きくない値で、最も大きい整数値を浮動小数点値として返します。

**呼出しインターフェイス:**

```
long double floorl(long double x);
double floor(double x);
float floorf(float x);
```

### LRINT

**説明:** `lrint`関数は、`long int`として丸めた整数値を返します。

**呼出しインターフェイス:**

```
long int lrintl(long double x);
long int lrint(double x);
long int lrintf(float x);
```

### LLRINT

**説明:** `llrint`関数は、`long long int`として丸めた整数値を返します。

**呼出しインターフェイス:**

```
long long int llrintl(long double x);
long long int llrint(double x);
long long int llrintf(float x);
```

### LROUND

**説明:** `lround`関数は、`long int`として丸めた整数値を返します。

**呼出しインターフェイス:**

```
long int lroundl(long double x);
long int lround(double x);
long int lroundf(float x);
```

## LLROUND

**説明:** llround関数は、long long intとして丸めた整数値を返します。

**呼出しインターフェイス:**

```
long long int llroundl(long double x);
long long int llround(double x);
long long int llroundf(float x);
```

## MODF

**説明:** modf関数は、xの符号付き小数点部分の値を返し、\*iptrに浮動小数点形式で整数部分をストアします。

**呼出しインターフェイス:**

```
long double modfl(long double x, long double *iptr);
double modf(double x, double *iptr);
float modff(float x, float *iptr);
```

## NEARBYINT

**説明:** nearbyint関数は、浮動小数点数として丸めた整数値を返します。

**呼出しインターフェイス:**

```
long double nearbyintl(long double x);
double nearbyint(double x);
float nearbyintf(float x);
```

## RINT

**説明:** rint関数は、浮動小数点数として丸めた整数値を返します。

**呼出しインターフェイス:**

```
long double rintl(long double x);
double rint(double x);
float rintf(float x);
```

## ROUND

**説明:** round関数は、浮動小数点数として丸めた整数値を返します。

**呼出しインターフェイス:**

```
long double roundl(long double x);
double round(double x);
float roundf(float x);
```

## TRUNC

**説明:** trunc関数は、浮動小数点数として切り捨てられた整数値を返します。

**呼出しインターフェイス:**

```
long double trunc1(long double x);
double trunc(double x);
float truncf(float x);
```

## 剰余関数

インテル数値演算ライブラリは、次の剰余関数をサポートしています。



## FMOD

**説明:** fmod関数は、整数nの値 $x - n * y$ を返します。yが非ゼロの場合、その結果は、xと同じ符号を持ち、yの絶対値より小さい値になります。

**呼出しインターフェイス:**

```
long double fmodl(long double x, long double y);
double fmod(double x, double y);
float fmodf(float x, float y);
```

## REMAINDER

**説明:** remainder関数は、 $x \text{ REM } y$ の値を返します。

**呼出しインターフェイス:**

```
long double remainderl(long double x, long double y);
double remainder(double x, double y);
float remainderf(float x, float y);
```

## REMQUO

**説明:** remquo関数は、 $x \text{ REM } y$ の値を返します。

**呼出しインターフェイス:**

```
long double remquol(long double x, long double y, int
*quo);
double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
/* All remquo functions: Itanium®-based systems only*/
```

## その他の関数

インテル数値演算ライブラリは、次に示すその他の関数をサポートします。

## COPYSIGN

**説明:** copysign関数は、xの絶対値とyの符号を返します。

**呼出しインターフェイス:**

```
long double copysignl(long double x, long double y);
double copysign(double x, double y);
float copysignf(float x, float y);
```

## FABS

**説明:** fabs関数は、xの絶対値を返します。

**呼出しインターフェイス:**

```
long double fabsl(long double x);
double fabs(double x);
float fabsf(float x);
```

## FDIM

**説明:** fdim関数は、正の異なる値、 $x - y$  ( $x > y$ ) または  $+0$  ( $x \leq y$ ) を返します。

**呼出しインターフェイス:**

```
long double fdiml(long double x, long double y);
```

```
double fdim(double x, double y);
float fdimf(float x, float y);
```

## FINITE

**説明:** finite関数は、xがNaNまたは+/-無限大ではない場合、1を返します。それ以外の場合は、0 を返します。

**呼出しインターフェイス:**

```
int finitel(long double x);
int finite(double x);
int finitelf(float x);
/* All finite functions: Itanium®-based systems only*/
```

## FMA

**説明:** fma関数は、 $(x*y)+z$ を返します。

**呼出しインターフェイス:**

```
long double fmal(long double x, long double y, long double
z);
double fma(double x, double y, long double z);
float fmaf(float x, float y, long double z);
/* All the fma functions: Itanium-based systems only */
```

## FMAX

**説明:** fmax関数は、引数の最大値を返します。

**呼出しインターフェイス:**

```
long double fmaxl(long double x, long double y);
double fmax(double x, double y);
float fmaxf(float x, float y);
```

## FMIN

**説明:** fmin関数は、引数の最小値を返します。

**呼出しインターフェイス:**

```
long double fminl(long double x, long double y);
double fmin(double x, double y);
float fminf(float x, float y);
```

## ISNAN

**説明:** isnan関数は、xがNaN値を持つ場合のみ、非ゼロ値を返します。

**呼出しインターフェイス:**

```
int isnanl(long double x);
int isnan(double x);
int isnanf(float x);
```

## NEXTAFTER

**説明:** nextafter関数は、y 方向での x の次に表現可能な値を指定した形式で返します。

**呼出しインターフェイス:**

```
long double nextafterl(long double x, long double y);
```

```
double nextafter(double x, double y);
float nextafterf(float x, float y);
```

## NEXTTOWARD

**説明:** nextafter関数は、 $y$  方向での  $x$  の次に表現可能な値を指定した形式で返します。 $x$ と $y$ が等しい場合、関数のタイプに変換した $y$ を返します。

**呼出しインターフェイス:**

```
long double nexttowardl(long double x, long double y);
double nexttoward(double x, double y);
float nexttowardf(float x, float y);
/* All nexttoward functions: Itanium-based systems only
*/
```

## 複素数関数

インテル数値演算ライブラリは、次の複素数関数をサポートします。

### CABS

**説明:** cabs関数は、 $z$ の複素数絶対値を返します。

**呼出しインターフェイス:**

```
double cabs(double _Complex z);
float cabsf(float _Complex z);
```

### CACOS

**説明:** cacos関数は、 $z$ の複素数逆余弦を返します。

**呼出しインターフェイス:**

```
double _Complex cacos(double _Complex z);
float _Complex cacosf(float _Complex z);
```

### CACOSH

**説明:** cacosh関数は、 $z$ の複素数逆双曲線余弦を返します。

**呼出しインターフェイス:**

```
double _Complex cacosh(double _Complex z);
float _Complex cacoshf(float _Complex z);
```

### CARG

**説明:** carg関数は、 $[-\pi, +\pi]$ の範囲で引数の値を返します。

**呼出しインターフェイス:**

```
double carg(double _Complex z);
float cargf(float _Complex z);
```

### CASIN

**説明:** casin関数は、 $z$ の複素数逆正弦を返します。

**呼出しインターフェイス:**

```
double _Complex casin(double _Complex z);
float _Complex casinf(float _Complex z);
```

## CASINH

説明: casinh関数は、 $z$ の複素数逆双曲線正弦を返します。

呼出しインターフェイス:

```
double _Complex casinh(double _Complex z);
float _Complex casinhf(float _Complex z);
```

## CATAN

説明: catan関数は、 $z$ の複素数逆正接を返します。

呼出しインターフェイス:

```
double _Complex catan(double _Complex z);
float _Complex catanf(float _Complex z);
```

## CATANH

説明: catanh関数は、 $z$ の複素数逆双曲線正接を返します。

呼出しインターフェイス:

```
double _Complex catanh(double _Complex z);
float _Complex catanhf(float _Complex z);
```

## CCOS

説明: ccos関数は、 $z$ の複素数余弦を返します。

呼出しインターフェイス:

```
double _Complex ccos(double _Complex z);
float _Complex ccosf(float _Complex z);
```

## CCOSH

説明: ccosh関数は、 $z$ の複素数双曲線余弦を返します。

呼出しインターフェイス:

```
double _Complex ccosh(double _Complex z);
float _Complex ccoshf(float _Complex z);
```

## CEXP

説明: cexp関数は、 $z$ の複素数指数を計算します。

呼出しインターフェイス:

```
double _Complex cexp(double _Complex z);
float _Complex cexpf(float _Complex z);
```

## CIMAG

説明: cimag関数は、 $z$ の虚数部分値を返します。

呼出しインターフェイス:

```
double cimag(double _Complex z);
float cimagf(float _Complex z);
```

## CIS

説明: cis関数は、弧度法で $z$ の余弦と正弦(複素数値)を返します。

呼出しインターフェイス:

```
double _Complex cis(double z);
float _Complex cisf(float z);
```

## CLOG

説明: clog関数は、 $z$ の複素数自然対数を返します。

呼出しインターフェイス:

```
double _Complex clog(double _Complex z);
float _Complex clogf(float _Complex z);
```

## CONJ

説明: conj関数は、虚数部の正弦を反転して、 $z$ の共役複素数を返します。

呼出しインターフェイス:

```
double _Complex conj(double _Complex z);
float _Complex conjf(float _Complex z);
```

## CPOW

説明: cpow関数は、べき乗関数 $x^y$ 複素数を返します。

呼出しインターフェイス:

```
double _Complex cpow(double _Complex x, double _Complex
y);
float _Complex cpowf(float _Complex x, float _Complex
y);
```

## CPROJ

説明: cproj関数は、リーマン球上での $z$ の投影を返します。

呼出しインターフェイス:

```
double _Complex cproj(double _Complex z);
float _Complex cprojf(float _Complex z);
```

## CREAL

説明: creal関数は、 $z$ の実数部の値を返します。

呼出しインターフェイス:

```
double creal(double _Complex z);
float crealf(float _Complex z);
```

## CSIN

説明: csin関数は、 $z$ の複素数正弦を返します。

呼出しインターフェイス:

```
double _Complex csin(double _Complex z);
float _Complex csinf(float _Complex z);
```

## CSINH

説明: csinh関数は、 $z$ の複素数双曲線正弦を返します。

呼出しインターフェイス:

```
double _Complex csinh(double _Complex z);
float _Complex csinhf(float _Complex z);
```

## CSQRT

説明: csqrt関数は、 $z$ の複素数平方根を返します。

呼出しインターフェイス:

```
double _Complex csqrt(double _Complex z);
```

```
float _Complex csqrtf(float _Complex z);
```

#### **CTAN**

**説明:** ctan関数は、 $z$ の複素数正接を返します。

**呼出しインターフェイス:**

```
double _Complex ctan(double _Complex z);  
float _Complex ctanf(float _Complex z);
```

#### **CTANH**

**説明:** ctanh関数は、 $z$ の複素数双曲線正接を返します。

**呼出しインターフェイス:**

```
double _Complex ctanh(double _Complex z);  
float _Complex ctanhf(float _Complex z);
```

# 診断およびメッセージ

## 診断およびメッセージの概要

本書では、コンパイラから出力する各種メッセージについて説明します。メッセージの種類としては、開始メッセージ(サインオン・メッセージ)および、リマーク、警告、エラーのそれぞれに関連した診断メッセージがあります。診断メッセージの場合は必ず、エラーを含むソース行も一緒に標準出力に出力します。

本書では、診断メッセージの重要度の制御方法についても説明します。

## 診断メッセージ

オプション	説明
-w0, -w	エラー・メッセージだけを表示します。-w0でも-wでもまったく同じメッセージが表示されます。
-w1, -w2	警告メッセージとエラー・メッセージを表示します。-w1でも-w2でもまったく同じメッセージが表示されます。デフォルトでは、コンパイラはこの設定になっています。

## 言語診断

ソースファイルの処理中に報告された診断結果を示すメッセージです。この診断結果は次の形式で出力します。

filename (linenum):type [#nn]:message

filename	現在処理しているソースファイルの名前です。
linenum	問題が検出されたソース行番号です。
type	診断メッセージの重要度です。warning (警告)、remark (リマーク)、error (エラー)、catastrophic error (致命的エラー)のいずれかです。
[#nn]	error (または warning)メッセージの番号です。ハードエラーまたは致命的エラーに番号は付きません。
message	診断結果の説明文です。

警告メッセージの例を次に示します。

```
tantst.cpp(3): warning #328: Local variable "increment" never used.
```

標準的なエラーに関する内部エラー・メッセージを表示する場合があります。コンパイル時に

内部エラーが表示されたときは、インテルのカスタマサポートまでご連絡ください。内部エラー・メッセージは次の形式で出力します。

FATAL COMPILER ERROR: *message*

## lintコメントを使用した警告メッセージの出力停止

UNIXのlintプログラムは、C/C++プログラムの中にバグ、移植不能部分、無駄の多い部分がないかどうかを検出します。このコンパイラでは、以下に示した、lint固有のコメントが3つ認識されます。

1. `/*ARGSUSED*/`
2. `/*NOTREACHED*/`
3. `/*VARARGS*/`

lintプログラムと同じくこのコンパイラでも、ソースの特定の位置にこれらのコメントが書き込んであれば、何らかの条件に関する警告を表示しないようにできます。

## 警告メッセージの出力停止方法とリマーク・メッセージの出力方法

警告メッセージを非表示にするときやリマーク・メッセージ機能を有効にするには、前処理やコンパイルの各フェーズで `-w0` オプションか `-Wn` オプションを使用します。このオプションは、次のいずれかの引数とともに入力できます。

オプション	説明
<code>-w0</code> , <code>-w</code>	エラー・メッセージだけを表示します。 <code>-w0</code> でも <code>-w</code> でもまったく同じメッセージが表示されます。
<code>-w1</code> , <code>-w2</code>	警告メッセージとエラー・メッセージを表示します。 <code>-w1</code> でも <code>-w2</code> でもまったく同じメッセージが表示されます。デフォルトでは、コンパイラはこの設定になっています。

不正な箇所がコードに含まれていても、それについて既に把握しているときや実際には問題のないとき(K&RのCコンストラクトなど)にまで警告メッセージを表示させたくない場合があります。例えば、次のコマンドを実行すると `newprog.c` がコンパイルされ、コンパイラ・エラーは表示されますが、警告メッセージは出力されません。

- IA-32 システム: `prompt>icc -W0 newprog.c`
- Itanium® ベース・システム `prompt>ecc -W0 newprog.c`



## エラーの出力個数の制限

エラー・メッセージがいくつ表示したらコンパイラを終了するのは、`-wnn` オプションで指定します。デフォルトでは、エラー数が100を超えるとコンパイルの処理が止まります。

オプション	説明
<code>-wnn</code>	$n$ で指定した個数を超すエラー・メッセージが出力されるとコンパイルの処理が止まります。リマーク・メッセージと警告メッセージはこの個数には入りません。

例えば、次のコマンドラインを実行した場合は、ファイル`a.c`のコンパイル中にエラー・メッセージの出力個数が50個を超えたときにコンパイルの処理が止まります。

- **IA-32 システム:** `prompt>icc -wn50 -c a.c`
- **Itanium® ベース・システム:** `prompt>ecc -wn50 -c a.c`

## リマーク・メッセージ

C/C++で一般的によく使用されるが実際には規約に違反している恐れのある用法を報告するメッセージです。「警告メッセージの出力停止方法とリマーク・メッセージの出力方法」で説明したように、`-W` オプションをレベル4に指定しない限り、リマーク・メッセージは出力も表示もされません。リマーク・メッセージが出ても、変換もリンクも停止しません。リマーク・メッセージを表示しても出力ファイルは影響を受けません。次に、リマーク・メッセージの代表例をいくつか示します。

- `function declared implicitly`
- `type qualifiers are meaningless in this declaration`
- `controlling expression is constant`

# gcc との互換性

## gcc との互換性

インテル® C++ コンパイラで作成される C 言語のオブジェクト・ファイルは、GNU\* gcc コンパイラと GNU C 言語ライブラリである glibc とバイナリ互換があります。C 言語のオブジェクト・ファイルは、インテル・コンパイラまたは gcc コンパイラのいずれかにリンクすることができます。しかし、インテル・ライブラリをリンクに正確に渡すには、インテル・コンパイラを使用してください。詳細は、「リンクとデフォルト・ライブラリ」を参照してください。

GNU C には、ISC 標準準拠の C にはない機能がいくつか含まれています。このバージョンのインテル C++ コンパイラは、これら多くの C 言語の拡張機能をサポートしています。詳細は、Webサイト <http://www.gnu.org> を参照してください。

C 言語の gcc 拡張機能	インテル サポート	GNU の説明と例
Statements and Declarations in Expressions (式の中の複合文と宣言)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Statement-Exprs.html#Statement%20Exprs">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Statement-Exprs.html#Statement%20Exprs</a>
Locally Declared Labels (ローカル宣言レベル)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Local-Labels.html#Local%20Labels">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Local-Labels.html#Local%20Labels</a>
Labels as Values (値としてのラベル)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Labels-as-Values.html#Labels%20as%20Values">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Labels-as-Values.html#Labels%20as%20Values</a>
Nested Functions (入れ子の関数)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Nested-Functions.html#Nested%20Functions">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Nested-Functions.html#Nested%20Functions</a>
Constructing Function Calls (関数呼出しの構築)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Constructing-Calls.html#Constructing%20Calls">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Constructing-Calls.html#Constructing%20Calls</a>
Naming an Expression's Type (式の型の指定)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Naming-Types.html#Naming%20Types">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Naming-Types.html#Naming%20Types</a>
Referring to a Type with typeof (typeof による型の参照)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/typeof.html#typeof">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/typeof.html#typeof</a>
Generalized Lvalues (一般化された左辺値)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Lvalues.html#Lvalues">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Lvalues.html#Lvalues</a>
Conditionals with Omitted Operands (省略されたオペランドを備えた三項条件式)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Conditionals.html#Conditionals">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Conditionals.html#Conditionals</a>
Double-Word Integers (ダブルワード整数)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Long-Long.html#Long%20Long">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Long-Long.html#Long%20Long</a>

Complex Numbers (複素数)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Complex.html#Complex">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Complex.html#Complex</a>
Hex Floats (16進浮動)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Hex-Floats.html#Hex%20Floats">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Hex-Floats.html#Hex%20Floats</a>
Arrays of Length Zero (長さ0の配列)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Zero-Length.html#Zero%20Length">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Zero-Length.html#Zero%20Length</a>
Arrays of Variable Length (引数の数が可変のマクロ)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Variable-Length.html#Variable%20Length">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Variable-Length.html#Variable%20Length</a>
Macros with a Variable Number of Arguments (引数の数が可変のマクロ)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Variadic-Macros.html#Variadic%20Macros">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Variadic-Macros.html#Variadic%20Macros</a>
Slightly Looser Rules for Escaped Newlines (エスケープされた改行に対するより柔軟な規則)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Escaped-Newlines.html#Escaped%20Newlines">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Escaped-Newlines.html#Escaped%20Newlines</a>
String Literals with Embedded Newlines (埋め込まれた改行による文字列リテラル)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Multi-line-Strings.html#Multi-line%20Strings">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Multi-line-Strings.html#Multi-line%20Strings</a>
Non-Lvalue Arrays May Have Subscripts (左辺値でない配列には添字があるかもしれません)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Subscripting.html#Subscripting">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Subscripting.html#Subscripting</a>
Arithmetic on void-Pointers (voidポインタの演算)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Pointer-Arith.html#Pointer%20Arith">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Pointer-Arith.html#Pointer%20Arith</a>
Arithmetic on Function-Pointers (関数ポインタの演算)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Pointer-Arith.html#Pointer%20Arith">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Pointer-Arith.html#Pointer%20Arith</a>
Non-Constant Initializers (非定数の初期化式)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Initializers.html#Initializers">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Initializers.html#Initializers</a>
Compound Literals (コンパウンド・リテラル)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Compound-Literals.html#Compound%20Literals">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Compound-Literals.html#Compound%20Literals</a>
Designated Initializers (指定初期化子)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Designated-Inits.html#Designated%20Inits">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Designated-Inits.html#Designated%20Inits</a>
Cast to a Union Type (Union型へのキャスト)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Cast-to-Union.html#Cast%20to%20Union">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Cast-to-Union.html#Cast%20to%20Union</a>
Case Ranges (範囲付きCase文)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Case-Ranges.html#Case%20Ranges">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Case-Ranges.html#Case%20Ranges</a>
Mixed Declarations and Code (混在する宣言とコード)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Mixed-Declarations.html#Mixed%20Declarations">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Mixed-Declarations.html#Mixed%20Declarations</a>

Declaring Attributes of Functions（関数の属性の宣言）	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Attributes.html#Function%20Attributes">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Attributes.html#Function%20Attributes</a>
Attribute Syntax（属性の構文）	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Attribute-Syntax.html#Attribute%20Syntax">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Attribute-Syntax.html#Attribute%20Syntax</a>
Prototypes and Old-Style Function Definitions（プロトタイプとオールドスタイル関数定義）	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Prototypes.html#Function%20Prototypes">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Prototypes.html#Function%20Prototypes</a>
C++ Style Comments（C++スタイルのコメント）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/C---Comments.html#C++%20Comments">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/C---Comments.html#C++%20Comments</a>
Dollar Signs in Identifier Names（識別子名の中のドル記号）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Dollar-Signs.html#Dollar%20Signs">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Dollar-Signs.html#Dollar%20Signs</a>
The Character ESC in Constants（定数の中のESC文字）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Character-Escapes.html#Character%20Escapes">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Character-Escapes.html#Character%20Escapes</a>
Specifying Attributes of Variables（変数の属性の指定）	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Variable-Attributes.html#Variable%20Attributes">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Variable-Attributes.html#Variable%20Attributes</a>
Specifying Attributes of Types（型の属性の指定）	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Type-Attributes.html#Type%20Attributes">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Type-Attributes.html#Type%20Attributes</a>
Inquiring on Alignment of Types or Variables（型または変数のアラインメントの問い合わせ）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Alignment.html#Alignment">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Alignment.html#Alignment</a>
An Inline Function is As Fast As a Macro（マクロと同じくらい速いインライン関数）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Inline.html#Inline">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Inline.html#Inline</a>
Assembler Instructions with C Expression Operands（C式オペランドを備えたアセンブラ命令）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Extended-Asm.html#Extended%20Asm">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Extended-Asm.html#Extended%20Asm</a>
Controlling Names Used in Assembler Code（アセンブラコードの中で使用される名前の制御）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Asm-Labels.html#Asm%20Labels">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Asm-Labels.html#Asm%20Labels</a>
Variables in Specified Registers（指定されたレジスタの変数）	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Explicit-Reg-Vars.html#Explicit%20Reg%20Vars">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Explicit-Reg-Vars.html#Explicit%20Reg%20Vars</a>
Alternate Keywords（代替	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Alternat">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Alternat</a>

キーワード)		<a href="#">e-Keywords.html#Alternate%20Keywords</a>
Incomplete enum Types (不完全なenum型)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Incomplete-Enums.html#Incomplete%20Enums">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Incomplete-Enums.html#Incomplete%20Enums</a>
Function Names as Strings (文字列としての関 数名)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Names.html#Function%20Names">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Function-Names.html#Function%20Names</a>
Getting the Return or Frame Address of a Function (関数の戻り値ま たはフレームアドレスの取 得)	有	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Return-Address.html#Return%20Address">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Return-Address.html#Return%20Address</a>
Using Vector Instructions Through Built-in Functions (ビルトイン関数のベクトル 命令の使用)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Vector-Extensions.html#Vector%20Extensions">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Vector-Extensions.html#Vector%20Extensions</a>
Other built-in functions provided by GCC (GCC 提供のその他のビルトイン 関数)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Other-Builtins.html#Other%20Builtins">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Other-Builtins.html#Other%20Builtins</a>
Built-in Functions Specific to Particular Target Machines (特定のマシン 用のビルトイン関数)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Target-Builtins.html#Target%20Builtins">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Target-Builtins.html#Target%20Builtins</a>
Pragmas Accepted by GCC (GCCで指定したプラ グマ)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Pragmas.html#Pragmas">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Pragmas.html#Pragmas</a>
Unnamed struct/union fields within structs/unions (構造体と共用体内の名前 なしの構造体と共用体の項 目)	無	<a href="http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Unnamed-Fields.html#Unnamed%20Fields">http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/Unnamed-Fields.html#Unnamed%20Fields</a>

## 参照情報

## コンパイラの制限

### コンパイラの制限

次の表に、コンパイラが処理できる各アイテムのサイズまたは数を示します。表に示したすべ

ての容量は、テスト済みの値です。実際の値は、表の値より大きくなる可能性があります。

アイテム	テスト済みの値
制御構造のネスト(ブロックネスト)	512
条件付きコンパイルのネスト	512
宣言子の修飾子	512
カッコのネストレベル	512
有効文字数(内部識別子)	2048
外部識別子名の長さ	64K
外部識別子/ファイルの数	128K
1つのブロック内の識別子の数	2048
同時に定義されるマクロの数	128K
関数呼び出しのパラメータの数	512
マクロ1つ当たりのパラメータの数	512
文字列内の文字数	128K
オブジェクト内のバイト数	512K
インクルード・ファイルのネストの深さ	512
スイッチ内のケースラベル	32K
1つの構造体または共用体内のメンバ数	32K
1つの列挙子内の列挙定数	8192
構造体のネストレベル	320

## 主要ファイル

### IA-32 コンパイラの主要なファイルの概要

次の表に、IA-32 版コンパイラ用にインストールされるファイルの一覧と簡単な説明を示します。

#### /bin ファイル

ファイル	説明
iccvars.sh	環境変数を設定するためのバッチ・ファイル
icc.cfg	コマンド・ラインから使用するための設定ファイル
icc	インテル® C++ コンパイラ
profmerge	プロファイルに基づく最適化に使用するユーティリティ
profororder	プロファイルに基づく最適化に使用するユーティリティ
xild	プロシージャ間の最適化に使用するツール

#### /lib ファイル

ファイル	説明
libcprts.a	C++ 標準言語ライブラリ
libcxa.so	I/O データの位置を示す C++ 言語ライブラリ
libguide.a	OpenMP* ライブラリ
libguide.so	共用 OpenMP ライブラリ
libimf.a	Linux *専用の数値演算ライブラリ関数(超越関数を含む)
libintrins.a	組込み関数ライブラリ
libirc.a	インテル固有のライブラリ(最適化)
libunwind.a	Unwinder ライブラリ
libsvml.a	ショート・ベクトル数値演算ライブラリ(ベクトライザによって使用されます)

### Itanium® コンパイラの主要なファイル概要

次の表に、Itanium® コンパイラにインストールされるファイルの一覧と簡単な説明を示します。

#### /bin ファイル

ファイル	説明
eccvars.sh	環境変数を設定するためのバッチ・ファイル
ecc.cfg	コマンド・ラインから使用するための設定ファイル
ecc	インテル® C++ コンパイラ
ias	アセンブラ

profmerge	プロファイルに基づく最適化に使用するユーティリティ
proforder	プロファイルに基づく最適化に使用するユーティリティ
xild	プロシージャ間の最適化に使用するツール

## **/lib ファイル**

ファイル	説明
libcprts.a	C++ 標準言語ライブラリ
libcxa.so	I/O データの位置を示す C++ 言語ライブラリ
libirc.a	インテル固有のライブラリ(最適化)
libm.a	数値演算ライブラリ
libguide.a	OpenMP ライブラリ
libguide.so	共用 OpenMP ライブラリ
libmofl.a	インテルのアセンブラが使用する、複数オブジェクト・フォーマット・ライブラリ
libmofl.so	インテル・アセンブラが使用する、共用複数オブジェクト・フォーマット・ライブラリ
libunwinder.a	Unwinder ライブラリ
libintrins.a	組込み関数ライブラリ



# インテル C++ 組込み関数リファレンス

## 概要

### 組込み関数の種類

インテル® Pentium® 4 プロセッサおよびその他のインテル・プロセッサには、最適化されたマルチメディア・アプリケーションの開発用の命令を用意しています。これらの命令は、以前に実装した命令を拡張したものです。これらの命令は、SIMD (Single Instruction, Multiple Data) テクノLOGYを使用します。SIMD命令を使用してデータ要素を並列処理をすると、大量のマルチメディア・データ・ストリームを処理するアプリケーションのパフォーマンスが大きく向上します。インテル® Itanium® プロセッサもまた、そうした命令をサポートします。

これらの命令を使用する最も直接的な方法は、ソースコード内でインライン・アセンブリ言語命令を使用することです。しかし、これは時間のかかる作業です。また、一部のコンパイラは、アセンブリ言語のインライン・プログラミングに対応していません。そこで、インテルでは、組込み関数と呼ばれるAPI拡張機能セットを使用する方法を用意しました。

組込み関数は、ハードウェア・レジスタを直接操作するのではなく、C の関数呼び出しと C の変数の構文を使用できる、コーディング用の拡張機能です。組込み関数によって、プログラマは、アセンブリ言語のプログラミングとレジスタの管理を行う必要がなくなります。また、コンパイラは、命令のスケジューリングを最適化して、実行ファイルの処理速度を上げられます。

さらに、プログラマは、Itanium プロセッサ向けのネイティブ組込み関数を使用して、C および C++ 言語の標準的な構文では生成できない Itanium 命令を利用できます。また、インテル® C++ コンパイラは、すべての IA-32 プラットフォームおよび Itanium ベースのプラットフォーム上で動作する汎用組込み関数もサポートしています。

組込み関数の詳細については、次の参考資料を参照してください。

『インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻: 命令セット・リファレンス』インテル(資料番号243191J)を参照してください。

『Itanium-based Application Developer's Architecture Guide』(インテル)

### 各インテル・プロセッサの組込み関数への対応

プロセッサ:	MMX テクノ ロジの組込 み関数	ストリーミン グSIMD拡張 命令	ストリーミン グSIMD拡張 命令2	Itanium プロ セッサ命令
Itanium プロ セッサ	X	X	N/A	X
Pentium 4 プロセッサ	X	X	X	N/A
Pentium III プロセッサ	X	X	N/A	N/A
Pentium II プロセッサ	X	N/A	N/A	N/A

MMX® テク ノロジ Pentium プ ロセッサ	X	N/A	N/A	N/A
Pentium Pro プロセッサ	N/A	N/A	N/A	N/A
Pentium プ ロセッサ	N/A	N/A	N/A	N/A

## 組込み関数を使用するメリット

組込み関数を使用する大きなメリットは、従来のコーディング手法では使用できない重要な機能を利用できることです。組込み関数を使用すれば、アセンブリ言語の代わりに、C の関数呼び出しと変数の構文を使用してコーディングできます。ほとんどの MMX® テクノロジ、ストリーミングSIMD拡張命令およびストリーミングSIMD拡張命令2の組込み関数は、これらの命令を直接使用するCの関数に対応します。これによって、プログラマは、レジスタを管理する必要がなくなり、コンパイラは命令のスケジューリングを最適化できます。

MMX テクノロジ命令の組込み関数とストリーミングSIMD拡張命令の組込み関数では、次の新しい機能を使用します。

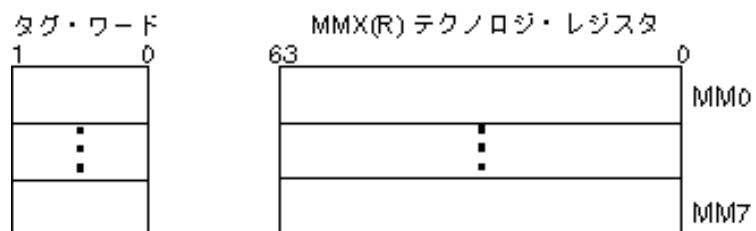
- 新しいレジスタ--最大128ビットのパックドデータの最適なSIMD処理が可能です。
- 新しいデータ型--最大16要素のデータを1つのレジスタにパックできます。

ストリーミングSIMD拡張命令2の組込み関数は、IA-32 についてのみ定義されており、Itanium® ベースのシステムについては定義されていません。ストリーミングSIMD拡張命令2は、128ビットデータ(2個の64ビット倍精度浮動小数点値)を操作します。Itanium アーキテクチャでは、ストリーミングSIMD拡張命令2をサポートしていないため、Itanium ベースのシステム上では、ストリーミングSIMD拡張命令2は実行できません。

## 新しいレジスタ

新しいプロセッサ・アーキテクチャには、新しいレジスタセットが追加されています。MMX 命令は、8個の64ビットレジスタ (mm0～mm7) を使用します。これらのレジスタは、浮動小数点スタックレジスタに別名を付けて使用されます。

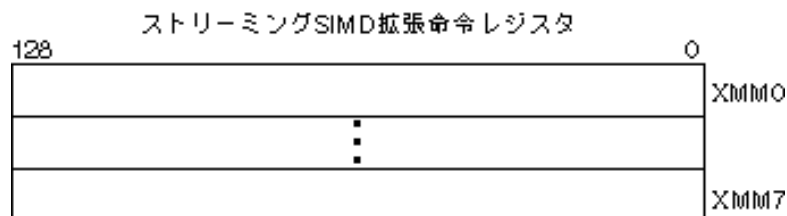
### MMX テクノロジ・レジスタ



OM06832

### ストリーミングSIMD拡張命令レジスタ

ストリーミングSIMD拡張命令レジスタは、8個の128ビットレジスタ(xmm0～xmm7)を使用します。



OM06833

これらの新しいデータレジスタによって、データ要素の並列処理が可能になります。各レジスタが複数のデータ要素を保持できるため、プロセッサは複数のデータ要素を同時に処理できます。このような処理方法は、SIMD (Single Instruction, Multiple Data) 処理と呼ばれます。新しい拡張命令セットのそれぞれの計算命令とデータ操作命令について、その命令を直接に実行する C 組込み関数を用意しています。これにより、プログラマは、レジスタの管理とアセンブリ言語のプログラミングを行う必要がなくなります。また、コンパイラは、命令のスケジューリングを最適化して、実行ファイルの処理速度を上げられます。



#### 注

MMX テクノロジ・レジスタと XMM レジスタは、それぞれ MMX テクノロジの組込み関数とストリーミングSIMD拡張命令/ストリーミングSIMD拡張命令2の組込み関数を実行するために、IA-32 プラットフォーム上で使用されるSIMDレジスタです。Itanium ベースのプラットフォーム上では、MMX テクノロジの組込み関数とストリーミングSIMD拡張命令の組込み関数は、64ビット汎用レジスタと、80ビット浮動小数点レジスタの64ビットの仮数部を使用します。

#### 新しいデータ型

組込み関数は、4つの新しい C データ型をオペランドとして使用します。4つのデータ型は、組込み関数に対するオペランドとして使用される新しいレジスタを表しています。次の表に、プロセッサごとに、これらの新しいデータ型が使用可能かどうかを示します。

## 新しいデータ型への対応

新しいデータ型	MMX テクノロジ	ストリーミング SIMD 拡張命令	ストリーミング SIMD 拡張命令2	Itanium プロセッサ
m64	X	X	X	X
m128	N/A	X	X	X
m128d	N/A	N/A	X	X
m128i	N/A	N/A	X	X

## \_\_m64 データ型

\_\_m64 データ型は、MMX テクノロジの組込み関数に使用される MMX テクノロジレジスタの内容を表します。\_\_m64 データ型は、8個の8ビット値、4個の16ビット値、2個の32ビット値、または1個の64ビット値を保持できます。

## \_\_m128 データ型

\_\_m128 データ型は、ストリーミング SIMD 拡張命令の組込み関数に使用するストリーミング SIMD 拡張命令レジスタの内容を表します。\_\_m128 データ型は、4つの32ビット浮動小数点値を保持できます。

\_\_m128d データ型は、2つの64ビット浮動小数点値を保持できます。

\_\_m128i データ型は、16個の8ビット整数値、8個の16ビット整数値、4個の32ビット整数値、または2個の64ビット整数値を保持できます。

コンパイラは、\_\_m128 型のローカルデータとグローバルデータのアライメントを、スタック上の16バイト境界に合わせます。integer 型、float 型、または double 型の配列のアライメントを合わせるには、declspec 文を使用します。

## 新しいデータ型を使用する際のガイドライン

これらの新しいデータ型は、基本的な ANSI C データ型ではありません。このため、次のような使用上の制限があります。

- 新しいデータ型は、代入文の左辺または右辺で、戻り値またはパラメータとして使用してください。他の算術式("+", "-", "など")にこのデータ型を使用することはできません。
- 新しいデータ型は、バイト要素/構造にアクセスするための共用体など、共用体などの集合体のオブジェクトとして使用してください。
- 新しいデータ型は、本書で説明する組込み関数でのみ使用してください。新しいデータ型は、代入文の両辺で、関数呼び出しに渡すパラメータおよび関数呼び出しからの戻り値として使用できます。

## 命名と使用する構文

ほとんどの組込み関数名は、次の表記規則に従います。

`mm <intrin_op> <suffix>`

<code>&lt;intrin_op&gt;</code>	組込み関数の基本操作を示します。例えば、加算の場合は <code>add</code> 、減算の場合は <code>sub</code> になります。
<code>&lt;suffix&gt;</code>	<p>命令の操作対象となるデータの型を示します。各サフィックスの最初の1文字または2文字は、データがパックドデータ(p)、拡張パックドデータ(ep)、またはスカラデータ(s)であることを示します。その他の文字は、次のとおりデータ型を示します。</p> <ul style="list-style-type: none"> <li>• s 単精度浮動小数点値</li> <li>• d 倍精度浮動小数点値</li> <li>• i128 符号付き128ビット整数</li> <li>• i64 符号付き64ビット整数</li> <li>• u64 符号なし64ビット整数</li> <li>• i32 符号付き32ビット整数</li> <li>• u32 符号なし32ビット整数</li> <li>• i16 符号付き16ビット整数</li> <li>• u16 符号なし16ビット整数</li> <li>• i8 符号付き8ビット整数</li> <li>• u8 符号なし8ビット整数</li> </ul>

変数名を付加した数字は、パックされたオブジェクトの要素を示します。例えば、`r0`は`r`の最下位ワードです。一部の組込み関数は、2つ以上の命令で実行するため、「複合組込み関数」と呼ばれます。

パックされた値は、右から左の順序で表現し、最下位の値がスカラ操作に使用されます。次の操作の例について考えます。

```
double a[2] = {1.0, 2.0};
__m128d t = _mm_load_pd(a);
```

上の操作の結果は、次のそれぞれの操作と同じになります。

```
__m128d t = _mm_set_pd(2.0, 1.0);
__m128d t = _mm_setr_pd(1.0, 2.0);
```

つまり、値`t`を保持する`xmm`レジスタは、次のようになります。

```
127 ┌───┴───┐
    │ 2.0 │ 1.0 │
    └───┴───┘
      0
```

「スカラ」要素は `1.0` です。一部の組込み関数では、命令の性質上、引数として即値(定数整数リテラル)を指定しなければなりません。

## 組込み関数の構文

コードの中で組込み関数を使用するには、次の構文の行を挿入します。

```
data_type intrinsic_name (parameters)
```

各項の意味は次のとおりです。

data_type	戻りデータ型。void、int、__m64、__m128、__m128d、__m128i、__int64のうちのいずれかです。すべてのインテル・アーキテクチャでサポートする組込み関数は、組込み関数の構文の定義に従って、その他のデータ型を返す場合があります。
intrinsic_name	この名前は、実際の命令をインライン展開する代わりに C++ コード内で使用できる関数として機能します。
parameters	各組込み関数が要求するパラメータを表します。

## すべての IA プロセッサでサポートされる組込み関数

### すべての IA プロセッサでサポートされる組込み関数

この節で説明する組込み関数は、すべての IA-32 プラットフォームおよび Itanium® ベースのプラットフォーム上で動作します。これらの組込み関数は、プログラマにとって便利なものです。これらの組込み関数は、次のグループに分類されます。

- 整数演算に関連する組込み関数
- 浮動小数点に関連する組込み関数
- 文字列とブロックのコピーに関連する組込み関数
- その他

### 整数演算に関連する組込み関数



ローテート組込み関数内で定数シフト値を渡すと、パフォーマンスが向上します。

組込み関数	説明
<code>int abs(int)</code>	整数の絶対値を返します。
<code>long labs(long)</code>	long型整数の絶対値を返します。
<code>unsigned long _lrotl(unsigned long value, int shift)</code>	符号なしlong型整数の各ビットを左にローテートします。
<code>unsigned long _lrotr(unsigned long value, int shift)</code>	符号なしlong型整数の各ビットを右にローテートします。
<code>unsigned int __rotl(unsigned int value, int shift)</code>	符号なし整数の各ビットを左にローテートします。
<code>unsigned int rotr(unsigned int value, int shift)</code>	符号なし整数の各ビットを右にローテートします。

### 浮動小数点に関連する組込み関数

組込み関数	説明
<code>double fabs(double)</code>	浮動小数点値の絶対値を返します。
<code>double log(double)</code>	倍精度を使用して、自然対数 $\ln(x)$ , $x>0$ を返します。
<code>float logf(float)</code>	単精度を使用して、自然対数 $\ln(x)$ , $x>0$ を返します。
<code>double log10(double)</code>	倍精度を使用して、10を基数とする対数 $\log_{10}(x)$ , $x>0$ を返します。

<code>float log10f(float)</code>	単精度を使用して、10を基数とする対数 $\log_{10}(x)$ , $x > 0$ を返します。
<code>double exp(double)</code>	倍精度を使用して、指数関数を返します。
<code>float expf(float)</code>	単精度を使用して、指数関数を返します。
<code>double pow(double, double)</code>	倍精度を使用して、 $x$ を $y$ 乗した値を返します。
<code>float powf(float, float)</code>	単精度を使用して、 $x$ を $y$ 乗した値を返します。
<code>double sin(double)</code>	倍精度を使用して、 $x$ の正弦を返します。
<code>float sinf(float)</code>	単精度を使用して、 $x$ の正弦を返します。
<code>double cos(double)</code>	倍精度を使用して、 $x$ の余弦を返します。
<code>float cosf(float)</code>	単精度を使用して、 $x$ の余弦を返します。
<code>double tan(double)</code>	倍精度を使用して、 $x$ の正接を返します。
<code>float tanf(float)</code>	単精度を使用して、 $x$ の正接を返します。
<code>double acos(double)</code>	倍精度を使用して、 $x$ の逆余弦を返します。
<code>float acosf(float)</code>	単精度を使用して、 $x$ の逆余弦を返します。
<code>double acosh(double)</code>	倍精度を使用して、引数の逆双曲線余弦を計算します。
<code>float acoshf(float)</code>	単精度を使用して、引数の逆双曲線余弦を計算します。
<code>double asin(double)</code>	倍精度を使用して、引数の逆正弦を計算します。
<code>float asinf(float)</code>	単精度を使用して、引数の逆正弦を計算します。
<code>double asinh(double)</code>	倍精度を使用して、引数の逆双曲線正弦を計算します。
<code>float asinhf(float)</code>	単精度を使用して、引数の逆双曲線正弦を計算します。
<code>double atan(double)</code>	倍精度を使用して、引数の逆正接を計算します。
<code>float atanf(float)</code>	単精度を使用して、引数の逆正接を計算します。
<code>double atanh(double)</code>	倍精度を使用して、引数の逆双曲線正接を計算します。
<code>float atanhf(float)</code>	単精度を使用して、引数の逆双曲線正接を計算します。
<code>float cabs(double)**</code>	複素数の絶対値を計算します。
<code>double ceil(double)</code>	引数より小さくない倍精度引数の整数値



	のうち、最も小さい値を計算します。
<code>float ceilf(float)</code>	引数より小さくない単精度引数の整数値のうち、最も小さい値を計算します。
<code>double cosh(double)</code>	倍精度引数の双曲線余弦を計算します。
<code>float coshf(float)</code>	単精度引数の双曲線余弦を計算します。
<code>float fabsf(float)</code>	単精度引数の絶対値を計算します。
<code>double floor(double)</code>	引数より大きくない倍精度引数の整数値のうち、最も大きい値を計算します。
<code>float floorf(float)</code>	引数より大きくない単精度引数の整数値のうち、最も大きい値を計算します。
<code>double fmod(double)</code>	倍精度を使用して、第1の引数を第2の引数で割ったときの浮動小数点剰余を計算します。
<code>float fmodf(float)</code>	単精度を使用して、第1の引数を第2の引数で割ったときの浮動小数点剰余を計算します。
<code>double hypot(double, double)</code>	倍精度を使用して、直角三角形の斜辺の長さを計算します。
<code>float hypotf(float)</code>	単精度を使用して、直角三角形の斜辺の長さを計算します。
<code>double rint(double)</code>	IEEE丸めモードを使用して、倍精度で表現される整数値を計算します。
<code>float rintf(float)</code>	IEEE丸めモードを使用して、単精度で表現される整数値を計算します。
<code>double sinh(double)</code>	倍精度引数の双曲線正弦を計算します。
<code>float sinhf(float)</code>	単精度引数の双曲線正弦を計算します。
<code>float sqrtf(float)</code>	単精度引数の平方根を計算します。
<code>double tanh(double)</code>	倍精度引数の双曲線正接を計算します。
<code>float tanhf(float)</code>	単精度引数の双曲線正接を計算します。

\* Itanium® ベースのシステムではサポートしていません。

\*\* この場合の`double`は、2個の単精度(32ビット浮動小数点)要素(実数部と虚数部)で構成される複素数です。

## 文字列とブロックのコピーに関連する組み込み関数



注

次の関数は、Itanium® ベースのプラットフォーム上では組み込み関数としてサポートしていません。

組み込み関数	説明
<code>char *_strset(char *, _int32)</code>	文字列のすべての文字を固定値に設定します。
<code>void *memcmp(const void *cs, const void *ct, size_t n)</code>	メモリの2つの領域を比較します。Return <0 if cs<ct, 0 if cs=ct, or >0 if cs>ct.
<code>void *memcpy(void *s, const void *ct, size_t n)</code>	メモリからコピーします。sを返します。
<code>void *memset(void *s, int c, size_t n)</code>	メモリを固定値に設定します。sを返します。
<code>char *strcat(char *s, const char *ct)</code>	文字列に追加します。sを返します。
<code>int strcmp(const char *, const char *)</code>	2つの文字列を比較します。Return <0 if cs<ct, 0 if cs=ct, or >0 if cs>ct.
<code>char *strcpy(char *s, const char *ct)</code>	文字列をコピーします。sを返します。
<code>size_t strlen(const char *cs)</code>	文字列csの長さを返します。
<code>int strncmp(char *, char *, int)</code>	指定した文字数だけ、2つの文字列を比較します。
<code>int strncpy(char *, char *, int)</code>	指定した文字数だけ、文字列をコピーします。

## その他の組み込み関数



注

`_enable()` と `_disable()` を除いて、ここに記載した関数は、Itanium® 命令ではサポートしていません。

組み込み関数	説明
<code>void *_alloca(int)</code>	バッファを割り当てます。
<code>int _setjmp(jmp_buf) *</code>	<code>setjmp()</code> の高速版。終了処理が省略されます。呼び出し先セーブのレジスタ、スタックポインタ、およびリターンアドレスを保存します。
<code>_exception_code(void)</code>	例外コードを返します。
<code>_exception_info(void)</code>	例外情報を返します。

<code>_abnormal_termination(void)</code>	終了ハンドラ以外では起動できません。対応する最後に試行する領域が予定より早く終了したために終了ハンドラが起動された場合は、TRUEを返します。
<code>void _enable()</code>	割り込みを有効にします。
<code>void _disable()</code>	割り込みを無効にします。
<code>int _bswap(int)</code>	IA-32 命令BSWAP (swap bytes)に対応付けられる組込み関数。リトル/ビッグ・エンディアン形式の32ビット引数を、ビッグ/リトル・エンディアン形式に変換します。
<code>int _in_byte(int)</code>	IA-32 命令INに対応付けられる組込み関数。引数で指定されたポートからデータ・バイトを転送します。
<code>int _in_dword(int)</code>	IA-32 命令INに対応付けられる組込み関数。引数で指定されたポートからダブルワードを転送します。
<code>int _in_word(int)</code>	IA-32 命令INに対応付けられる組込み関数。引数で指定されたポートからワードを転送します。
<code>int _inp(int)</code>	<code>_in_byte</code> と同じです。
<code>int _inpd(int)</code>	<code>_in_dword</code> と同じです。
<code>int _inpw(int)</code>	<code>_in_word</code> と同じです。
<code>int _out_byte(int, int)</code>	IA-32 命令OUTに対応付けられる組込み関数。第2の引数内のデータバイトを、第1の引数で指定するポートに転送します。
<code>int _out_dword(int, int)</code>	IA-32 命令OUTに対応付けられる組込み関数。第2の引数内のダブルワードを、第1の引数で指定するポートに転送します。
<code>int _out_word(int, int)</code>	IA-32 命令OUTに対応付けられる組込み関数。第2の引数内のワードを、第1の引数で指定するポートに転送します。
<code>int _outp(int, int)</code>	<code>_out_byte</code> と同じです。
<code>int _outpd(int, int)</code>	<code>_out_dword</code> と同じです。
<code>int _outpw(int, int)</code>	<code>_out_word</code> と同じです。

\* ライブラリ関数呼び出しとしてサポートします。

## MMX(R) テクノロジーの組込み関数

### MMX® テクノロジーのサポート

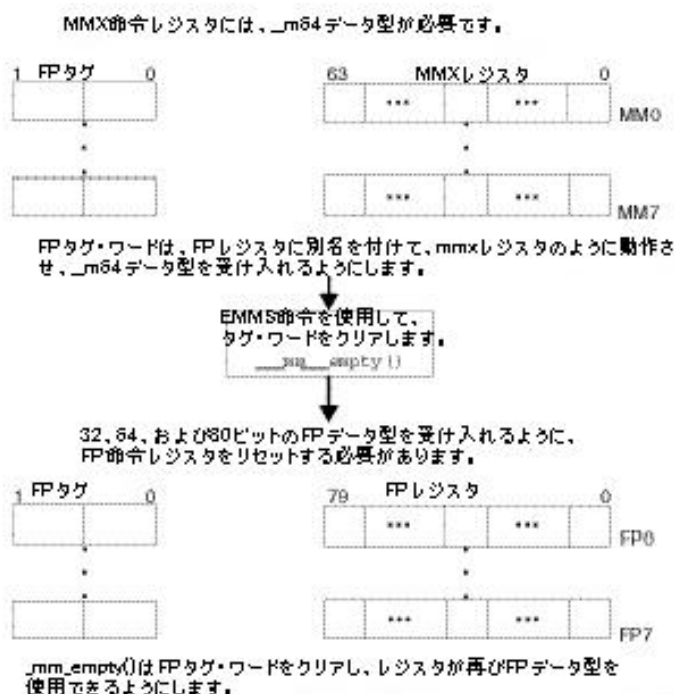
インテル® MMX® テクノロジーは、インテル・アーキテクチャ(IA)命令セットを拡張したものです。MMX テクノロジー命令セットには、57のオペコード、64ビット・クワッドワード・データ型、および8個の64ビットレジスタが追加されました。各レジスタは、レジスタ名mm0～mm7を使用して、直接にアドレス指定できます。

MMX テクノロジーの組込み関数のプロトタイプは、ヘッダ・ファイルmmintrin.h内にあります。

### EMMS 命令: 必要な理由

EMMS 命令には、レジスタを一度空にして、別のデータに使用できるようにする働きがあります。例えば、MMX® テクノロジー命令は、FPレジスタ内のFPタグ・ワードを自動的に有効にして、`_m64`型を使用できるようにします。これによって、FPレジスタセットがリセットされ、MMX テクノロジー・レジスタセットとして別名が付けられます。再びFPレジスタ・セットとして使用できるようにするには、EMMS命令か `_mm_empty()` 組込み関数を使用して、レジスタの状態をリセットします。

### MMX テクノロジー命令の実行後、EMMS命令でレジスタをリセットする理由



CRW06851



### 注意

MMX 命令の実行後、マルチメディア・ステートを空にせずに浮動小数点命令を実行すると、予期しない動作が行われたり、パフォーマンスが低下する場合があります。

## EMMSを使用する際のガイドライン

次に、EMMSを使用する際のガイドラインを示します。

- Itanium® ベース・システム上では使用しないでください。EMMS組込み関数がサポートされている場合でも、Itanium ベースのシステム上には、MMX® 命令やストリーミング SIMD 拡張命令用の特殊なレジスタ(またはオーバーレイ)はありません。
- MMX 命令の実行後、次に浮動小数点(FP)命令を実行する(例えば、float、double、またはlong double型データを計算する)場合は、その前に `_mm_empty()` を使用してください。インテル® C++ コンパイラを使用して MMX 命令を生成したコードでは、次のすべての状況に注意してください。
  - MMX テクノロジー組込み関数を使用する場合
  - `__m64` データ型を使用する、ストリーミングSIMD拡張命令の整数組込み関数を使用する場合
  - `__m64` データ型の変数を参照する場合
  - インライン・アセンブリによって MMX 命令を使用する場合
- MMX 命令の前に `_mm_empty()` を使用しないでください。MMX 命令の前に `_mm_empty()` を使用すると、何のメリットもない操作(no-op)が行われます。
- FP命令を使用する操作と MMX 命令を使用する操作は、別々の関数に分けてください。そうすれば、性能に大きな影響を及ぼすループの中で、マルチメディア・ステートを空にする必要はありません。
- 実行時に `__m64` およびFPデータ型を初期化するときは、`_mm_empty()` を使用してください。これによって、データ型が切り替わる間に、レジスタが確実にリセットされます。
- 次の「正しい使用方法」のコーディング例を参照してください。

誤った使用方法	正しい使用方法
<code>__m64 x = _m_paddb(y, z); float f = init();</code>	<code>__m64 x = _m_paddb(y, z); float f = (_mm_empty(), init());</code>

EMMSの詳細については、<http://www.intel.co.jp/jp/developer/> のインテル Web サイトにアクセスして、EMMSを検索してください。

## MMX® テクノロジーの一般的な組込み関数

MMX® テクノロジーの組込み関数のプロトタイプは、ヘッダ・ファイル `mmintrin.h` 内にあります。

組込み関数名	別名	対応する 命令	操作	符号	飽和
<code>_mm_empty</code>	<code>_mm_empty</code> <code>y</code>	EMMS	MMステートを空にする	--	--

<code>_m_from_int</code>	<code>_mm_cvtsi32_si64</code>	MOVD	intからの変換	--	--
<code>_m_to_int</code>	<code>_mm_cvtsi64_si32</code>	MOVD	intからの変換	--	--
<code>_m_packsswb</code>	<code>_mm_packss_pi16</code>	PACKSSWB	パック	可	可
<code>_m_packssdw</code>	<code>_mm_packss_pi32</code>	PACKSSDW	パック	可	可
<code>_m_packuswb</code>	<code>_mm_packus_pi16</code>	PACKUSWB	パック	符号なし	可
<code>_m_punpckhbw</code>	<code>_mm_unpackhi_pi8</code>	PUNPCKHBW	インターリーブ	--	--
<code>_m_punpckhwd</code>	<code>_mm_unpackhi_pi16</code>	PUNPCKHWD	インターリーブ	--	--
<code>_m_punpckhdq</code>	<code>_mm_unpackhi_pi32</code>	PUNPCKHDQ	インターリーブ	--	--
<code>_m_punpcklbw</code>	<code>_mm_unpacklo_pi8</code>	PUNPCKLBW	インターリーブ	--	--
<code>_m_punpcklwd</code>	<code>_mm_unpacklo_pi16</code>	PUNPCKLWD	インターリーブ	--	--
<code>_m_punpckldq</code>	<code>_mm_unpacklo_pi32</code>	PUNPCKLDQ	インターリーブ	--	--

`void _m_empty(void)`

マルチメディア・ステートを空にします。

詳細は、「EMMS命令: 必要な理由」の図を参照してください。

`__m64 _m_from_int(int i)`

整数オブジェクト*i*を、64ビットの `__m64` オブジェクトに変換します。整数値は0で拡張して64ビットに変換します。

`int _m_to_int(__m64 m)`

`__m64` オブジェクト*m*の下位32ビットを、整数に変換します。

`__m64 _m_packsswb(__m64 m1, __m64 m2)`

符号付き飽和処理を使用して、*m1*の4個の16ビット値を、結果の下位4個の8ビット値にパックします。符号付き飽和処理を使用して、*m2*の4個の16ビット値を、結果の上位4個の8ビット値にパックします。

`__m64 _m_packssdw(__m64 m1, __m64 m2)`

符号付き飽和処理を使用して、*m1*の2個の32ビット値を、結果の下位2個の16ビット値にパックします。符号付き飽和処理を使用して、*m2*の2個の32ビット値を、結果の上位

2個の16ビット値にパックします。

\_\_m64 \_\_m\_packuswb(\_\_m64 m1, \_\_m64 m2)

符号なし飽和処理を使用して、m1の4個の16ビット値を、結果の下位4個の8ビット値にパックします。符号なし飽和処理を使用して、m2の4個の16ビット値を、結果の上位4個の8ビット値にパックします。

\_\_m64 \_\_m\_punpckhbw(\_\_m64 m1, \_\_m64 m2)

m1の上位半分の4個の8ビット値と、m2の上位半分の4個の値をインターリーブ(交互に配置)します。インターリーブはm1のデータから始めます。

\_\_m64 \_\_m\_punpckhwd(\_\_m64 m1, \_\_m64 m2)

m1の上位半分の2個の16ビット値と、m2の上位半分の2個の値をインターリーブします。インターリーブはm1のデータから始めます。

\_\_m64 \_\_m\_punpckhdq(\_\_m64 m1, \_\_m64 m2)

m1の上位半分の32ビット値と、m2の上位半分の32ビット値をインターリーブします。インターリーブはm1のデータから始めます。

\_\_m64 \_\_m\_punpcklbw(\_\_m64 m1, \_\_m64 m2)

m1の下位半分の4個の8ビット値と、m2の下位半分の4個の値をインターリーブします。インターリーブはm1のデータから始めます。

\_\_m64 \_\_m\_punpcklwd(\_\_m64 m1, \_\_m64 m2)

m1の下位半分の2個の16ビット値と、m2の下位半分の2個の値をインターリーブします。インターリーブはm1のデータから始めます。

\_\_m64 \_\_m\_punpckldq(\_\_m64 m1, \_\_m64 m2)

m1の下位半分の32ビット値と、m2の下位半分の32ビット値をインターリーブします。インターリーブはm1のデータから始めます。

## MMX® テクノロジーのパックド算術演算組込み関数

MMX® テクノロジーの組込み関数のプロトタイプは、ヘッダ・ファイル `mmintrin.h` 内にあります。

組込み関数名	別名	対応する命令	操作	符号	引数 - 値の数/ビット数	結果値の数/ビット数
<code>_mm_paddb</code>	<code>_mm_add_pi8</code>	PADDB	加算	--	8/8	8/8
<code>_mm_paddw</code>	<code>_mm_add_pi16</code>	PADDW	加算	--	4/16	4/16
<code>_mm_paddq</code>	<code>_mm_add_pi32</code>	PADDQ	加算	--	2/32	2/32
<code>_mm_paddsb</code>	<code>_mm_adds_pi8</code>	PADDSB	加算	可	8/8	8/8
<code>_mm_paddsw</code>	<code>_mm_adds_pi16</code>	PADDSW	加算	可	4/16	4/16
<code>_mm_paddusb</code>	<code>_mm_adds_pu8</code>	PADDUSB	加算	符号なし	8/8	8/8
<code>_mm_paddusw</code>	<code>_mm_adds_pu16</code>	PADDUSW	加算	符号なし	4/16	4/16
<code>_mm_psubb</code>	<code>_mm_sub_pi8</code>	PSUBB	減算	--	8/8	8/8
<code>_mm_psubw</code>	<code>_mm_sub_pi16</code>	PSUBW	減算	--	4/16	4/16
<code>_mm_psubq</code>	<code>_mm_sub_pi32</code>	PSUBQ	減算	--	2/32	2/32
<code>_mm_psubsb</code>	<code>_mm_subs_pi8</code>	PSUBSB	減算	可	8/8	8/8
<code>_mm_psubsw</code>	<code>_mm_subs_pi16</code>	PSUBSW	減算	可	4/16	4/16
<code>_mm_psubusb</code>	<code>_mm_subs_pu8</code>	PSUBUSB	減算	符号なし	8/8	8/8
<code>_mm_psubusw</code>	<code>_mm_subs_pu16</code>	PSUBUSW	減算	符号なし	4/16	4/16
<code>_mm_pmaddwd</code>	<code>_mm_madd_pi16</code>	PMADDWD	乗算	--	4/16	2/32
<code>_mm_pmulhw</code>	<code>_mm_mulhi_pi16</code>	PMULHW	乗算	可	4/16	4/16 (上位)
<code>_mm_pmulld</code>	<code>_mm_mullo_pi16</code>	PMULLW	乗算	--	4/16	4/16 (下位)

`__m64 _mm_paddb(__m64 m1, __m64 m2)`  
 m1の8個の8ビット値を、m2の8個の8ビット値に加算します。



`__m64 __m_paddw(__m64 m1, __m64 m2)`  
 m1の4個の16ビット値を、m2の4個の16ビット値に加算します。

`__m64 __m_padd (__m64 m1, __m64 m2)`  
 m1の2個の32ビット値を、m2の2個の32ビット値に加算します。

`__m64 __m_paddsb(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の8つの符号付き8ビット値を、m2の8つの符号付き8ビット値に加算します。

`__m64 __m_paddsw(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の4つの符号付き16ビット値を、m2の4つの符号付き16ビット値に加算します。

`__m64 __m_paddusb(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の8つの符号なし8ビット値を、m2の8つの符号なし8ビット値に加算します。

`__m64 __m_paddusw(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の4つの符号なし16ビット値を、m2の4つの符号なし16ビット値に加算します。

`__m64 __m_psubb(__m64 m1, __m64 m2)`  
 m1の8個の8ビット値から、m2の8個の8ビット値を引きます。

`__m64 __m_psubw(__m64 m1, __m64 m2)`  
 m1の4個の16ビット値から、m2の4個の16ビット値を引きます。

`__m64 __m_psubd(__m64 m1, __m64 m2)`  
 m1の2個の32ビット値から、m2の2個の32ビット値を引きます。

`__m64 __m_psubsb(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の8つの符号付き8ビット値から、m2の8つの符号付き8ビット値を引きます。

`__m64 __m_psubsw(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の4つの符号付き16ビット値から、m2の4つの符号付き16ビット値を引きます。

`__m64 __m_psubusb(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の8個の符号なし8ビット値から、m2の8個の符号なし8ビット値を引きます。

`__m64 __m_psubusw(__m64 m1, __m64 m2)`  
 飽和演算を使用して、m1の4つの符号なし16ビット値から、m2の4つの符号なし16ビット値を引きます。

`__m64 __m_pmaddwd(__m64 m1, __m64 m2)`  
 m1の4つの16ビット値にm2の4つの16ビット値を掛けて、4つの32ビット即値を求め、それらを2つずつ合計して2つの32ビットの結果を求めます。

`__m64 __m_pmulhw(__m64 m1, __m64 m2)`  
 m1の4つの符号付き16ビット値にm2の4つの符号付き16ビット値を掛けて、4つの結果

の上位16ビットを求めます。

```
__m64 _m_pmulw(__m64 m1, __m64 m2)
```

m1の4つの16ビット値にm2の4つの16ビット値を掛けて、4つの結果の下位16ビットを求めます。

## MMX® テクノロジのシフト組込み関数

MMX® テクノロジの組込み関数のプロトタイプは、ヘッダ・ファイルmmintrin.h内にあります。

組込み関数名	別名	シフトの方向	シフトの種類	対応する命令
_m_psllw	_mm_sll_pi16	左	論理	PSLLW
_m_psllwi	_mm_slli_pi16	左	論理	PSLLWI
_m_psllld	_mm_sll_pi32	左	論理	PSLLD
_m_psllldi	_mm_slli_pi32	左	論理	PSLLDI
_m_psllq	_mm_sll_si64	左	論理	PSLLQ
_m_psllqi	_mm_slli_si64	左	論理	PSLLQI
_m_psraw	_mm_sra_pi16	右	算術	PSRAW
_m_psrawi	_mm_srai_pi16	右	算術	PSRAWI
_m_psrad	_mm_sra_pi32	右	算術	PSRAD
_m_psradi	_mm_srai_pi32	右	算術	PSRADI
_m_psrlw	_mm_srl_pi16	右	論理	PSRLW
_m_psrlwi	_mm_srli_pi16	右	論理	PSRLWI
_m_psrld	_mm_srl_pi32	右	論理	PSRLD
_m_psrldi	_mm_srli_pi32	右	論理	PSRLDI
_m_psrlq	_mm_srl_si64	右	論理	PSRLQ
_m_psrlqi	_mm_srli_si64	右	論理	PSRLQI

```
__m64 _m_psllw(__m64 m, __m64 count)
```

mの4つの16ビット値を、countで指定した値だけ左にシフトし、下位ビットを0で埋めます。

\_\_m64 \_\_m\_psllwi(\_\_m64 m, int count)

mの4つの16ビット値を、countで指定した値だけ左にシフトし、下位ビットを0で埋めます。パフォーマンス上の理由で、countは定数にしてください。

\_\_m64 \_\_m\_psllld(\_\_m64 m, \_\_m64 count)

mの2つの32ビット値を、countで指定した値だけ左にシフトし、下位ビットを0で埋めます。

\_\_m64 \_\_m\_psllldi(\_\_m64 m, int count)

mの2つの32ビット値を、countで指定した値だけ左にシフトし、下位ビットを0で埋めます。パフォーマンス上の理由で、countは定数にしてください。

\_\_m64 \_\_m\_psllq(\_\_m64 m, \_\_m64 count)

mの64ビット値を、countで指定した値だけ左にシフトし、下位ビットを0で埋めます。

\_\_m64 \_\_m\_psllqi(\_\_m64 m, int count)

mの64ビット値を、countで指定した値だけ左にシフトし、下位ビットを0で埋めます。パフォーマンス上の理由で、countは定数にしてください。

\_\_m64 \_\_m\_psrw(\_\_m64 m, \_\_m64 count)

mの4つの16ビット値を、countで指定した値だけ右にシフトし、上位ビットを符号ビットで埋めます。

\_\_m64 \_\_m\_psrwi(\_\_m64 m, int count)

mの4つの16ビット値を、countで指定した値だけ右にシフトし、上位ビットを符号ビットで埋めます。パフォーマンス上の理由で、countは定数にしてください。

\_\_m64 \_\_m\_psrld(\_\_m64 m, \_\_m64 count)

mの2つの32ビット値を、countで指定した値だけ右にシフトし、上位ビットを符号ビットで埋めます。

\_\_m64 \_\_m\_psrldi(\_\_m64 m, int count)

mの2つの32ビット値を、countで指定した値だけ右にシフトし、上位ビットを符号ビットで埋めます。パフォーマンス上の理由で、countは定数にしてください。

\_\_m64 \_\_m\_psrldi(\_\_m64 m, \_\_m64 count)

mの4つの16ビット値を、countで指定した値だけ右にシフトし、上位ビットを0で埋めます。

\_\_m64 \_\_m\_psrldi(\_\_m64 m, int count)

mの4つの16ビット値を、countで指定した値だけ右にシフトし、上位ビットを0で埋めます。パフォーマンス上の理由で、countは定数にしてください。

\_\_m64 \_\_m\_psrldi(\_\_m64 m, \_\_m64 count)

mの2つの32ビット値を、countで指定した値だけ右にシフトし、上位ビットを0で埋めます。

\_\_m64 \_\_m\_psrldi(\_\_m64 m, int count)

mの2つの32ビット値を、countで指定した値だけ右にシフトし、上位ビットを0で埋めま

す。パフォーマンス上の理由で、countは定数にしてください。

```
__m64 __m_srlq(__m64 m, __m64 count)
```

mの64ビット値を、countで指定した値だけ右にシフトし、上位ビットを0で埋めます。

```
__m64 __m_srlqi (__m64 m, int count)
```

mの64ビット値を、countで指定した値だけ右にシフトし、上位ビットを0で埋めます。パフォーマンス上の理由で、countは定数にしてください。

## MMX® テクノロジーの論理演算組込み関数

MMX® テクノロジーの組込み関数のプロトタイプは、ヘッダ・ファイルmmintrin.h内にあります。

組込み関数名	別名	操作	対応する命令
<code>__m_pand</code>	<code>_mm_and_si64</code>	ビット単位のAND(論理積)	PAND
<code>__m_pandn</code>	<code>_mm_andnot_si64</code>	NOT(否定)	PANDN
<code>__m_por</code>	<code>_mm_or_si64</code>	ビット単位のOR(論理和)	POR
<code>__m_pxor</code>	<code>_mm_xor_si64</code>	ビット単位のXOR(排他的論理和)	PXOR

```
__m64 __m_pand(__m64 m1, __m64 m2)
```

m1 の64ビット値と m2 の64ビット値について、ビット単位の AND (排他的論理和)演算を実行します。

```
__m64 __m_pandn(__m64 m1, __m64 m2)
```

m1 の64ビット値のNOT (否定)演算を実行し、その結果とm2 の64ビット値について、ビット単位の AND (論理積)演算を実行します。

```
__m64 __m_por(__m64 m1, __m64 m2)
```

m1 の64ビット値と m2 の64ビット値について、ビット単位の OR (排他的論理和)演算を実行します。

```
__m64 __m_pxor(__m64 m1, __m64 m2)
```

m1 の64ビット値と m2 の64ビット値について、ビット単位の XOR (排他的論理和)演算を実行します。

## MMX® テクノロジーの比較組込み関数

MMX® テクノロジーの組込み関数のプロトタイプは、ヘッダ・ファイルmmintrin.h内にあります。

組込み関数名	別名	比較条件	要素の数	要素のサイズ(ビット)	対応する命令
<code>__m_pcmpe</code>	<code>_mm_cmpe</code>	等しい	8	8	PCMPEQB

qb	q_pi8				
_m_pcmpeqw	_mm_cmpeqw_pi16	等しい	4	16	PCMPEQW
_m_pcmpeqd	_mm_cmpeqd_pi32	等しい	2	32	PCMPEQD
_m_pcmpgtbtb	_mm_cmpgtbtb_pi8	より大きい	8	8	PCMPGTB
_m_pcmpgtwtw	_mm_cmpgtwtw_pi16	より大きい	4	16	PCMPGTW
_m_pcmpgtdtd	_mm_cmpgtdtd_pi32	より大きい	2	32	PCMPGTD

\_\_m64 \_m\_pcmpeqb(\_\_m64 m1, \_\_m64 m2)

m1の各8ビット値が、それに対応するm2の8ビット値に等しい場合は、それに対応する結果の8ビット値をすべて1に設定します。それ以外の場合は、すべて0に設定します。

\_\_m64 \_m\_pcmpeqw(\_\_m64 m1, \_\_m64 m2)

m1の各16ビット値が、それに対応するm2の16ビット値に等しい場合は、それに対応する結果の16ビット値をすべて1に設定します。それ以外の場合は、すべて0に設定します。

\_\_m64 \_m\_pcmpeqd(\_\_m64 m1, \_\_m64 m2)

m1の各32ビット値が、それに対応するm2の32ビット値に等しい場合は、それに対応する結果の32ビット値をすべて1に設定します。それ以外の場合は、すべて0に設定します。

\_\_m64 \_m\_pcmpgtbtb(\_\_m64 m1, \_\_m64 m2)

m1の各8ビット値が、それに対応するm2の8ビット値より大きい場合は、それに対応する結果の8ビット値をすべて1に設定します。それ以外の場合は、すべて0に設定します。

\_\_m64 \_m\_pcmpgtwtw(\_\_m64 m1, \_\_m64 m2)

m1の各16ビット値が、それに対応するm2の16ビット値より大きい場合は、それに対応する結果の16ビット値をすべて1に設定します。それ以外の場合は、すべて0に設定します。

\_\_m64 \_m\_pcmpgtdtd(\_\_m64 m1, \_\_m64 m2)

m1の各32ビット値が、それに対応するm2の32ビット値より大きい場合は、それに対応する結果の32ビット値をすべて1に設定します。それ以外の場合は、すべて0に設定します。

## MMX® テクノロジの設定組込み関数

MMX® テクノロジの組込み関数のプロトタイプは、ヘッダ・ファイルmmintrin.h内にあります。

組込み関数名	操作	要素の数	要素のサイズ(ビット)	符号	逆順
_mm_setz	0に設定	1	64	符号なし	符号なし

ero si64					
_mm_set_pi32	整数値の設定	2	32	符号なし	符号なし
_mm_set_pi16	整数値の設定	4	16	符号なし	符号なし
_mm_set_pi8	整数値の設定	8	8	符号なし	符号なし
_mm_set1_pi32	整数値の設定	2	32	可	符号なし
_mm_set1_pi16	整数値の設定	4	16	可	符号なし
_mm_set1_pi8	整数値の設定	8	8	可	符号なし
_mm_setr_pi32	整数値の設定	2	32	符号なし	可
_mm_setr_pi16	整数値の設定	4	16	符号なし	可
_mm_setr_pi8	整数値の設定	8	8	符号なし	可



次の説明では、MMX テクノロジ・レジスタのビット0が最下位ビット、ビット63が最上位ビットになります。

```
__m64 _mm_setzero_si64()
```

PXOR

64ビット値を0に設定します。

```
r := 0x0
```

```
__m64 _mm_set_pi32(int i1, int i0)(composite)
```

2個の符号付き32ビット整数値を設定します。

```
r0 := i0
```

```
r1 := i1
```

```
__m64 _mm_set_pi16(short s3, short s2, short s1, short s0)
```

(複合) 4個の符号付き16ビット整数値を設定します。

```
r0 := w0
```

```
r1 := w1
```

```
r2 := w2
```

```
r3 := w3
```

```
__m64 _mm_set_pi8(char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)
```

(複合) 8個の符号付き8ビット整数値を設定します。

```
r0 := b0
```

```
r1 := b1
```

```

    ...
    r7 := b7
__m64 _mm_setl_pi32(int i)
    (composite) Sets the 2 signed 32-bit integer values to i.
    r0 := i
    r1 := i
__m64 _mm_setl_pi16(short s)
    (composite) Sets the 4 signed 16-bit integer values to w.
    r0 := w
    r1 := w
    r2 := w
    r3 := w
__m64 _mm_setl_pi8(char b)
    (composite) Sets the 8 signed 8-bit integer values to b.
    r0 := b
    r1 := b
    ...
    r7 := b
__m64 _mm_setr_pi32(int i1, int i0)
    (複合) 2個の符号付き32ビット整数値を逆順で設定します。
    r0 := i0
    r1 := i1
__m64 _mm_setr_pi16(short s3, short s2, short s1, short s0)
    (複合) 4個の符号付き16ビット整数値を逆順で設定します。
    r0 := w0
    r1 := w1
    r2 := w2
    r3 := w3
__m64 _mm_setr_pi8(char b7, char b6, char b5, char b4, char
b3, char b2, char b1, char b0)
    (複合) 8個の符号付き8ビット整数値を逆順で設定します。
    r0 := b0
    r1 := b1
    ...
    r7 := b7

```

## Itanium® アーキテクチャ上での MMX® テクノロジー組込み関数の使用

MMX® テクノロジーの組込み関数を使用して、Itanium® ベースのシステム上で MMX テクノロジー命令セットを利用できます。IA-32 アーキテクチャのソースコードとの互換性を保つために、これらの組込み関数の名前と機能は、IA-32 ベースの MMX テクノロジー組込み関数セットと同等になっています。

一部の組込み関数には、2つ以上の名前があります。1つの組込み関数に2つの名前がある

場合は、いずれの名前も同じ命令を生成しますが、最初の名前の方が新しい命名規則に適合しているため、最初の名前を使用することをお勧めします。

MMX テクノロジの組込み関数のプロトタイプは、ヘッダ・ファイル `mmintrin.h` 内にあります。

## データ型

MMX テクノロジの組込み関数は、C データ型 `__m64` を使用します。このデータ型は、8つの8ビット値、4つの16ビット値、2つの32ビット値、または1つの64ビット値を保持できます。

`__m64` データ型は、基本的なANSI C データ型ではありません。このため、次のような使用上の制限があります。

- この新しいデータ型は、代入文の左辺で、戻り値またはパラメータとして使用してください。他の算術式("+", "-", など)にこのデータ型を使用することはできません。
- この新しいデータ型は、バイト要素/構造にアクセスするための共用体などの集合体のオブジェクトとして使用してください。`__m64` オブジェクトのアドレスを指定できます。
- 新しいデータ型は、本書で説明する組込み関数でのみ使用してください。

ハードウェア命令の詳細については、『*Intel Architecture MMX Technology Programmer's Reference Manual*』を参照してください。データ型の詳細については、『*インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻*』を参照してください。



# ストリーミングSIMD拡張命令

## ストリーミングSIMD拡張命令のサポート

この節では、インテル® C++ コンパイラ内でストリーミングSIMD拡張命令をサポートする、C++ 言語レベルの機能について説明します。ここでは、ストリーミングSIMD拡張命令の組み込み関数の次のような機能について説明します。

- 浮動小数点演算組み込み関数
- 算術演算組み込み関数
- 論理演算組み込み関数
- 比較組み込み関数
- 変換組み込み関数
- ロード操作
- 設定操作
- ストア操作
- キャッシュ制御
- 整数演算組み込み関数
- メモリ操作と初期化操作の組み込み関数
- その他の組み込み関数
- Itanium® アーキテクチャ上でのストリーミング SIMD 拡張命令の使用

ストリーミングSIMD拡張命令の組み込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

## ストリーミングSIMD拡張命令を使用する浮動小数点演算組み込み関数

ストリーミング SIMD 拡張命令の組み込み関数を使用してプログラムを作成する際は、ストリーミング SIMD 拡張命令によって提供されるハードウェア機能をよく理解している必要があります。特に、次の4つの重要な点に注意してください。

- `_mm_loadr_ps` や `_mm_cmpgt_ss` などの一部の組み込み関数は、特定の命令セットによって直接にはサポートしていません。これらの組み込み関数は、プログラミング上の便宜のために用意したもので、実際には2つ以上のマシン言語命令で構成されています。
- `__m128` オブジェクトとしてロードまたはストアされる浮動小数点データは、通常は16バイトにアライメントが合っていなければなりません。
- 一部の組み込み関数は、命令の性質上、引数を即値で、すなわち定数整数(リテラル)で指定する必要があります。
- 2つのNaN (Not a Number) 引数を操作する算術演算の結果は未定義です。したがって、NaN引数を使用するFP (浮動小数点) 演算は、対応するアセンブリ言語命令の予想される動作とは一致しません。

## ストリーミングSIMD拡張命令の算術演算

ストリーミングSIMD拡張命令の組み込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組み込み関数	命令	操作	R0	R1	R2	R3
<code>_mm_add_ss</code>	ADDSS	加算	a0 [op] b0	a1	a2	a3
<code>_mm_add_ps</code>	ADDPS	加算	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
<code>_mm_sub_ss</code>	SUBSS	減算	a0 [op] b0	a1	a2	a3
<code>_mm_sub_ps</code>	SUBPS	減算	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
<code>_mm_mul_ss</code>	MULSS	乗算	a0 [op] b0	a1	a2	a3
<code>_mm_mul_ps</code>	MULPS	乗算	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
<code>_mm_div_ss</code>	DIVSS	除算	a0 [op] b0	a1	a2	a3
<code>_mm_div_ps</code>	DIVPS	除算	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
<code>_mm_sqrt_ss</code>	SQRTSS	平方根	[op] a0	a1	a2	a3
<code>_mm_sqrt_ps</code>	SQRTPS	平方根	[op] a0	[op] b1	[op] b2	[op] b3
<code>_mm_rcp_ss</code>	RCPSS	逆数	[op] a0	a1	a2	a3
<code>_mm_rcp_ps</code>	RCPPS	逆数	[op] a0	[op] b1	[op] b2	[op] b3
<code>_mm_rsqrt_ss</code>	RSQRTSS	平方根の逆数	[op] a0	a1	a2	a3
<code>_mm_rsqrt_ps</code>	RSQRTPS	平方根の逆数	[op] a0	[op] b1	[op] b2	[op] b3
<code>_mm_min_ss</code>	MINSS	最小値の計算	[op]( a0,b0 )	a1	a2	a3
<code>_mm_min_ps</code>	MINPS	最小値の計算	[op]( a0,b0 )	[op] (a1, b1)	[op] (a2, b2)	[op] (a3, b3)
<code>_mm_max_ss</code>	MAXSS	最大値の計算	[op]( a0,b0 )	a1	a2	a3
<code>_mm_max_ps</code>	MAXPS	最大値の計算	[op]( a0,b0 )	[op] (a1, b1)	[op] (a2, b2)	[op] (a3, b3)

```

__m128 _mm_add_ss(__m128 a, __m128 b)
    aとbの最下位の単精度浮動小数点値を加算します。上位3個の単精度浮動小数点値
    は、aからそのまま渡されます。
    r0 := a0 + b0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 _mm_add_ps(__m128 a, __m128 b)
    aとbの4個の単精度浮動小数点値を加算します。
    r0 := a0 + b0
    r1 := a1 + b1
    r2 := a2 + b2
    r3 := a3 + b3
__m128 _mm_sub_ss(__m128 a, __m128 b)
    aの最下位の単精度浮動小数点値から、bの最下位の単精度浮動小数点値を引きます。
    上位3個の単精度浮動小数点値は、aからそのまま渡されます。
    r0 := a0 - b0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 _mm_sub_ps(__m128 a, __m128 b)
    aの4個の単精度浮動小数点値から、bの4個の単精度浮動小数点値を引きます。
    r0 := a0 - b0
    r1 := a1 - b1
    r2 := a2 - b2
    r3 := a3 - b3
__m128 _mm_mul_ss(__m128 a, __m128 b)
    aとbの最下位の単精度浮動小数点値を乗算します。上位3個の単精度浮動小数点値
    は、aからそのまま渡されます。
    r0 := a0 * b0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 _mm_mul_ps(__m128 a, __m128 b)
    aとbの4個の単精度浮動小数点値を乗算します。
    r0 := a0 * b0
    r1 := a1 * b1
    r2 := a2 * b2
    r3 := a3 * b3
__m128 _mm_div_ss(__m128 a, __m128 b)
    aの最下位の単精度浮動小数点値を、bの最下位の単精度浮動小数点値で割ります。
    上位3個の単精度浮動小数点値は、aからそのまま渡されます。
    r0 := a0 / b0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 _mm_div_ps(__m128 a, __m128 b)
    aの4個の単精度浮動小数点値を、bの4個の単精度浮動小数点値で割ります。
    r0 := a0 / b0

```

```

    r1 := a1 / b1
    r2 := a2 / b2
    r3 := a3 + b3
__m128 __mm_sqrt_ss(__m128 a)
    aの最下位の単精度浮動小数点値の平方根を計算します。上位3個の単精度浮動小
    数点値はそのまま渡されます。
    r0 := sqrt(a0)
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_sqrt_ps(__m128 a)
    aの4個の単精度浮動小数点値の平方根を計算します。
    r0 := sqrt(a0)
    r1 := sqrt(a1)
    r2 := sqrt(a2)
    r3 := sqrt(a3)
__m128 __mm_rcp_ss(__m128 a)
    aの最下位の単精度浮動小数点値の逆数の近似値を計算します。上位3個の単精度
    浮動小数点値はそのまま渡されます。
    r0 := recip(a0)
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_rcp_ps(__m128 a)
    aの4個の単精度浮動小数点値の逆数の近似値を計算します。
    r0 := recip(a0)
    r1 := recip(a1)
    r2 := recip(a2)
    r3 := recip(a3)
__m128 __mm_rsqrt_ss(__m128 a)
    aの最下位の単精度浮動小数点値の平方根の逆数の近似値を計算します。上位3個
    の単精度浮動小数点値はそのまま渡されます。
    r0 := recip(sqrt(a0))
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_rsqrt_ps(__m128 a)
    aの4個の単精度浮動小数点値の平方根の逆数の近似値を計算します。
    r0 := recip(sqrt(a0))
    r1 := recip(sqrt(a1))
    r2 := recip(sqrt(a2))
    r3 := recip(sqrt(a3))
__m128 __mm_min_ss(__m128 a, __m128 b)
    aとbの最下位の単精度浮動小数点値について、小さい方の値を計算をします。上位3
    個の単精度浮動小数点値は、aからそのまま渡されます。
    r0 := min(a0, b0)
    r1 := a1 ; r2 := a2 ; r3 := a3

```

```

__m128 _mm_min_ps(__m128 a, __m128 b)
    aとbの4個の単精度浮動小数点値について、それぞれ小さい方の値を計算します。
    r0 := min(a0, b0)
    r1 := min(a1, b1)
    r2 := min(a2, b2)
    r3 := min(a3, b3)
__m128 _mm_max_ss(__m128 a, __m128 b)
    aとbの最下位の単精度浮動小数点値について、大きい方の値を計算します。上位3個
    の単精度浮動小数点値は、aからそのまま渡されます。
    r0 := max(a0, b0)
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 _mm_max_ps(__m128 a, __m128 b)
    aとbの4個の単精度浮動小数点値について、それぞれ小さい方の値を計算します。
    r0 := max(a0, b0)
    r1 := max(a1, b1)
    r2 := max(a2, b2)
    r3 := max(a3, b3)

```

## ストリーミングSIMD拡張命令の論理演算

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組込み関数名	操作	対応する命令
<code>_mm_and_ps</code>	ビット単位のAND(論理積)	ANDPS
<code>_mm_andnot_ps</code>	NOT(否定)	ANDNPS
<code>_mm_or_ps</code>	ビット単位のOR(論理和)	ORPS
<code>_mm_xor_ps</code>	ビット単位のXOR(排他的論理和)	XORPS

```

__m128 _mm_and_ps(__m128 a, __m128 b)
    aの4個の単精度浮動小数点値とbの4個の単精度浮動小数点値について、ビット単位
    のAND (論理積)を計算します。
    r0 := a0 & b0
    r1 := a1 & b1
    r2 := a2 & b2
    r3 := a3 & b3
__m128 _mm_andnot_ps(__m128 a, __m128 b)
    aの4個の単精度浮動小数点値のNOT (否定)演算を実行し、その結果とbの4個の単精
    度浮動小数点値について、ビット単位のAND (論理積)を計算します。
    r0 := ~a0 & b0
    r1 := ~a1 & b1
    r2 := ~a2 & b2
    r3 := ~a3 & b3

```

```

r2 := ~a2 & b2
非対応
__m128 __mm_or_ps(__m128 a, __m128 b)
aの4個の単精度浮動小数点値とbの4個の単精度浮動小数点値について、ビット単位
のOR (論理和)を計算します。
r0 := a0 | b0
r1 := a1 | b1
r2 := a2 | b2
r3 := a3 | b3
__m128 __mm_xor_ps(__m128 a, __m128 b)
aの4個の単精度浮動小数点値とbの4個の単精度浮動小数点値について、ビット単位
のXOR (排他的論理和)を計算します。
r0 := a0 ^ b0
r1 := a1 ^ b1
r2 := a2 ^ b2
r3 := a3 ^ b3

```

## ストリーミング SIMD 拡張命令の比較操作

比較組込み関数は、aとbの比較を実行します。パックド形式では、aとbの4個の単精度浮動小数点値を比較して、128ビット・マスクを返します。スカラ形式では、aとbの最下位の単精度浮動小数点値を比較して、32ビット・マスクを返します。上位3個の単精度浮動小数点値は、aからそのまま渡されます。マスクは、各要素について、比較の結果が真の場合は0xffffffffに設定し、偽の場合は0x0に設定されます。

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組込み関数名	比較条件	対応する命令
<code>__mm_cmpeq_ss</code>	等しい	CMPEQSS
<code>__mm_cmpeq_ps</code>	等しい	CMPEQPS
<code>__mm_cmplt_ss</code>	より大きい	CMPLTSS
<code>__mm_cmplt_ps</code>	より大きい	CMPLTPS
<code>__mm_cmple_ss</code>	より小さいか等しい	CMPLESS
<code>__mm_cmple_ps</code>	より小さいか等しい	CMPLEPS
<code>__mm_cmpgt_ss</code>	より大きい	CMPLTSS
<code>__mm_cmpgt_ps</code>	より大きい	CMPLTPS
<code>__mm_cmpge_ss</code>	より大きいか等しい	CMPLESS
<code>__mm_cmpge_ps</code>	より大きいか等しい	CMPLEPS
<code>__mm_cmpneq_ss</code>	等しくない	CMPNEQSS
<code>__mm_cmpneq_ps</code>	等しくない	CMPNEQPS
<code>__mm_cmpnlt_ss</code>	より小さくない	CMPNLTSS
<code>__mm_cmpnlt_ps</code>	より小さくない	CMPNLTPS

<code>_mm_cmpnle_ss</code>	より小さくなく等しくない	CMPNLESS
<code>_mm_cmpnle_ps</code>	より小さくなく等しくない	CMPNLEPS
<code>_mm_cmpngt_ss</code>	より大きくない	CMPNLTSS
<code>_mm_cmpngt_ps</code>	より大きくない	CMPNLTPS
<code>_mm_cmpnge_ss</code>	より大きくなく等しくない	CMPNLESS
<code>_mm_cmpnge_ps</code>	より大きくなく等しくない	CMPNLEPS
<code>_mm_cmpord_ss</code>	順序化可能	CMPORDSS
<code>_mm_cmpord_ps</code>	順序化可能	CMPORDPS
<code>_mm_cmpunord_ss</code>	順序化不可能	CMPUNORDSS
<code>_mm_cmpunord_ps</code>	順序化不可能	CMPUNORDPS
<code>_mm_comieq_ss</code>	等しい	COMISS
<code>_mm_comilt_ps</code>	より大きい	COMISS
<code>_mm_comile_ss</code>	より小さいか等しい	COMISS
<code>_mm_comigt_ss</code>	より大きい	COMISS
<code>_mm_comige_ss</code>	より大きい等しい	COMISS
<code>_mm_comineq_ss</code>	等しくない	COMISS
<code>_mm_ucomieq_ss</code>	等しい	UCOMISS
<code>_mm_ucomilt_ss</code>	より大きい	UCOMISS
<code>_mm_ucomile_ss</code>	より小さいか等しい	UCOMISS
<code>_mm_ucomigt_ss</code>	より大きい	UCOMISS
<code>_mm_ucomige_ss</code>	より大きい等しい	UCOMISS
<code>_mm_ucomineq_ss</code>	等しくない	UCOMISS

`__m128 _mm_cmpeq_ss(__m128 a, __m128 b)`

等しいかどうか比較します。

`r0 := (a0 == b0) ? 0xffffffff : 0x0`

`r1 := a1 ; r2 := a2 ; r3 := a3`

`__m128 _mm_cmpeq_ps(__m128 a, __m128 b)`

等しいかどうか比較します。

`r0 := (a0 == b0) ? 0xffffffff : 0x0`

`r1 := (a1 == b1) ? 0xffffffff : 0x0`

`r2 := (a2 == b2) ? 0xffffffff : 0x0`

`r3 := (a3 == b3) ? 0xffffffff : 0x0`

`__m128 _mm_cmlt_ss(__m128 a, __m128 b)`

より小さいかどうか比較します。

`r0 := (a0 < b0) ? 0xffffffff : 0x0`

`r1 := a1 ; r2 := a2 ; r3 := a3`

`__m128 _mm_cmlt_ps(__m128 a, __m128 b)`

より小さいかどうか比較します。

`r0 := (a0 < b0) ? 0xffffffff : 0x0`

```

    r1 := (a1 < b1) ? 0xffffffff : 0x0
    r2 := (a2 < b2) ? 0xffffffff : 0x0
    r3 := (a3 < b3) ? 0xffffffff : 0x0
__m128 __mm_cmple_ss(__m128 a, __m128 b)
    より小さいか等しいかどうか比較します。
    r0 := (a0 <= b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmple_ps(__m128 a, __m128 b)
    より小さいか等しいかどうか比較します。
    r0 := (a0 <= b0) ? 0xffffffff : 0x0
    r1 := (a1 <= b1) ? 0xffffffff : 0x0
    r2 := (a2 <= b2) ? 0xffffffff : 0x0
    r3 := (a3 <= b3) ? 0xffffffff : 0x0
__m128 __mm_cmpgt_ss(__m128 a, __m128 b)
    より大きいかどうか比較します。
    r0 := (a0 > b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpgt_ps(__m128 a, __m128 b)
    より大きいかどうか比較します。
    r0 := (a0 > b0) ? 0xffffffff : 0x0
    r1 := (a1 > b1) ? 0xffffffff : 0x0
    r2 := (a2 > b2) ? 0xffffffff : 0x0
    r3 := (a3 > b3) ? 0xffffffff : 0x0
__m128 __mm_cmpge_ss(__m128 a, __m128 b)
    より大きいか等しいかどうか比較します。
    r0 := (a0 >= b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpge_ps(__m128 a, __m128 b)
    より大きいか等しいかどうか比較します。
    r0 := (a0 >= b0) ? 0xffffffff : 0x0
    r1 := (a1 >= b1) ? 0xffffffff : 0x0
    r2 := (a2 >= b2) ? 0xffffffff : 0x0
    r3 := (a3 >= b3) ? 0xffffffff : 0x0
__m128 __mm_cmpneq_ss(__m128 a, __m128 b)
    等しくないかどうか比較します。
    r0 := (a0 != b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpneq_ps(__m128 a, __m128 b)
    等しくないかどうか比較します。

```



```

    r0 := (a0 != b0) ? 0xffffffff : 0x0
    r1 := (a1 != b1) ? 0xffffffff : 0x0
    r2 := (a2 != b2) ? 0xffffffff : 0x0
    r3 := (a3 != b3) ? 0xffffffff : 0x0
__m128 __mm_cmpnlt_ss(__m128 a, __m128 b)
    より小さくないかどうか比較します。
    r0 := !(a0 < b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpnlt_ps(__m128 a, __m128 b)
    より小さくないかどうか比較します。
    r0 := !(a0 < b0) ? 0xffffffff : 0x0
    r1 := !(a1 < b1) ? 0xffffffff : 0x0
    r2 := !(a2 < b2) ? 0xffffffff : 0x0
    r3 := !(a3 < b3) ? 0xffffffff : 0x0
__m128 __mm_cmpnle_ss(__m128 a, __m128 b)
    より小さくなく等しくないかどうか比較します。
    r0 := !(a0 <= b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpnle_ps(__m128 a, __m128 b)
    より小さくなく等しくないかどうか比較します。
    r0 := !(a0 <= b0) ? 0xffffffff : 0x0
    r1 := !(a1 <= b1) ? 0xffffffff : 0x0
    r2 := !(a2 <= b2) ? 0xffffffff : 0x0
    r3 := !(a3 <= b3) ? 0xffffffff : 0x0
__m128 __mm_cmpngt_ss(__m128 a, __m128 b)
    より大きくないかどうか比較します。
    r0 := !(a0 > b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpngt_ps(__m128 a, __m128 b)
    より大きくないかどうか比較します。
    r0 := !(a0 > b0) ? 0xffffffff : 0x0
    r1 := !(a1 > b1) ? 0xffffffff : 0x0
    r2 := !(a2 > b2) ? 0xffffffff : 0x0
    r3 := !(a3 > b3) ? 0xffffffff : 0x0
__m128 __mm_cmpnge_ss(__m128 a, __m128 b)
    より大きくなく等しくないかどうか比較します。
    r0 := !(a0 >= b0) ? 0xffffffff : 0x0
    r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpnge_ps(__m128 a, __m128 b)

```

より大きくなく等しくないかどうか比較します。

```
r0 := !(a0 >= b0) ? 0xffffffff : 0x0
r1 := !(a1 >= b1) ? 0xffffffff : 0x0
r2 := !(a2 >= b2) ? 0xffffffff : 0x0
r3 := !(a3 >= b3) ? 0xffffffff : 0x0
__m128 __mm_cmpord_ss(__m128 a, __m128 b)
```

順序化可能かどうか判定します。

```
r0 := (a0 ord? b0) ? 0xffffffff : 0x0
r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpord_ps(__m128 a, __m128 b)
```

順序化可能かどうか判定します。

```
r0 := (a0 ord? b0) ? 0xffffffff : 0x0
r1 := (a1 ord? b1) ? 0xffffffff : 0x0
r2 := (a2 ord? b2) ? 0xffffffff : 0x0
r3 := (a3 ord? b3) ? 0xffffffff : 0x0
__m128 __mm_cmpunord_ss(__m128 a, __m128 b)
```

順序化不可能かどうか判定します。

```
r0 := (a0 unord? b0) ? 0xffffffff : 0x0
r1 := a1 ; r2 := a2 ; r3 := a3
__m128 __mm_cmpunord_ps(__m128 a, __m128 b)
```

順序化不可能かどうか判定します。

```
r0 := (a0 unord? b0) ? 0xffffffff : 0x0
r1 := (a1 unord? b1) ? 0xffffffff : 0x0
r2 := (a2 unord? b2) ? 0xffffffff : 0x0
r3 := (a3 unord? b3) ? 0xffffffff : 0x0
```

```
int __mm_comieq_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aとbが等しいかどうか比較します。aとbが等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 == b0) ? 0x1 : 0x0
```

```
int __mm_comilt_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより小さいかどうか比較します。aがbより小さい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 < b0) ? 0x1 : 0x0
```

```
int __mm_comile_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより小さいか等しいかどうか比較します。aがbより小さいか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 <= b0) ? 0x1 : 0x0
```

```
int __mm_comigt_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより大きいかどうか比較します。

aがbより大きい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 > b0) ? 0x1 : 0x0
```

```
int __mm_comige_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより大きいとか等しいかどうか比較します。aがbより大きいとか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 >= b0) ? 0x1 : 0x0
```

```
int __mm_comineq_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aとbが等しくないかどうか比較します。aとbが等しくない場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 != b0) ? 0x1 : 0x0
```

```
int __mm_ucomieq_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aとbが等しいかどうか比較します。aとbが等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 == b0) ? 0x1 : 0x0
```

```
int __mm_ucomilt_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより小さいかどうか比較します。aがbより小さい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 < b0) ? 0x1 : 0x0
```

```
int __mm_ucomile_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより小さいとか等しいかどうか比較します。aがbより小さいとか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 <= b0) ? 0x1 : 0x0
```

```
int __mm_ucomigt_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより大きいかどうか比較します。aがbより大きいとか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 > b0) ? 0x1 : 0x0
```

```
int __mm_ucomige_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aがbより大きいとか等しいかどうか比較します。aがbより大きいとか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 >= b0) ? 0x1 : 0x0
```

```
int __mm_ucomineq_ss(__m128 a, __m128 b)
```

aとbの最下位の単精度浮動小数点値について、aとbが等しくないかどうか比較します。aとbが等しくない場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 != b0) ? 0x1 : 0x0
```

## ストリーミングSIMD拡張命令の変換操作

次の表に、変換操作のリストを示します。表の後に、最新のニーモニック命名規則に従って、各組込み関数の説明を示します。これらの組込み関数を以前に使用していた場合に備えて、別名を用意しています。

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組込み関数名	別名	対応する命令
<code>mm_cvt_ss2si</code>	<code>mm_cvtss_si32</code>	CVTSS2SI
<code>int mm_cvt_ps2pi</code>	<code>mm_cvtps_pi32</code>	CVTPS2PI
<code>mm_cvtt_ss2si</code>	<code>mm_cvttss_si32</code>	CVTTSS2SI
<code>mm_cvtt_ps2pi</code>	<code>mm_cvttps_pi32</code>	CVTTPS2PI
<code>mm_cvt_si2ss</code>	<code>mm_cvtsi32_ss</code>	CVTSI2SS
<code>mm_cvt_pi2ps</code>	<code>mm_cvtpi32_ps</code>	CVTTPS2PI
<code>_mm_cvtpi16_ps</code>		複合
<code>_mm_cvtpu16_ps</code>		複合
<code>_mm_cvtpi8_ps</code>		複合
<code>_mm_cvtpu8_ps</code>		複合
<code>_mm_cvtpi32x2_ps</code>		複合
<code>_mm_cvtps_pi16</code>		複合
<code>_mm_cvtps_pi8</code>		複合

```
int _mm_cvt_ss2si(__m128 a)
```

現在の丸めモードに従って、aの最下位の単精度浮動小数点値を32ビット整数に変換します。

```
    r := (int) a0
```

```
__m64 _mm_cvt_ps2pi(__m128 a)
```

現在の丸めモードに従って、aの下位2個の単精度浮動小数点値を2個の32ビット整数に変換し、パックド形式で返します。

```
    r0 := (int) a0
```

```
    r1 := (int) a1
```

```
int _mm_cvtt_ss2si(__m128 a)
```

切り捨てを使用して、aの最下位の単精度浮動小数点値を32ビット整数に変換します。

```
    r := (int) a0
```

```
__m64 _mm_cvtt_ps2pi(__m128 a)
```

切り捨てを使用して、aの下位2個の単精度浮動小数点値を2個の32ビット整数に変換し、パックド形式で返します。

```
    r0 := (int) a0
```

```
    r1 := (int) a1
```

```
__m128 _mm_cvt_si2ss(__m128, int)
```

32ビット整数bを単精度浮動小数点値に変換します。上位3個の単精度浮動小数点値

は、aからそのまま渡されます。

```
r0 := (float)b  
r1 := a1 ; r2 := a2 ; r3 := a3
```

```
__m128 __mm_cvt_pi2ps(__m128, __m64)
```

bのパックド形式の2個の32ビット整数値を、2個の単精度浮動小数点値に変換します。  
上位2個の単精度浮動小数点値は、aからそのまま渡されます。

```
r0 := (float)b0  
r1 := (float)b1  
r2 := a2  
r3 := a3
```

```
__inline __m128 __mm_cvtpi16_ps(__m64 a)
```

aの4個の符号付き16ビット整数値を、4個の単精度浮動小数点値に変換します。

```
r0 := (float) a0  
r1 := (float) a1  
r2 := (float) a2  
r3 := (float) a3
```

```
__inline __m128 __mm_cvtpu16_ps(__m64 a)
```

aの4個の符号なし16ビット整数値を、4個の単精度浮動小数点値に変換します。

```
r0 := (float) a0  
r1 := (float) a1  
r2 := (float) a2  
r3 := (float) a3
```

```
__inline __m128 __mm_cvtpi8_ps(__m64 a)
```

aの下位4個の符号付き8ビット整数値を、4個の単精度浮動小数点値に変換します。

```
r0 := (float) a0  
r1 := (float) a1  
r2 := (float) a2  
r3 := (float) a3
```

```
__inline __m128 __mm_cvtpu8_ps(__m64 a)
```

aの下位4個の符号なし8ビット整数値を、4個の単精度浮動小数点値に変換します。

```
r0 := (float) a0  
r1 := (float) a1  
r2 := (float) a2  
r3 := (float) a3
```

```
__inline __m128 __mm_cvtpi32x2_ps(__m64 a, __m64 b)
```

aの2個の符号付き32ビット整数値とbの2個の符号付き32ビット整数値を、4個の単精度浮動小数点値に変換します。

```
r0 := (float) a0  
r1 := (float) a1  
r2 := (float)b0  
r3 := (float)b1
```

```
__inline __m64 __mm_cvtps_pi16(__m128 a)
```

aの4個の単精度浮動小数点値を、4個の符号付き16ビット整数値に変換します。

```
r0 := (short)a0
r1 := (short)a1
r2 := (short)a2
r3 := (short)a3
```

\_\_\_ inline \_\_m64 \_mm\_cvtps\_pi8(\_\_m128 a)

aの4個の単精度浮動小数点値を、結果の下位4個の符号付き8ビット整数値に変換します。

```
r0 := (char)a0
r1 := (char)a1
r2 := (char)a2
r3 := (char)a3
```

## ストリーミングSIMD拡張命令のロード操作

「メモリ操作と初期化操作のまとめ」にある一覧表を参照してください。

ストリーミングSIMD拡張命令の組み込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

\_\_\_ m128 \_mm\_load\_ss(float \* p)

単精度浮動小数点値を最下位ワードにロードし、上位3ワードをクリアします。

```
r0 := *p
r1 := 0.0 ; r2 := 0.0 ; r3 := 0.0
```

\_\_\_ m128 \_mm\_load\_ps1(float \* p)

1個の単精度浮動小数点値をロードして、その値を4ワードすべてにコピーします。

```
r0 := *p
r1 := *p
r2 := *p
r3 := *p
```

\_\_\_ m128 \_mm\_load\_ps(float \* p)

4つの単精度浮動小数点値をロードします。アドレスは16バイトにアライメントが合っていないければなりません。

```
r0 := p[0]
r1 := p[1]
r2 := p[2]
r3 := p[3]
```

\_\_\_ m128 \_mm\_loadu\_ps(float \* p)

4つの単精度浮動小数点値をロードします。アドレスは16バイトにアライメントが合っていないてもかまいません。

```
r0 := p[0]
```

```

    r1 := p[1]
    r2 := p[2]
    r3 := p[3]
__m128 _mm_loadr_ps(float * p)
    4つの単精度浮動小数点値を逆順でロードします。アドレスは16バイトにアライメントが
    合っていないかもしれません。
    r0 := p[3]
    r1 := p[2]
    r2 := p[1]
    r3 := p[0]

```

## ストリーミングSIMD拡張命令の設定操作

「メモリ操作と初期化操作」にある一覧表を参照してください。

ストリーミングSIMD拡張命令の組み込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

```

__m128 _mm_set_ss(float w )
    単精度浮動小数点値の最下位ワードをwに設定し、上位3ワードをクリアします。
    r0 := w
    r1 := r2 := r3 := 0.0
__m128 _mm_set_ps1(float w )
    4個の単精度浮動小数点値をwに設定します。
    r0 := r1 := r2 := r3 := w
__m128 _mm_set_ps(float z, float y, float x, float w )
    4つの単精度浮動小数点値を、4つの入力値に設定します。
    r0 := w
    r1 := x
    r2 := y
    r3 := z
__m128 _mm_setr_ps(float z, float y, float x, float w )
    4つの単精度浮動小数点値を、逆順で4つの入力値に設定します。
    r0 := z
    r1 := y
    r2 := x
    r3 := w
__m128 _mm_setzero_ps(void)
    4つの単精度浮動小数点値をクリアします。
    r0 := r1 := r2 := r3 := 0.0

```

## ストリーミングSIMD拡張命令のストア操作

「メモリ操作と初期化操作のまとめ」にある一覧表を参照してください。

ストリーミングSIMD拡張命令の組み込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

```
void _mm_store_ss(float * p, __m128 a)
```

最下位の単精度浮動小数点値をストアします。

```
*p := a0
```

```
void _mm_store_ps1(float * p, __m128 a )
```

最下位の単精度浮動小数点値を、4ワードにストアします。

```
p[0] := a0
```

```
p[1] := a0
```

```
p[2] := a0
```

```
p[3] := a0
```

```
void _mm_store_ps(float *p, __m128 a )
```

4個の単精度浮動小数点値をストアします。アドレスは16バイトにアライメントが合っていないとなりません。

```
p[0] := a0
```

```
p[1] := a1
```

```
p[2] := a2
```

```
p[3] := a3
```

```
void _mm_storeu_ps(float *p, __m128 a)
```

4個の単精度浮動小数点値をストアします。アドレスは16バイトにアライメントが合っていない場合でもかまいません。

```
p[0] := a0
```

```
p[1] := a1
```

```
p[2] := a2
```

```
p[3] := a3
```

```
void _mm_storer_ps(float * p, __m128 a )
```

4個の単精度浮動小数点値を逆順でストアします。アドレスは16バイトにアライメントが合っていないとなりません。

```
p[0] := a3
```

```
p[1] := a2
```

```
p[2] := a1
```

```
p[3] := a0
```

```
__m128 _mm_move_ss(__m128 a, __m128 b)
```

最下位ワードを、bの単精度浮動小数点値に設定します。上位3個の単精度浮動小数点値は、aからそのまま渡されます。

```
r0 := b0
```

```
r1 := a1
```



```
r2 := a2
r3 := a3
```

## ストリーミングSIMD拡張命令によるキャッシュ制御

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

```
void _mm_pause(void)
```

プロセッサに固有の時間の間、次の命令の実行を遅らせます。この命令を実行しても、アーキテクチャ上の状態は変化しません。この組込み関数を使用すると、パフォーマンスが大きく向上します。この組込み関数については、以降で詳しく説明します。

### PAUSE組込み関数

PAUSE 組込み関数は、ダイナミック・エグゼキューション(特に、アウトオブオーダー実行)をサポートするプロセッサ上で、spin-wait ループに使用します。spin-wait ループ内でPAUSEを使用すると、ロックの解放を検出するコードの処理速度が向上します。動的スケジューリングに PAUSE 命令を使用すると、スピンループの終了時のペナルティが軽減されます。

### PAUSE命令を使用したループの例

```
spin_loop: pause
cmp eax, A
jne spin_loop
```

上の例では、メモリ・ロケーションAがレジスタeaxの値と一致するまで、プログラムはスピンのままです。次のコード・シーケンスは、test-and-test-and-set 操作を示しています。この例では、ロックの取得に失敗した場合にのみ、スピンが発生します。

```
get_lock: mov eax, 1
xchg eax, A ; Try to get lock
cmp eax, 0 ; Test if successful
jne spin_loop
クリティカル・セクション:
<critical_section code>
mov A, 0 ; Release lock
jmp continue
spin_loop: pause; Spin-loop hint
cmp 0, A ; Check lock availability
jne spin_loop
jmp get_lock
continue: <other code>
```

この例では、ロックの取得に成功すると予測して、最初の条件分岐は分岐せずに、そのままクリティカル・セクションの処理に移ります。すべての spin-wait ループに、PAUSE 命令を使用するのを強くお勧めします。PAUSE命令は既存のすべてのIA-32プロセッサで使用可能なため、プロセッサのタイプをテストする(CPUIDテスト)必要はありません。すべての従来のプロセ

ッサは、PAUSE を NOP として実行しますが、PAUSE をヒントとして使用するプロセッサでは、パフォーマンスが大きく向上する可能性があります。

## ストリーミングSIMD拡張命令を使用する整数演算組込み関数

次の表に、整数演算組込み関数のリストを示します。表の後に、最新のニーモニック命名規則に従って、各組込み関数の説明を示します。

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組込み関数名	別名	操作	対応する命令
<code>_m_pextrw</code>	<code>_mm_extract_pi16</code>	4ワードのうち1つを抽出する	PEXTRW
<code>_m_pinsrw</code>	<code>_mm_insert_pi16</code>	1ワードを挿入する	PINSRW
<code>_m_pmaxsw</code>	<code>_mm_max_pi16</code>	最大値を計算する	PMAXSW
<code>_m_pmaxub</code>	<code>_mm_max_pu8</code>	最大値を計算する (符号なし)	PMAXUB
<code>_m_pminsw</code>	<code>_mm_min_pi16</code>	最小値を計算する	PMINSW
<code>_m_pminub</code>	<code>_mm_min_pu8</code>	最小値を計算する (符号なし)	PMINUB
<code>_m_pmovmskb</code>	<code>_mm_movemask_pi8</code>	8ビットマスクを作成する	PMOVMASKB
<code>_m_pmulhuw</code>	<code>_mm_mulhi_pu16</code>	乗算(上位ビットを返す)	PMULHUW
<code>_m_pshufw</code>	<code>_mm_shuffle_pi16</code>	4ワードを組み合わせて返す	PSHUFW
<code>_m_maskmovq</code>	<code>_mm_maskmove_si64</code>	条件付きストア	MASKMOVQ
<code>_m_pavgb</code>	<code>_mm_avg_pu8</code>	丸め平均を計算する	PAVGB
<code>_m_pavgw</code>	<code>_mm_avg_pu16</code>	丸め平均を計算する	PAVGW
<code>_m_psadbw</code>	<code>_mm_sad_pu8</code>	差の絶対値の合計を計算する	PSADBW

ここで説明する組込み関数を使用する場合は、mmxレジスタのマルチメディア・ステートを空にする必要があります。詳細は、「EMMS命令: 必要な理由と使用の条件」の項目を参照してください。

```
int _m_pextrw(__m64 a, int n)
```

aの4ワードのうち1つを抽出します。セレクトaは即値でなければなりません。

```
    r := (n==0) ? a0 : ( (n==1) ? a1 : ( (n==2) ? a2 : a3 ) )
__m64 _m_pinsrw(__m64 a, int d, int n)
```

aの4ワードのうち1つに、ワードdを挿入します。セクタnは即値でなければなりません。

```
r0 := (n==0) ? d : a0;  
r1 := (n==1) ? d : a1;  
r2 := (n==2) ? d : a2;  
r3 := (n==3) ? d : a3;
```

```
__m64 __m_pmaxsw(__m64 a, __m64 b)
```

aとbの4ワードについて、要素ごとに最大値を計算します。

```
r0 := min(a0, b0)  
r1 := min(a1, b1)  
r2 := min(a2, b2)  
r3 := min(a3, b3)
```

```
__m64 __m_pmaxub(__m64 a, __m64 b)
```

aとbの8個の符号なしバイトについて、要素ごとに最大値を計算します。

```
r0 := min(a0, b0)  
r1 := min(a1, b1)  
...  
r7 := min(a7, b7)
```

```
__m64 __m_pminsw(__m64 a, __m64 b)
```

aとbの4ワードについて、要素ごとに最小値を計算します。

```
r0 := min(a0, b0)  
r1 := min(a1, b1)  
r2 := min(a2, b2)  
r3 := min(a3, b3)
```

```
__m64 __m_pminub(__m64 a, __m64 b)
```

aとbの8個の符号なしバイトについて、要素ごとに最小値を計算します。

```
r0 := min(a0, b0)  
r1 := min(a1, b1)  
...  
r7 := min(a7, b7)
```

```
int __m_pmovmskb(__m64 a)
```

aの各バイトの最上位ビットから、8ビット・マスクを作成します。

```
r := sign(a7)<<7 | sign(a6)<<6 | ... | sign(a0)
```

```
__m64 __m_pmulhuw(__m64 a, __m64 b)
```

aとbの対応する符号なしワードを乗算し、得られた32ビットの中間結果の上位16ビットを返します。

```
r0 := hiword(a0 * b0)  
r1 := hiword(a1 * b1)  
r2 := hiword(a2 * b2)  
r3 := hiword(a3 * b3)
```

```
__m64 __m_pshufw(__m64 a, int n)
```

aの4ワードを組み合わせて返します。セクタnは即値でなければなりません。

```
r0 := word (n&0x3) of a
r1 := word ((n>>2)&0x3) of a
r2 := word ((n>>4)&0x3) of a
r3 := word ((n>>6)&0x3) of a
void _m_maskmovq(__m64 d, __m64 n, char *p)
d のバイト要素を、条件付きでアドレス p にストアします。セクタ n の各バイトの最
上位ビットによって、それに対応する d の各バイトがストアされるかどうかが決まりま
す。
if (sign(n0)) p[0] := d0
if (sign(n1)) p[1] := d1
...
if (sign(n7)) p[7] := d7
__m64 _m_pavgb(__m64 a, __m64 b)
aとbの対応する符号なしバイトについて、(丸め)平均を計算します。
t = (unsigned short)a0 + (unsigned short)b0
r0 = (t >> 1) | (t & 0x01)
...
t = (unsigned short)a7 + (unsigned short)b7
r7 = (unsigned char)((t >> 1) | (t & 0x01))
__m64 _m_pavgw(__m64 a, __m64 b)
aとbの対応する符号なしワードについて、(丸め)平均を計算します。
t = (unsigned int)a0 + (unsigned int)b0
r0 = (t >> 1) | (t & 0x01)
...
t = (unsigned word)a7 + (unsigned word)b7
r7 = (unsigned short)((t >> 1) | (t & 0x01))
__m64 _m_psadbw(__m64 a, __m64 b)
aとbの対応する符号なしバイトの差の絶対値の合計を計算し、最下位ワードの値を返
します。上位3ワードはクリアされます。
r0 = abs(a0-b0) + ... + abs(a7-b7)
r1 = r2 = r3 = 0
```

## ストリーミングSIMD拡張命令を使用するメモリ操作と初期化操作

この節では、ロード操作、設定操作、およびストア操作を行う組込み関数について説明しま
す。ロード組込み関数と設定組込み関数はよく似ており、いずれも \_\_m128 型のデータを
初期化します。しかし、設定組込み関数は、データを定数で初期化するための関数で、float
引数を使用します。ロード組込み関数は、メモリからデータをロードする命令を模倣するた
めの関数で、浮動小数点引数を使用します。ストア組込み関数は、初期化したデータを、指定し

たアドレスに割り当てます。

次の表に、各種の組込み関数のリストを示します。以降の項には、各組込み関数の構文と簡単な説明を示します。

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組込み関数名	別名	操作	対応する命令
<code>_mm_load_ss</code>		最下位の値をロードして、上位3つの値をクリアする。	MOVSS
<code>_mm_load_ps1</code>	<code>_mm_load1_ps</code>	1つの値を4ワードすべてにロードする。	MOVSS + シャッフルリング
<code>_mm_load_ps</code>		4つの値をロードする (アドレスのアライメントが合っていないといけない)。	MOVAPS
<code>_mm_loadu_ps</code>		4つの値をロードする (アドレスのアライメントが合っている必要はない)。	MOVUPS
<code>_mm_loadr_ps</code>		4つの値を逆順でロードする。	MOVAPS + シャッフルリング
<code>_mm_set_ss</code>		最下位の値を設定し、上位3つの値をクリアする。	複合
<code>_mm_set_ps1</code>	<code>_mm_set1_ps</code>	4ワードすべてを同じ値に設定する。	複合
<code>_mm_set_ps</code>		4つの値を設定する (アドレスのアライメントが合っていないといけない)。	複合
<code>_mm_setr_ps</code>		4つの値を逆順で設定する。	複合
<code>_mm_setzero_ps</code>		4つの値をすべてクリアする。	複合
<code>_mm_store_ss</code>		最下位の値をストアする。	MOVSS
<code>_mm_store_ps1</code>	<code>_mm_store1_ps</code>	最下位の値を4ワードすべてにストアする。アドレスは、16バイトにアライメントが	Shuffling + MOVSS

		合っていないかもしれません。	
<code>_mm_store_ps</code>		4つの値をストアする (アドレスのアライメントが合っていない なければならない)。	MOVAPS
<code>_mm_storeu_ps</code>		4つの値をストアする (アドレスのアライメントが合っている 必要はない)。	MOVUPS
<code>_mm_storer_ps</code>		4つの値を逆順で ストアする。	MOVAPS + シャ ッフリング
<code>_mm_move_ss</code>		最下位ワードを設定 し、上位3つの値は そのまま渡す。	MOVSS
<code>_mm_getcsr</code>		レジスタの内容を返 す	STMXCSR
<code>_mm_setcsr</code>		コントロール・レジ スタ	LDMXCSR
<code>mm_prefetch</code>			
<code>_mm_stream_pi</code>			
<code>_mm_stream_ps</code>			
<code>mm_sfence</code>			

```

__m128 _mm_load_ss(float const*a)
    単精度浮動小数点値を最下位ワードにロードし、上位3ワードをクリアします。
    r0 := *a
    r1 := 0.0 ; r2 := 0.0 ; r3 := 0.0
__m128 _mm_load_ps1(float const*a)
    1個の単精度浮動小数点値をロードして、その値を4ワードすべてにコピーします。
    r0 := *a
    r1 := *a
    r2 := *a
    r3 := *a
__m128 _mm_load_ps(float const*a)
    4つの単精度浮動小数点値をロードします。アドレスは16バイトにアライメントが合ってい  
なければなりません。
    r0 := a[0]
    r1 := a[1]

```

```

    r2 := a[2]
    r3 := a[3]
__m128 __mm_loadu_ps(float const*a)
    4つの単精度浮動小数点値をロードします。アドレスは16バイトにアライメントが合ってい
    なくてもかまいません。
    r0 := a[0]
    r1 := a[1]
    r2 := a[2]
    r3 := a[3]
__m128 __mm_loadr_ps(float const*a)
    4つの単精度浮動小数点値を逆順でロードします。アドレスは16バイトにアライメントが
    合っていなければなりません。
    r0 := a[3]
    r1 := a[2]
    r2 := a[1]
    r3 := a[0]
__m128 __mm_set_ss ( float w)
    単精度浮動小数点値の最下位ワードをaに設定し、上位3ワードをクリアします。
    r0 := c
    r1 := r2 := r3 := 0.0
__m128 __mm_set_ss ( float w)
    4個の単精度浮動小数点値をaに設定します。
    r0 := r1 := r2 := r3 := a
__m128 __mm_set_ps ( float z, float y, float x, float w)
    4つの単精度浮動小数点値を、4つの入力値に設定します。
    r0 := a
    r1 := b
    r2 := c
    r3 := d
__m128 __mm_setr_ps(float a, float b, float c, float d)
    4つの単精度浮動小数点値を、逆順で4つの入力値に設定します。
    r0 := d
    r1 := c
    r2 := b
    r3 := a
__m128 __mm_setzero_ps(void)
    4つの単精度浮動小数点値をクリアします。
    r0 := r1 := r2 := r3 := 0.0
void __mm_store_ss(float *v, __m128 a)
    最下位の単精度浮動小数点値をストアします。
    *v := a0

```

```

void _mm_store_ps1(float *v, __m128 a)
    最下位の単精度浮動小数点値を、4ワードにストアします。
    v[0] := a0
    v[1] := a0
    v[2] := a0
    v[3] := a0
void _mm_store_ps(float *v, __m128 a)
    4個の単精度浮動小数点値をストアします。アドレスは16バイトにアライメントが合ってい
    なければなりません。
    v[0] := a0
    v[1] := a1
    v[2] := a2
    v[3] := a3
void _mm_storeu_ps(float *v, __m128 a)
    4個の単精度浮動小数点値をストアします。アドレスは16バイトにアライメントが合ってい
    なくてもかまいません。
    v[0] := a0
    v[1] := a1
    v[2] := a2
    v[3] := a3
void _mm_store_ss(float *v, __m128 a)
    4個の単精度浮動小数点値を逆順でストアします。アドレスは16バイトにアライメントが
    合っていなければなりません。
    v[0] := a3
    v[1] := a2
    v[2] := a1
    v[3] := a0
__m128 _mm_move_ss(__m128 a, __m128 b)
    最下位ワードを、bの単精度浮動小数点値に設定します。上位3個の単精度浮動小数
    点値は、aからそのまま渡されます。
    r0 := b0
    r1 := a1
    r2 := a2
    r3 := a3
unsigned int _mm_getcsr(void)
    コントロール・レジスタの内容を返します。
void _mm_setcsr(unsigned int i)
    コントロール・レジスタを指定された値に設定します。
void _mm_prefetch(char const*a, int sel)
    (PREFETCH を使用)1キャッシュ・ライン分のデータを、アドレスaからプロセッサに「近
    い」位置にロードします。sel の値は、プリフェッチ操作のタイプを指定します。この

```



値には、プリフェッチ命令のタイプに応じて、定数 `_MM_HINT_T0`、`_MM_HINT_T1`、`_MM_HINT_T2`、または `_MM_HINT_NTA` を指定してください。

```
void _mm_stream_pi(__m64 *p, __m64 a)
```

(`MOVNTQ` を使用) `a` のデータを、キャッシュを介さずに、アドレス `p` にストアします。この組込み関数を使用する前に、`mmx` レジスタのマルチメディア・ステートを空にする必要があります。詳細は、「EMMS 命令: 必要な理由」を参照してください。

```
void _mm_stream_ps(float *p, __m128 a)
```

(`MOVNTPS` を参照) `a` のデータを、キャッシュを介さずに、アドレス `p` にストアします。アドレスは16バイトにアライメントが合っていないかもしれません。

```
void _mm_sfence(void)
```

(`SFENCE` を使用) すべての先行するストアが、後に続くストアより前に、グローバルにアクセス可能になるのを保証します。

## ストリーミングSIMD拡張命令を使用するその他の組込み関数

ストリーミングSIMD拡張命令の組込み関数のプロトタイプは、ヘッダ・ファイル `xmmintrin.h` 内にあります。

組込み関数名	操作	対応する命令
<code>_mm_shuffle_ps</code>	シャッフル	<code>SHUFPS</code>
<code>_mm_unpackhi_ps</code>	上位の値のアンパック	<code>UNPCKHPS</code>
<code>_mm_unpacklo_ps</code>	下位の値のアンパック	<code>UNPCKLPS</code>
<code>_mm_loadh_pi</code>	上位の値のロード	<code>MOVHPS reg, mem</code>
<code>_mm_storeh_pi</code>	上位の値のストア	<code>MOVHPS mem, reg</code>
<code>_mm_movehl_ps</code>	上位から下位への移動	<code>MOVHLPS</code>
<code>_mm_movelh_ps</code>	下位から上位への移動	<code>MOVLHPS</code>
<code>_mm_loadl_pi</code>	下位の値のロード	<code>MOVLPS reg, mem</code>
<code>_mm_storel_pi</code>	下位の値のストア	<code>MOVLPS mem, reg</code>
<code>_mm_movemask_ps</code>	4ビットマスクの作成	<code>MOVMSKPS</code>

```
__m128 _mm_shuffle_ps(__m128 a, __m128 b, unsigned int imm8)
```

マスク `imm8` に基づいて、`a` と `b` から4個の単精度浮動小数点値を選択します。マスクは即値でなければなりません。シャッフルのセマンティクスについては、「ストリーミングSIMD 拡張命令を使用してシャッフルを行うマクロ関数」を参照してください。

```
__m128 _mm_unpackhi_ps(__m128 a, __m128 b)
```

`a` と `b` から上位2個の単精度浮動小数点値を選択し、インターリーブ(交互に配置)します。

```
    r0 := a2
    r1 := b2
    r2 := a3
    r3 := b3
```

```
__m128 _mm_unpacklo_ps(__m128 a, __m128 b)
```

aとbから下位2個の単精度浮動小数点値を選択し、インターリーブします。

```
r0 := a0
r1 := b0
r2 := a1
r3 := b1
```

```
__m128 __mm_loadh_pi(__m128, __m64 const *p)
```

アドレスpからロードされた64ビットのデータで、上位2個の単精度浮動小数点値を設定します。

```
r0 := a0
r1 := a1
r2 := *p0
r3 := *p1
```

```
void __mm_storeh_pi(__m64 *p, __m128 a)
```

aの上位2個の単精度浮動小数点値を、アドレスpにストアします。

```
*p0 := a2
*p1 := a3
```

```
__m128 __mm_movehl_ps(__m128 a, __m128 b)
```

bの上位2つの単精度浮動小数点値を、結果の下位2つの単精度浮動小数点値に移動します。aの上位2つの単精度浮動小数点値は、そのまま結果に渡されます。

```
r3 := a3
r2 := a2
r1 := b3
r0 := b2
```

```
__m128 __mm_movelh_ps(__m128 a, __m128 b)
```

bの下位2つの単精度浮動小数点値を、結果の上位2つの単精度浮動小数点値に移動します。aの下位2つの単精度浮動小数点値は、そのまま結果に渡されます。

```
r3 := b1
r2 := b0
r1 := a1
r0 := a0
```

```
__m128 __mm_loadl_pi(__m128 a, __m64 const *p)
```

アドレスpからロードされた64ビットのデータで、下位2個の単精度浮動小数点値を設定します。上位2個の値は、aからそのまま渡されます。

```
r0 := *p0
r1 := *p1
r2 := a2
r3 := a3
```

```
void __mm_storel_pi(__m64 *p, __m128 a)
```

aの下位2個の単精度浮動小数点値を、アドレスpにストアします。

```
*p0 := a0
*p1 := a1
```

```
int __mm_movemask_ps(__m128 a)
```

4つの単精度浮動小数点値の最上位ビットを使用して、4ビットマスクを作成します。

```
r := sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)
```

## Itanium® アーキテクチャ上でのストリーミング SIMD 拡張命令の使用

ストリーミング SIMD 拡張命令の組込み関数によって、Itanium® アーキテクチャ上でストリーミング SIMD 拡張命令を利用できます。IA-32 アーキテクチャとのソースコードの互換性を保つために、これらの組込み関数の名前と機能は、IA-32 ベースのストリーミング SIMD 拡張命令の組込み関数セットと同等になっています。

これらの組込み関数を使用してプログラムを作成するには、ストリーミング SIMD 拡張命令によって提供されるハードウェア機能をよく理解している必要があります。特に、次の3つの重要な点に注意してください。

- 一部の組込み関数は、以前に定義した IA-32 組込み関数との互換性のためにのみ用意しています。Itanium ベースのシステム上でこれらの組込み関数を使用すると、通常はパフォーマンスが低下します。下記の「互換性とパフォーマンス」の項を参照してください。
- \_\_m128 オブジェクトとしてロードまたはストアされる浮動小数点 (FP) データは、16 バイトにアライメントが合っていなければなりません。
- 一部の組込み関数は、命令の性質上、引数を即値で、すなわち定数整数 (リテラル) で指定する必要があります。

### データ型

ストリーミング SIMD 拡張命令の組込み関数は、新しいデータ型の \_\_m128 を使用します。このデータ型は、4つの単精度浮動小数点値で構成される128ビットデータを表します。これは128ビットのIA-32 ストリーミング SIMD 拡張命令レジスタに対応します。

コンパイラは、\_\_m128 型のローカルデータのアライメントを、スタック上の16バイト境界に合わせます。これらのデータ型のグローバル・データも、16バイトにアライメントを合わせます。integer 型、float 型、または double 型の配列のアライメントを合わせるには、declspec ディレクティブを使用できます。

Itanium 命令は、パックドデータの操作でもスカラデータの操作でも、同じ方法でストリーミング SIMD 拡張命令レジスタを操作します。したがって、スカラデータを表す \_\_m32 データ型はありません。スカラ操作には、\_\_m128 オブジェクトと「スカラ」形式の組込み関数を使用します。コンパイラとプロセッサは、32ビットのメモリ参照によって、これらの操作を実行します。ただし、パフォーマンス上の理由で、できるだけスカラ形式の操作をパックド形式の操作で置き換えるのをお勧めします。

\_\_m128 オブジェクトのアドレスを指定できます。

詳細については、『インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻: 命令セット・リファレンス』インテル(資料番号243191J)を参照してください。

Itanium ベース・システム上での実装手法

ストリーミング SIMD 拡張命令の組込み関数は、\_\_m128 データ型について定義していま

す。このデータ型は、4個の単精度浮動小数点値で構成される128ビット・データです。Itanium ベースのシステム上のSIMD命令は、2つの単精度浮動小数点値を保持する64ビット浮動小数点レジスタのデータを操作します。したがって、それぞれの `__m128` オペランドは、実際には浮動小数点レジスタのペアになります。このため、各組込み関数は、浮動小数点レジスタ・オペランドのペアを操作する、少なくとも2つの Itanium 命令に対応します。

### 互換性とパフォーマンス

Itanium ベースのシステム用のストリーミングSIMD拡張命令の組込み関数はほとんど、パフォーマンスの向上のためではなく、既存の IA-32 組込み関数との互換性を保つために用意したものです。IA-32 システム上でパフォーマンスを向上させる組込み関数を使用しても、Itanium ベースのシステム上ではパフォーマンスが向上しない場合があります。この理由の1つとして、一部の組込み関数は、IA-32 命令セットには厳密に対応付けられますが、Itanium 命令セットには対応付けられていない場合があります。したがって、Itanium ベースのシステム上でのパフォーマンスの向上のために用意している組込み関数と、単に既存の IA-32 コードとの互換性を保つために用意している組込み関数は、区別して使用する必要があります。次の組込み関数を使用すると、パフォーマンスが低下する可能性があります。これらの組込み関数は、既存のコードを移植する場合や、重要でないコード・セクションにのみ使用してください。

- ストリーミングSIMD拡張命令のスカラー組込み関数(名前の最後に `_ss` が付くもの) - できるだけパックド(`_ps`)版を使用してください。
- ストリーミングSIMD拡張命令の比較組込み関数 `comi` および `ucomi` - これらの組込み関数は、IA-32 命令 `COMISS` および `UCOMISS` にのみ対応します。これらの組込み関数を実行するには、一連の Itanium 命令を実行する必要があります。
- 変換操作には、通常は複数の命令が必要です。`_mm_cvtpi16_ps`、`_mm_cvtpu16_ps`、`_mm_cvtpi8_ps`、`_mm_cvtpu8_ps`、`_mm_cvtpi32x2_ps`、`_mm_cvtps_pi16`、`_mm_cvtps_pi8` は、特にパフォーマンス上のコストがかかります。
- ストリーミングSIMD拡張命令ユーティリティ組込み関数 `_mm_movemask_ps`

精度が多少低下してもかまわない場合は、真の `div` 組込み関数や `sqr` 組込み関数の代わりに、逆数の近似値を計算するSIMDの組込み関数(`rcp`)や逆数の平方根の近似値を計算するSIMDの組込み関数(`rsqr`)を使用すれば、処理速度が大幅に向上します。

## マクロ関数

### ストリーミングSIMD拡張命令を使用してシャッフルを行うマクロ関数

ストリーミングSIMD拡張命令には、シャッフル操作を記述する定数を生成するマクロ関数を用意しています。このマクロは、4個の小さな整数(0～3の範囲)を組み合わせて、SHUFPS命令が使用する8ビット即値を生成します。次の例を参照してください。

シャッフル関数のマクロ

```
_MM_SHUFFLE(z,y,x,w)
/* expands to the following value */
(z<<6) | (y<<4) | (x<<2) | w
```

4個の整数は、第1入力オペランドと第2入力オペランドからそれぞれどの2ワードを取り出して結果のワードに入れるかを選択するセレクトアとして機能します。

シャッフル関数のマクロの元のワードと結果のワード

```
      127      0
; m1 = [a b c d]
      127      0
; m2 = [e f g h]
m3 = _mm_shuffle_ps(m1, m2,
  _MM_SHUFFLE(1,0,3,2))
      127      0
; m3 = [g h a b]
```

### コントロール・レジスタを読み書きするマクロ関数

次のマクロ関数を使用して、コントロール・レジスタの各ビットを読み書きできます。詳細は、「設定操作」を参照してください。Itanium® ベースのシステムでは、これらのマクロではアクセスできないFPSRビットもあります。詳細については、「Itanium® 命令のネイティブ組込み関数」のgetfpsr() およびsetfpsr() 組込み関数を参照してください。

例外状態マクロ	マクロ引数
_MM_SET_EXCEPTION_STATE(x)	_MM_EXCEPT_INVALID
MM_GET_EXCEPTION_STATE()	MM_EXCEPT_DIV_ZERO
	MM_EXCEPT_DENORM
マクロの定義	_MM_EXCEPT_OVERFLOW
コントロール・レジスタの最下位から6番目のビットを読み書きします。	
	MM_EXCEPT_UNDERFLOW
	MM_EXCEPT_INEXACT

次の例では、ゼロ除算例外が発生したかどうかをテストします。

## \_MM\_EXCEPT\_DIV\_ZEROマクロによる例外状態の確認

```
if ( _MM_GET_EXCEPTION_STATE(x) & _MM_EXCEPT_DIV_ZERO) {
    /* Exception has occurred */
}
```

例外マスクマクロ	マクロ引数
<code>MM_SET_EXCEPTION_MASK(x)</code>	<code>MM_MASK_INVALID</code>
<code>MM_GET_EXCEPTION_MASK()</code>	<code>MM_MASK_DIV_ZERO</code>
	<code>MM_MASK_DENORM</code>
<b>マクロの定義</b> コントロール・レジスタの第7ビット～第12ビットを読み書きします。 注: 6つの例外マスクビットのすべてが、常に影響を受けます。 明示的にセットされないビットはクリアされます。	<code>_MM_MASK_OVERFLOW</code>
	<code>MM_MASK_UNDERFLOW</code>
	<code>MM_MASK_INEXACT</code>

次の例では、オーバーフロー例外とアンダーフロー例外をマスクし、その他のすべての例外をマスク解除します。

<code>_MM_MASK_OVERFLOW</code> と <code>_MM_MASK_UNDERFLOW</code> による例外マスクの変更
<code>_MM_SET_EXCEPTION_MASK(MM_MASK_OVERFLOW   MM_MASK_UNDERFLOW)</code>

丸めモード	マクロ引数
<code>MM_SET_ROUNDING_MODE(x)</code>	<code>MM_ROUND_NEAREST</code>
<code>MM_GET_ROUNDING_MODE()</code>	<code>MM_ROUND_DOWN</code>
<b>マクロの定義</b> コントロール・レジスタのビット13とビット14を読み書きします。	<code>_MM_ROUND_UP</code>
	<code>MM_ROUND_TOWARD_ZERO</code>

次の例では、丸めモードがゼロ方向への丸めになっているかどうかをテストします。

<code>_MM_ROUND_TOWARD_ZERO</code> による丸めモードの確認
<pre>if ( MM_GET_ROUNDING_MODE() == MM_ROUND_TOWARD_ZERO) {     /* Rounding mode is round toward zero */ }</pre>

ゼロ・フラッシュ・モード	マクロ引数
--------------	-------

<code>_MM_SET_FLUSH_ZERO_MODE(x)</code>	<code>_MM_FLUSH_ZERO_ON</code>
<code>MM_GET_FLUSH_ZERO_MODE()</code>	<code>MM_FLUSH_ZERO_OFF</code>
<b>マクロの定義</b> コントロール・レジスタのビット15を読み書きします。	
次の例では、ゼロ・フラッシュ・モードを無効にします。	
<b><code>_MM_FLUSH_ZERO_OFF</code>によるゼロ・フラッシュ・モードの変更</b>	
<code>MM_SET_FLUSH_ZERO_MODE( MM_FLUSH_ZERO_OFF)</code>	

## 行列の転置を行うマクロ関数

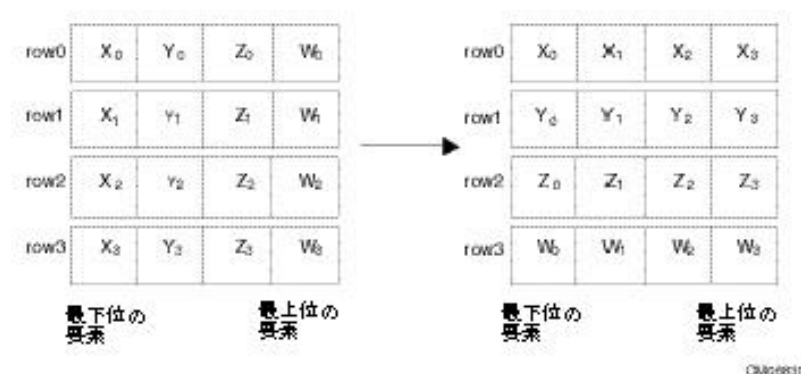
ストリーミングSIMD拡張命令には、単精度浮動小数点値の4×4行列を転置する、次のマクロ関数も用意しています。

`_MM_TRANSPOSE4_PS(row0, row1, row2, row3)`

引数row0、row1、row2、およびrow3は\_\_m128値であり、引数内の各要素が、4×4行列の行に対応します。転置した行列は、引数row0、row1、row2、およびrow3の形式で返されます。row0には元の行列の列0が格納され、row1には元の行列の列1が格納されます(以下同様)。

「`_MM_TRANSPOSE4_PS`による行列の転置」の図に、このマクロの転置機能を示します。

`_MM_TRANSPOSE4_PS`マクロによる行列の転置



## ストリーミングSIMD拡張命令2

### ストリーミングSIMD拡張命令2 の組込み関数の概要

この節では、インテル® C++ コンパイラ内でインテル® Pentium® 4 プロセッサのストリーミング SIMD 拡張命令 2 をサポートする、C++ 言語レベルの機能について説明します。これらの機能は、次の2種類に分類されます。

- 浮動小数点組込み関数 -- 倍精度浮動小数点データ型(\_\_m128d)に対する、算術



演算、論理演算、比較、変換、メモリ操作、および初期化を行う組込み関数について説明します。

- 整数組込み関数 -- 拡張精度整数データ型(`__m128i`)に対する、算術演算、論理演算、比較、変換、メモリ操作、および初期化を行う組込み関数について説明します。



インテル Pentium 4 プロセッサのストリーミングSIMD拡張命令2 の組込み関数は、IA-32 プラットフォームのみ定義しており、Itanium® ベース用のプラットフォームについては定義していません。インテル Pentium 4 プロセッサのストリーミングSIMD拡張命令2は、128ビットデータ(2個の64ビット倍精度浮動小数点値)を操作します。インテル Pentium 4 プロセッサのストリーミングSIMD拡張命令2 は、Itanium ベースのシステム上では使用できません。

詳細については、『*Pentium® 4 processor Streaming SIMD Extensions 2 External Architecture Specification (EAS)*』などのインテル Pentium 4 プロセッサのマニュアルを参照してください。また、これらの資料は、Webサイト <http://www.intel.co.jp/jp/developer/> からダウンロードできます。ストリーミングSIMD拡張命令2 の組込み関数を使用してプログラムを作成する際は、ストリーミングSIMD拡張命令2 によって提供されるハードウェア機能についてよく理解している必要があります。特に、次の3つの点に注意してください。

- `_mm_loadr_pd`や`_mm_cmpgt_sd`などの一部の組込み関数は、命令セットによって直接にはサポートされていません。これらの組込み関数は、プログラミング上の便宜のために用意されたものであり、実行時にはコストがかかります。
- `__m128d`オブジェクトとしてロードまたはストアされるデータは、通常は16バイトにアライメントが合っていないかもしれません。
- 一部の組込み関数は、命令の性質上、引数を即値で、すなわち定数整数(リテラル)で指定する必要があります。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

## 浮動小数点演算組込み関数

### ストリーミングSIMD拡張命令2 の浮動小数点算術演算

次の表に、ストリーミング SIMD 拡張命令 2の算術演算組込み関数のリストを示します。表の後に、各組込み関数の説明を示します。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

組込み関数名	対応する命令	操作	R0 の値	R1 の値
<code>_mm_add_sd</code>	ADDSD	加算	<code>a0 [op] b0</code>	<code>a1</code>
<code>_mm_add_pd</code>	ADDPD	加算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>
<code>_mm_sub_sd</code>	SUBSD	減算	<code>a0 [op] b0</code>	<code>a1</code>



<code>mm_sub_pd</code>	SUBPD	減算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>
<code>mm_mul_sd</code>	MULSD	乗算	<code>a0 [op] b0</code>	<code>a1</code>
<code>mm_mul_pd</code>	MULPD	乗算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>
<code>mm_div_sd</code>	DIVSD	除算	<code>a0 [op] b0</code>	<code>a1</code>
<code>mm_div_pd</code>	DIVPD	除算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>
<code>mm_sqrt_sd</code>	SQRTSD	平方根の計算	<code>a0 [op] b0</code>	<code>a1</code>
<code>mm_sqrt_pd</code>	SQRTPD	平方根の計算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>
<code>mm_min_sd</code>	MINSD	最小値の計算	<code>a0 [op] b0</code>	<code>a1</code>
<code>mm_min_pd</code>	MINPD	最小値の計算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>
<code>mm_max_sd</code>	MAXSD	最大値の計算	<code>a0 [op] b0</code>	<code>a1</code>
<code>mm_max_pd</code>	MAXPD	最大値の計算	<code>a0 [op] b0</code>	<code>a1 [op] b1</code>

```

__m128d __mm_add_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値を加算します。上位の倍精度浮動小数点値は、a
    からそのまま渡されます。
    r0 := a0 + b0
    r1 := a1
__m128d __mm_add_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値を加算します。
    r0 := a0 + b0
    r1 := a1 + b1
__m128d __mm_sub_sd(__m128d a, __m128d b)
    aの下位の倍精度浮動小数点値から、b の下位の倍精度浮動小数点値を引きます。上
    位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := a0 - b0
    r1 := a1
__m128d __mm_sub_pd(__m128d a, __m128d b)
    aの2個の倍精度浮動小数点値から、bの2個の倍精度浮動小数点値を引きます。
    r0 := a0 - b0
    r1 := a1 - b1
__m128d __mm_mul_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値を乗算します。上位の倍精度浮動小数点値は、a
    からそのまま渡されます。
    r0 := a0 * b0
    r1 := a1
__m128d __mm_mul_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値を乗算します。
    r0 := a0 * b0
    r1 := a1 * b1
__m128d __mm_div_sd(__m128d a, __m128d b)
    aの下位の倍精度浮動小数点値を、bの下位の倍精度浮動小数点値で割ります。上位

```

の倍精度浮動小数点値は、aからそのまま渡されます。

```
r0 := a0 / b0  
r1 := a1
```

```
__m128d _mm_div_pd(__m128d a, __m128d b)
```

aの2個の倍精度浮動小数点値を、bの2個の倍精度浮動小数点値で割ります。

```
r0 := a0 / b0  
r1 := a1 / b1
```

```
__m128d _mm_sqrt_sd(__m128d a, __m128d b)
```

bの下位の倍精度浮動小数点値の平方根を計算します。上位の倍精度浮動小数点値は、aからそのまま渡されます。

```
r0 := sqrt(b0)  
r1 := a1
```

```
__m128d _mm_sqrt_pd(__m128d a)
```

aの2個の倍精度浮動小数点値の平方根を計算します。

```
r0 := sqrt(a0)  
r1 := sqrt(a1)
```

```
__m128d _mm_min_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、小さい方の値を計算します。上位の倍精度浮動小数点値は、aからそのまま渡されます。

```
r0 := min(a0, b0)  
r1 := a1
```

```
__m128d _mm_min_pd(__m128d a, __m128d b)
```

aとbの2個の倍精度浮動小数点値について、それぞれ小さい方の値を計算します。

```
r0 := min(a0, b0)  
r1 := min(a1, b1)
```

```
__m128d _mm_max_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、大きい方の値を計算します。上位の倍精度浮動小数点値は、aからそのまま渡されます。

```
r0 := max(a0, b0)  
r1 := a1
```

```
__m128d _mm_max_pd(__m128d a, __m128d b)
```

aとbの2個の倍精度浮動小数点値について、それぞれ大きい方の値を計算します。

```
r0 := max(a0, b0)  
r1 := max(a1, b1)
```

## ストリーミングSIMD拡張命令2 の論理演算

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

```
__m128d _mm_and_pd(__m128d a, __m128d b)
```

(ANDPD を使用) a と b の2個の倍精度浮動小数点値について、ビット単位の AND

(論理積)を計算します。

```
r0 := a0 & b0
```

```
r1 := a1 & b1
```

```
__m128d _mm_andnot_pd(__m128d a, __m128d b)
```

(ANDNPD を使用)a の128ビット値のビット単位の NOT (否定)を実行し、その結果と b の128ビット値について、ビット単位の AND (論理積)を計算します。

```
r0 := (~a0) & b0
```

```
r1 := (~a1) & b1
```

```
__m128d _mm_or_pd(__m128d a, __m128d b)
```

(ORPD を使用)a と b の2個の倍精度浮動小数点値について、ビット単位の OR (論理和)を計算します。

```
r0 := a0 | b0
```

```
r1 := a1 | b1
```

```
__m128d _mm_xor_pd(__m128d a, __m128d b)
```

(XORPD を使用)a と b の2個の倍精度浮動小数点値について、ビット単位の XOR (論理和)を計算します。

```
r0 := a0 ^ b0
```

```
r1 := a1 ^ b1
```

## ストリーミングSIMD拡張命令2 の比較操作

比較組込み関数は、aとbの比較を実行します。パワード形式の場合は、a と b の2つの倍精度浮動小数点値を比較して、128ビットマスクを返します。スカラ形式の場合は、a と b の下位の倍精度浮動小数点値を比較して、64ビットマスクを返します。上位の倍精度浮動小数点値は、a からそのまま渡されます。マスクは、各要素について、比較の結果が真の場合は 0xffffffffffffffffに設定し、偽の場合は0x0に設定します。命令名の後の r は、SIMD命令の実行時にオペランドが逆順にされることを示します。次の表に、ストリーミング SIMD 拡張命令 2の比較組込み関数のリストを示します。表の後に、各組込み関数の説明を示します。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

組込み関数名	対応する命令	比較の条件
<code>_mm_cmpeq_pd</code>	CMPEQPD	等しい
<code>_mm_cmplt_pd</code>	CMPLTPD	より大きい
<code>_mm_cmple_pd</code>	CMPLEPD	より小さいか等しい
<code>_mm_cmpgt_pd</code>	CMPLTPDr	より大きい
<code>_mm_cmpge_pd</code>	CMPLEPD <sub>r</sub>	より大きいか等しい
<code>_mm_cmpord_pd</code>	CMPODPD	順序化可能
<code>_mm_cmpunord_pd</code>	CMPUNORDPD	順序化不可能
<code>_mm_cmpneq_pd</code>	CMPNEQPD	≠
<code>_mm_cmpnlt_pd</code>	CMPNLTPD	より小さくない

<code>mm_cmpnle_pd</code>	<code>CMPNLEPD</code>	より小さくなく等しくない
<code>mm_cmpngt_pd</code>	<code>CMPNLTPDr</code>	より大きくない
<code>mm_cmpnge_pd</code>	<code>CMPLEPDr</code>	より大きくなく等しくない
<code>mm_cmpeq_sd</code>	<code>CMPEQSD</code>	等しい
<code>mm_cmplt_sd</code>	<code>CMPLTSD</code>	より大きい
<code>mm_cmple_sd</code>	<code>CMPLESD</code>	より小さいか等しい
<code>mm_cmpgt_sd</code>	<code>CMPLTSDr</code>	より大きい
<code>mm_cmpge_sd</code>	<code>CMPLESDr</code>	より大きい等しい
<code>mm_cmpord_sd</code>	<code>CMPORDSD</code>	順序化可能
<code>mm_cmpunord_sd</code>	<code>CMPUNORDSD</code>	順序化不可能
<code>mm_cmpneq_sd</code>	<code>CMPNEQSD</code>	等しくない
<code>mm_cmpnlt_sd</code>	<code>CMPNLTSD</code>	より小さくない
<code>mm_cmpnle_sd</code>	<code>CMPNLESD</code>	より小さくなく等しくない
<code>mm_cmpngt_sd</code>	<code>CMPNLTPSDr</code>	より大きくない
<code>mm_cmpnge_sd</code>	<code>CMPNLESDr</code>	より大きくなく等しくない
<code>mm_comieq_sd</code>	<code>COMISD</code>	等しい
<code>mm_comilt_sd</code>	<code>COMISD</code>	より大きい
<code>mm_comile_sd</code>	<code>COMISD</code>	より小さいか等しい
<code>mm_comigt_sd</code>	<code>COMISD</code>	より大きい
<code>mm_comige_sd</code>	<code>COMISD</code>	より大きい等しい
<code>mm_comineq_sd</code>	<code>COMISD</code>	等しくない
<code>mm_ucomieq_sd</code>	<code>UCOMISD</code>	等しい
<code>mm_ucomilt_sd</code>	<code>UCOMISD</code>	より大きい
<code>mm_ucomile_sd</code>	<code>UCOMISD</code>	より小さいか等しい
<code>mm_ucomigt_sd</code>	<code>UCOMISD</code>	より大きい
<code>mm_ucomige_sd</code>	<code>UCOMISD</code>	より大きい等しい
<code>mm_ucomineq_sd</code>	<code>UCOMISD</code>	等しくない

`__m128d __mm_cmpeq_pd(__m128d a, __m128d b)`

a と b の2つの倍精度浮動小数点値が等しいかどうか比較します。

`r0 := (a0 == b0) ? 0xffffffffffffffff : 0x0`

`r1 := (a1 == b1) ? 0xffffffffffffffff : 0x0`

`__m128d __mm_cmplt_pd(__m128d a, __m128d b)`

aとbの2個の倍精度浮動小数点値について、aがbより小さいかどうか比較します。

`r0 := (a0 < b0) ? 0xffffffffffffffff : 0x0`

`r1 := (a1 < b1) ? 0xffffffffffffffff : 0x0`

`__m128d __mm_cmple_pd(__m128d a, __m128d b)`

aとbの2個の倍精度浮動小数点値について、aがbより小さいか等しいかどうか比較します。

`r0 := (a0 <= b0) ? 0xffffffffffffffff : 0x0`

```

    r1 := (a1 <= b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpgt_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値について、aがbより大きいかどうか比較します。
    r0 := (a0 > b0) ? 0xffffffffffffffff : 0x0
    r1 := (a1 > b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpge_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値について、aがbより大きいか等しいかどうか比較します。
    r0 := (a0 >= b0) ? 0xffffffffffffffff : 0x0
    r1 := (a1 >= b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpord_pd(__m128d a, __m128d b)
    a と b の2つの倍精度浮動小数点値が順序化可能かどうか判定します。
    r0 := (a0 ord b0) ? 0xffffffffffffffff : 0x0
    r1 := (a1 ord b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpunord_pd(__m128d a, __m128d b)
    a と b の2つの倍精度浮動小数点値が順序化不可能かどうか判定します。
    r0 := (a0 unord b0) ? 0xffffffffffffffff : 0x0
    r1 := (a1 unord b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpneq_pd ( __m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値が等しくないかどうか比較します。
    r0 := (a0 != b0) ? 0xffffffffffffffff : 0x0
    r1 := (a1 != b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpnlt_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値について、aがbより小さくないかどうか比較します。
    r0 := !(a0 < b0) ? 0xffffffffffffffff : 0x0
    r1 := !(a1 < b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpnle_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値について、aがbより小さくなく等しくないかどうか比較します。
    r0 := !(a0 <= b0) ? 0xffffffffffffffff : 0x0
    r1 := !(a1 <= b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpngt_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値について、aがbより大きくないかどうか比較します。
    r0 := !(a0 > b0) ? 0xffffffffffffffff : 0x0
    r1 := !(a1 > b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpnge_pd(__m128d a, __m128d b)
    aとbの2個の倍精度浮動小数点値について、aがbより大きくなく等しくないかどうか比較します。
    r0 := !(a0 >= b0) ? 0xffffffffffffffff : 0x0

```

```

    r1 := !(a1 >= b1) ? 0xffffffffffffffff : 0x0
__m128d _mm_cmpeq_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値が等しいかどうか比較します。上位の倍精度浮動
    小数点値は、aからそのまま渡されます。
    r0 := (a0 == b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d _mm_cmplt_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより小さいかどうか比較します。上
    位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 < b0) ? 0xffffffffffffffff : 0x0
    r1 := i1
__m128d _mm_cmple_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより小さいか等しいかどうか比較し
    ます。上位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 <= b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d _mm_cmpgt_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより大きいかどうか比較します。上
    位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 > b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d _mm_cmpge_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより大きいか等しいかどうか比較し
    ます。上位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 >= b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d _mm_cmpord_sd(__m128d a, __m128d b)
    a と b の下位の倍精度浮動小数点値が順序化可能かどうか判定します。上位の倍精
    度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 ord b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d _mm_cmpunord_sd(__m128d a, __m128d b)
    a と b の下位の倍精度浮動小数点値が順序化不可能かどうか判定します。上位の倍
    精度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 unord b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d _mm_cmpneq_sd(__m128d a, __m128d b)
    a と b の下位の倍精度浮動小数点値が等しくないかどうか比較します。上位の倍精
    度浮動小数点値は、aからそのまま渡されます。
    r0 := (a0 != b0) ? 0xffffffffffffffff : 0x0

```

```

    r1 := a1
__m128d __mm_cmpnlt_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより小さくないかどうか比較します。
    上位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := !(a0 < b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d __mm_cmpnle_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより小さくなく等しくないかどうか比較します。
    上位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := !(a0 <= b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d __mm_cmpngt_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより大きくないかどうか比較します。
    上位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := !(a0 > b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
__m128d __mm_cmpnge_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより大きくなく等しくないかどうか比較します。
    上位の倍精度浮動小数点値は、aからそのまま渡されます。
    r0 := !(a0 >= b0) ? 0xffffffffffffffff : 0x0
    r1 := a1
int __mm_comieq_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aとbが等しいかどうか比較します。aとb
    が等しい場合は、1を返します。それ以外の場合は、0を返します。
    r := (a0 == b0) ? 0x1 : 0x0
int __mm_comilt_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより小さいかどうか比較します。a
    がbより小さい場合は、1を返します。それ以外の場合は、0を返します。
    r := (a0 < b0) ? 0x1 : 0x0
int __mm_comile_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより小さいか等しいかどうか比較し
    ます。aがbより小さいか等しい場合は、1を返します。それ以外の場合は、0を返します。
    r := (a0 <= b0) ? 0x1 : 0x0
int __mm_comigt_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより大きいかどうか比較します。a
    がbより大きい場合は、1を返します。それ以外の場合は、0を返します。
    r := (a0 > b0) ? 0x1 : 0x0
int __mm_comige_sd(__m128d a, __m128d b)
    aとbの下位の倍精度浮動小数点値について、aがbより大きいか等しいかどうか比較し

```

ます。aがbより大きいか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 >= b0) ? 0x1 : 0x0
```

```
int __mm_comineq_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aとbが等しくないかどうか比較します。aとbが等しくない場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 != b0) ? 0x1 : 0x0
```

```
int __mm_ucomieq_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aとbが等しいかどうか比較します。aとbが等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 == b0) ? 0x1 : 0x0
```

```
int __mm_ucomilt_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aがbより小さいかどうか比較します。aがbより小さい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 < b0) ? 0x1 : 0x0
```

```
int __mm_ucomile_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aがbより小さいか等しいかどうか比較します。aがbより小さいか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 <= b0) ? 0x1 : 0x0
```

```
int __mm_ucomigt_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aがbより大きいかどうか比較します。aがbより大きい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 > b0) ? 0x1 : 0x0
```

```
int __mm_ucomige_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aがbより大きいか等しいかどうか比較します。aがbより大きいか等しい場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 >= b0) ? 0x1 : 0x0
```

```
int __mm_ucomineq_sd(__m128d a, __m128d b)
```

aとbの下位の倍精度浮動小数点値について、aとbが等しくないかどうか比較します。aとbが等しくない場合は、1を返します。それ以外の場合は、0を返します。

```
r := (a0 != b0) ? 0x1 : 0x0
```

## ストリーミングSIMD拡張命令2 の変換操作

変換組込み関数は、データ型を他のデータ型に変換します。`__mm_cvtpd_ps` などの変換を実行すると、データの精度が低下します。このような場合に使用する丸めモードは、MXCSRレジスタの値によって決まります。デフォルトの丸めモードは、最近値への丸めです。ただし、C および C++ 言語は、型変換の実行時に切り捨てモードを使用します。

`__mm_cvttpd_epi32` および `__mm_cvttss_si32` 組込み関数は、MXCSRレジスタで



指定した丸めモードに関係なく、切り捨てモードを使用します。

次の表に、ストリーミングSIMD拡張命令2 の変換操作組込み関数のリストを示します。表の後に、各組込み関数の説明を示します。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

組込み関数名	対応する命令	戻り値の型	パラメータ
<code>mm_cvtpd_ps</code>	CVTPD2PS	<code>m128</code>	<code>( m128d a)</code>
<code>mm_cvtps_pd</code>	CVTPS2PD	<code>m128d</code>	<code>( m128 a)</code>
<code>_mm_cvtepi32_pd</code>	CVTDQ2PD	<code>__m128d</code>	<code>(__m128i a)</code>
<code>_mm_cvtpd_epi32</code>	CVTPD2DQ	<code>__m128i</code>	<code>(__m128d a)</code>
<code>_mm_cvtsd_si32</code>	CVTSD2SI	<code>int</code>	<code>(__m128d a)</code>
<code>_mm_cvtsd_ss</code>	CVTSD2SS	<code>__m128</code>	<code>(__m128 a, m128d b)</code>
<code>_mm_cvtsi32_sd</code>	CVTSI2SD	<code>__m128d</code>	<code>(__m128d a, int b)</code>
<code>_mm_cvtss_sd</code>	CVTSS2SD	<code>__m128d</code>	<code>(__m128d a, m128 b)</code>
<code>_mm_cvttpd_epi32</code>	CVTTPD2DQ	<code>__m128i</code>	<code>(__m128d a)</code>
<code>_mm_cvttsd_si32</code>	CVTTSD2SI	<code>int</code>	<code>(__m128d a)</code>
<code>_mm_cvtpd_pi32</code>	CVTPD2PI	<code>__m64</code>	<code>(__m128d a)</code>
<code>_mm_cvttpd_pi32</code>	CVTTPD2PI	<code>__m64</code>	<code>(__m128d a)</code>
<code>_mm_cvtpi32_pd</code>	CVTPI2PD	<code>__m128d</code>	<code>(__m64 a)</code>

`__m128 _mm_cvtpd_ps(__m128d a)`

`a` の2つの倍精度浮動小数点値を単精度浮動小数点値に変換します。

```
r0 := (float) a0
r1 := (float) a1
r2 := 0.0 ; r3 := 0.0
```

`__m128d _mm_cvtps_pd(__m128 a)`

`a` の下位2つの単精度浮動小数点値を倍精度浮動小数点値に変換します。

```
r0 := (double) a0
r1 := (double) a1
```

`__m128d _mm_cvtepi32_pd(__m128i a)`

`a` の下位2つの符号付き 32ビット整数値を倍精度浮動小数点値に変換します。

```
r0 := (double) a0
r1 := (double) a1
```

```

__m128i _mm_cvtpd_epi32(__m128d a)
    a の2つの倍精度浮動小数点値を符号付き 32ビット整数値に変換します。
    r0 := (int) a0
    r1 := (int) a1
    r2 := 0x0 ; r3 := 0x0
int _mm_cvtsd_si32(__m128d a)
    a の下位の倍精度浮動小数点値を符号付き 32ビット整数値に変換します。
    r := (int)a0
__m128 _mm_cvtsd_ss(__m128 a, __m128d b)
    b の下位の倍精度浮動小数点値を単精度浮動小数点値に変換します。a の上位の単
    精度浮動小数点値はそのまま渡されます。
    r0 := (float)b0
    r1 := a1; r2 := a2 ; r3 := a3
__m128d _mm_cvtsi32_sd(__m128d a, int b)
    b の符号付き整数値を倍精度浮動小数点値に変換します。a の上位の倍精度浮動小
    数点値はそのまま渡されます。
    r0 := (double) b
    r1 := a1
__m128d _mm_cvtss_sd(__m128d a, __m128 b)
    b の最下位の単精度浮動小数点値を倍精度浮動小数点値に変換します。a の上位の
    倍精度浮動小数点値はそのまま渡されます。
    r0 := (double) b0
    r1 := a1
__m128i _mm_cvttpd_epi32(__m128d a)
    切り捨てを使用して、aの2個の倍精度浮動小数点値を符号付き32ビット整数値に変換
    します。
    r0 := (int) a0
    r1 := (int) a1
    r2 := 0x0 ; r3 := 0x0
int _mm_cvttss_si32(__m128d a)
    切り捨てを使用して、aの下位の倍精度浮動小数点値を符号付き32ビット整数に変換し
    ます。
    r := (int)a0
__m64 _mm_cvtpd_pi32(__m128d a)
    a の2つの倍精度浮動小数点値を符号付き 32ビット整数値に変換します。
    r0 := (int) a0
    r1 := (int) a1
__m64 _mm_cvttpd_pi32(__m128d a)
    切り捨てを使用して、a の2つの倍精度浮動小数点値を符号付き 32ビット整数値に変
    換します。

```

```
    r0 := (int) a0
    r1 := (int) a1
__m128d __mm_cvtpi32_pd(__m64 a)
    a の2つの符号付き 32ビット整数値を倍精度浮動小数点値に変換します。
    r0 := (double) a0
    r1 := (double) a1
```

## 浮動小数点のメモリ操作と初期化操作

### ストリーミングSIMD拡張命令2 の浮動小数点メモリ操作と初期化操作

この節では、ロード操作、設定操作、およびストア操作を行う組込み関数について説明します。ロード組込み関数と設定組込み関数はよく似ており、いずれも `__m128d` 型のデータを初期化します。しかし、設定組込み関数は、データを定数で初期化するための関数で、`double` 引数を使用します。ロード組込み関数は、メモリからデータをロードする命令を模倣するための関数で、`double` ポインタ引数を使用します。ストア組込み関数は、初期化したデータを、指定したアドレスに割り当てます。



注

移動操作の組込み関数はありません。1つのレジスタから別のレジスタへデータを移動するには、`A = B` のように単に代入します。AとB は移動操作のソースで対象レジスタです。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

### ストリーミングSIMD拡張命令2 のロード操作

次のロード操作組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

```
__m128d _mm_load_pd(double const*dp)
    (MOVAPD を使用) 2個の倍精度浮動小数点値をロードします。アドレス p は、16バイトにアライメントが合っていなければなりません。
    r0 := p[0]
    r1 := p[1]
__m128d _mm_load1_pd(double const*dp)
    (MOVSD とシャッフリングを使用) 1個の倍精度浮動小数点値をロードして、その値を両方の要素にコピーします。アドレス dp は、16バイトにアライメントが合っていないかまいません。
    r0 := *p
    r1 := *p
__m128d _mm_loadr_pd(double const*dp)
    (MOVAPD とシャッフリングを使用) 2個の倍精度浮動小数点値を逆順でロードします。アドレス p は、16バイトにアライメントが合っていなければなりません。
    r0 := p[1]
    r1 := p[0]
__m128d _mm_loadu_pd(double const*dp)
    (MOVUPD を使用) 2個の倍精度浮動小数点値をロードします。アドレス dp は、16バイトにアライメントが合っていないかまいません。
    r0 := p[0]
```

```

    r1 := p[1]
__m128d __mm_load_sd(double const*dp)
    (MOVSD を使用) 1個の倍精度浮動小数点値をロードします。上位の倍精度浮動小数
    点値は0に設定されます。アドレス dp は、16バイトにアライメントが合っていないてもか
    まいません。
    r0 := *p
    r1 := 0.0
__m128d __mm_loadh_pd(__m128d a, double const*dp)
    (MOVHPD を使用)結果の上位の倍精度浮動小数点値として、1個の倍精度浮動小数
    点値をロードします。下位の倍精度浮動小数点値は、a からそのまま渡されます。アド
    レス dp は、16バイトにアライメントが合っていないてもかまいません。
    r0 := a0
    r1 := *p
__m128d __mm_loadl_pd(__m128d a, double const*dp)
    (MOVLPD を使用)結果の下位の倍精度浮動小数点値として、1個の倍精度浮動小数
    点値をロードします。上位の倍精度浮動小数点値は、aからそのまま渡されます。アドレ
    ス dp は、16バイトにアライメントが合っていないてもかまいません。
    r0 := *p
    r1 := a1

```

## ストリーミングSIMD拡張命令2 の設定操作

次の設定操作組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポ  
ートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル  
emmintrin.h内にあります。

```

__m128d __mm_set_sd(double w)
    (複合)下位の倍精度浮動小数点値をwに設定し、上位の倍精度浮動小数点値を0に設
    定します。
    r0 := w
    r1 := 0.0
__m128d __mm_set1_pd(double w)
    (複合)2個の倍精度浮動小数点値をwに設定します。
    r0 := w
    r1 := w
__m128d __mm_set_pd(double w, double x)
    (複合)下位の倍精度浮動小数点値をxに設定し、上位の倍精度浮動小数点値をwに設
    定します。
    r0 := x
    r1 := w
__m128d __mm_setr_pd(double w, double x)
    (複合)下位の倍精度浮動小数点値をwに設定し、上位の倍精度浮動小数点値をxに設

```

定します。

```
r0 := w  
r1 := x
```

```
__m128d _mm_setzero_pd(void)  
(XORPD を使用)2個の倍精度浮動小数点値を0に設定します。
```

```
r0 := 0.0  
r1 := 0.0
```

```
__m128d _mm_move_sd(__m128d a, __m128d b)  
(MOVSD を使用)下位の倍精度浮動小数点値を、bの下位の倍精度浮動小数点値に  
設定します。上位の倍精度浮動小数点値は、aからそのまま渡されます。
```

```
r0 := b0  
r1 := a1
```

### ストリーミングSIMD拡張命令2 のストア操作

次のストア操作組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

```
void _mm_store_sd(double *dp, __m128d a)  
(MOVSD を使用)aの下位の倍精度浮動小数点値をストアします。アドレス dp は、16  
バイトにアライメントが合っていないかまいません。
```

```
*dp := a0
```

```
void _mm_storel_pd(double *dp, __m128d a)  
(MOVAPD とシャッフリングを使用)aの下位の倍精度浮動小数点値を2回ストアします。  
アドレス dp は、16バイトにアライメントが合っていなければなりません。
```

```
dp[0] := a0  
dp[1] := a0
```

```
void _mm_store_pd(double *dp, __m128d a)  
(MOVAPD を使用)2個の倍精度浮動小数点値をストアします。アドレス dp は、16バ  
イトにアライメントが合っていなければなりません。
```

```
dp[0] := a0  
dp[1] := a1
```

```
void _mm_storeu_pd(double *dp, __m128d a)  
(MOVUPD を使用)2個の倍精度浮動小数点値をストアします。アドレス dp は、16バ  
イトにアライメントが合っていないかまいません。
```

```
dp[0] := a0  
dp[1] := a1
```

```
void _mm_storer_pd(double *dp, __m128d a)  
(MOVAPD とシャッフリングを使用)2個の倍精度浮動小数点値を逆順でストアします。  
アドレス dp は、16バイトにアライメントが合っていなければなりません。
```

```
dp[0] := a1
```

```

    dp[1] := a0
void _mm_storeh_pd(double *dp, __m128d a)
    (MOVHPPD を使用)aの上位の倍精度浮動小数点値をストアします。
    *dp := a1
void _mm_storel_pd(double *dp, __m128d a)
    (MOVLPPD を使用)aの下位の倍精度浮動小数点値をストアします。
    *dp := a0

```

## ストリーミングSIMD拡張命令2 その他の操作

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

```

__m128d _mm_unpackhi_pd(__m128d a, __m128d b)
    (UNPCKHPD を使用)a と b の上位の倍精度浮動小数点値をインターリーブ(交互に
    配置)します。
    r0 := a1
    r1 := b1
__m128d _mm_unpacklo_pd(__m128d a, __m128d b)
    (UNPCKLPD を使用)a と b の下位の倍精度浮動小数点値をインターリーブします。
    r0 := a0
    r1 := b0
int _mm_movemask_pd(__m128d a)
    (MOVMSKPD を使用)a の2個の倍精度浮動小数点値の符号ビットから、2ビットマスク
    を作成します。
    r := sign(a1) << 1 | sign(a0)
__m128d _mm_shuffle_pd(__m128d a, __m128d b, int i)
    (SHUFFPD を使用)マスク i に基づいて、a と b から2個の倍精度浮動小数点値を
    選択します。マスクは即値でなければなりません。シャッフルのセマンティクスについて
    は、「シャッフルを行うマクロ関数」を参照してください。

```

## 整数演算組込み関数

### ストリーミングSIMD拡張命令2 の整数算術演算

次の表に、ストリーミング SIMD 拡張命令 2の整数算術演算組込み関数のリストを示します。表の後に、各組込み関数の説明を示します。ストリーミング SIMD 拡張命令 2のパックド算術演算組込み関数については、「浮動小数点算術演算」を参照してください。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

組込み関数	命令	操作
<code>mm_add_epi8</code>	<code>PADDB</code>	加算
<code>mm_add_epi16</code>	<code>PADDW</code>	加算
<code>mm_add_epi32</code>	<code>PADDQ</code>	加算
<code>mm_add_si64</code>	<code>PADDQ</code>	加算
<code>mm_add_epi64</code>	<code>PADDQ</code>	加算
<code>mm_adds_epi8</code>	<code>PADDSB</code>	加算
<code>mm_adds_epi16</code>	<code>PADDSW</code>	加算
<code>mm_adds_epu8</code>	<code>PADDUSB</code>	加算
<code>mm_adds_epu16</code>	<code>PADDUSW</code>	加算
<code>mm_avg_epu8</code>	<code>PAVGB</code>	平均値の計算
<code>mm_avg_epu16</code>	<code>PAVGW</code>	平均値の計算
<code>mm_madd_epi16</code>	<code>PMADDWD</code>	乗算/加算
<code>mm_max_epi16</code>	<code>PMAXSW</code>	最大値の計算
<code>mm_max_epu8</code>	<code>PMAXUB</code>	最大値の計算
<code>mm_min_epi16</code>	<code>PMINSW</code>	最小値の計算
<code>mm_min_epu8</code>	<code>PMINUB</code>	最小値の計算
<code>mm_mulhi_epi16</code>	<code>PMULHW</code>	乗算
<code>mm_mulhi_epu16</code>	<code>PMULHUW</code>	乗算
<code>mm_mullo_epi16</code>	<code>PMULLW</code>	乗算
<code>mm_mul_su32</code>	<code>PMULUDQ</code>	乗算
<code>mm_mul_epu32</code>	<code>PMULUDQ</code>	乗算
<code>mm_sad_epu8</code>	<code>PSADBW</code>	差の計算/加算
<code>mm_sub_epi8</code>	<code>PSUBB</code>	減算
<code>mm_sub_epi16</code>	<code>PSUBW</code>	減算
<code>mm_sub_epi32</code>	<code>PSUBD</code>	減算
<code>mm_sub_si64</code>	<code>PSUBQ</code>	減算
<code>mm_sub_epi64</code>	<code>PSUBQ</code>	減算
<code>mm_subs_epi8</code>	<code>PSUBSB</code>	減算
<code>mm_subs_epi16</code>	<code>PSUBSW</code>	減算



<code>mm_subs_epu8</code>	<code>PSUBUSB</code>	減算
<code>mm_subs_epu16</code>	<code>PSUBUSW</code>	減算

```

__m128i _mm_add_epi8 (__m128i a, __m128i b)
    aの16個の符号付きまたは符号なし8ビット整数を、bの16個の符号付きまたは符号なし8ビット整数に加算します。
    r0 := a0 + b0
    r1 := a1 + b1
    ...
    r15 := a15 + b15
__mm128i _mm_add_epi16(__m128i a, __m128i b)
    aの8個の符号付きまたは符号なし16ビット整数を、bの8個の符号付きまたは符号なし16ビット整数に加算します。
    r0 := a0 + b0
    r1 := a1 + b1
    ...
    r7 := a7 + b7
__m128i _mm_add_epi32(__m128i a, __m128i b)
    aの4個の符号付きまたは符号なし32ビット整数を、bの4個の符号付きまたは符号なし32ビット整数に加算します。
    r0 := a0 + b0
    r1 := a1 + b1
    r2 := a2 + b2
    r3 := a3 + b3
__m64 _mm_add_si64(__m64 a, __m64 b)
    aの符号付きまたは符号なし64ビット整数を、bの符号付きまたは符号なし64ビット整数に加算します。
    r := a + b
__m128i _mm_add_epi64(__m128i a, __m128i b)
    aの2個の符号付きまたは符号なし64ビット整数を、bの2個の符号付きまたは符号なし64ビット整数に加算します。
    r0 := a0 + b0
    r1 := a1 + b1
__m128i _mm_adds_epi8(__m128i a, __m128i b)
    飽和演算を使用して、aの16個の符号付き8ビット整数を、bの16個の符号付き8ビット整数に加算します。
    r0 := SignedSaturate(a0 + b0)
    r1 := SignedSaturate(a1 + b1)
    ...
    r15 := SignedSaturate(a15 + b15)
__m128i _mm_adds_epi16(__m128i a, __m128i b)

```

飽和演算を使用して、aの8個の符号付き16ビット整数を、bの8個の符号付き16ビット整数に加算します。

```
r0 := SignedSaturate(a0 + b0)
r1 := SignedSaturate(a1 + b1)
```

...

```
r7 := SignedSaturate(a7 + b7)
```

\_\_m128i \_mm\_adds\_epu8(\_\_m128i a, \_\_m128i b)

飽和演算を使用して、aの16個の符号なし8ビット整数を、bの16個の符号なし8ビット整数に加算します。

```
r0 := UnsignedSaturate(a0 + b0)
r1 := UnsignedSaturate(a1 + b1)
```

...

```
r15 := UnsignedSaturate(a15 + b15)
```

\_\_m128i \_mm\_adds\_epu16(\_\_m128i a, \_\_m128i b)

飽和演算を使用して、aの8個の符号なし16ビット整数を、bの8個の符号なし16ビット整数に加算します。

```
r0 := UnsignedSaturate(a0 + b0)
r1 := UnsignedSaturate(a1 + b1)
```

...

```
r15 := UnsignedSaturate(a7 + b7)
```

\_\_m128i \_mm\_avg\_epu8(\_\_m128i a, \_\_m128i b)

aの16個の符号なし8ビット整数とbの16個の符号なし8ビット整数について、対応する値の平均値を計算し、その結果を丸めます。

```
r0 := (a0 + b0) / 2
r1 := (a1 + b1) / 2
```

...

```
r15 := (a15 + b15) / 2
```

\_\_m128i \_mm\_avg\_epu16(\_\_m128i a, \_\_m128i b)

aの8個の符号なし16ビット整数とbの8個の符号なし16ビット整数について、対応する値の平均値を計算し、その結果を丸めます。

```
r0 := (a0 + b0) / 2
r1 := (a1 + b1) / 2
```

...

```
r7 := (a7 + b7) / 2
```

\_\_m128i \_mm\_madd\_epi16(\_\_m128i a, \_\_m128i b)

aの8個の符号付き16ビット整数に、bの8個の符号付き16ビット整数を掛けます。得られた符号付き32ビット整数を2つずつ加算して、4個の符号付き32ビット整数としてパックします。

```
r0 := (a0 * b0) + (a1 * b1)
r1 := (a2 * b2) + (a3 * b3)
r2 := (a4 * b4) + (a5 * b5)
r3 := (a6 * b6) + (a7 * b7)
```

```

__m128i __mm_max_epi16(__m128i a, __m128i b)
    aの8個の符号付き16ビット整数とbの8個の符号付き16ビット整数について、それぞれの
    値のペアの最大値を計算します。
    r0 := max(a0, b0)
    r1 := max(a1, b1)
    ...
    r7 := max(a7, b7)
__m128i __mm_max_epu8(__m128i a, __m128i b)
    aの16個の符号なし8ビット整数とbの16個の符号なし8ビット整数について、それぞれの
    値のペアの最大値を計算します。
    r0 := max(a0, b0)
    r1 := max(a1, b1)
    ...
    r15 := max(a15, b15)
__m128i __mm_min_epi16(__m128i a, __m128i b)
    aの8個の符号付き16ビット整数とbの8個の符号付き16ビット整数について、それぞれの
    値のペアの最小値を計算します。
    r0 := min(a0, b0)
    r1 := min(a1, b1)
    ...
    r7 := min(a7, b7)
__m128i __mm_min_epu8(__m128i a, __m128i b)
    aの16個の符号なし8ビット整数とbの16個の符号なし8ビット整数について、それぞれの
    値のペアの最小値を計算します。
    r0 := min(a0, b0)
    r1 := min(a1, b1)
    ...
    r15 := min(a15, b15)
__m128i __mm_mulhi_epi16(__m128i a, __m128i b)
    aの8個の符号付き16ビット整数に、bの8個の符号付き16ビット整数を掛けます。得ら
    れた8個の符号付き32ビット整数の上位16ビットをパックします。
    r0 := (a0 * b0)[31:16]
    r1 := (a1 * b1)[31:16]
    ...
    r7 := (a7 * b7)[31:16]
__m128i __mm_mulhi_epu16(__m128i a, __m128i b)
    aの8個の符号なし16ビット整数に、bの8個の符号なし16ビット整数を掛けます。得られ
    た8個の符号なし32ビット整数の上位16ビットをパックします。
    r0 := (a0 * b0)[31:16]
    r1 := (a1 * b1)[31:16]
    ...
    r7 := (a7 * b7)[31:16]

```

```

__m128i mm_mullo_epi16(__m128i a, __m128i b)
    aの8個の符号付きまたは符号なし16ビット整数に、bの8個の符号付きまたは符号なし
    16ビット整数を掛けます。得られた8個の符号付きまたは符号なし32ビット整数の下位
    16ビットをパックします。
    r0 := (a0 * b0) [15:0]
    r1 := (a1 * b1) [15:0]
    ...
    r7 := (a7 * b7) [15:0]
__m64 mm_mul_su32(__m64 a, __m64 b)
    aの下位の32ビット整数に、bの下位の32ビット整数を掛けて、64ビット整数の結果を返
    します。
    r := a0 * b0
__m128i mm_mul_epu32(__m128i a, __m128i b)
    aの2個の符号なし32ビット整数に、bの2個の符号なし32ビット整数を掛けます。得られ
    た2個の符号なし64ビット整数をパックします。
    r0 := a0 * b0
    r1 := a2 * b2
__m128i mm_sad_epu8(__m128i a, __m128i b)
    aの16個の符号なし8ビット整数とbの16個の符号なし8ビット整数について、それぞれ
    の差の絶対値を計算します。上位の8つの差と下位の8つの差をそれぞれに合計して、
    得られた2個の符号なし16ビット整数を、結果の上位および下位の64ビット要素の中に
    パックします。
    r0 := abs(a0 - b0) + abs(a1 - b1) + ... + abs(a7 - b7)
    r1 := 0x0 ; r2 := 0x0 ; r3 := 0x0
    r4 := abs(a8 - b8) + abs(a9 - b9) + ... + abs(a15 - b15)
    r5 := 0x0 ; r6 := 0x0 ; r7 := 0x0
__m128i mm_sub_epi8(__m128i a, __m128i b)
    aの16個の符号付きまたは符号なし8ビット整数から、bの16個の符号付きまたは符号
    なし8ビット整数を引きます。
    r0 := a0 - b0
    r1 := a1 - b1
    ...
    r15 := a15 - b15
__m128i mm_sub_epi16(__m128i a, __m128i b)
    aの8個の符号付きまたは符号なし16ビット整数から、bの8個の符号付きまたは符号な
    し16ビット整数を引きます。
    r0 := a0 - b0
    r1 := a1 - b1
    ...
    r7 := a7 - b7
__m128i mm_sub_epi32(__m128i a, __m128i b)

```

aの4個の符号付きまたは符号なし32ビット整数から、bの4個の符号付きまたは符号なし32ビット整数を引きます。

```
r0 := a0 - b0
r1 := a1 - b1
r2 := a2 - b2
r3 := a3 - b3
```

\_\_\_m64 \_\_mm\_sub\_si64 (\_\_m64 a, \_\_m64 b)

aの符号付きまたは符号なし64ビット整数から、bの符号付きまたは符号なし64ビット整数を引きます。

```
r := a - b
```

\_\_\_m128i \_\_mm\_sub\_epi64(\_\_m128i a, \_\_m128i b)

aの2個の符号付きまたは符号なし64ビット整数から、bの2個の符号付きまたは符号なし64ビット整数を引きます。

```
r0 := a0 - b0
r1 := a1 - b1
```

\_\_\_m128i \_\_mm\_subs\_epi8(\_\_m128i a, \_\_m128i b)

飽和演算を使用して、aの16個の符号付き8ビット整数から、bの16個の符号付き8ビット整数を引きます。

```
r0 := SignedSaturate(a0 - b0)
r1 := SignedSaturate(a1 - b1)
```

...

```
r15 := SignedSaturate(a15 - b15)
```

\_\_\_m128i \_\_mm\_subs\_epi16(\_\_m128i a, \_\_m128i b)

飽和演算を使用して、aの8個の符号付き16ビット整数から、bの8個の符号付き16ビット整数を引きます。

```
r0 := SignedSaturate(a0 - b0)
r1 := SignedSaturate(a1 - b1)
```

...

```
r7 := SignedSaturate(a7 - b7)
```

\_\_\_m128i \_\_mm\_subs\_epu8(\_\_m128i a, \_\_m128i b)

飽和演算を使用して、aの16個の符号なし8ビット整数から、bの16個の符号なし8ビット整数を引きます。

```
r0 := UnsignedSaturate(a0 - b0)
r1 := UnsignedSaturate(a1 - b1)
```

...

```
r15 := UnsignedSaturate(a15 - b15)
```

\_\_\_m128i \_\_mm\_subs\_epu16(\_\_m128i a, \_\_m128i b)

飽和演算を使用して、aの8個の符号なし16ビット整数から、bの8個の符号なし16ビット整数を引きます。

```
r0 := UnsignedSaturate(a0 - b0)
r1 := UnsignedSaturate(a1 - b1)
```

```
...
r7 := UnsignedSaturate(a7 - b7)
```

## ストリーミングSIMD拡張命令2 の整数論理演算

次の4つの論理演算組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

```
__m128i _mm_and_si128(__m128i a, __m128i b)
(PAND を使用)a の128ビット値と b の128ビット値について、ビット単位の AND (論理和)を計算します。
```

```
r := a & b
```

```
__m128i _mm_andnot_si128(__m128i a, __m128i b)
(PANDNを使用)a の128ビット値のビット単位の NOT (否定)を実行し、その結果と b の128ビット値について、ビット単位の AND (論理積)を計算します。
```

```
r := (~a) & b
```

```
__m128i _mm_or_si128(__m128i a, __m128i b)
(PORを使用)a の128ビット値と b の128ビット値について、ビット単位の OR (論理和)を計算します。
```

```
r := a | b
```

```
__m128i _mm_xor_si128(__m128i a, __m128i b)
(PXOR を使用)a の128ビット値と b の128ビット値について、ビット単位の XOR (論理和)を計算します。
```

```
r := a ^ b
```

## ストリーミングSIMD拡張命令2 の整数シフト操作

ストリーミングSIMD拡張命令2 に対応したシフト演算組込み関数とその説明を下表に示します。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダファイル `emmintrin.h` 内にあります。

組込み関数	シフトの方向	シフトの種類	対応する命令
<code>_mm_slli_si128</code>	左	論理	PSLLDQ
<code>_mm_slli_epi16</code>	左	論理	PSLLW
<code>_mm_sll_epi16</code>	左	論理	PSLLW
<code>_mm_slli_epi32</code>	左	論理	PSLLD
<code>_mm_sll_epi32</code>	左	論理	PSLLD
<code>_mm_slli_epi64</code>	左	論理	PSLLQ
<code>_mm_sll_epi64</code>	左	論理	PSLLQ

mm_srai_epi16	右	算術	PSRAW
mm_sra_epi16	右	算術	PSRAW
mm_srai_epi32	右	算術	PSRAD
mm_sra_epi32	右	算術	PSRAD
mm_srli_si128	右	論理	PSRLDQ
mm_srli_epi16	右	論理	PSRLW
mm_srl_epi16	右	論理	PSRLW
mm_srli_epi32	右	論理	PSRLD
mm_srl_epi32	右	論理	PSRLD
mm_srli_epi64	右	論理	PSRLQ
mm_srl_epi64	右	論理	PSRLQ

```

__m128i _mm_slli_si128(__m128i a, int imm)
    a の128ビット値を imm バイトだけ左にシフトし、下位ビットを0で埋めます。imm は
    即値でなければなりません。
    r := a << (imm * 8)
__m128i _mm_slli_epi16(__m128i a, int count)
    aに含まれている8個の16ビット整数(符号付き/符号なし)を左にcountビットだけシフトし、
    ゼロをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
    ...
    r7 := a7 << count
__m128i _mm_sll_epi16(__m128i a, __m128i count)
    aに含まれている8個の16ビット整数(符号付き/符号なし)を左にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
    ...
    r7 := a7 << count
__m128i _mm_slli_epi32(__m128i a, int count)
    aに含まれている4個の32ビット整数(符号付き/符号なし)を左にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
    r2 := a2 << count
    r3 := a3 << count
__m128i _mm_sll_epi32(__m128i a, __m128i count)
    aに含まれている4個の32ビット整数(符号付き/符号なし)を左にcountビットだけシフト
    し、ゼロをシフトインします。

```

```

    r0 := a0 << count
    r1 := a1 << count
    r2 := a2 << count
    r3 := a3 << count
__m128i _mm_slli_epi64(__m128i a, int count)
    aに含まれている2個の64ビット整数(符号付き/符号なし)を左にcountビットだけシフトし、ゼロをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
__m128i _mm_sll_epi64(__m128i a, __m128i count)
    aに含まれている2個の64ビット整数(符号付き/符号なし)を左にcountビットだけシフトし、ゼロをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
__m128i _mm_srai_epi16(__m128i a, int count)
    aに含まれている8個の符号付き16ビット整数を右にcountビットだけシフトし、その符号ビットをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
    ...
    r7 := a7 << count
__m128i _mm_sra_epi16(__m128i a, __m128i count)
    aに含まれている8個の符号付き16ビット整数を右にcountビットだけシフトし、その符号ビットをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
    ...
    r7 := a7 << count
__m128i _mm_srai_epi32(__m128i a, int count)
    aに含まれている4個の符号付き32ビット整数を右にcountビットだけシフトし、その符号ビットをシフトインします。
    r0 := a0 << count
    r1 := a1 << count
    r2 := a2 << count
    r3 := a3 << count
__m128i _mm_sra_epi32(__m128i a, __m128i count)
    aに含まれている4個の符号付き32ビット整数を右にcountビットだけシフトし、その符号ビットをシフトインします。
    r0 := a0 << count

```



```

    r1 := a1 << count
    r2 := a2 << count
    r3 := a3 << count
__m128i _mm_srli_si128(__m128i a, int imm)
    aに含まれている128ビット値を右にimmバイトだけシフトし、ゼロをシフトインします。
    imm は即値でなければなりません。
    r := srl(a, imm * 8)
__m128i _mm_srli_epi16(__m128i a, int count)
    aに含まれている8個の16ビット整数(符号付き/符号なし)を右にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := srl(a0, count)
    r1 := srl(a1, count)
    r7 := srl(a7, count)
__m128i _mm_srl_epi16(__m128i a, __m128i count)
    aに含まれている8個の16ビット整数(符号付き/符号なし)を右にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := srl(a0, count)
    r1 := srl(a1, count)
    r7 := srl(a7, count)
__m128i _mm_srli_epi32(__m128i a, int count)
    aに含まれている4個の32ビット整数(符号付き/符号なし)を右にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := srl(a0, count)
    r1 := srl(a1, count)
    r2 := srl(a2, count)
    r3 := srl(a3, count)
__m128i _mm_srl_epi32(__m128i a, __m128i count)
    aに含まれている4個の32ビット整数(符号付き/符号なし)を右にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := srl(a0, count)
    r1 := srl(a1, count)
    r2 := srl(a2, count)
    r3 := srl(a3, count)
__m128i _mm_srli_epi64(__m128i a, int count)
    aに含まれている2個の64ビット整数(符号付き/符号なし)を右にcountビットだけシフト
    し、ゼロをシフトインします。
    r0 := srl(a0, count)
    r1 := srl(a1, count)
__m128i _mm_srl_epi64(__m128i a, __m128i count)
    aに含まれている2個の64ビット整数(符号付き/符号なし)を右にcountビットだけシフト
    し、ゼロをシフトインします。

```

```

r0 := srl(a0, count)
r1 := srl(a1, count)

```

## ストリーミングSIMD拡張命令2 の整数比較操作

ストリーミングSIMD拡張命令2 に対応した比較演算組込み関数とその説明を下表に示します。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

組込み関数名	命令	比較条件	要素数	要素のサイズ
<code>_mm_cmpeq_epi8</code>	PCMPEQB	=	16	8
<code>_mm_cmpeq_epi16</code>	PCMPEQW	=	8	16
<code>_mm_cmpeq_epi32</code>	PCMPEQD	=	4	32
<code>_mm_cmpgt_epi8</code>	PCMPGTB	より大きい	16	8
<code>_mm_cmpgt_epi16</code>	PCMPGTW	より大きい	8	16
<code>_mm_cmpgt_epi32</code>	PCMPGTD	より大きい	4	32
<code>_mm_cmpltr_epi8</code>	PCMPGTB	<	16	8
<code>_mm_cmpltr_epi16</code>	PCMPGTWr	<	8	16
<code>_mm_cmpltr_epi32</code>	PCMPGTD	<	4	32

```
__m128i _mm_cmpeq_epi8(__m128i a, __m128i b)
```

a の16個の符号付きまたは符号なし8ビット整数と、b の16個の符号付きまたは符号なし8ビット整数が等しいかどうか比較します。

```

r0 := (a0 == b0) ? 0xff : 0x0
r1 := (a1 == b1) ? 0xff : 0x0
...

```

```

r15 := (a15 == b15) ? 0xff : 0x0

```

```
__m128i _mm_cmpeq_epi16(__m128i a, __m128i b)
```

a の8個の符号付きまたは符号なし16ビット整数と、b の8個の符号付きまたは符号なし16ビット整数が等しいかどうか比較します。

```

r0 := (a0 == b0) ? 0xffff : 0x0

```

```

r1 := (a1 == b1) ? 0xffff : 0x0
...
r7 := (a7 == b7) ? 0xffff : 0x0
__m128i _mm_cmpeq_epi32(__m128i a, __m128i b)
a の4個の符号付きまたは符号なし32ビット整数と、b の4個の符号付きまたは符号なし32ビット整数が等しいかどうか比較します。
r0 := (a0 == b0) ? 0xffffffff : 0x0
r1 := (a1 == b1) ? 0xffffffff : 0x0
r2 := (a2 == b2) ? 0xffffffff : 0x0
r3 := (a3 == b3) ? 0xffffffff : 0x0
__m128i _mm_cmpgt_epi8(__m128i a, __m128i b)
a の16個の符号付き8ビット整数が、b の16個の符号付き8ビット整数より大きいかどうか比較します。
r0 := (a0 > b0) ? 0xff : 0x0
r1 := (a1 > b1) ? 0xff : 0x0
...
r15 := (a15 > b15) ? 0xff : 0x0
__m128i _mm_cmpgt_epi16(__m128i a, __m128i b)
a の8個の符号付き16ビット整数が、b の8個の符号付き16ビット整数より大きいかどうか比較します。
r0 := (a0 > b0) ? 0xffff : 0x0
r1 := (a1 > b1) ? 0xffff : 0x0
...
r7 := (a7 > b7) ? 0xffff : 0x0
__m128i _mm_cmpgt_epi32(__m128i a, __m128i b)
a の4個の符号付き32ビット整数が、b の4個の符号付き32ビット整数より大きいかどうか比較します。
r0 := (a0 > b0) ? 0xffff : 0x0
r1 := (a1 > b1) ? 0xffff : 0x0
r2 := (a2 > b2) ? 0xffff : 0x0
r3 := (a3 > b3) ? 0xffff : 0x0
__m128i _mm_cmplt_epi8(__m128i a, __m128i b)
a の16個の符号付き8ビット整数が、b の16個の符号付き8ビット整数より小さいかどうか比較します。
r0 := (a0 < b0) ? 0xff : 0x0
r1 := (a1 < b1) ? 0xff : 0x0
...
r15 := (a15 < b15) ? 0xff : 0x0
__m128i _mm_cmplt_epi16(__m128i a, __m128i b)
a の8個の符号付き16ビット整数が、b の8個の符号付き16ビット整数より小さいかどうか

```

か比較します。

```
r0 := (a0 < b0) ? 0xffff : 0x0
r1 := (a1 < b1) ? 0xffff : 0x0
...
r7 := (a7 < b7) ? 0xffff : 0x0
```

\_\_m128i \_mm\_cmplt\_epi32(\_\_m128i a, \_\_m128i b)

a の4個の符号付き32ビット整数が、b の4個の符号付き32ビット整数より小さいかどうか比較します。

```
r0 := (a0 < b0) ? 0xffff : 0x0
r1 := (a1 < b1) ? 0xffff : 0x0
r2 := (a2 < b2) ? 0xffff : 0x0
r3 := (a3 < b3) ? 0xffff : 0x0
```

## ストリーミングSIMD拡張命令2 の変換操作

次の2つの変換組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

\_\_m128i \_mm\_cvtsi32\_si128(int a)

(MOVD を使用)32ビット整数a を \_\_m128i オブジェクトの最下位32ビットに移動し、a の符号ビットを \_\_m128i オブジェクトの上位96ビットにコピーします。

```
r0 := a
r1 := 0x0 ; r2 := 0x0 ; r3 := 0x0
```

int \_mm\_cvtsi128\_si32(\_\_m128i a)

(MOVD を使用) a の最下位の32ビットを、32ビット整数に移動します。

```
r := a0
```

\_\_m128 \_mm\_cvtepi32\_ps(\_\_m128i a)

aの4個の符号付き32ビット整数値を単精度浮動小数点値に変換します。

```
r0 := (float) a0
r1 := (float) a1
r2 := (float) a2
r3 := (float) a3
```

\_\_m128i \_mm\_cvtps\_epi32(\_\_m128 a)

aの4個の単精度浮動小数点値を符号付き32ビット整数値に変換します。

```
r0 := (int) a0
r1 := (int) a1
r2 := (int) a2
r3 := (int) a3
```

\_\_m128i \_mm\_cvttps\_epi32(\_\_m128 a)

切り捨てを使用して、aの4個の単精度浮動小数点値を符号付き 32ビット整数値に変換します。

```
r0 := (int) a0
r1 := (int) a1
r2 := (int) a2
r3 := (int) a3
```

## シャッフルを行うマクロ関数

ストリーミングSIMD拡張命令2 には、シャッフル操作を記述する定数を生成するマクロ関数を用意しています。このマクロは、2個の小さな整数( 0~1 の範囲)を組み合わせて、SHUFPSD 命令が使用する2ビット即値を生成します。次の例を参照してください。

シャッフル関数のマクロ

```
_MM_SHUFFLE2(x, y)
expands to the value of
(x<<1) | y
```

2個の整数は、第1入力オペランドと第2入力オペランドからそれぞれの2ワードを取り出して結果のワードに入れるかを選択するセレクトアとして機能します。

シャッフル関数のマクロの元のワードと結果のワード

```
; m1 = 127 [ a | b ] 0
; m2 = 127 [ c | d ] 0
m3 = _mm_shuffle_pd(m1, m2, _MM_SHUFFLE2(1,0))
; m3 = 127 [ c | b ] 0
```

## ストリーミングSIMD拡張命令2 のキャッシュ操作

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h` 内にあります。

```
void _mm_stream_pd(double *p, __m128d a)
```

(MOVNTPD を使用) a のデータを、キャッシュを介さずに、アドレス p にストアします。アドレス p は、16バイトにアライメントが合っていないとなりません。アドレス p を含むキャッシュ・ラインが既にキャッシュ内にある場合、キャッシュは更新されます。

```
p[0] := a0
p[1] := a1
```

```
void _mm_stream_si128(__m128i *p, __m128i a)
```

a のデータを、キャッシュを介さずに、アドレス p にストアします。アドレス p を含むキャッシュ・ラインが既にキャッシュ内にある場合、キャッシュは更新されます。アドレス p は、16バイトにアライメントが合っていないとなりません。

```

    *p := a
void _mm_stream_si32(int *p, int a)
    a のデータを、キャッシュを介さずに、アドレス p にストアします。アドレス p を含むキ
    ャッシュ・ラインが既にキャッシュ内にある場合、キャッシュは更新されます。
    *p := a
void _mm_clflush(void const*p)
    コヒーレンシ・ドメイン内のすべてのキャッシュから、p を含むキャッシュ・ラインをフラッ
    シュし、無効化します。
void _mm_lfence(void)
    プログラムの順序でロードフェンス命令に先行するすべてのロード命令が、フェンスに続
    くロード命令より前に、グローバルにアクセス可能になるのを保証します。
void _mm_mfence(void)
    プログラムの順序でメモリフェンス命令に先行するすべてのメモリアクセス命令が、フェ
    ンスに続くメモリアクセス命令より前に、グローバルにアクセス可能になるのを保証しま
    す。
void _mm_pause(void)
    プロセッサに固有の時間の間、次の命令の実行を遅らせます。この命令を実行しても、
    アーキテクチャ上の状態は変化しません。この組み込み関数を使用すると、パフォーマンス
    が大きく向上します。この組み込み関数については、以降で詳しく説明します。

```

## PAUSE組み込み関数

PAUSE 組み込み関数は、ダイナミック・エグゼキューション(特に、アウトオブオーダー実行)をサポートするプロセッサ上で、spin-wait ループに使用します。spin-wait ループ内でPAUSEを使用すると、ロックの解放を検出するコードの処理速度が向上します。動的スケジューリングに PAUSE 命令を使用すると、スピンループの終了時のペナルティが軽減されます。

PAUSE命令を使用したループの例

```
spin_loop:pause
```

```
cmp eax, A
```

```
jne spin_loop
```

↑上の例では、メモリ・ロケーション A がレジスタ eax の値と一致するまで、プログラムはスピンします。次のコード・シーケンスは、test-and-test-and-set 操作を示しています。この例では、ロックの取得に失敗した場合にのみ、スピンが発生します。

```
get_lock: mov eax, 1
```

```
xchg eax, A ; Try to get lock
```

```
cmp eax, 0 ; Test if successful
```

```
jne spin_loop
```

```
<critical_section code>
```

```
mov A, 0 ; Release lock
```

```
jmp continue
```

```
spin_loop: pause ; Spin-loop hint
cmp 0, A ; Check lock availability
jne spin_loop
jmp get_lock
continue: <other code>
```

この例では、ロックの取得に成功すると予測して、最初の条件分岐は分岐せずに、そのままクリティカル・セクションの処理に移ります。すべての spin-wait ループに、PAUSE 命令を使用するのを強くお勧めします。PAUSE は、既存のすべての IA-32 プロセッサで使用可能なため、プロセッサのタイプをテストする(CPUID テスト)必要はありません。すべての従来のプロセッサは、PAUSE を NOP として実行しますが、PAUSE をヒントとして使用するプロセッサでは、パフォーマンスが大きく向上する可能性があります。

## ストリーミングSIMD拡張命令2 のその他の操作

次の表に、ストリーミング SIMD 拡張命令 2のその他の組込み関数のリストを示します。表の後に、各組込み関数の説明を示します。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

組込み関数	対応する命令	操作
<code>mm_packs_epi16</code>	PACKSSWB	パックド飽和
<code>mm_packs_epi32</code>	PACKSSDW	パックド飽和
<code>mm_packus_epi16</code>	PACKUSWB	パックド飽和
<code>mm_extract_epi16</code>	PEXTRW	抽出
<code>mm_insert_epi16</code>	PINSRW	挿入
<code>mm_movemask_epi8</code>	PMOVBMSKB	マスクの作成
<code>mm_shuffle_epi32</code>	PSHUFD	シャッフル
<code>mm_shufflehi_epi16</code>	PSHUFW	シャッフル
<code>mm_shufflelo_epi16</code>	PSHUFLW	シャッフル
<code>mm_unpackhi_epi8</code>	PUNPCKHBW	インターリーブ
<code>mm_unpackhi_epi16</code>	PUNPCKHWD	インターリーブ
<code>mm_unpackhi_epi32</code>	PUNPCKHDQ	インターリーブ
<code>mm_unpackhi_epi64</code>	PUNPCKHQDQ	インターリーブ
<code>mm_unpacklo_epi8</code>	PUNPCKLBW	インターリーブ
<code>mm_unpacklo_epi16</code>	PUNPCKLWD	インターリーブ
<code>mm_unpacklo_epi32</code>	PUNPCKLDQ	インターリーブ
<code>mm_unpacklo_epi64</code>	PUNPCKLQDQ	インターリーブ
<code>mm_movepi64_pi64</code>	MOVDQ2Q	移動
<code>_mm_movelpi64_pi64</code>	MOVQ2DQ	移動

<code>__mm_move_epi64</code>	<code>MOVQ</code>	移動
------------------------------	-------------------	----

```

__m128i __mm_packs_epi16(__m128i a, __m128i b)
    aとbの16個の符号付き16ビット整数を8ビット整数にパックして、飽和処理します。
    r0 := SignedSaturate(a0)
    r1 := SignedSaturate(a1)
    ...
    r7 := SignedSaturate(a7)
    r8 := SignedSaturate(b0)
    r9 := SignedSaturate(b1)
    ...
    r15 := SignedSaturate(b7)
__m128i __mm_packs_epi32(__m128i a, __m128i b)
    aとbの8個の符号付き32ビット整数を符号付き16ビット整数にパックして、飽和処理します。
    r0 := SignedSaturate(a0)
    r1 := SignedSaturate(a1)
    r2 := SignedSaturate(a2)
    r3 := SignedSaturate(a3)
    r4 := SignedSaturate(b0)
    r5 := SignedSaturate(b1)
    r6 := SignedSaturate(b2)
    r7 := SignedSaturate(b3)
__m128i __mm_packus_epi16(__m128i a, __m128i b)
    aとbの16個の符号付き16ビット整数を符号なし8ビット整数にパックして、飽和処理します。
    r0 := UnsignedSaturate(a0)
    r1 := UnsignedSaturate(a1)
    ...
    r7 := UnsignedSaturate(a7)
    r8 := UnsignedSaturate(b0)
    r9 := UnsignedSaturate(b1)
    ...
    r15 := UnsignedSaturate(b7)
int __mm_extract_epi16(__m128i a, int imm)
    選択された符号付きまたは符号なし16ビット整数をaから抽出して、0で拡張します。セクタimmは、即値でなければなりません。
    r := (imm == 0) ? a0 :
        ( (imm == 1) ? a1 :
            ...
            (imm == 7) ? a7 )
__m128i __mm_insert_epi16(__m128i a, int b, int imm)
    bの最下位16ビットを、aの選択された16ビット整数に挿入します。セクタimmは、即

```



値でなければなりません。

```
r0 := (imm == 0) ? b : a0;  
r1 := (imm == 1) ? b : a1;
```

```
...  
r7 := (imm == 7) ? b : a7;
```

```
int __mm_movemask_epi8(__m128i a)
```

aの16個の符号付きまたは符号なし8ビット整数の最上位ビットを使用して16ビットマスクを作成し、上位ビットを0で拡張します。

```
r := a15[7] << 15 |  
a14[7] << 14 |  
r := a1[7] << 1 |
```

```
__m128i __mm_shuffle_epi32(__m128i a, int imm)
```

immの指定に従って、aの4個の符号付きまたは符号なし32ビット整数をシャッフルします。シャッフル値immは、即値でなければなりません。シャッフルのセマンティクスについては、この節の最後の「シャッフルを行うマクロ関数」を参照してください。

```
__m128i __mm_shufflehi_epi16(__m128i a, int imm)
```

immの指定に従って、aの上位4個の符号付きまたは符号なし16ビット整数をシャッフルします。シャッフル値immは、即値でなければなりません。シャッフルのセマンティクスについては、この節の最後の「シャッフルを行うマクロ関数」を参照してください。

```
__m128i __mm_shufflelo_epi16(__m128i a, int imm)
```

immの指定に従って、aの下位4個の符号付きまたは符号なし16ビット整数をシャッフルします。シャッフル値immは、即値でなければなりません。シャッフルのセマンティクスについては、この節の最後の「シャッフルを行うマクロ関数」を参照してください。

```
__m128i __mm_unpackhi_epi8(__m128i a, __m128i b)
```

aの上位8個の符号付きまたは符号なし8ビット整数と、bの上位8個の符号付きまたは符号なし8ビット整数をインターリーブ(交互に配置)します。

```
r0 := a8 ; r1 := b8  
r2 := a9 ; r3 := b9
```

```
...  
r14 := a15 ; r15 := b15
```

```
__m128i __mm_unpackhi_epi16(__m128i a, __m128i b)
```

aの上位4個の符号付きまたは符号なし16ビット整数と、bの上位4個の符号付きまたは符号なし16ビット整数をインターリーブします。

```
r0 := a4 ; r1 := b4  
r2 := a5 ; r3 := b5  
r4 := a6 ; r5 := b6  
r6 := a7 ; r7 := b7
```

```
__m128i __mm_unpackhi_epi32(__m128i a, __m128i b)
```

aの上位2個の符号付きまたは符号なし32ビット整数と、bの上位2個の符号付きまたは符号なし32ビット整数をインターリーブします。

```
r0 := a2 ; r1 := b2
```

```

    r2 := a3 ; r3 := b3
__m128i __mm_unpackhi_epi64(__m128i a, __m128i b)
    aの上位の符号付きまたは符号なし64ビット整数と、bの上位の符号付きまたは符号なし64ビット整数をインターリーブします。
    r0 := a1 ; r1 := b1
__m128i __mm_unpacklo_epi8(__m128i a, __m128i b)
    aの下位8個の符号付きまたは符号なし8ビット整数と、bの下位8個の符号付きまたは符号なし8ビット整数をインターリーブします。
    r0 := a0 ; r1 := b0
    r2 := a1 ; r3 := b1
    ...
    r14 := a7 ; r15 := b7
__m128i __mm_unpacklo_epi16(__m128i a, __m128i b)
    aの下位4個の符号付きまたは符号なし16ビット整数と、bの下位4個の符号付きまたは符号なし16ビット整数をインターリーブします。
    r0 := a0 ; r1 := b0
    r2 := a1 ; r3 := b1
    r4 := a2 ; r5 := b2
    r6 := a3 ; r7 := b3
__m128i __mm_unpacklo_epi32(__m128i a, __m128i b)
    aの下位2個の符号付きまたは符号なし32ビット整数と、bの下位2個の符号付きまたは符号なし32ビット整数をインターリーブします。
    r0 := a0 ; r1 := b0
    r2 := a1 ; r3 := b1
__m128i __mm_unpacklo_epi64(__m128i a, __m128i b)
    aの下位の符号付きまたは符号なし64ビット整数と、bの下位の符号付きまたは符号なし64ビット整数をインターリーブします。
    r0 := a0 ; r1 := b0
__m64 __mm_movepi64_pi64(__m128i a)
    aの下位64ビットを、__m64型として返します。
    r0 := a0 ;
__128i __mm_movpi64_pi64(__m64 a)
    aの64ビットを結果の下位64ビットに移動し、上位ビットを0に設定します。
    r0 := a0 ; r1 := 0X0 ;
__128i __mm_move_epi64(__128i a)
    aの下位64ビットを結果の下位64ビットに移動し、上位ビットを0に設定します。
    r0 := a0 ; r1 := 0X0 ;

```

## 整数のメモリ操作と初期化操作

### ストリーミングSIMD拡張命令2 の整数メモリ操作と初期化操作

整数のロード、設定、およびストア組込み関数とそれに対応する命令によって、ストリーミングSIMD拡張命令2 のメモリ操作と初期化操作を実行できます。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

- ロード操作
- 設定操作
- ストア操作

### ストリーミングSIMD拡張命令2 の整数ロード演算

次のロード操作組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

```
__m128i _mm_load_si128(__m128i const*p)
    (MOVDQA を使用) 128ビット値をロードします。アドレス p は、16バイトにアライメント
    が合っていなければなりません。
    r := *p
__m128i _mm_loadu_si128(__m128i const*p)
    (MOVDQU を使用) 128ビット値をロードします。アドレス p は、16バイトにアライメント
    が合っていなくてもかまいません。
    r := *p
__m128i _mm_loadl_epi64(__m128i const*p)
    (MOVQ を使用)p で指定された値の下位64ビットを結果の下位64ビットにロードし、結
    果の上位64ビットは0に設定します。
    r0 := *p[63:0]
    r1 := 0x0
```

### ストリーミングSIMD拡張命令2 の整数設定操作

次の設定操作組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2の組込み関数のプロトタイプは、ヘッダ・ファイル `emmintrin.h`内にあります。

```
__m128i _mm_set_epi64(__m64 q1, __m64 q0)
    2個の64ビット整数値を設定します。
    r0 := q0
    r1 := q1
```

```

__m128i _mm_set_epi32(int i3, int i2, int i1, int i0)
    4個の符号付き32ビット整数値を設定します。
    r0 := i0
    r1 := i1
    r2 := i2
    r3 := i3
__m128i _mm_set_epi16(short w7, short w6, short w5, short
w4, short w3, short w2, short w1, short w0)
    8個の符号付き16ビット整数値を設定します。
    r0 := w0
    r1 := w1
    ...
    r7 := w7
__m128i _mm_set_epi8(char b15, char b14, char b13, char b12,
char b11, char b10, char b9, char b8, char b7, char b6, char
b5, char b4, char b3, char b2, char b1, char b0)
    16個の符号付き8ビット整数値を設定します。
    r0 := b0
    r1 := b1
    ...
    r15 := b15
__m128i _mm_set1_epi64(__m64 q)
    2個の64ビット整数値を q に設定します。
    r0 := q
    r1 := q
__m128i _mm_set1_epi32(int i)
    4個の符号付き32ビット整数値を i に設定します。
    r0 := i
    r1 := i
    r2 := i
    r3 := i
__m128i _mm_set1_epi16(short w)
    8個の符号付き16ビット整数値を w に設定します。
    r0 := w
    r1 := w
    ...
    r7 := w
__m128i _mm_set1_epi8(char b)
    16個の符号付き8ビット整数値を b に設定します。
    r0 := b
    r1 := b
    ...

```

```

    r15 := b
__m128i __mm_setr_epi64(__m64 q0, __m64 q1)
    2個の64ビット整数値を逆順で設定します。
    r0 := q0
    r1 := q1
__m128i __mm_setr_epi32(int i0, int i1, int i2, int i3)
    4個の符号付き32ビット整数値を逆順で設定します。
    r0 := i0
    r1 := i1
    r2 := i2
    r3 := i3
__m128i __mm_setr_epi16(short w0, short w1, short w2, short
w3, short w4, short w5, short w6, short w7)
    8個の符号付き16ビット整数値を逆順で設定します。
    r0 := w0
    r1 := w1
    ...
    r7 := w7
__m128i __mm_setr_epi8(char b15, char b14, char b13, char
b12, char b11, char b10, char b9, char b8, char b7, char
b6, char b5, char b4, char b3, char b2, char b1, char b0)
    16個の符号付き8ビット整数値を逆順で設定します。
    r0 := b0
    r1 := b1
    ...
    r15 := b15
__m128i __mm_setzero_si128()
    128ビット値を0に設定します。
    r := 0x0

```

## ストリーミングSIMD拡張命令2 の整数ストア操作

次のストア操作組込み関数とそれに対応する命令は、ストリーミングSIMD拡張命令2 をサポートするプロセッサ上で有効です。

ストリーミングSIMD拡張命令2 組込み関数のプロトタイプは、ヘッダファイル `emmintrin.h` 内にあります。

```

void __mm_store_si128(__m128i *p, __m128i b)
    (MOVDQA を使用) 128ビット値をストアします。アドレス p は、16バイトにアライメント
    が合っていないとなりません。
    *p := a
void __mm_storeu_si128(__m128i *p, __m128i b)

```

(MOVDQU を使用) 128ビット値をストアします。アドレス p は、16バイトにアライメントが合っていないかまいません。

```
*p := a
```

```
void _mm_maskmoveu_si128(__m128i d, __m128i n, char *p)
```

(MASKMOVDQU を使用) d のバイト要素を、条件付きでアドレス p にストアします。セクタ n の各バイトの最上位ビットによって、それに対応する d の各バイトがストアされるかどうかが決まります。アドレス p は、16バイトにアライメントが合っていないかまいません。

```
if (n0[7]) p[0] := d0
if (n1[7]) p[1] := d1
```

```
...
```

```
if (n15[7]) p[15] := d15
```

```
void _mm_storel_epi64(__m128i *p, __m128i q)
```

(MOVQ を使用) p で指定された値の下位64ビットをストアします。

```
*p[63:0] := a0
```

## Itanium(R) 命令の組込み関数

### Itanium® 命令の組込み関数の概要

この節では、Itanium® 命令のネイティブ組込み関数を一覧表示し説明します。これらの組込み関数は、IA-32 アーキテクチャ上では使用できません。Itanium 命令の組込み関数によって、プログラマは、C および C++ 言語の標準的な構文では生成できない Itanium 命令を利用できます。

Itanium アーキテクチャ用の組込み関数のプロトタイプは、ヘッダファイル `ia64intrin.h` 内にあります。



注

Itanium ベース・アプリケーションのインテル® C++ コンパイラは、コンパイラの最適化を妨げず、命令スケジューリングに影響を与えないインライン・アセンブリと同等の機能を持つ組込み関数を提供します。

### Itanium® 命令のネイティブ組込み関数

Itanium® アーキテクチャ用の組込み関数のプロトタイプは、ヘッダファイル `ia64intrin.h` 内にあります。

#### 整数演算

組込み関数	対応する命令
<code>__int64 _m64_dep_mr(__int64 r, __int64 s, const int pos, const int len)</code>	dep (Deposit)
<code>__int64 m64_dep_mi(const int v, __int64 s, const int p, const int len)</code>	dep (Deposit)
<code>__int64 _m64_dep_zr(__int64 s, const int pos, const int len)</code>	dep.z (Deposit)
<code>__int64 _m64_dep_zi(const int v, const int pos, const int len)</code>	dep.z (Deposit)
<code>__int64 m64_extr(__int64 r, const int pos, const int len)</code>	extr (Extract)
<code>__int64 m64_extru(__int64 r, const int pos, const int len)</code>	extr.u (Extract)
<code>__int64 m64_xmal(__int64 a, __int64 b, __int64 c)</code>	xma.l (Fixed-point multiply add. 128ビットの下位64ビットを使用します。結果は符

	号付きです。)
<code>int64 m64 xmalu( int64 a, __int64 b, __int64 c)</code>	<code>xma.lu</code> (Fixed-point multiply add. 128 ビットの下位64ビットを使用します。結果は符号なしです。)
<code>int64 m64 xmah( int64 a, __int64 b, __int64 c)</code>	<code>xma.hl</code> (Fixed-point multiply add. 128 ビットの上位64ビットを使用します。結果は符号付きです。)
<code>int64 m64 xmahu( int64 a, __int64 b, __int64 c)</code>	<code>xma.hu</code> (Fixed-point multiply add. 128 ビットの上位64ビットを使用します。結果は符号なしです。)
<code>__int64 m64 popcnt( __int64 a)</code>	<code>popcnt</code> (Population count)
<code>__int64 m64 shladd( __int64 a, const int count, int64 b)</code>	<code>shladd</code> (Shift left and add)
<code>int64 m64 shrp( int64 a, __int64 b, const int count)</code>	<code>shrp</code> (Shift right pair)

## FSR演算

組込み関数	説明
<code>void fsetc(int amask, int omask)</code>	FPSR.sf0 の制御ビットをセットします。 <code>fsetc.sf0 r, r</code> 命令に対応付けられます。これに対応する、制御ビットの読み取り命令はありません。 <code>_mm_getfpsr()</code> を使用してください。
<code>void _fclrf(void)</code>	浮動小数点ステータス・フラグ (FPSR.sf0 の6ビットフラグ) をクリアします。 <code>fclrf.sf0</code> 命令に対応付けられます。

`__int64 _m64_dep_mr(__int64 r, __int64 s, const int pos, const int len)`

右揃えした64ビット値 `r` を、`s` の値の中の任意のビット位置にデポジットし、その結果を返します。デポジットしたビット・フィールドは、ビット位置 `pos` を始点として、`len` で指定したビット数だけ左に(最上位ビットの方向に)延長します。

`__int64 _m64_dep_mi(const int v, __int64 s, const int p, const int len)`

符号で拡張した値 `v` (すべて1またはすべて0)を、`s` の値の中の任意のビット位置にデポジットし、その結果を返します。デポジットしたビット・フィールドは、ビット位置 `p` を始点として、`len` で指定したビット数だけ左に(最上位ビットの方向に)延長します。



`__int64 _m64_dep_zr(__int64 s, const int pos, const int len)`  
 右揃えした64ビット値 `s` を、すべて0の64ビット・フィールド内の任意のビット位置にデポジットし、その結果を返します。デポジットしたビット・フィールドは、ビット位置 `pos` を始点として、`len`で指定したビット数だけ左に(最上位ビットの方向に)延長します。

`__int64 _m64_dep_zi(const int v, const int pos, const int len)`  
 符号で拡張した値 `v` (すべて1またはすべて0)を、すべて0の64ビット・フィールド内の任意のビット位置にデポジットし、その結果を返します。デポジットしたビット・フィールドは、ビット位置 `pos` を始点として、`len`で指定したビット数だけ左に(最上位ビットの方向に)延長します。

`__int64 _m64_extr(__int64 r, const int pos, const int len)`  
 64ビット値 `r` から1つのフィールドを抽出し、右揃えにして符号で拡張した値を返します。抽出したフィールドは、`pos` の位置を始点として、`len` ビットだけ左に延長します。抽出したフィールドの最上位ビットの符号が使用されます。

`__int64 _m64_extru(__int64 r, const int pos, const int len)`  
 64ビット値 `r` から1つのフィールドを抽出し、右揃えにして0で拡張した値を返します。抽出したフィールドは、`pos` の位置を始点として、`len` ビットだけ左に延長します。

`__int64 _m64_xmal(__int64 a, __int64 b, __int64 c)`  
 64ビット値 `a` と `b` を符号付き整数と見なして乗算し、全128ビットの符号付きの結果を求めます。64ビット値 `c` を0で拡張してこの積に加算し、得られた和の最下位64ビットを返します。

`__int64 _m64_xmalu(__int64 a, __int64 b, __int64 c)`  
 64ビット値 `a` と `b` を符号付き整数と見なして乗算し、全128ビットの符号なしの結果を求めます。64ビット値 `c` を0で拡張してこの積に加算し、得られた和の最下位64ビットを返します。

`__int64 _m64_xmah(__int64 a, __int64 b, __int64 c)`  
 64ビット値 `a` と `b` を符号付き整数と見なして乗算し、全128ビットの符号付きの結果を求めます。64ビット値 `c` を0で拡張してこの積に加算し、得られた和の最上位64ビットを返します。

`__int64 _m64_xmahu(__int64 a, __int64 b, __int64 c)`  
 64ビット値 `a` と `b` を符号なし整数と見なして乗算し、全128ビットの符号なしの結果を求めます。64ビット値 `c` を0で拡張してこの積に加算し、得られた和の最上位64ビットを返します。

`__int64 _m64_popcnt(__int64 a)`  
 64ビット整数 `a` の中のビットのうち、値が1のビットをカウントし、得られたビット数を返します。

`__int64 _m64_shladd(__int64 a, const int count, __int64 b)`  
`a` を `count` ビットだけ左にシフトして、`b`に加算します。結果を返します。

`__int64 _m64_shrp(__int64 a, __int64 b, const int count)`  
`a` と `b` を連結して128ビット値を作成し、`count` ビットだけ右にシフトします。結果の

下位64ビットを返します。

## ロックおよびアトミック操作に関連する組み込み関数

Itanium® アーキテクチャ用の組み込み関数のプロトタイプは、ヘッダファイル `ia64intrin.h` 内にあります。

組み込み関数	説明
<code>unsigned __int64 _InterlockedExchange8(volatile unsigned char *Target, unsigned int64 value)</code>	xchg1 命令に対応付けます。第1引数で指定されたアドレスに第2引数の最下位のバイトをアトミックに書きます。
<code>unsigned __int64 _InterlockedCompareExchange8(rel(volatile unsigned char *Destination, unsigned int64 Exchange, unsigned __int64 Comparand)</code>	第1引数で指定されたアドレスの最下位のバイトをアトミックに比較/交換します。適切な設定の <code>cmpxchg1.rel</code> 命令に対応付けます。
<code>unsigned __int64 _InterlockedCompareExchange8(acq(volatile unsigned char *Destination, unsigned int64 Exchange, unsigned __int64 Comparand)</code>	上記と同じですが、acquire セマンティックを使用します。
<code>unsigned __int64 _InterlockedExchange16(volatile unsigned short *Target, unsigned int64 value)</code>	xchg2 命令に対応付けます。第1引数で指定されたアドレスに第2引数の最下位のワードをアトミックに書きます。
<code>unsigned __int64 _InterlockedCompareExchange16(rel(volatile unsigned short *Destination, unsigned int64 Exchange, unsigned __int64 Comparand)</code>	第1引数で指定されたアドレスの最下位のワードをアトミックに比較/交換します。適切な設定の <code>cmpxchg2.rel</code> 命令に対応付けます。
<code>unsigned __int64 _InterlockedCompareExchange16(acq(volatile unsigned short *Destination, unsigned int64 Exchange, unsigned __int64 Comparand)</code>	上記と同じですが、acquire セマンティックを使用します。
<code>int</code>	引数で指定された値を1ずつアトミックに増

<code>_InterlockedIncrement (volatile int *addend)</code>	分します。fetchadd4 命令に対応付けます。
<code>int _InterlockedDecrement (volatile int *addend)</code>	引数で指定された値を1ずつアトミックに減分します。fetchadd4 命令に対応付けます。
<code>int _InterlockedExchange (volatile int *Target, int value)</code>	交換操作をアトミックに実行します。xchg4 命令に対応付けます。
<code>int _InterlockedCompareExchange (volatile int *Destination, int Exchange, int Comparand)</code>	適切な設定のcmpxchg4 命令に対応付けます。第1引数(32ビット・ポインタ)で指定された値をアトミックに比較/交換します。
<code>int _InterlockedExchangeAdd (volatile int *addend, int increment)</code>	比較/交換操作を使用して、加数に対して増分値をアトミックに加算します。cmpxchg4 命令を使用するループに対応付けられ、アトミックな性質が保証されます。
<code>int _InterlockedAdd (volatile int *addend, int increment)</code>	上記と同じですが、元の値ではなく、新しい値を返します。
<code>void * _InterlockedCompareExchangePointer (void * volatile *Destination, void *Exchange, void *Comparand)</code>	exch8 命令に対応付けます。第1引数(すべての引数はポインタ)で指定されたポインタ値をアトミックに比較/交換します。
<code>unsigned __int64 _InterlockedExchangeU (volatile unsigned int *Target, unsigned int64 value)</code>	第1引数で指定された32ビット・データをアトミックに交換します。xchg4 命令に対応付けます。
<code>unsigned __int64 _InterlockedCompareExchange_rel (volatile unsigned int *Destination, unsigned int64 Exchange, unsigned int64 Comparand)</code>	適切な設定のcmpxchg4.rel 命令に対応付けます。第1引数(64ビット・ポインタ)で指定された値をアトミックに比較/交換します。
<code>unsigned __int64 _InterlockedCompareExchange_acq (volatile unsigned int *Destination, unsigned int64 Exchange, unsigned int64 Comparand)</code>	上記と同じですが、cmpxchg4.acq 命令に対応付けます。
<code>void</code>	スピンロックを解放します。

<code>ReleaseSpinLock(volatile int *x)</code>	
<code>__int64 _InterlockedIncrement64(volatile int64 *addend)</code>	引数で指定された値を1ずつ増分します。fetchadd命令に対応付けます。
<code>__int64 _InterlockedDecrement64(volatile int64 *addend)</code>	引数で指定された値を1ずつ減分します。fetchadd命令に対応付けます。
<code>__int64 _InterlockedExchange64(volatile __int64 *Target, int64 value)</code>	交換操作をアトミックに実行します。xchg命令に対応付けます。
<code>unsigned __int64 _InterlockedExchangeU64(volatile unsigned __int64 *Target, unsigned __int64 value)</code>	InterlockedExchange64と同じです(符号なしのデータ)。
<code>unsigned __int64 _InterlockedCompareExchange64 rel(volatile unsigned __int64 *Destination, unsigned int64 Exchange, unsigned __int64 Comparand)</code>	適切な設定のcmpxchg.rel 命令に対応付けます。第1引数(64ビット・ポインタ)で指定された値をアトミックに比較/交換します。
<code>unsigned __int64 _InterlockedCompareExchange64 acq(volatile unsigned __int64 *Destination, unsigned int64 Exchange, unsigned __int64 Comparand)</code>	適切な設定のcmpxchg.acq 命令に対応付けます。第1引数(64ビット・ポインタ)で指定された値をアトミックに比較/交換します。
<code>__int64 _InterlockedCompareExchange64(volatile __int64 *Destination, __int64 Exchange, __int64 Comparand)</code>	符号付きデータについては上記と同じです。
<code>__int64 _InterlockedExchangeAdd64(volatile int64 *addend, __int64 increment)</code>	比較/交換操作を使用して、加数に対して増分値をアトミックに加算します。cmpxchg 命令を使用するループに対応付けられ、アトミックな性質を保証します。
<code>__int64 _InterlockedAdd64(volatile int64 *addend, int64 increment)</code>	上記と同じです。元の値ではなく、新しい値を返します。下記の注を参照してください。

increment);	い。
-------------	----



注:

`_InterlockedSub64`は`_InterlockedAdd64`に基づいたマクロ定義として提供されています。

```
#define _InterlockedSub64(target, incr)
_InterlockedAdd64((target), (- (incr))).
```

`cmpxchg`を使用して、`target`に対して`incr`値をアトミックに減算します。`cmpxchg` 命令を使用するループに対応付けられ、アトミックな性質を保証します。

## ロードとストア

ロードとストアの組込み関数を使用して、特定のデータ・オブジェクトのメモリアクセスの順序を制限することができます。これを使用するのは、`no-serialize-volatile` オプションを使用して、ユーザのメモリアクセスの順序を厳密に抑止する場合です。

組込み関数	プロトタイプ	説明
<code>__st1_rel</code>	<code>void __st1_rel(void *dst, const char value);</code>	<code>st1.rel</code> 命令を生成します。
<code>__st2_rel</code>	<code>void __st2_rel(void *dst, const short value);</code>	<code>st2.rel</code> 命令を生成します。
<code>__st4_rel</code>	<code>void __st4_rel(void *dst, const int value);</code>	<code>st4.rel</code> 命令を生成します。
<code>__st8_rel</code>	<code>void __st8_rel(void *dst, const int64 value);</code>	<code>st8.rel</code> 命令を生成します。
<code>__ld1_acq</code>	<code>unsigned char __ld1_acq(void *src);</code>	<code>ld1.rel</code> 命令を生成します。
<code>__ld2_acq</code>	<code>unsigned short __ld2_acq(void *src);</code>	<code>ld2.rel</code> 命令を生成します。
<code>__ld4_acq</code>	<code>unsigned int __ld4_acq(void *src);</code>	<code>ld4.rel</code> 命令を生成します。
<code>__ld8_acq</code>	<code>unsigned int64 __ld8_acq(void *src);</code>	<code>ld8.rel</code> 命令を生成します。

## Itanium® ベース・システムのオペレーティング・システムに関連する組込み関数

Itanium® アーキテクチャ用の組込み関数のプロトタイプは、ヘッダファイル `ia64intrin.h` 内にあります。

組込み関数	説明
<code>unsigned __int64 __getReg(const int whichReg)</code>	指定されたインデックスに基づいて、ハードウェア・レジスタから値を取得します。対応する <code>mov = r</code> 命令を生成します。以下のレジスタにアクセスできます。 <code>getReg()</code> および <code>setReg()</code> のレジスタ名を参照してください。
<code>void __setReg(const int whichReg, unsigned __int64 value)</code>	指定されたインデックスに基づいて、ハードウェア・レジスタの値を設定します。対応する <code>mov = r</code> 命令を生成します。 <code>getReg()</code> および <code>setReg()</code> のレジスタ名を参照してください。
<code>unsigned __int64 __getIndReg(const int whichIndReg, __int64 index)</code>	インデックス付きレジスタの値を返します。インデックスは第2引数で、レジスタ・ファイルは第1引数です。
<code>void setIndReg(const int whichIndReg, __int64 index, unsigned __int64 value)</code>	インデックス付きレジスタで値をコピーします。インデックスは第2引数で、レジスタ・ファイルは第1引数です。
<code>void *_rdteb(void)</code>	TEB アドレスを取得します。TEB アドレスは、 <code>r13</code> に保持しています。 <code>move r=tp</code> 命令に対応付けます。
<code>void __isrlz(void)</code>	<code>serialize</code> 命令を実行します。 <code>srlz.i</code> 命令に対応付けます。
<code>void __dsrlz(void)</code>	データをシリアル化します。 <code>srlz.d</code> 命令に対応付けます。
<code>unsigned __int64 __fetchadd4_acq(unsigned int *addend, const int increment)</code>	<code>fetchadd4.acq</code> 命令に対応付けます。
<code>unsigned __int64 __fetchadd4_rel(unsigned int *addend, const int increment)</code>	<code>fetchadd4.rel</code> 命令に対応付けます。
<code>unsigned __int64 __fetchadd8_acq(unsigned __int64 *addend, const int</code>	<code>fetchadd8.acq</code> 命令に対応付けます。

increment)	
unsigned __int64 __fetchadd8_rel(unsigned int64 *addend, const int increment)	fetchadd8.rel 命令を対応付けま す。
void __fwb(void)	書き込みバッファをフラッシュします。fwb 命令に対応付けます。
void __ldfs(const int whichFloatReg, void *src)	ldfs 命令を対応付けます。指定したレ ジスタに単精度値をロードします。
void __ldfd(const int whichFloatReg, void *src)	ldfd 命令を対応付けます。指定したレ ジスタに倍精度値をロードします。
void __ldfe(const int whichFloatReg, void *src)	ldfe 命令を対応付けます。指定したレ ジスタに拡張精度値をロードします。
void __ldf8(const int whichFloatReg, void *src)	ldf8 命令を対応付けます。
void __ldf_fill(const int whichFloatReg, void *src)	ldf.fill 命令を対応付けます。
void __stfs(void *dst, const int whichFloatReg)	sfts 命令を対応付けます。
void __stfd(void *dst, const int whichFloatReg)	stfd 命令を対応付けます。
void __stfe(void *dst, const int whichFloatReg)	stfe 命令を対応付けます。
void __stf8(void *dst, const int whichFloatReg)	stf8 命令を対応付けます。
void __stf_spill(void *dst, const int whichFloatReg)	stf.spill 命令を対応付けます。
void __mf(void)	メモリフェンス命令を実行します。mf 命 令に対応付けます。
void __mfa(void)	メモリフェンス(受け入れ形式)命令を実行 します。mf.a 命令に対応付けます。
void __synci(void)	メモリの同期化を有効にします。sync.i 命令に対応付けます。
void __thash(__int64)	変換ハッシュ・エントリ・アドレスを生成しま す。thash r = r 命令に対応付けま す。
void __ttag(__int64)	変換ハッシュ・エントリ・タグを生成します。 ttag r=r 命令に対応付けます。
void __itcd(__int64 pa)	エントリをデータ変換キャッシュに挿入しま す (itc.d 命令を対応付けます)。
void __itci(__int64 pa)	エントリを命令変換キャッシュに挿入しま す

	(itc.i を対応付けます)。
void __itrld(__int64 whichTransReg, int64 pa)	itr.d 命令を対応付けます。
void __itri(__int64 whichTransReg, int64 pa)	itr.i 命令を対応付けます。
void __ptce(__int64 va)	ptc.ei 命令を対応付けます。
void __ptcl(__int64 va, __int64 pagesz)	ローカル変換キャッシュをページします。 ptc.l r, r 命令に対応付けます。
void __ptcg(__int64 va, __int64 pagesz)	グローバル変換キャッシュをページしま す。ptc.g r, r 命令に対応付けます。
void __ptcg(__int64 va, __int64 pagesz)	グローバル変換キャッシュと ALAT をパ ージします。ptc.ga r, r 命令に対応 付けます。
void __ptri(__int64 va, __int64 pagesz)	変換レジスタをページします。ptr.i r, r 命令に対応付けます。
void __ptrd(__int64 va, __int64 pagesz)	変換レジスタをページします。ptr.d r, r 命令に対応付けます。
__int64 __tpa(__int64 va)	tpa 命令を対応付けます。
void __invalat(void)	ALAT を無効化します。invala 命 令に対応付けます。
void __invala (void)	void __invalat(void) と同じで す。
void __invala gr(const int whichGeneralReg)	whichGeneralReg = 0-127
void __invala fr(const int whichFloatReg)	whichFloatReg = 0-127
void __break(const int)	ブレーク命令と即値を生成します。
void __nop(const int)	nop 命令を生成します。
void __debugbreak(void)	デバッグ・ブレーク命令フォルトを生成しま す。
void __fc(__int64)	引数で指定されたアドレスに関連するキャ ッシュ・ラインをフラッシュします。fcr 命 令に対応付けます。
void __sum(int mask)	PSR のユーザマスクビットを設定します。 sum imm24 命令に対応付けます。
void __rum(int mask)	ユーザマスクをリセットします。
void __ssm (int mask)	システムマスクを設定します。
void __rsm (int mask)	PSR のシステム・マスク・ビットをリセットし ます。rsm imm24 命令に対応付けま す。
__int64	呼び出し元のアドレスを返します。



<code>ReturnAddress(void)</code>	
<code>void lfeth(int lfhint, void *y)</code>	<code>lfeth.lfhint</code> 命令を生成します。第1引数の値はヒント・タイプを指定します。
<code>void __lfeth_fault(int lfhint, void *y)</code>	<code>lfeth.fault.lfhint</code> 命令を生成します。第1引数の値はヒント・タイプを指定します。
<code>unsigned int __cacheSize(unsigned int cacheLevel)</code>	<code>__cacheSize(n)</code> は <code>n</code> レベルのキャッシュのサイズをバイトで返します。1 は一次キャッシュを表します。キャッシュ・レベルが存在しない場合、0 が返されます。例えば、アプリケーションはキャッシュ・サイズをクエリし、行列を操作するアルゴリズムでブロックサイズを選択するのにキャッシュ・サイズを使用します。
<code>void __memory_barrier(void)</code>	コンパイラがデータアクセス命令をスケジューリングしないbarrierを生成します。コンパイラはメモリbarrierのレジストリにローカルデータを割り当てます。しかしグローバルデータは割り当てません。

## Itanium® ベース・システム用の変換組込み関数

Itanium® アーキテクチャ用の組込み関数のプロトタイプは、ヘッダファイル `ia64intrin.h`内にあります。

組込み関数	説明
<code>int64 m to int64( m64 a)</code>	<code>m64</code> 型の <code>a</code> を <code>__int64</code> 型に変換します。Itanium ベースのシステムでは、両方のデータ型が同じレジスタに置かれるため、 <code>nop</code> に変換します。
<code>m64 __m_from_int64(__int64 a)</code>	<code>__int64</code> 型の <code>a</code> を <code>__m64</code> 型に変換します。Itanium ベースのシステムでは、両方のデータ型が同じレジスタに置かれるため、 <code>nop</code> に変換します。
<code>__int64 __round_double_to_int64(double d)</code>	倍精度引数を符号付き整数に変換します。
<code>unsigned __int64 __getf_exp(double d)</code>	<code>getf.exp</code> 命令をマップし、オペランドの16ビット指数と符号を返します。

## getReg() および setReg() のレジスタ名

`getReg()` と `setReg()` の組込み関数のプロトタイプは、ヘッダ・ファイル `ia64regs.h`内に

あります。

名	whichReg
IA64 REG IP	1016
IA64 REG PSR	1019
IA64 REG PSR L	1019

## 一般的な整数レジスタ

名	whichReg
IA64 REG GP	1025
IA64 REG SP	1036
IA64 REG TP	1037

## アプリケーション・レジスタ

名	whichReg
IA64 REG AR KR0	3072
IA64 REG AR KR1	3073
IA64 REG AR KR2	3074
IA64 REG AR KR3	3075
IA64 REG AR KR4	3076
IA64 REG AR KR5	3077
IA64 REG AR KR6	3078
IA64 REG AR KR7	3079
IA64 REG AR RSC	3088
IA64 REG AR BSP	3089
IA64 REG AR BSPSTORE	3090
IA64 REG AR RNAT	3091
IA64 REG AR FCR	3093
IA64 REG AR EFLAG	3096
IA64 REG AR CSD	3097
IA64 REG AR SSD	3098
IA64 REG AR CFLAG	3099
IA64 REG AR FSR	3100
IA64 REG AR FIR	3101
IA64 REG AR FDR	3102
IA64 REG AR CCV	3104
IA64 REG AR UNAT	3108
IA64 REG AR FPSR	3112
IA64 REG AR ITC	3116
IA64 REG AR PFS	3136
IA64 REG AR LC	3137
IA64 REG AR EC	3138

## コントロール・レジスタ

名	whichReg
IA64 REG CR DCR	4096
IA64 REG CR ITM	4097
IA64 REG CR IVA	4098
IA64 REG CR PTA	4104
IA64 REG CR IPSR	4112
IA64 REG CR ISR	4113
IA64 REG CR IIP	4115
IA64 REG CR IFA	4116
IA64 REG CR ITIR	4117
IA64 REG CR IIPA	4118
IA64 REG CR IFS	4119
IA64 REG CR IIM	4120
IA64 REG CR IHA	4121
IA64 REG CR LID	4160
IA64 REG CR IVR	4161 *
IA64 REG CR TPR	4162
IA64 REG CR EOI	4163
IA64 REG CR IRR0	4164 *
IA64 REG CR IRR1	4165 *
IA64 REG CR IRR2	4166 *
IA64 REG CR IRR3	4167 *
IA64 REG CR ITV	4168
IA64 REG CR PMV	4169
IA64 REG CR CMCV	4170
IA64 REG CR LRR0	4176
IA64 REG CR LRR1	4177

\* getReg のみ

## getIndReg() および setIndReg() の間接レジスタ

名	whichReg
IA64 REG INDR CPUID	9000 *
IA64 REG INDR DBR	9001
IA64 REG INDR IBR	9002
IA64 REG INDR PKR	9003
IA64 REG INDR PMC	9004
IA64 REG INDR PMD	9005
IA64 REG INDR RR	9006
IA64 REG INDR RESERVED	9007

\* getIndReg のみ

## Itanium® ベース・システム用のマルチメディア乗算

Itanium® アーキテクチャ用の組込み関数のプロトタイプは、ヘッダファイル `ia64intrin.h` 内にあります。

組込み関数	対応する命令
<code>__int64 _m64_czx1l(__m64 a)</code>	<code>czx1.l</code> (Compute Zero Index)
<code>__int64 _m64_czx1r(__m64 a)</code>	<code>czx1.r</code> (Compute Zero Index)
<code>__int64 _m64_czx2l(__m64 a)</code>	<code>czx2.l</code> (Compute Zero Index)
<code>__int64 _m64_czx2r(__m64 a)</code>	<code>czx2.r</code> (Compute Zero Index)
<code>m64 m64 mix1l(m64 a, m64 b)</code>	<code>mix1.l</code> (Mix)
<code>m64 m64 mix1r(m64 a, m64 b)</code>	<code>mix1.r</code> (Mix)
<code>m64 m64 mix2l(m64 a, m64 b)</code>	<code>mix2.l</code> (Mix)
<code>__m64 _m64_mix2r(m64 a, m64 b)</code>	<code>mix2.r</code> (Mix)
<code>m64 m64 mix4l(m64 a, m64 b)</code>	<code>mix4.l</code> (Mix)
<code>m64 m64 mix4r(m64 a, m64 b)</code>	<code>mix4.r</code> (Mix)
<code>__m64 _m64_mux1(__m64 a, const int n)</code>	<code>mux1</code> (Mux)
<code>__m64 _m64_mux2(__m64 a, const int n)</code>	<code>mux2</code> (Mux)
<code>__m64 _m64_padd1uus(__m64 a, m64 b)</code>	<code>padd1.uus</code> (Parallel add)
<code>m64 m64 padd2uus(m64 a, m64 b)</code>	<code>padd2.uus</code> (Parallel add)
<code>__m64 _m64_pavg1_nraz(__m64 a, m64 b)</code>	<code>pavg1</code> (Parallel average)
<code>__m64 _m64_pavg2_nraz(__m64 a, m64 b)</code>	<code>pavg2</code> (Parallel average)
<code>__m64 _m64_pavgsub1(_m64 a, m64 b)</code>	<code>pavgsub1</code> (Parallel average subtract)
<code>m64 m64 pavgsub2(m64 a, m64 b)</code>	<code>pavgsub2</code> (Parallel average subtract)
<code>m64 m64 pmpy2r(m64 a, m64 b)</code>	<code>pmpy2.r</code> (Parallel multiply)

m64 m64 pmpy2l( m64 a, m64 b)	pmpy2.l (Parallel multiply)
m64 m64 pmpyshr2( m64 a, __m64 b, const int count)	pmpyshr2 (Parallel multiply and shift right)
m64 m64 pmpyshr2u( m64 a, __m64 b, const int count)	pmpyshr2.u (Parallel multiply and shift right)
m64 m64 pshladd2( m64 a, const int count, m64 b)	pshladd2 (Parallel shift left and add)
m64 m64 pshradd2( m64 a, const int count, m64 b)	pshradd2 (Parallel shift right and add)
m64 m64 psub1uus( m64 a, m64 b)	psub1.uus (Parallel subtract)
m64 m64 psub2uus( m64 a, m64 b)	psub2.uus (Parallel subtract)

\_\_int64 \_\_m64\_czx1l(\_\_m64 a)

64ビット値a の中の0の要素を、最上位の要素から最下位の要素に向かってスキャンし、最初に見つかった0の要素のインデックスを返します。要素の幅は8ビットであるため、結果の範囲は 0～7 になります。0の要素が見つからない場合は、デフォルトでは結果は 8 になります。

\_\_int64 \_\_m64\_czx1r(\_\_m64 a)

64ビット値a の中の0の要素を、最下位の要素から最上位の要素に向かってスキャンし、最初に見つかった0の要素のインデックスを返します。要素の幅は8ビットであるため、結果の範囲は 0～7 になります。0の要素が見つからない場合は、デフォルトでは結果は 8 になります。

\_\_int64 \_\_m64\_czx2l(\_\_m64 a)

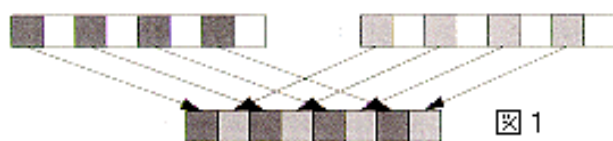
64ビット値a の中の0の要素を、最上位の要素から最下位の要素に向かってスキャンし、最初に見つかった0の要素のインデックスを返します。要素の幅は16ビットであるため、結果の範囲は 0～3 になります。0の要素が見つからない場合は、デフォルトでは結果は 4 になります。

\_\_int64 \_\_m64\_czx2r(\_\_m64 a)

64ビット値a の中の0の要素を、最下位の要素から最上位の要素に向かってスキャンし、最初に見つかった0の要素のインデックスを返します。要素の幅は16ビットであるため、結果の範囲は 0～3 になります。0の要素が見つからない場合は、デフォルトでは結果は 4 になります。

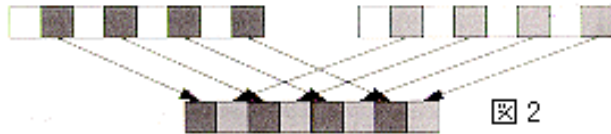
\_\_m64 \_\_m64\_mix1l(\_\_m64 a, \_\_m64 b)

64ビット値a と b を、図1に示すように、1バイト・グループ単位で左から順にインターリーブし、得られた結果を返します。



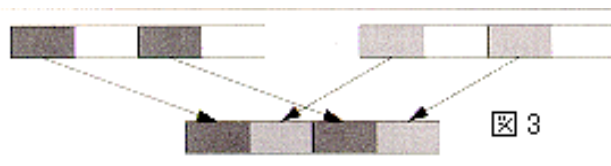
\_\_m64 \_\_m64\_mix1r(\_\_m64 a, \_\_m64 b)

64ビット値 a と b を、図2に示すように、1バイト・グループ単位で右から順にインターリーブし、得られた結果を返します。



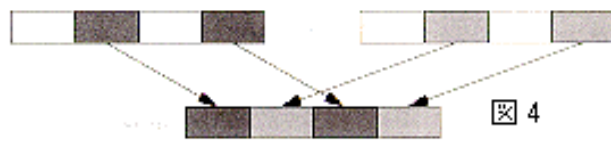
\_\_m64 \_\_m64\_mix2l(\_\_m64 a, \_\_m64 b)

64ビット値 a と b を、図3に示すように、2バイト・グループ単位で左から順にインターリーブし、得られた結果を返します。



\_\_m64 \_\_m64\_mix2r(\_\_m64 a, \_\_m64 b)

64ビット値 a と b を、図4に示すように、4バイト・グループ単位で右から順にインターリーブし、得られた結果を返します。



\_\_m64 \_\_m64\_mix4l(\_\_m64 a, \_\_m64 b)

64ビット値 a と b を、図5に示すように、4バイト・グループ単位で左から順にインターリーブし、得られた結果を返します。



\_\_m64 \_\_m64\_mix4r(\_\_m64 a, \_\_m64 b)

64ビット値 a と b を、図6に示すように、4バイトグループ単位で右から順にインターリーブし、得られた結果を返します。



\_\_m64 \_m64\_mux1(\_\_m64 a, const int n)

n の値に基づいて、図7に示すように a の並べ替えを実行し、その結果を返します。表1に、指定可能な n の値を示します。

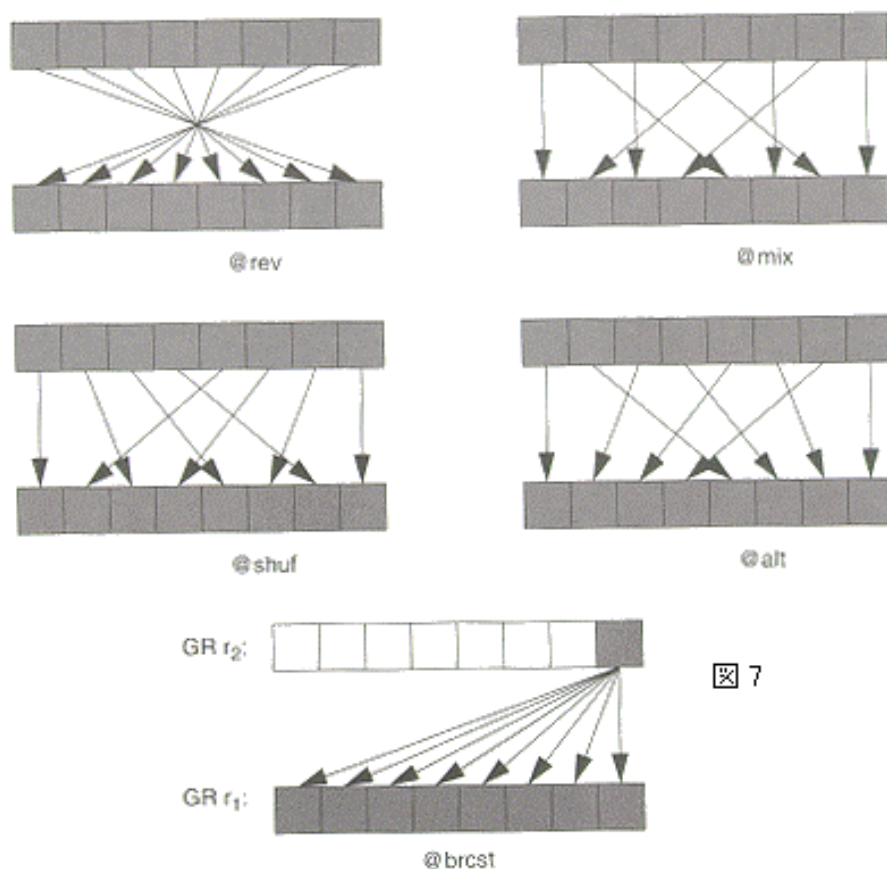


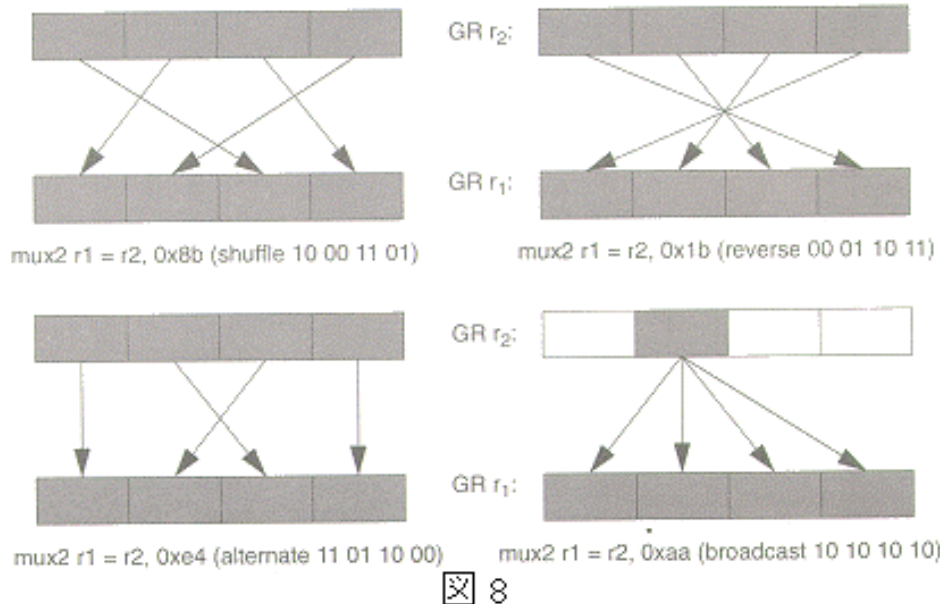
図 7

表1. `_m64_mux1` の n の値

	n
@brcst	0
@mix	8
@shuf	9
@alt	0xA
@rev	0xB

\_\_m64 \_m64\_mux2(\_\_m64 a, const int n)

n の値に基づいて、図8に示すように a の並べ替えを実行し、その結果を返します。



\_\_m64 \_\_m64\_pavgsub1(\_\_m64 a, \_\_m64 b)

a の符号なしデータ要素(バイト)から b の符号なしデータ要素(バイト)を引き、減算の結果をそれぞれ1ポジションだけ右にシフトします。各要素の上位ビットは、減算のボロービットで埋められます。

\_\_m64 \_\_m64\_pavgsub2(\_\_m64 a, \_\_m64 b)

a の符号なしデータ要素(ダブルバイト)から b の符号なしデータ要素(ダブルバイト)を引き、減算の結果をそれぞれ1ポジションだけ右にシフトします。各要素の上位ビットは、減算のボロービットで埋められます。

\_\_m64 \_\_m64\_pmpy2l(\_\_m64 a, \_\_m64 b)

図9に示すように、a の2個の符号付き16ビット・データ要素に、最上位の要素から順に、それに対応する b の2個の符号付き16ビット・データ要素を掛けて、2個の32ビットの結果を返します。

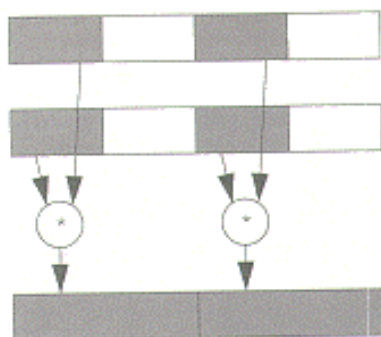


図 9

\_\_m64 \_\_m64\_pmpy2r(\_\_m64 a, \_\_m64 b)

図10に示すように、a の2個の符号付き16ビット・データ要素に、最下位の要素から順



に、それに対応する  $b$  の2個の符号付き16ビット・データ要素を掛けて、2個の32ビットの結果を返します。

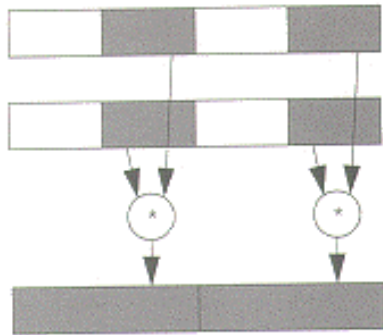


図 10

- ```
__m64 __m64_pmpyshr2(__m64 a, __m64 b, const int count)
```
- $a$  の4個の符号付き16ビット・データ要素に、それに対応する  $b$  の符号付き16ビット・データ要素を掛けて、4個の32ビットの積を求めます。それぞれの積を、 $\text{count}$  ビットだけ右にシフトします。シフトしたそれぞれの積の最下位16ビットから、4つの16ビットの結果を生成し、1つの64ビットワードとして返します。
- ```
__m64 __m64_pmpyshr2u(__m64 a, __m64 b, const int count)
```
- $a$  の4個の符号なし16ビット・データ要素に、それに対応する  $b$  の符号なし16ビット・データ要素を掛けて、4個の32ビットの積を求めます。それぞれの積を、 $\text{count}$  ビットだけ右にシフトします。シフトしたそれぞれの積の最下位16ビットから、4つの16ビットの結果を生成し、1つの64ビットワードとして返します。
- ```
__m64 __m64_pshladd2(__m64 a, const int count, __m64 b)
```
- $a$  を  $\text{count}$  ビットだけ左にシフトして、 $b$  に加算します。結果の上位32ビットを 0 にクリアして、 $b$  のビット [31:30] を結果のビット[62:61] にコピーし、その結果を返します。
- ```
__m64 __m64_pshradd2(__m64 a, const int count, __m64 b)
```
- $a$  の4個の符号付き16ビットデータ要素を、それぞれ  $\text{count}$  ビットだけ右にシフトします(各要素の上位ビットは、 $a$  のデータ要素の符号ビットの初期値で埋められます)。次に、これらの要素を、 $b$  の4個の符号付き16ビットデータ要素に加算し、その結果を返します。
- ```
__m64 __m64_padd1uus(__m64 a, __m64 b)
```
- 8個の1バイト要素として、 $a$  を  $b$  に加算します。 $a$  の要素は符号なしデータとして処理され、 $b$  の要素は符号付きデータとして処理します。演算の結果は符号なしデータとして処理され、1つの64ビットワードとして返します。
- ```
__m64 __m64_padd2uus(__m64 a, __m64 b)
```
- 4個の16ビット要素として、 $a$  を  $b$  に加算します。 $a$  の要素は符号なしデータとして処理され、 $b$  の要素は符号付きデータとして処理します。演算の結果は符号なしデータとして処理され、1つの64ビットワードとして返します。
- ```
__m64 __m64_psub1uus(__m64 a, __m64 b)
```

8個の1バイト要素として、b から a を引きます。a の要素は符号なしデータとして処理され、b の要素は符号付きデータとして処理します。演算の結果は符号なしデータとして処理され、1つの64ビットワードとして返します。

\_\_m64 \_\_m64\_psub2uus(\_\_m64 a, \_\_m64 b)

4個の16ビット要素として、b から a を引きます。a の要素は符号なしデータとして処理され、b の要素は符号付きデータとして処理します。演算の結果は符号なしデータとして処理され、1つの64ビットワードとして返します。

\_\_m64 \_\_m64\_pavg1\_nraz(\_\_m64 a, \_\_m64 b)

a の符号なしのバイト・データ要素を、b の符号なしのバイト・データ要素に加算し、それぞれの加算の結果を1ポジションだけ右にシフトします。各要素の上位ビットは、和のキャリービットで埋められます。

\_\_m64 \_\_m64\_pavg2\_nraz(\_\_m64 a, \_\_m64 b)

a の符号なしの16ビット・データ要素を、b の符号なしの16ビット・データ要素に加算し、それぞれの加算の結果を1ポジションだけ右にシフトします。各要素の上位ビットは、和のキャリービットで埋められます。

# データのアライメント、メモリの割り当て組込み関数、およびインライン・アセンブリ

## データのアライメント、メモリの割り当て組込み関数、およびインライン・アセンブリの概要

この節では、組込み関数の使用をサポートする機能について説明します。次の項目について説明します。

- アライメントのサポート
- アライメントの合ったメモリブロックの割り当てと解放
- インライン・アセンブリ

### アライメントのサポート

組込み関数のパフォーマンスを向上させるには、データをアライメントする必要があります。例えば、ストリーミングSIMD拡張命令を使用する場合は、メモリ操作の際にデータのアライメントを16バイトに合わせて、パフォーマンスを向上させてください。特に、`_mm_load`、`_mm_store`の組込み関数にアドレスを渡したときは、`__m128`オブジェクトのアライメントを合わせなければなりません。float型の配列を宣言し、キャストすることによってそれを`__m128`オブジェクトとして扱う場合は、float型配列のアライメントが正しく揃うようにする必要があります。

より正確なデータのアライメントを合わせるためにコンパイラへ命令するときは、`__declspec(align)` を使用してください。そうすると、IA-32、いずれの Itanium® ベースのシステムでもアライメントの正確性が上がります。例えばint型のデータ・オブジェクトは、特に指定しなければ4の倍数(すなわちintのサイズ)のバイトアドレスに配置されます。一方、`__declspec(align)` を使えば、4の代わりに8、16、32のいずれかの倍数のアドレスを使用するようにコンパイラに命令できます。ただしIA-32の場合は次の制約があります。

- 32バイト・アドレスは、静的に割り当てなければなりません。
- 16バイト・アドレスは、ローカルに割り当てたり、静的に割り当てたりできます。

この機能を使用して、キャッシュ・ラインの使用効率を最適化できます。よく使用する小さなオブジェクトをいくつか組み合わせて1個の構造体を作り、その構造体を強制的にキャッシュ・ラインの先頭に配置すれば、どのオブジェクトにアクセスしたときでも、すぐにそのオブジェクトがキャッシュにロードされるため、パフォーマンスが大きく向上します。

この拡張属性の構文は、次のとおりです。

`align(n)`

ここで、n は、2の何乗かを示す値(32またはそれ以下)です。データのアライメントは、この値のバイト境界に合わせてられます。



注意

このリリースでは、`__declspec(align(8))` が正しく動作しません。代わりに `__declspec(align(16))` を使用してください。



#### 注

対象となるデータ型のサイズより小さい値を指定した場合は、無効になります。言い換えれば、データは、元々設定されているアライメントの最大値か、`__declspec(align)` で指定したアライメントかのいずれかに揃います。

変数の種類が静的か自動かに関わらず、個々の変数についてアライメント要求を設定できません。特に指定しない限り、グローバル変数と静的変数には静的記憶域期間が設定されて、ローカル変数には自動記憶域期間が設定されています。パラメータのアライメントは調整できません。struct フィールド、class フィールドのアライメントの調整もできません。ただし struct (または union、class) のアライメントは拡張されます。その場合は、該当する型のオブジェクトがすべて影響を受けます。

例えば、ある関数が2次元配列の添字にローカル変数 `i`、`j` を使用する場合、これらの変数は次のように宣言されます。これらの組込み関数は、次のグループに分類されます。

```
int i, j;
```

2つの変数は、通常は組み合わせて使用します。しかし、これらの変数が異なるキャッシュ・ラインに割り当てられると、パフォーマンスが低下します。これを防ぐには、次のように2つの変数を宣言します。

```
__declspec(align(8)) struct { int i, j; } sub;
```

これで、コンパイラは、2つの変数を必ず同じキャッシュ・ラインに割り当てます。C++ では、struct 変数名(上の例では `sub`) が省略できます。ただしCの場合は必要であり、`i`、`j` への参照を、`sub.i`、`sub.j` のように記述しなければなりません。

この添字ペアの付いた関数を多く使用する場合は、次の例のように、その関数に対して struct 型を宣言して使用するほうが便利です。

```
typedef struct __declspec(align(8)) { int i, j; } Sub;
```

キーワード `struct` の後に `__declspec(align)` を置くと、該当する型のオブジェクトすべてに対してしかるべきアライメント要求が出ます。ただし、各パラメータの配置は `__declspec(align)` に影響されません。必要であれば、アライメントの正しく揃っているローカル変数にパラメータの値が代入できます。

また、配列などのグローバル変数についても、アライメントを指定できます。

```
__declspec(align(16)) float array[1000];
```

## アライメントの合ったメモリブロックの割り当てと解放

`_mm_malloc` および `_mm_free` 組込み関数を使用して、アライメントの合ったメモリブロックの割り当てと解放を実行できます。これらの組込み関数は、`libirc.a` 内にある `malloc` および `free` 関数に基づいています。これらの組込み関数の構文は、次のとおりです。

```
void* _mm_malloc (int size, int align)
```

`void _mm_free (void *p)`

`_mm_malloc` ルーチンには、追加のパラメータを1つ指定する必要があります。このパラメータは、アライメントの制約条件です。この制約条件は、2の何乗かを示す値です。

`_mm_malloc` で返すポインタは、指定した境界にアライメントが合っているのを保証します。



`_mm_malloc` を使用して割り当てられたメモリは、`_mm_free` を使用して解放しなければなりません。`_mm_malloc` で割り当てられたメモリ上で `free` を呼び出したり、`malloc` で割り当てられたメモリ上で `_mm_free` を呼び出したりすると、予測できない動作が発生します。

## インライン・アセンブリ

デフォルトでは、C、C++、および数値演算の各標準ライブラリ関数のいくつかはコンパイラによってインライン化します。通常、この処理によってプログラムの実行速度が速くなります。

ライブラリ関数をインライン展開すると、予期しない結果になる場合があります。インライン化されたライブラリ関数では、`errno`変数は設定されません。したがって、`errno`変数を設定するかしないかによって動作が異なるコードに対しては、`-nolib_inline`オプションを使用しなければなりません。そうすれば、ライブラリ関数のインライン展開は禁止されます。また、コンパイラから提供されるライブラリ関数と同じ名前を持つ関数はライブラリ関数と見なすため、元々の呼び出し命令は、インライン化したものと置き換わります。したがって、既知のライブラリ・ルーチンのいずれかと同じ名前を持つ関数がプログラムの中で定義している場合は、その関数そのものが必ず使用されるように`-nolib_inline`オプションを使用する必要があります。



ライブラリ関数の自動インライン展開は、プロシージャ間の最適化処理中にコンパイラが行うインライン展開とは関連がありません。例えば、次のコマンドを実行すると、`sum.c` というプログラムがコンパイルされます。このとき、ライブラリ関数の展開は行いませんが、プロシージャ間の最適化(IPO)によるインライン展開は行います。

- **IA-32 システム:** `prompt>icc -ip -nolib_inline sum.c`
- **Itanium® ベース・システム:** `prompt>ecc -ip -nolib_inline sum.c`

IPOの詳細は、「プロシージャ間の最適化とプロファイルに基づく最適化」を参照してください。

### MASM\* スタイルのインライン・アセンブリ

インテル® C++ コンパイラは、`-use_msasm`オプションによってMASM\* スタイルのインライン・アセンブリをサポートしています。構文については、MASM のマニュアルを参照してください。

### GNU\* 式スタイルのインライン・アセンブリ

インテル C++ コンパイラは、GNU式 スタイルのインライン・アセンブリをサポートしています。構文は、次のとおりです。

```
asm-keyword [ volatile-keyword ] ( asm-template
[ asm-interface ] ) ;
```

| 構文要素             | 説明                                                                                                                                                                                                                                                                                           |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| asm-keyword      | asm文はasmキーワードで始まります。または、互換性のため、__asmもしくは__asm__ が使用されることもあります。                                                                                                                                                                                                                               |
| volatile-keyword | オプションの volatile キーワードが指定されたら、asm は volatile です。2 つのvolatile asm文は、お互いに移動しません。volatile変数への参照は、volatile asmへ相対移動しません。または、互換性のため、__volatileもしくはvolatile__ が使用されることもあります。                                                                                                                        |
| asm-template     | asm-templateは、アセンブリ・コードを出力する方法を指定する C 言語の ASCII 文字列です。テンプレートの多くは固定文字列です。代入ディレクティブ以外のすべては、アセンブリにそのまま渡されます。代入ディレクティブの構文は、%の後に1または2文字続きます。サポートされた代入ディレクティブは、次のセクションで指定されます。                                                                                                                     |
| asm-interface    | asm-interface は次の3つの部分で構成されます。<br>1. output-list(オプション)<br>2. input-list(オプション)<br>3. clobber-list(オプション)<br>これらは、コロン(:)で区切ります。output-listがなく、input-list が指定される場合、output-list の代わりに、input-list は2つのコロン(::) に続きます。asm-interfaceがすべて省略された場合、volatile-keywordの指定の有無にかかわらず、asm文はvolatileとみなされます。 |
| output-list      | output-list は、カンマで区切られた1つ以上のoutput-specs から構成されます。<br>asm-template に代入するために、各output-specには番号が付けられます。<br>output-list の最初のオペランドは 0 で、次は 1 のようになります。番号付けは、output-listからinput-list へ続行します。オペランドの合計数は10個までです(0-9)。                                                                                |
| input-list       | output-listと類似して、input-list はカンマで区切られた1つ以上のinput-specs から構                                                                                                                                                                                                                                   |

|              |                                                                                                                                                                                                                                                                                                                       |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | 成されます。asm-templateに代入するために、各 input-specは番号が付けられます。番号は、output-listのオペランドから続きます。                                                                                                                                                                                                                                        |
| clobber-list | clobber-list はasm が特定のマシンレジスタを使用または変更することをコンパイラに伝えます。特定のマシンレジスタは直接asmにコードされるか、アセンブリの命令によって暗黙的に変更されます。clobber-list は、カンマで区切られた clobber-specs のリストです。                                                                                                                                                                   |
| input-spec   | input-specs は、挿入されたアセンブリの命令によって必要とされる値の式をコンパイラに伝えます。asm の必須入力をすべて示すのに、実際には asm-templateに参照されないinput-specを一覧表示できます。                                                                                                                                                                                                    |
| clobber-spec | 各clobber-specは、壊れた1つのマシン・レジスタ名を指定します。レジスタ名は、オプションで先頭に%を使用できます。有効なレジスタ名: eax, ebx, ecx, edx, esi, edi, ebp, esp, ax, bx, cx, dx, si, di, bp, sp, al, bl, cl, dl, ah, bh, ch, dh, st, st(1) ?Est(7), mm0 ?Emm7, xmm0 ?Exmm7 および cc また、clobber-spec で"メモリ" を指定することもできます。これを指定すると、コンパイラはレジスタにキャッシュされたデータをasm文に渡さないようにします。 |

## 各種のプロセッサでの組込み関数の使用

### 各種のプロセッサでの組込み関数の使用

この節には、各種のアーキテクチャ上での組込み関数のパフォーマンスを比較する一連の表を記載しています。各アーキテクチャ上で組込み関数を使用する前に、次の点に注意してください。

- 組込み関数を使用すると、一部の IA プロセッサ上で正常に動作しないコードが生成される可能性があります。したがって、プログラマが CPUID 命令を使用してプロセッサを検出し、適切なコードを生成しなければなりません。
- 組込み関数は、特定のプロセッサ向けにではなく、プロセッサ・ファミリごとに使用してください。組込み関数がどちらのファミリ(IA-32 または Itanium® プロセッサ)上でサポートされているかは、互換性ではなく、主にパフォーマンスによって決められています。両方のファミリでパフォーマンスが向上する場合は、同じ組込み関数が使用されます。

### すべての IA プロセッサでサポートされる組込み関数

表中の項目の意味は、次のとおりです。

- A = 組込み関数を使用しない同等のコードと比べて、パフォーマンスが大きく向上すると予想されるもの。
- B = 組込み関数を使用しないソースコードを使用した方がよいもの。この場合、組込み関数はネイティブ命令に直接に対応付けられますが、パフォーマンスはほとんど向上しません。
- C = 特定のマイクロアーキテクチャでは対応する命令が存在しないもの。組込み関数を使用すると、パフォーマンスが大きく低下します。

| 組込み関数                                                              | すべての IA | MMX® テクノロジ | ストリーミング SIMD 拡張命令 | ストリーミング SIMD 拡張命令2 | Itanium® アーキテクチャ |
|--------------------------------------------------------------------|---------|------------|-------------------|--------------------|------------------|
| <code>int abs(int)</code>                                          | A       | A          | A                 | A                  | A                |
| <code>long labs(long)</code>                                       | A       | A          | A                 | A                  | A                |
| <code>unsigned long __lrotl(unsigned long value, int shift)</code> | A       | A          | A                 | A                  | A                |
| <code>unsigned long __lrotr(unsigned long value, int shift)</code> | A       | A          | A                 | A                  | A                |
| <code>unsigned int __rotr(unsigned int value, int shift)</code>    | A       | A          | A                 | A                  | A                |



|                                                             |   |   |   |   |   |
|-------------------------------------------------------------|---|---|---|---|---|
| shift)                                                      |   |   |   |   |   |
| unsigned int<br>__rotr(unsigned<br>int value, int<br>shift) | A | A | A | A | A |
| __int64<br>__i64_rotl(__int6<br>4 value, int shift)         | A | A | A | A | A |
| __int64<br>__i64_rotr(__int6<br>4 value, int shift)         | A | A | A | A | A |
| double<br>fabs(double)                                      | A | A | A | A | A |
| double log(double)                                          | A | A | A | A | A |
| float logf(float)                                           | A | A | A | A | A |
| double<br>log10(double)                                     | A | A | A | A | A |
| float<br>log10f(float)                                      | A | A | A | A | A |
| double exp(double)                                          | A | A | A | A | A |
| float expf(float)                                           | A | A | A | A | A |
| double pow(double,<br>double)                               | A | A | A | A | A |
| float powf(float,<br>float)                                 | A | A | A | A | A |
| double sin(double)                                          | A | A | A | A | A |
| float sinf(float)                                           | A | A | A | A | A |
| double cos(double)                                          | A | A | A | A | A |
| float cosf(float)                                           | A | A | A | A | A |
| double tan(double)                                          | A | A | A | A | A |
| float tanf(float)                                           | A | A | A | A | A |
| double<br>acos(double)                                      | A | A | A | A | A |
| float acosf(float)                                          | A | A | A | A | A |
| double<br>acosh(double)                                     | A | A | A | A | A |
| float<br>acoshf(float)                                      | A | A | A | A | A |
| double<br>asin(double)                                      | A | A | A | A | A |
| float asinf(float)                                          | A | A | A | A | A |
| double<br>asinh(double)                                     | A | A | A | A | A |
| float<br>asinhf(float)                                      | A | A | A | A | A |

|                                                              |   |   |   |   |   |
|--------------------------------------------------------------|---|---|---|---|---|
| double<br>atan(double)                                       | A | A | A | A | A |
| float atanf(float)                                           | A | A | A | A | A |
| double<br>atanh(double)                                      | A | A | A | A | A |
| float<br>atanhf(float)                                       | A | A | A | A | A |
| float<br>cabs(double) *                                      | A | A | A | A | A |
| double<br>ceil(double)                                       | A | A | A | A | A |
| float ceilf(float)                                           | A | A | A | A | A |
| double<br>cosh(double)                                       | A | A | A | A | A |
| float coshf(float)                                           | A | A | A | A | A |
| float fabsf(float)                                           | A | A | A | A | A |
| double<br>floor(double)                                      | A | A | A | A | A |
| float<br>floorf(float)                                       | A | A | A | A | A |
| double<br>fmod(double)                                       | A | A | A | A | A |
| float fmodf(float)                                           | A | A | A | A | A |
| double<br>hypot(double,<br>double)                           | A | A | A | A | A |
| float<br>hypotf(float)                                       | A | A | A | A | A |
| double<br>rint(double)                                       | A | A | A | A | A |
| float rintf(float)                                           | A | A | A | A | A |
| double<br>sinh(double)                                       | A | A | A | A | A |
| float sinhf(float)                                           | A | A | A | A | A |
| float sqrtf(float)                                           | A | A | A | A | A |
| double<br>tanh(double)                                       | A | A | A | A | A |
| float tanhf(float)                                           | A | A | A | A | A |
| char * strset(char<br>*, int32)                              | A | A | A | A | A |
| void *memcpy(const<br>void *cs, const<br>void *ct, size t n) | A | A | A | A | A |
| void *memcpy(void<br>*s, const void *ct,                     | A | A | A | A | A |

|                                         |   |   |   |   |   |
|-----------------------------------------|---|---|---|---|---|
| size_t n)                               |   |   |   |   |   |
| void *memset(void *s, int c, size_t n)  | A | A | A | A | A |
| char *Strcat(char *s, const char *ct)   | A | A | A | A | A |
| int *strcmp(const char *, const char *) | A | A | A | A | A |
| char *strcpy(char *s, const char *ct)   | A | A | A | A | A |
| size_t strlen(const char *cs)           | A | A | A | A | A |
| int strncmp(char *, char *, int)        | A | A | A | A | A |
| int strncpy(char *, char *, int)        | A | A | A | A | A |
| void *alloca(int)                       | A | A | A | A | A |
| int setjmp(jmp_buf)                     | A | A | A | A | A |
| _exception_code(void)                   | A | A | A | A | A |
| _exception_info(void)                   | A | A | A | A | A |
| _abnormal_termination(void)             | A | A | A | A | A |
| void enable()                           | A | A | A | A | A |
| void disable()                          | A | A | A | A | A |
| int bswap(int)                          | A | A | A | A | A |
| int in_byte(int)                        | A | A | A | A | A |
| int in_dword(int)                       | A | A | A | A | A |
| int in_word(int)                        | A | A | A | A | A |
| int inp(int)                            | A | A | A | A | A |
| int inpd(int)                           | A | A | A | A | A |
| int inpw(int)                           | A | A | A | A | A |
| int out_byte(int, int)                  | A | A | A | A | A |
| int _out_dword(int, int)                | A | A | A | A | A |
| int out_word(int, int)                  | A | A | A | A | A |
| int outp(int, int)                      | A | A | A | A | A |

|                      |   |   |   |   |   |
|----------------------|---|---|---|---|---|
| int)                 |   |   |   |   |   |
| int _outpd(int, int) | A | A | A | A | A |
| int _outpw(int, int) | A | A | A | A | A |

## MMX® テクノロジーの組込み関数

表中の項目の意味は、次のとおりです。

- A = 組込み関数を使用しない同等のコードと比べて、パフォーマンスが大きく向上すると予想されるもの。
- B = 組込み関数を使用しないソースコードを使用した方がよいもの。この場合、組込み関数はネイティブ命令に直接に対応付けられますが、パフォーマンスはほとんど向上しません。
- C = 特定のマイクロアーキテクチャでは対応する命令が存在しないもの。組込み関数を使用すると、パフォーマンスが大きく低下します。

| 組込み関数名       | 別名                | すべてのIA | MMX®テクノロジー | ストリーミングSIMD拡張命令 | ストリーミングSIMD拡張命令2 | Itanium®アーキテクチャ |
|--------------|-------------------|--------|------------|-----------------|------------------|-----------------|
| _m_empty     | _mm_empty         | N/A    | A          | A               | A                | B               |
| _m_from_int  | _mm_cvt_si32_si64 | N/A    | A          | A               | A                | A               |
| _m_to_int    | _mm_cvt_si64_si32 | N/A    | A          | A               | A                | A               |
| _m_packsswb  | _mm_packs_pi16    | N/A    | A          | A               | A                | A               |
| _m_packssdw  | _mm_packs_pi32    | N/A    | A          | A               | A                | A               |
| _m_packuswb  | _mm_packs_pu16    | N/A    | A          | A               | A                | A               |
| _m_punpckhbw | _mm_unpackhi_pi8  | N/A    | A          | A               | A                | A               |
| _m_punpckhwd | _mm_unpackhi_pi16 | N/A    | A          | A               | A                | A               |
| _m_punpckhdq | _mm_unpackhi_pi32 | N/A    | A          | A               | A                | A               |

|                  |                       |     |   |   |   |   |
|------------------|-----------------------|-----|---|---|---|---|
| _m_punp<br>cklbw | _mm_unp<br>acklo_pi8  | N/A | A | A | A | A |
| _m_punp<br>cklwd | _mm_unp<br>acklo_pi16 | N/A | A | A | A | A |
| _m_punp<br>ckldq | _mm_unp<br>acklo_pi32 | N/A | A | A | A | A |
| _m_padd<br>b     | _mm_add<br>pi8        | N/A | A | A | A | A |
| _m_padd<br>w     | _mm_add<br>pi16       | N/A | A | A | A | A |
| _m_padd<br>d     | _mm_add<br>pi32       | N/A | A | A | A | A |
| _m_padd<br>sb    | _mm_add<br>s_pi8      | N/A | A | A | A | A |
| _m_padd<br>sw    | _mm_add<br>s_pi16     | N/A | A | A | A | A |
| _m_padd<br>usb   | _mm_add<br>s_pu8      | N/A | A | A | A | A |
| _m_padd<br>usw   | _mm_add<br>s_pu16     | N/A | A | A | A | A |
| _m_psub<br>b     | _mm_sub<br>pi8        | N/A | A | A | A | A |
| _m_psub<br>w     | _mm_sub<br>pi16       | N/A | A | A | A | A |
| _m_psub<br>d     | _mm_sub<br>pi32       | N/A | A | A | A | A |
| _m_psub<br>sb    | _mm_sub<br>s_pi8      | N/A | A | A | A | A |
| _m_psub<br>sw    | _mm_sub<br>s_pi16     | N/A | A | A | A | A |
| _m_psub<br>usb   | _mm_sub<br>s_pu8      | N/A | A | A | A | A |
| _m_psub<br>usw   | _mm_sub<br>s_pu16     | N/A | A | A | A | A |
| _m_pmad<br>dwd   | _mm_mad<br>d_pi16     | N/A | A | A | A | C |
| _m_pmul<br>hw    | _mm_mul<br>hi_pi16    | N/A | A | A | A | A |
| _m_pmul<br>lw    | _mm_mul<br>lo_pi16    | N/A | A | A | A | A |
| _m_psl1<br>w     | _mm_sl1<br>pi16       | N/A | A | A | A | A |

|                |                     |     |   |   |   |   |
|----------------|---------------------|-----|---|---|---|---|
| _m_psl1<br>wi  | _mm_sll<br>i_pi16   | N/A | A | A | A | A |
| _m_psl1<br>d   | _mm_sll<br>pi32     | N/A | A | A | A | A |
| _m_psl1<br>di  | _mm_sll<br>i_pi32   | N/A | A | A | A | A |
| _m_psl1<br>q   | _mm_sll<br>si64     | N/A | A | A | A | A |
| _m_psl1<br>qi  | _mm_sll<br>i_si64   | N/A | A | A | A | A |
| _m_psra<br>w   | _mm_sra<br>pi16     | N/A | A | A | A | A |
| _m_psra<br>wi  | _mm_sra<br>i_pi16   | N/A | A | A | A | A |
| _m_psra<br>d   | _mm_sra<br>pi32     | N/A | A | A | A | A |
| _m_psra<br>di  | _mm_sra<br>i_pi32   | N/A | A | A | A | A |
| _m_psrl<br>w   | _mm_srl<br>pi16     | N/A | A | A | A | A |
| _m_psrl<br>wi  | _mm_srl<br>i_pi16   | N/A | A | A | A | A |
| _m_psrl<br>d   | _mm_srl<br>pi32     | N/A | A | A | A | A |
| _m_psrl<br>di  | _mm_srl<br>i_pi32   | N/A | A | A | A | A |
| _m_psrl<br>q   | _mm_srl<br>si64     | N/A | A | A | A | A |
| _m_psrl<br>qi  | _mm_srl<br>i_si64   | N/A | A | A | A | A |
| _m_pand<br>n   | _mm_and<br>si64     | N/A | A | A | A | A |
| _m_pand<br>n   | _mm_and<br>not_si64 | N/A | A | A | A | A |
| _m_por         | _mm_or_<br>si64     | N/A | A | A | A | A |
| _m_pxor        | _mm_xor_<br>si64    | N/A | A | A | A | A |
| _m_pcmp<br>eqb | _mm_cmp<br>eq_pi8   | N/A | A | A | A | A |
| _m_pcmp<br>eqw | _mm_cmp<br>eq_pi16  | N/A | A | A | A | A |
| _m_pcmp<br>eqd | _mm_cmp<br>eq_pi32  | N/A | A | A | A | A |

|                                |                              |     |   |   |   |   |
|--------------------------------|------------------------------|-----|---|---|---|---|
| <code>_m_pcmp_gtb</code>       | <code>_mm_cmp_gt_pi8</code>  | N/A | A | A | A | A |
| <code>_m_pcmp_gtw</code>       | <code>_mm_cmp_gt_pi16</code> | N/A | A | A | A | A |
| <code>_m_pcmp_gtd</code>       | <code>_mm_cmp_gt_pi32</code> | N/A | A | A | A | A |
| <code>_mm_set_zero_si64</code> |                              | N/A | A | A | A | A |
| <code>_mm_set_pi32</code>      |                              | N/A | A | A | A | A |
| <code>_mm_set_pi16</code>      |                              | N/A | A | A | A | C |
| <code>_mm_set_pi8</code>       |                              | N/A | A | A | A | C |
| <code>_mm_set1_pi32</code>     |                              | N/A | A | A | A | A |
| <code>_mm_set1_pi16</code>     |                              | N/A | A | A | A | A |
| <code>_mm_set1_pi8</code>      |                              | N/A | A | A | A | A |
| <code>_mm_setr_pi32</code>     |                              | N/A | A | A | A | A |
| <code>_mm_setr_pi16</code>     |                              | N/A | A | A | A | C |
| <code>_mm_setr_pi8</code>      |                              | N/A | A | A | A | C |

`_mm_empty` は、Itanium 命令では、ソースの互換性のためにのみ NOP としてサポートしています。

## ストリーミングSIMD拡張命令の組込み関数

通常のストリーミングSIMD拡張命令の組込み関数は、4個の32ビット単精度浮動小数点値を操作します。Itanium® ベースのシステム上では、加算や比較などの基本演算に2つのSIMD命令が必要です。2つのSIMD命令を同じサイクルで実行できるため、スループットは、1サイクル当たり1つのストリーミングSIMD拡張命令基本演算、すなわち1サイクル当たり4個の32ビット単精度浮動小数点値の演算になります。

表中の項目の意味は、次のとおりです。

- A = 組込み関数を使用しない同等のコードと比べて、パフォーマンスが大きく向上すると予想されるもの。
- B = 組込み関数を使用しないソース・コードを使用した方がよいもの。組込み関数はネイティブ命令に直接に対応付けられますが、パフォーマンスはほとんど向上しません。

- C = 特定のマイクロアーキテクチャでは対応する命令が存在しないもの。組込み関数を使用すると、パフォーマンスが大きく低下します。

| 組込み関数名            | 別名 | すべての<br>IA | MMX®<br>テクノロ<br>ジ | ストリーミ<br>ング<br>SIMD拡<br>張命令 | ストリーミ<br>ング<br>SIMD<br>拡張命<br>令2 | Itanium<br>アーキ<br>テクチャ |
|-------------------|----|------------|-------------------|-----------------------------|----------------------------------|------------------------|
| mm add ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm add ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm sub ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm sub ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm mul ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm mul ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm div ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm div ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm sqrt ss        |    | N/A        | N/A               | B                           | B                                | B                      |
| mm sqrt ps        |    | N/A        | N/A               | A                           | A                                | A                      |
| mm rcp ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm rcp ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm rsqrt ss       |    | N/A        | N/A               | B                           | B                                | B                      |
| mm rsqrt ps       |    | N/A        | N/A               | A                           | A                                | A                      |
| mm min ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm min ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm max ss         |    | N/A        | N/A               | B                           | B                                | B                      |
| mm max ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm and ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| _mm_andnot_p<br>s |    | N/A        | N/A               | A                           | A                                | A                      |
| mm or ps          |    | N/A        | N/A               | A                           | A                                | A                      |
| mm xor ps         |    | N/A        | N/A               | A                           | A                                | A                      |
| mm cmpeq ss       |    | N/A        | N/A               | B                           | B                                | B                      |
| mm cmpeq ps       |    | N/A        | N/A               | A                           | A                                | A                      |
| mm cmplt ss       |    | N/A        | N/A               | B                           | B                                | B                      |
| mm cmplt ps       |    | N/A        | N/A               | A                           | A                                | A                      |
| mm cmple ss       |    | N/A        | N/A               | B                           | B                                | B                      |
| mm cmple ps       |    | N/A        | N/A               | A                           | A                                | A                      |
| mm cmpgt ss       |    | N/A        | N/A               | B                           | B                                | B                      |
| mm cmpgt ps       |    | N/A        | N/A               | A                           | A                                | A                      |
| mm cmpge ss       |    | N/A        | N/A               | B                           | B                                | B                      |
| mm cmpge ps       |    | N/A        | N/A               | A                           | A                                | A                      |
| _mm_cmpneq_s      |    | N/A        | N/A               | B                           | B                                | B                      |
| _mm_cmpneq_p<br>s |    | N/A        | N/A               | A                           | A                                | A                      |



|                 |  |     |     |   |   |   |
|-----------------|--|-----|-----|---|---|---|
| _mm_cmpnlt_s    |  | N/A | N/A | B | B | B |
| _mm_cmpnlt_p    |  | N/A | N/A | A | A | A |
| _mm_cmpnle_s    |  | N/A | N/A | B | B | B |
| _mm_cmpnle_p    |  | N/A | N/A | A | A | A |
| _mm_cmpngt_s    |  | N/A | N/A | B | B | B |
| _mm_cmpngt_p    |  | N/A | N/A | A | A | A |
| _mm_cmpnge_s    |  | N/A | N/A | B | B | B |
| _mm_cmpnge_p    |  | N/A | N/A | A | A | A |
| _mm_cmpord_s    |  | N/A | N/A | B | B | B |
| _mm_cmpord_p    |  | N/A | N/A | A | A | A |
| _mm_cmpunord_ss |  | N/A | N/A | B | B | B |
| _mm_cmpunord_ps |  | N/A | N/A | A | A | A |
| _mm_comieq_s    |  | N/A | N/A | B | B | B |
| _mm_comilt_s    |  | N/A | N/A | B | B | B |
| _mm_comile_s    |  | N/A | N/A | B | B | B |
| _mm_comigt_s    |  | N/A | N/A | B | B | B |
| _mm_comige_s    |  | N/A | N/A | B | B | B |
| _mm_comineq_ss  |  | N/A | N/A | B | B | B |
| _mm_ucomieq_ss  |  | N/A | N/A | B | B | B |
| _mm_ucomilt_ss  |  | N/A | N/A | B | B | B |
| _mm_ucomile_ss  |  | N/A | N/A | B | B | B |
| _mm_ucomigt_ss  |  | N/A | N/A | B | B | B |
| mm ucomige      |  | N/A | N/A | B | B | B |

|                   |                 |     |     |   |   |   |
|-------------------|-----------------|-----|-----|---|---|---|
| ss                |                 |     |     |   |   |   |
| _mm_ucomineqss    |                 | N/A | N/A | B | B | B |
| _mm_cvt_ss2si     | _mm_cvtss_si32  | N/A | N/A | A | A | B |
| int _mm_cvt_ps2pi | _mm_cvtps_pi32  | N/A | N/A | A | A | A |
| _mm_cvtt_ss2si    | _mm_cvttss_si32 | N/A | N/A | A | A | B |
| _mm_cvtt_ps2pi    | _mm_cvttps_pi32 | N/A | N/A | A | A | A |
| _mm_cvt_si2ss     | _mm_cvtss_si32  | N/A | N/A | A | A | B |
| _mm_cvt_pi2ps     | _mm_cvtps_pi32  | N/A | N/A | A | A | C |
| _mm_cvtpi16_ps    |                 | N/A | N/A | A | A | C |
| _mm_cvtpu16_ps    |                 | N/A | N/A | A | A | C |
| _mm_cvtpi8_ps     |                 | N/A | N/A | A | A | C |
| _mm_cvtpu8_ps     |                 | N/A | N/A | A | A | C |
| _mm_cvtpi32x2_ps  |                 | N/A | N/A | A | A | C |
| _mm_cvtps_pi16    |                 | N/A | N/A | A | A | C |
| _mm_cvtps_pi8     |                 | N/A | N/A | A | A | C |
| mm move ss        |                 | N/A | N/A | A | A | A |
| _mm_shuffle_ps    |                 | N/A | N/A | A | A | A |
| _mm_unpackhi_ps   |                 | N/A | N/A | A | A | A |
| _mm_unpacklo_ps   |                 | N/A | N/A | A | A | A |
| _mm_movehl_ps     |                 | N/A | N/A | A | A | A |
| _mm_movelh_ps     |                 | N/A | N/A | A | A | A |
| _mm_movemask_ps   |                 | N/A | N/A | A | A | C |

|                |                  |     |     |   |   |   |
|----------------|------------------|-----|-----|---|---|---|
| mm_getcsr      |                  | N/A | N/A | A | A | A |
| mm_setcsr      |                  | N/A | N/A | A | A | A |
| mm_loadh_pi    |                  | N/A | N/A | A | A | A |
| mm_loadl_pi    |                  | N/A | N/A | A | A | A |
| mm_load_ss     |                  | N/A | N/A | A | A | B |
| _mm_load_ps1   | _mm_loadl_ps     | N/A | N/A | A | A | A |
| mm_load_ps     |                  | N/A | N/A | A | A | A |
| mm_loadu_ps    |                  | N/A | N/A | A | A | A |
| mm_loadr_ps    |                  | N/A | N/A | A | A | A |
| _mm_storeh_pi  |                  | N/A | N/A | A | A | A |
| _mm_storel_pi  |                  | N/A | N/A | A | A | A |
| mm_store_ss    |                  | N/A | N/A | A | A | A |
| mm_store_ps    |                  | N/A | N/A | A | A | A |
| _mm_store_ps1  | _mm_storel_ps    | N/A | N/A | A | A | A |
| _mm_storeu_ps  |                  | N/A | N/A | A | A | A |
| _mm_storer_ps  |                  | N/A | N/A | A | A | A |
| mm_set_ss      |                  | N/A | N/A | A | A | A |
| _mm_set_ps1    | _mm_setl_ps      | N/A | N/A | A | A | A |
| mm_set_ps      |                  | N/A | N/A | A | A | A |
| mm_setr_ps     |                  | N/A | N/A | A | A | A |
| _mm_setzero_ps |                  | N/A | N/A | A | A | A |
| mm_prefetch    |                  | N/A | N/A | A | A | A |
| _mm_stream_pi  |                  | N/A | N/A | A | A | A |
| _mm_stream_ps  |                  | N/A | N/A | A | A | A |
| mm_sfence      |                  | N/A | N/A | A | A | A |
| _mm_pextrw     | _mm_extract_pi16 | N/A | N/A | A | A | A |
| _mm_pinsrw     | _mm_insert_pi16  | N/A | N/A | A | A | A |
| _mm_pmaxsw     | _mm_max_pi16     | N/A | N/A | A | A | A |
| _mm_pmaxub     | _mm_max_pu8      | N/A | N/A | A | A | A |

|             |                           |     |     |   |   |   |
|-------------|---------------------------|-----|-----|---|---|---|
| _m_pminsw   | _mm_min<br>pi16           | N/A | N/A | A | A | A |
| _m_pminub   | _mm_min<br>pu8            | N/A | N/A | A | A | A |
| _m_pmovmskb | _mm_mov<br>emask_p<br>i8  | N/A | N/A | A | A | C |
| _m_pmulhuw  | _mm_mul<br>hi_pu16        | N/A | N/A | A | A | A |
| _m_pshufw   | _mm_shu<br>ffle_pi<br>16  | N/A | N/A | A | A | A |
| _m_maskmovq | _mm_mas<br>kmove_s<br>i64 | N/A | N/A | A | A | C |
| _m_pavgb    | _mm_avg<br>pu8            | N/A | N/A | A | A | A |
| _m_pavgw    | _mm_avg<br>pu16           | N/A | N/A | A | A | A |
| _m_psadbw   | _mm_sad<br>pu8            | N/A | N/A | A | A | A |

## ストリーミングSIMD拡張命令2 の組込み関数

ストリーミングSIMD拡張命令2 では、128ビットデータ(2個の64ビット倍精度浮動小数点値)を操作します。Itanium® ベースのシステム上ではストリーミングSIMD拡張命令2 はサポートされていません。

表中の項目の意味は、次のとおりです。

- A = 組込み関数を使用しない同等のコードと比べて、パフォーマンスが大きく向上すると予想されるもの。
- B = 組込み関数を使用しないソースコードを使用した方がよいもの。この場合、組込み関数はネイティブ命令に直接に対応付けられますが、パフォーマンスはほとんど向上しません。
- C = 特定のマイクロアーキテクチャでは対応する命令が存在しないもの。組込み関数を使用すると、パフォーマンスが大きく低下します。

| 組込み関数     | すべての<br>IA | MMX®<br>テクノロジー | ストリーミング<br>SIMD拡張<br>命令 | インテル<br>Pentium®<br>4 プロセッ<br>サ<br>ストリーミ<br>ングSIMD<br>拡張命令2 | Itanium<br>アーキテ<br>クチャ |
|-----------|------------|----------------|-------------------------|-------------------------------------------------------------|------------------------|
| mm_add_sd | N/A        | N/A            | N/A                     | A                                                           | N/A                    |

|                  |     |     |     |   |     |
|------------------|-----|-----|-----|---|-----|
| mm add pd        | N/A | N/A | N/A | A | N/A |
| mm sub sd        | N/A | N/A | N/A | A | N/A |
| mm sub pd        | N/A | N/A | N/A | A | N/A |
| mm mul sd        | N/A | N/A | N/A | A | N/A |
| mm mul pd        | N/A | N/A | N/A | A | N/A |
| mm sqrt sd       | N/A | N/A | N/A | A | N/A |
| mm sqrt pd       | N/A | N/A | N/A | A | N/A |
| mm div sd        | N/A | N/A | N/A | A | N/A |
| mm div pd        | N/A | N/A | N/A | A | N/A |
| mm min sd        | N/A | N/A | N/A | A | N/A |
| mm min pd        | N/A | N/A | N/A | A | N/A |
| mm max sd        | N/A | N/A | N/A | A | N/A |
| mm max pd        | N/A | N/A | N/A | A | N/A |
| mm and pd        | N/A | N/A | N/A | A | N/A |
| mm_andnot_p<br>d | N/A | N/A | N/A | A | N/A |
| mm or pd         | N/A | N/A | N/A | A | N/A |
| mm xor pd        | N/A | N/A | N/A | A | N/A |
| mm cmpeq sd      | N/A | N/A | N/A | A | N/A |
| mm cmpeq pd      | N/A | N/A | N/A | A | N/A |
| mm cmplt sd      | N/A | N/A | N/A | A | N/A |
| mm cmplt pd      | N/A | N/A | N/A | A | N/A |
| mm cmple sd      | N/A | N/A | N/A | A | N/A |
| mm cmple pd      | N/A | N/A | N/A | A | N/A |
| mm cmpgt sd      | N/A | N/A | N/A | A | N/A |
| mm cmpgt pd      | N/A | N/A | N/A | A | N/A |
| mm cmpge sd      | N/A | N/A | N/A | A | N/A |
| mm cmpge pd      | N/A | N/A | N/A | A | N/A |
| mm_cmpneq_s<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpneq_p<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpnlt_s<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpnlt_p<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpnle_s<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpnle_p<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpngt_s<br>d | N/A | N/A | N/A | A | N/A |
| mm_cmpngt_p<br>d | N/A | N/A | N/A | A | N/A |

|                  |     |     |     |   |     |
|------------------|-----|-----|-----|---|-----|
| _mm_cmpnge_sd    | N/A | N/A | N/A | A | N/A |
| _mm_cmpnge_pd    | N/A | N/A | N/A | A | N/A |
| _mm_cmpord_pd    | N/A | N/A | N/A | A | N/A |
| _mm_cmpord_sd    | N/A | N/A | N/A | A | N/A |
| _mm_cmpunord_pd  | N/A | N/A | N/A | A | N/A |
| _mm_cmpunord_sd  | N/A | N/A | N/A | A | N/A |
| _mm_comieq_sd    | N/A | N/A | N/A | A | N/A |
| _mm_comilt_sd    | N/A | N/A | N/A | A | N/A |
| _mm_comile_sd    | N/A | N/A | N/A | A | N/A |
| _mm_comigt_sd    | N/A | N/A | N/A | A | N/A |
| _mm_comige_sd    | N/A | N/A | N/A | A | N/A |
| _mm_comineq_sd   | N/A | N/A | N/A | A | N/A |
| _mm_ucomieq_sd   | N/A | N/A | N/A | A | N/A |
| _mm_ucomilt_sd   | N/A | N/A | N/A | A | N/A |
| _mm_ucomile_sd   | N/A | N/A | N/A | A | N/A |
| _mm_ucomigt_sd   | N/A | N/A | N/A | A | N/A |
| _mm_ucomige_sd   | N/A | N/A | N/A | A | N/A |
| _mm_ucomineq_sd  | N/A | N/A | N/A | A | N/A |
| _mm_cvtepi32_pd  | N/A | N/A | N/A | A | N/A |
| _mm_cvtpd_epi32  | N/A | N/A | N/A | A | N/A |
| _mm_cvttpd_epi32 | N/A | N/A | N/A | A | N/A |
| _mm_cvtepi32_ps  | N/A | N/A | N/A | A | N/A |
| _mm_cvtps_epi32  | N/A | N/A | N/A | A | N/A |

|                    |     |     |     |   |     |
|--------------------|-----|-----|-----|---|-----|
| i32                |     |     |     |   |     |
| _mm_cvttps_epi32   | N/A | N/A | N/A | A | N/A |
| _mm_cvtpd_ps       | N/A | N/A | N/A | A | N/A |
| _mm_cvtps_pd       | N/A | N/A | N/A | A | N/A |
| _mm_cvtsd_ss       | N/A | N/A | N/A | A | N/A |
| _mm_cvtss_sd       | N/A | N/A | N/A | A | N/A |
| _mm_cvtsd_si32     | N/A | N/A | N/A | A | N/A |
| _mm_cvttss_sd      | N/A | N/A | N/A | A | N/A |
| _mm_cvtsi32_sd     | N/A | N/A | N/A | A | N/A |
| _mm_cvtpd_pi32     | N/A | N/A | N/A | A | N/A |
| _mm_cvttpd_pi32    | N/A | N/A | N/A | A | N/A |
| _mm_cvtpi32_pd     | N/A | N/A | N/A | A | N/A |
| _mm_unpackhi_epi32 | N/A | N/A | N/A | A | N/A |
| _mm_unpacklo_epi32 | N/A | N/A | N/A | A | N/A |
| _mm_shuffle_epi32  | N/A | N/A | N/A | A | N/A |
| _mm_load_pd        | N/A | N/A | N/A | A | N/A |
| _mm_loadl_pd       | N/A | N/A | N/A | A | N/A |
| _mm_loadr_pd       | N/A | N/A | N/A | A | N/A |
| _mm_loadu_pd       | N/A | N/A | N/A | A | N/A |
| _mm_load_sd        | N/A | N/A | N/A | A | N/A |
| _mm_loadh_pd       | N/A | N/A | N/A | A | N/A |
| _mm_loadl_pd       | N/A | N/A | N/A | A | N/A |
| _mm_set_sd         | N/A | N/A | N/A | A | N/A |
| _mm_setl_pd        | N/A | N/A | N/A | A | N/A |
| _mm_set_pd         | N/A | N/A | N/A | A | N/A |
| _mm_setr_pd        | N/A | N/A | N/A | A | N/A |
| _mm_setzero_pd     | N/A | N/A | N/A | A | N/A |
| _mm_move_sd        | N/A | N/A | N/A | A | N/A |
| _mm_store_sd       | N/A | N/A | N/A | A | N/A |
| _mm_storel_pd      | N/A | N/A | N/A | A | N/A |
| _mm_store_pd       | N/A | N/A | N/A | A | N/A |

|                 |     |     |     |   |     |
|-----------------|-----|-----|-----|---|-----|
| _mm_storeu_pd   | N/A | N/A | N/A | A | N/A |
| _mm_storer_pd   | N/A | N/A | N/A | A | N/A |
| _mm_storeh_pd   | N/A | N/A | N/A | A | N/A |
| _mm_storel_pd   | N/A | N/A | N/A | A | N/A |
| _mm_add_epi8    | N/A | N/A | N/A | A | N/A |
| _mm_add_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_add_epi32   | N/A | N/A | N/A | A | N/A |
| _mm_add_si64    | N/A | N/A | N/A | A | N/A |
| _mm_add_epi64   | N/A | N/A | N/A | A | N/A |
| _mm_adds_epi8   | N/A | N/A | N/A | A | N/A |
| _mm_adds_epi16  | N/A | N/A | N/A | A | N/A |
| _mm_adds_epu8   | N/A | N/A | N/A | A | N/A |
| _mm_adds_epu16  | N/A | N/A | N/A | A | N/A |
| _mm_avg_epu8    | N/A | N/A | N/A | A | N/A |
| _mm_avg_epu16   | N/A | N/A | N/A | A | N/A |
| _mm_madd_epi16  | N/A | N/A | N/A | A | N/A |
| _mm_max_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_max_epu8    | N/A | N/A | N/A | A | N/A |
| _mm_min_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_min_epu8    | N/A | N/A | N/A | A | N/A |
| _mm_mulhi_epi16 | N/A | N/A | N/A | A | N/A |
| _mm_mulhi_epu16 | N/A | N/A | N/A | A | N/A |
| _mm_mullo_epi16 | N/A | N/A | N/A | A | N/A |
| _mm_mul_su32    | N/A | N/A | N/A | A | N/A |
| _mm_mul_epu32   | N/A | N/A | N/A | A | N/A |
| _mm_sad_epu8    | N/A | N/A | N/A | A | N/A |



|                 |     |     |     |   |     |
|-----------------|-----|-----|-----|---|-----|
| mm_sub_epi8     | N/A | N/A | N/A | A | N/A |
| mm_sub_epi16    | N/A | N/A | N/A | A | N/A |
| mm_sub_epi32    | N/A | N/A | N/A | A | N/A |
| mm_sub_si64     | N/A | N/A | N/A | A | N/A |
| mm_sub_epi64    | N/A | N/A | N/A | A | N/A |
| mm_subs_epi8    | N/A | N/A | N/A | A | N/A |
| mm_subs_epi16   | N/A | N/A | N/A | A | N/A |
| mm_subs_epu8    | N/A | N/A | N/A | A | N/A |
| mm_subs_epu16   | N/A | N/A | N/A | A | N/A |
| mm_and_si128    | N/A | N/A | N/A | A | N/A |
| mm_andnot_si128 | N/A | N/A | N/A | A | N/A |
| mm_or_si128     | N/A | N/A | N/A | A | N/A |
| mm_xor_si128    | N/A | N/A | N/A | A | N/A |
| mm_slli_si128   | N/A | N/A | N/A | A | N/A |
| mm_slli_epi16   | N/A | N/A | N/A | A | N/A |
| mm_sll_epi16    | N/A | N/A | N/A | A | N/A |
| mm_slli_epi32   | N/A | N/A | N/A | A | N/A |
| mm_sll_epi32    | N/A | N/A | N/A | A | N/A |
| mm_slli_epi64   | N/A | N/A | N/A | A | N/A |
| mm_sll_epi64    | N/A | N/A | N/A | A | N/A |
| mm_srai_epi16   | N/A | N/A | N/A | A | N/A |
| mm_sra_epi16    | N/A | N/A | N/A | A | N/A |
| mm_srai_epi32   | N/A | N/A | N/A | A | N/A |
| mm_sra_epi32    | N/A | N/A | N/A | A | N/A |

|                   |     |     |     |   |     |
|-------------------|-----|-----|-----|---|-----|
| _mm_srli_si128    | N/A | N/A | N/A | A | N/A |
| _mm_srli_epi16    | N/A | N/A | N/A | A | N/A |
| _mm_srl_epi16     | N/A | N/A | N/A | A | N/A |
| _mm_srli_epi32    | N/A | N/A | N/A | A | N/A |
| _mm_srl_epi32     | N/A | N/A | N/A | A | N/A |
| _mm_srli_epi64    | N/A | N/A | N/A | A | N/A |
| _mm_srl_epi64     | N/A | N/A | N/A | A | N/A |
| _mm_cmpeq_epi8    | N/A | N/A | N/A | A | N/A |
| _mm_cmpeq_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_cmpeq_epi32   | N/A | N/A | N/A | A | N/A |
| _mm_cmpgt_epi8    | N/A | N/A | N/A | A | N/A |
| _mm_cmpgt_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_cmpgt_epi32   | N/A | N/A | N/A | A | N/A |
| _mm_cmplt_epi8    | N/A | N/A | N/A | A | N/A |
| _mm_cmplt_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_cmplt_epi32   | N/A | N/A | N/A | A | N/A |
| _mm_cvtsi32_si128 | N/A | N/A | N/A | A | N/A |
| _mm_cvtsi128_si32 | N/A | N/A | N/A | A | N/A |
| _mm_packs_epi16   | N/A | N/A | N/A | A | N/A |
| _mm_packs_epi32   | N/A | N/A | N/A | A | N/A |
| _mm_packus_epi16  | N/A | N/A | N/A | A | N/A |
| _mm_extract_epi16 | N/A | N/A | N/A | A | N/A |
| _mm_insert_epi16  | N/A | N/A | N/A | A | N/A |

|                         |     |     |     |   |     |
|-------------------------|-----|-----|-----|---|-----|
| pi16                    |     |     |     |   |     |
| _mm_movemask<br>_epi8   | N/A | N/A | N/A | A | N/A |
| _mm_shuffle_<br>epi32   | N/A | N/A | N/A | A | N/A |
| _mm_shuffleh<br>i_epi16 | N/A | N/A | N/A | A | N/A |
| _mm_shufflel<br>o_epi16 | N/A | N/A | N/A | A | N/A |
| _mm_unpackhi<br>_epi8   | N/A | N/A | N/A | A | N/A |
| _mm_unpackhi<br>_epi16  | N/A | N/A | N/A | A | N/A |
| _mm_unpackhi<br>_epi32  | N/A | N/A | N/A | A | N/A |
| _mm_unpackhi<br>_epi64  | N/A | N/A | N/A | A | N/A |
| _mm_unpacklo<br>_epi8   | N/A | N/A | N/A | A | N/A |
| _mm_unpacklo<br>_epi16  | N/A | N/A | N/A | A | N/A |
| _mm_unpacklo<br>_epi32  | N/A | N/A | N/A | A | N/A |
| _mm_unpacklo<br>_epi64  | N/A | N/A | N/A | A | N/A |
| _mm_move_epi<br>64      | N/A | N/A | N/A | A | N/A |
| _mm_movpi64_<br>_epi64  | N/A | N/A | N/A | A | N/A |
| _mm_movepi64<br>_pi64   | N/A | N/A | N/A | A | N/A |
| _mm_load_si1<br>28      | N/A | N/A | N/A | A | N/A |
| _mm_loadu_si<br>128     | N/A | N/A | N/A | A | N/A |
| _mm_loadl_ep<br>i64     | N/A | N/A | N/A | A | N/A |
| _mm_set_epi6<br>4       | N/A | N/A | N/A | A | N/A |
| _mm_set_epi3<br>2       | N/A | N/A | N/A | A | N/A |
| _mm_set_epi1<br>6       | N/A | N/A | N/A | A | N/A |
| mm set epi8             | N/A | N/A | N/A | A | N/A |
| mm setl epi             | N/A | N/A | N/A | A | N/A |

|                     |     |     |     |   |     |
|---------------------|-----|-----|-----|---|-----|
| 64                  |     |     |     |   |     |
| _mm_set1_epi32      | N/A | N/A | N/A | A | N/A |
| _mm_set1_epi16      | N/A | N/A | N/A | A | N/A |
| _mm_set1_epi8       | N/A | N/A | N/A | A | N/A |
| _mm_setr_epi64      | N/A | N/A | N/A | A | N/A |
| _mm_setr_epi32      | N/A | N/A | N/A | A | N/A |
| _mm_setr_epi16      | N/A | N/A | N/A | A | N/A |
| _mm_setr_epi8       | N/A | N/A | N/A | A | N/A |
| _mm_setzero_si128   | N/A | N/A | N/A | A | N/A |
| _mm_store_si128     | N/A | N/A | N/A | A | N/A |
| _mm_storeu_si128    | N/A | N/A | N/A | A | N/A |
| _mm_storel_epi64    | N/A | N/A | N/A | A | N/A |
| _mm_maskmoveu_si128 | N/A | N/A | N/A | A | N/A |
| _mm_stream_pd       | N/A | N/A | N/A | A | N/A |
| _mm_stream_si128    | N/A | N/A | N/A | A | N/A |
| mm_clflush          | N/A | N/A | N/A | A | N/A |
| mm_lfence           | N/A | N/A | N/A | A | N/A |
| mm_mfence           | N/A | N/A | N/A | A | N/A |
| _mm_stream_si32     | N/A | N/A | N/A | A | N/A |
| mm_pause            | N/A | N/A | N/A | A | N/A |

# インテル(R) C++ クラス・ライブラリ

## クラス・ライブラリの紹介

### はじめに

インテル® C++ クラス・ライブラリを使用するとSIMD(Single-Instruction, Multiple-Data)演算を実行できます。SIMD演算の原理は、マイクロプロセッサ・アーキテクチャを並列処理に活用することです。並列処理を使用すると、より少ないクロックサイクル数で、より高いデータ・スループットが得られます。その目的は、オーディオ、ビデオ、グラフィックなど複雑で大量の計算を必要とするデータ・ビット・ストリームの処理効率を高めることです。

### ハードウェアとソフトウェアの要件

クラス・ライブラリを使用するには、インテル® C++ コンパイラ(バージョン4.0以上)をインストールしなければなりません。インテル C++ のクラス・ライブラリとは、次の表に示すように、各種インテル・プロセッサで使用できるすべての拡張命令の中からいくつか選び出した関数のことです。

#### クラス・ライブラリを使用するプロセッサの必要条件

| ヘッダファイル | 拡張命令セット          | 対応プロセッサ                                                                                                          |
|---------|------------------|------------------------------------------------------------------------------------------------------------------|
| ivec.h  | MMX テクノロジ        | MMX® テクノロジ Pentium® プロセッサ、Pentium II プロセッサ、Pentium III プロセッサ、Pentium 4 プロセッサ、インテル® Xeon™ プロセッサおよび Itanium® プロセッサ |
| fvec.h  | ストリーミングSIMD拡張命令  | Pentium III プロセッサ、Pentium 4 プロセッサ、インテル Xeon プロセッサおよび Itanium プロセッサ                                               |
| dvec.h  | ストリーミングSIMD拡張命令2 | Pentium 4 プロセッサおよびインテル Xeon プロセッサ                                                                                |

### クラスについて

SIMD演算用のインテル® C++ クラス・ライブラリには次のものがあります。

- 整数ベクトルクラス (Ivec)
- 浮動小数点ベクトルクラス (Fvec)

これら演算の定義は、ivec.h、fvec.h、およびdvec.hという3つのヘッダファイルに記述しています。クラス自体がこのように区分されているわけではありません。各クラスの名前は、基本となる演算の種類に従って付けられたものです。ヘッダファイルはアーキテクチャの種類に従って区分しています。

- ivec.h は MMX® テクノロジ命令のアーキテクチャ専用です
- fvec.h はストリーミングSIMD拡張命令のアーキテクチャ専用です
- dvec.h はストリーミングSIMD拡張命令2 のアーキテクチャ専用です

ストリーミングSIMD拡張命令2の組込み関数は Itanium® ベースのシステムでは使用できません。ヘッダファイル`mmclass.h`には、Itanium アーキテクチャで使用可能な各クラスが含まれています。

本書は、インテル・アーキテクチャ用のコードを作成するプログラマに向けて書かれたものですが、中でもSIMD命令を活用するコードを作成するプログラマを対象としています。C++ と C++ の各クラスの使用に精通する必要があります。

## 技術的な概要

### ライブラリの詳細

SIMD演算用のインテル® C++ コンパイラのクラス・ライブラリは、「クラス・ライブラリを使用するプロセッサの必要条件」に示した各種プロセッサ用の基本命令を利用するための便利なインターフェイスとなります。プロセッサ命令のこのような拡張機能によって、SIMD(single instruction-multiple data)手法を用いた並列処理が可能になります。SIMDのデータフローを次の図に示します。



特にこの命令では、命令1個で演算が4回実行できるため、効率が4倍改善されます。このような新しいプロセッサ命令は、インライン・アセンブリ、組込み関数、または C++ SIMD クラスのいずれを使用しても実装できます。その3種類のインターフェイスについて、32ビット浮動小数点値を4個加算するのに必要なコーディングを比較してみてください。

### インライン・アセンブリ、組込み関数、クラス・ライブラリの比較

| インライン・アセンブリ                                                                                           | 組込み関数                                                                            | SIMDクラス・ライブラリ                                                       |
|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|---------------------------------------------------------------------|
| <pre>...    m128 a,b,c; __asm{ movaps xmm0,b movaps xmm1,c addps xmm0,xmm1 movaps a, xmm0 } ...</pre> | <pre>#include &lt;mmintrin.h&gt; ... _m128 a,b,c; a = _mm_add_ps(b,c); ...</pre> | <pre>#include &lt;fvec.h&gt; ...F32v ec4 a,b,c; a = b +c; ...</pre> |

上の表は、単精度浮動小数点値を2個加算するコードについて、インライン・アセンブリ、組込み関数、およびSIMD クラス・ライブラリを用いた場合のそれぞれについて示したものです。インテル C++ SIMD クラス・ライブラリでコーディングをするのがいかに簡単かがわかります。キー入力数が減り、コードの行数が減るだけでなく、表記についてもC++の標準表記と似ているため、他の手法よりも簡単に実装できます。

### C++ クラスとSIMD演算

SIMD演算用の C++ クラスは、配列(ベクトルデータ)を並列処理するときに使用するのが基本です。例として、2個のベクトルAとBの加算を考えてみます。各ベクトルは4個の要素で構成されているとします。整数ベクトル(Ivec)クラスを使用して、各配列から取り出した要素

A[i]とB[i]を下の例のように加算します。

### ループを使用して複数の要素を加算するときの一般的な方法

```
short a[4], b[4], c[4];  
for (i=0; i<4; i++) /* needs four iterations */  
    c[i] = a[i] + b[i]; /* returns c[0], c[1], c[2], c[3] */
```

次の例を見ると、Ivecクラスを使用すれば1回の演算で同じ結果が得られることがわかります。

### Ivecクラスを使用して複数の要素を加算するSIMD手法

```
Is16vec4 ivecA, ivecB, ivec C; /*needs one iteration */  
ivecC = ivecA + ivecB; /*returns ivecC0, ivecC1, ivecC2,  
ivecC3 */
```

## 使用できるクラス

並列処理は、C++ では通常それほど簡単には実装できませんが、インテル® C++ SIMD クラスを使用すればそれが可能です。次の表は、インテル C++ SIMD クラスでクラスとライブラリがどのように使用されるかを列挙したものです。

### SIMDベクトルクラス

| 命令セット                                                 | クラス      | 符号の有<br>無 | データ型  | サイズ | 要素数 | ヘッダファイル |
|-------------------------------------------------------|----------|-----------|-------|-----|-----|---------|
| MMX® テクノロジ<br>(IA-32 ベースおよび<br>Itanium® ベース・システムで利用可) | I64vec1  | 不定        | __m64 | 64  | 1   | ivec.h  |
|                                                       | I32vec2  | 不定        | int   | 32  | 2   | ivec.h  |
|                                                       | Is32vec2 | 符号付き      | int   | 32  | 2   | ivec.h  |
|                                                       | Iu32vec2 | 符号なし      | int   | 32  | 2   | ivec.h  |
|                                                       | I16vec4  | 不定        | short | 16  | 4   | ivec.h  |
|                                                       | Is16vec4 | 符号付き      | short | 16  | 4   | ivec.h  |
|                                                       | Iu16vec4 | 符号なし      | short | 16  | 4   | ivec.h  |
| ストリーミングSIMD拡張                                         | I8vec8   | 不定        | char  | 8   | 8   | ivec.h  |
|                                                       | Is8vec8  | 符号付き      | char  | 8   | 8   | ivec.h  |
|                                                       | Iu8vec8  | 符号なし      | char  | 8   | 8   | ivec.h  |
|                                                       | F32vec4  | 符号付き      | float | 32  | 4   | fvec.h  |
|                                                       |          |           |       |     |     |         |



|                                               |          |      |          |     |    |        |
|-----------------------------------------------|----------|------|----------|-----|----|--------|
| 張命令<br>(IA-32 ベースおよび<br>Itanium ベース・システムで利用可) |          |      |          |     |    |        |
|                                               | F32vec1  | 符号付き | float    | 32  | 1  | fvec.h |
| ストリーミングSIMD拡張命令<br>2(IA-32 ベース・システムでのみ利用可)    | F64vec2  | 符号付き | double   | 64  | 2  | dvec.h |
|                                               | I128vec1 | 不定   | __m128i  | 128 | 1  | dvec.h |
|                                               | I64vec2  | 不定   | long int | 64  | 4  | dvec.h |
|                                               | Is64vec2 | 符号付き | long int | 64  | 4  | dvec.h |
|                                               | Iu64vec2 | 符号なし | long int | 32  | 4  | dvec.h |
|                                               | I32vec4  | 不定   | int      | 32  | 4  | dvec.h |
|                                               | Is32vec4 | 符号付き | int      | 32  | 4  | dvec.h |
|                                               | Iu32vec4 | 符号なし | int      | 32  | 4  | dvec.h |
|                                               | I16vec8  | 不定   | int      | 16  | 8  | dvec.h |
|                                               | Is16vec8 | 符号付き | int      | 16  | 8  | dvec.h |
|                                               | Iu16vec8 | 符号なし | int      | 16  | 8  | dvec.h |
|                                               | I8vec16  | 不定   | char     | 8   | 16 | dvec.h |
|                                               | Is8vec16 | 符号付き | char     | 8   | 16 | dvec.h |
|                                               | Iu8vec16 | 符号なし | char     | 8   | 16 | dvec.h |

ほとんどのクラスは、どのデータ型についても同じような機能を持っていて、利用できるすべての組込み関数で表現されています。ただし一部の機能については、データ型が変わるときにその機能を維持しようとするとパフォーマンスが下がる場合があるため、個々のクラスからは除外しています。



注

即値をとるためにクラスの中に簡単に表現できない組込み関数は実装していません。

(例えば、`_mm_shuffle_ps`, `_mm_shuffle_pi16`, `_mm_extract_pi16`, `_mm_insert_pi16`)

## ヘッダファイルを使用したクラスへのアクセス

必要なクラス・ヘッダファイルは、インテル C++ コンパイラと一緒にインクルード・ディレクトリにインストールされます。各クラスを有効にするときは、次の表に示すように、プログラム・ファイルの中で`#include`ディレクティブを使用してください。

## クラスを有効にするためのインクルード・ディレクティブ

| 拡張命令セット          | インクルード・ディレクティブ                       |
|------------------|--------------------------------------|
| MMX テクノロジ        | <code>#include &lt;ivec.h&gt;</code> |
| ストリーミングSIMD拡張命令  | <code>#include &lt;fvec.h&gt;</code> |
| ストリーミングSIMD拡張命令2 | <code>#include &lt;dvec.h&gt;</code> |

上の表で、`dvec.h`は`fvec.h`の内容を含んでいます。同様に`fvec.h`は`ivec.h`の内容を含んでいます。`Ivec`クラスと`Fvec`クラスの両方を使用したい場合は、`fvec.h`をインクルードするだけで済みます。同様に、ストリーミングSIMD拡張命令2を利用する場合に、その3つとも含むすべてのクラスを使用するときは、`dvec.h`ファイルをインクルードするだけで済みます。

## 使用上の注意事項

C++ クラスの使用方法については、一般的なガイドラインがいくつかあります。クラスごとの使用規則については、「整数ベクトルクラス」および「浮動小数点ベクトルクラス」を参照してください。

## MMX テクノロジ・レジスタの消去

`Ivec`クラスと`Fvec`クラスを同時に使用している場合は、`Ivec`クラスから呼び出される MMX 命令と、`Fvec`クラスから呼び出される Intel x87 アーキテクチャ浮動小数点命令がプログラムの中で混在していることがあります。次の`Fvec`関数の中には浮動小数点命令が含まれています。

- `fvec` コンストラクタ
- デバッグ関数各種 関数各種(`cout`および要素アクセス)
- `rsqrt_nr`



注

MMX テクノロジ・レジスタは浮動小数点レジスタ上にエイリアス化されています。したがって、x87浮動小数点命令を発行する前には、EMMS命令の組込み関数を使用してMMX テクノロジ・ステートを消去する必要があります。次に例を示します。

|                                         |                                                                                |
|-----------------------------------------|--------------------------------------------------------------------------------|
| <code>ivecA = ivecA &amp; ivecB;</code> | <code>/* Ivec logical operation that uses MMX instructions */</code>           |
| <code>empty ();</code>                  | <code>/* clear state */</code>                                                 |
| <code>cout &lt;&lt; f32vec4a;</code>    | <code>/* F32vec4 operation that uses x87 floating-point instructions */</code> |



### 注意

MMX テクノロジ・レジスタを消去しないとレジスタステートが不正な状態になります。そのため、処理が不正確になったり、パフォーマンスが低下したりすることがあります。

## EMMS命令のガイドラインに従う

「EMMS命令を使用する際のガイドライン」に従うことを強くお勧めします。Ivec クラスを使用してコーディングをするときは、こちらをご覧ください。

### 機能

C++ SIMDの各クラスには、次のような基本的な機能があります。

- 計算
- 水平データ移動
- 分岐の圧縮/削除
- キャッシング・ヒント

目的の結果を得るためには、これら諸機能についてそれぞれ理解し、これら諸機能が互いにどのように作用し合うのかを理解することが重要です。

### 計算

SIMD C++ の各クラスは、シフトや飽和をはじめとしてほとんどの算術演算用の垂直演算子に対応しています。

算術演算には、`+`、`-`、`*`、`/`、逆数(`rcp`、`rcp_nr`)、平方根(`sqr`)、平方根の逆数(`rsqr`、`rsqr_nr`)があります。

`rcp`および`rsqr`という演算は、非常に短いレイテンシで近似値の計算ができる新しい命令のことで、最低でも12ビットの精度を持つ計算結果が得られます。`rcp_nr`および`rsqr_nr`という演算は、ソフトウェア的にいくつか改良がなされていて、パフォーマンスにはごく小さい影響しか与えずに精度の高い近似値が得られます。「nr」は、近似値を使ってパフォーマンスを改善する数学的手法の1つである「Newton-Raphson法」の頭文字です。

### 水平データのサポート

C++ SIMDの各クラスは、一部の算術演算については水平処理もできます。「水平」とは、1個のベクトルに含まれているすべての要素について、横方向に端から端まで計算することを指します。一方「垂直」とは、2個の異なるベクトルについて、縦方向に要素単位で計算していくことを指します。

例えば`add_horizontal`、`unpack_low`、`pack_sat`という各関数は、水平データ

に対応しています。このように水平データに対応していると、SIMD命令の能力をすべては利用できない一部のアルゴリズムも使用できるようになります。

シャッフル組込み関数も水平データフローのもう1つの例です。シャッフル組込み関数は、即値引数を使用するため、C++ の各クラスの中には示されていません。ただし、C++ SIMD クラスでは、シャッフル組込み関数を他の C++ 関数と混在できます。次に例を示します。

```
F32vec4 fveca, fvecb, fvecd;  
fveca += fvecb;  
fvecd = _mm_shuffle_ps(fveca, fvecb, 0);
```

一般に、水平データフローを用いると、どの命令の場合も、実装に際しては少し無駄な部分ができます。できる限り、アルゴリズムを実装するときは水平機能は使用しないでください。

## 分岐の圧縮/削除

SIMD アーキテクチャでの分岐処理は、複雑で手間がかかることがあり、場合によっては分岐予測の精度が下がったり、コードのサイズが増えたりします。SIMD C++ クラスには、論理演算、最大値/最小値関数、条件付き選択、比較を使うことによって分岐をなくすための関数がいくつか用意されています。次の例について考えてみます。

```
short a[4], b[4], c[4];  
for (i=0; i<4; i++)  
    c[i] = a[i] > b[i] ? a[i] : b[i];
```

この演算は*i*の値とは無関係です。*i*の値が変わるごとに、その結果は、実際の値に応じてAかBのいずれかになります。分岐を完全になくす簡単な方法の1つは、次のように

`select_gt`関数を使うことです。

```
Is16vec4 a, b, c  
c = select_gt(a, b, a, b)
```

## キャッシング・ヒント

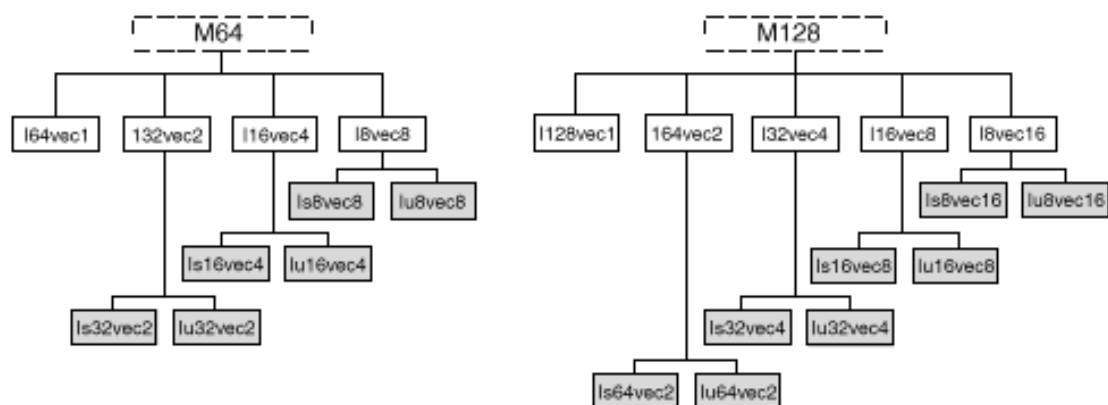
ストリーミングSIMD拡張命令には、プリフェッチとストリーミング・ヒントという機能があります。データをプリフェッチすると、メモリ・レイテンシの影響を最小限に抑えられます。ストリーミング・ヒントを使用すると、キャッシュしないほうが望ましいデータを示すことができます。これにより、キャッシュしたほうが望ましいデータのパフォーマンスが上がります。

# 整数ベクトルクラス

## 整数ベクトルクラスの概要

Ivecクラスは、さまざまなサイズの整数ベクトルを用いるSIMD処理へのインターフェースの機能を提供します。このクラスの派生関係図を次に示します。

### Ivecクラスの派生関係図



OM00834

M64およびM128という両クラスが `__m64` および `__m128i` という両データ型を定義しています。この2つのクラスから、残りのIvecクラスが派生します。第1世代の子クラスは、128、64、32、16、8というビットサイズだけで区分され、それぞれI128vec1、I64vec1、164vec2、I32vec2、I32vec4、I16vec4、I16vec8、I8vec16、I8vec8というクラスになります。そのうちのI128vec1とI64vec1以外は、符号の有無と飽和处理の有無を指定する必要があります。

### ⚠ 注意

M64データ型とM128データ型は混在させないでください。そうしないと、予期しない動作をします。

符号の有無は、クラス名の中のsとuという文字で表します。

I64vec2  
Iu64vec2  
I32vec4  
Iu32vec4  
I16vec8  
Iu16vec8  
I8vec16  
Iu8vec16  
I32vec2  
Iu32vec2  
I16vec4  
Iu16vec4  
I8vec8

## 用語、規則、および構文

本章では、各クラスとそれに関連した処理について説明するときに特殊な用語と構文を用いています。以下で、その説明をします。

### Ivecクラスの構文規則

クラス名は、データ型、符号の有無、ビットサイズ、要素数を表現したものです。一般的な形式で表すと次のようになります。

```
<type><signedness><bits>vec<elements>
{ F | I } { s | u } { 64 | 32 | 16 | 8 } vec { 8 | 4 | 2
| 1 }
```

各アイテムの意味は次のとおりです。

|            |                                                                                                       |
|------------|-------------------------------------------------------------------------------------------------------|
| type       | 浮動小数点(F)または整数(I)を示します。                                                                                |
| signedness | 符号付き(s)または符号なし(u)を示します。Ivecクラスの場合は、このフィールドが空のままだと中間クラスを表します。符号なしのFvecクラスはないため、Fvecクラスの場合、このフィールドは空です。 |
| bits       | 要素あたりのビット数です。                                                                                         |
| elements   | 要素の個数です。                                                                                              |

## 特殊な用語と規則

本書では、各クラスと演算についてその機能と特性を定義するときに次の用語を用いています。

- 最も近い共通の親クラス:** サイズの同じ2種類のクラスの間接クラスまたは親クラスです。例えば、Iu8vec8とIs8vec8の「最も近い共通の親クラス」はI8vec8です。同様に、Iu8vec8とI16vec4の「最も近い共通の親クラス」はM64です。
- キャスト:** クラスのデータ型を変更します。データ型の異なる複数のオペランドを使用して演算を行う場合は、その戻り値を同じ1種類のデータ型にする必要があります。したがって、1種類以上のデータ型を同じ種類の規定のデータ型に変換する必要があります。この変換処理のことを「型キャスト」と言います。型キャストは自動的に行われる場合もありますが、特殊な構文を使用して明示的に行う必要があります。
- 演算子の多重定義:** 何らかのクラスのデータ型をユーザが1個定義し、その1個のデータ型に対して複数の演算子を利用できる機能です。いったん変数を宣言すれば、加算、減算、乗算はもとより、複数の演算を連続して実行できます。どのクラスファミリも、規定の範囲の演算子が利用できますが、ヘッダファイルに定義されている、型キャストと演算子の多重定義に関する規則および制約に従う必要があります。本書では、型キャストや演算子の多重定義などの規則に次の表記を使用しています。

## クラス名の表記法

| クラス名                               | 説明                                                       |
|------------------------------------|----------------------------------------------------------|
| <code>I [s   u] [N] vec [N]</code> | <code>I128vec1</code> 以外および <code>I64vec1</code> 以外の任意の値 |
| <code>I64vec1</code>               | m64データ型                                                  |
| <code>I [s   u] 64vec2</code>      | 2個の64ビット値(符号付き、または符号なし)                                  |
| <code>I [s   u] 32vec4</code>      | 4個の32ビット値(符号付き、または符号なし)                                  |
| <code>I [s   u] 8vec16</code>      | 8個の16ビット値(符号付き、または符号なし)                                  |
| <code>I [s   u] 16vec8</code>      | 16個の8ビット値(符号付き、または符号なし)                                  |
| <code>I [s   u] 32vec2</code>      | 2個の32ビット値(符号付き、または符号なし)                                  |
| <code>I [s   u] 16vec4</code>      | 4個の16ビット値(符号付き、または符号なし)                                  |
| <code>I [s   u] 8vec8</code>       | 8個の8ビット値(符号付き、または符号なし)                                   |

## 演算子の規則

Ivecクラスで演算子を使用するときは、次の構文規則のいずれかに従う必要があります。

```
[ Ivec_Class ] R = [ Ivec_Class ] A [ operator ] [ Ivec_Class ] B
```

**例 1:** `I64vec1 R = I64vec1 A & I64vec1 B;`

```
[ Ivec_Class ] R = [ operator ] ( [ Ivec_Class ] A, [ Ivec_Class ] B )
```

**例 2:** `I64vec1 R = andnot(I64vec1 A, I64vec1 B);`

```
[ Ivec_Class ] R [ operator ] = [ Ivec_Class ] A
```

**例 3:** `I64vec1 R &= I64vec1 A;`

[operator] は演算子(例えば&, |, ^)

[ Ivec\_Class ]はIvecクラス

R,A,Bは、該当するIvecクラスを使用して宣言された変数

次の表は、符号とサイズを対象とした自動的型キャストと明示的型キャストを列挙したものです。「明示的」は、異なるデータ型をいくつか混在させるときは必ず型キャストを明示的に指定しなければならないという意味です。「自動的」は、複数のデータ型を混在させたときにコンパイラが自動的に型キャストを行うという意味です。

### 主要演算子の規則一覧

| 演算子 | 符号の<br>型キャスト | サイズの<br>型キャスト | 型キャストに関するその他の要件                                                   |
|-----|--------------|---------------|-------------------------------------------------------------------|
| 代入  | N/A          | N/A           | N/A                                                               |
| 論理  | 自動的          | 自動的<br>(左辺)   | 代入演算子の右辺に論理式以外の式が使用されていて、かつその中に、異なるデータ型が混在している場合は、明示的な型キャストが必要です。 |



|        |     |     |                                                |
|--------|-----|-----|------------------------------------------------|
|        |     |     | 「論理演算子の構文の使用」の例を参照してください。                      |
| 加算と減算  | 自動的 | 明示的 | N/A                                            |
| 乗算     | 自動的 | 明示的 | N/A                                            |
| シフト    | 自動的 | 明示的 | 算術演算シフトを確実に行うには型キャストが必要です。                     |
| 比較     | 自動的 | 明示的 | 「<」または「>」の比較を行う場合は、符号付きクラスについては明示的な型キャストが必要です。 |
| 条件付き選択 | 自動的 | 明示的 | 「<」または「>」の比較を行う場合は、符号付きクラスについては明示的な型キャストが必要です。 |

## データの宣言と初期化

次の表は、すべてのクラスサイズについて、コンストラクタの宣言とデータ型の初期化の例を列挙したものです。どの値についても、最上位要素が左側、最下位要素が右側で初期化されます。

### Ivecクラスのデータ型の宣言と初期化

| 操作                        | クラス     | 構文                                                                                                                                                                                            |
|---------------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 宣言                        | M128    | <code>I128vec1 A; Iu8vec16 A;</code>                                                                                                                                                          |
| 宣言                        | M64     | <code>I64vec1 A; Iu8vec16 A;</code>                                                                                                                                                           |
| <code>__m128</code> の初期化  | M128    | <code>I128vec1 A(__m128 m);<br/>Iu16vec8( m128 m);</code>                                                                                                                                     |
| <code>__m64</code> の初期化   | M64     | <code>I64vec1 A(__m64 m); Iu8vec8<br/>A( m64 m);</code>                                                                                                                                       |
| <code>__int64</code> の初期化 | M64     | <code>I64vec1 A = int64 m; Iu8vec8<br/>A = int64 m;</code>                                                                                                                                    |
| <code>int i</code> の初期化   | M64     | <code>I64vec1 A = int i; Iu8vec8 A =<br/>int i;</code>                                                                                                                                        |
| <code>int</code> の初期化     | I32vec2 | <code>I32vec2 A(int A1, int A0);<br/>Is32vec2 A(signed int A1,<br/>signed int A0);<br/>Iu32vec2 A(unsigned int A1,<br/>unsigned int A0);</code>                                               |
| <code>int</code> の初期化     | I32vec4 | <code>I32vec4 A(short A3, short A2,<br/>short A1, short A0);<br/>Is32vec4 A(signed short<br/>A3, ..., signed short A0);<br/>Iu32vec4 A(unsigned short<br/>A3, ..., unsigned short A0);</code> |



|                   |         |                                                                                                                                                                             |
|-------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| short int<br>の初期化 | I16vec4 | I16vec4 A(short A3, short A2,<br>short A1, short A0);<br>Is16vec4 A(signed short<br>A3, ..., signed short A0);<br>Iu16vec4 A(unsigned short<br>A3, ..., unsigned short A0); |
| short int<br>の初期化 | I16vec8 | I16vec8 A(short A7, short<br>A6, ..., short A1, short A0);<br>Is16vec8 A(signed A7, ...,<br>signed short A0);<br>Iu16vec8 A(unsigned short<br>A7, ..., unsigned short A0);  |
| char<br>の初期化      | I8vec8  | I8vec8 A(char A7, char A6, ...,<br>char A1, char A0);<br>Is8vec8 A(signed char A7, ...,<br>signed char A0);<br>Iu8vec8 A(unsigned char<br>A7, ..., unsigned char A0);       |
| char<br>の初期化      | I8vec16 | I8vec16 A(char A15, ..., char<br>A0);<br>Is8vec16 A(signed char<br>A15, ..., signed char A0);<br>Iu8vec16 A(unsigned char<br>A15, ..., unsigned char A0);                   |

## 代入演算子

どのlvecオブジェクトも別のlvecオブジェクトに代入できます。lvecオブジェクトから別のlvecオブジェクトに代入するときの変換は自動的に行われます。

### 代入演算子の例

```
Is16vec4 A;
Is8vec8 B;
I64vec1 C;
A = B; /* assign Is8vec8 to Is16vec4 */
B = C; /* assign I64vec1 to Is8vec8 */
B = A & C; /* assign M64 result of '&' to Is8vec8 */
```

## 論理演算子

論理演算子では、次の表に列挙した記号と組込み関数を使用します。

| ビット単位演算 | 演算子の記号 |      | 構文の使用 |      | 対応する組込み関数 |
|---------|--------|------|-------|------|-----------|
|         | 標準     | 代入付き | 標準    | 代入付き |           |

|        |                     |     |                                |        |                                                         |
|--------|---------------------|-----|--------------------------------|--------|---------------------------------------------------------|
| AND    | &                   | &=  | R = A & B                      | R &= A | <code>_mm_and_si64</code><br><code>_mm_and_si128</code> |
| OR     |                     | =   | R = A   B                      | R  = A | <code>_mm_and_si64</code><br><code>_mm_and_si128</code> |
| XOR    | ^                   | ^=  | R = A ^ B                      | R ^= A | <code>_mm_and_si64</code><br><code>_mm_and_si128</code> |
| ANDNOT | <code>andnot</code> | N/A | R = A<br><code>andnot</code> B | N/A    | <code>_mm_and_si64</code><br><code>_mm_and_si128</code> |

## 論理演算子と例外

```

/* A and B converted to M64. Result assigned to Iu8vec8.*/
I64vec1 A;
Is8vec8 B;
Iu8vec8 C;
C = A & B;
/* Same size and signedness operators return the nearest
common ancestor.*/
I32vec2 R = Is32vec2 A ^ Iu32vec2 B;
/* A&B returns M64, which is cast to Iu8vec8.*/
C = Iu8vec8(A&B) + C;

```

AとBが同じクラスの場合、戻り値はそれと同じ型になります。AとBとが別のクラスの場合、戻り値は、両者に共通する親データ型のうち最も近いデータ型になります。

クラスを複数組み合わせた場合に論理演算子の戻り値がどうなるかを次の表に示します。これは、AとBが別のクラスの場合に適用されるものです。

## Ivec論理演算子の多重定義

| 戻り値(R)       | AND | OR | XOR | NAND                | オペランドA             | オペランドB             |
|--------------|-----|----|-----|---------------------|--------------------|--------------------|
| I64vec1<br>R | &   |    | ^   | <code>andnot</code> | I[s u]6<br>4vec2 A | I[s u]6<br>4vec2 B |
| I64vec2<br>R | &   |    | ^   | <code>andnot</code> | I[s u]6<br>4vec2 A | I[s u]6<br>4vec2 B |
| I32vec2<br>R | &   |    | ^   | <code>andnot</code> | I[s u]3<br>2vec2 A | I[s u]3<br>2vec2 B |
| I32vec4<br>R | &   |    | ^   | <code>andnot</code> | I[s u]3<br>2vec4 A | I[s u]3<br>2vec4 B |
| I16vec4<br>R | &   |    | ^   | <code>andnot</code> | I[s u]1<br>6vec4 A | I[s u]1<br>6vec4 B |
| I16vec8<br>R | &   |    | ^   | <code>andnot</code> | I[s u]1<br>6vec8 A | I[s u]1<br>6vec8 B |
| I8vec8<br>R  | &   |    | ^   | <code>andnot</code> | I[s u]8<br>vec8 A  | I[s u]8<br>vec8 B  |
| I8vec16<br>R | &   |    | ^   | <code>andnot</code> | I[s u]8<br>vec16 A | I[s u]8<br>vec16 B |

代入付き論理演算子の場合は次表のように、戻り値Rは常に、事前に宣言した値Rと同じデータ型になります。

## 代入付きlvec論理演算子の多重定義

| 戻り値の型        | 左辺(R)          | AND | OR | XOR | 右辺(任意のlvec型)             |
|--------------|----------------|-----|----|-----|--------------------------|
| I128vec1     | I128vec1<br>R  | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I64vec1      | I64vec1 R      | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I64vec2      | I64vec2 R      | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I [x] 32vec4 | I [x] 32vec4 R | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I [x] 32vec2 | I [x] 32vec2 R | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I [x] 16vec8 | I [x] 16vec8 R | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I [x] 16vec4 | I [x] 16vec4 R | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I [x] 8vec16 | I [x] 8vec16 R | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |
| I [x] 8vec8  | I [x] 8vec8 R  | &=  | =  | ^=  | I [s   u] [N] vec [N] A; |

## 加算演算子と減算演算子

加算演算子と減算演算子は、右辺の各オペランドの符号が揃っていない場合は、共通の親クラスのうち最も近い親クラスを返します。使用例と例外をいくつか次のコードに示します。

### 加算演算子および減算演算子の構文の使用

```
/* Return nearest common ancestor type, I16vec4 */
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B;
/* Returns type left-hand operand type */
Is16vec4 A;
Iu16vec4 B;
A += B;
B -= A;
/* Explicitly convert B to Is16vec4 */
Is16vec4 A, C;
Iu32vec24 B;
C = A + C;
```

C = A + (Is16vec4) B;

### 加算演算子、減算演算子、およびそれらに対応する組込み関数

| 操作 | 記号      | 構文                  | 対応する組込み関数                                                                                                      |
|----|---------|---------------------|----------------------------------------------------------------------------------------------------------------|
| 加算 | +<br>+= | R = A + B<br>R += A | _mm_add_epi64<br>_mm_add_epi32<br>_mm_add_epi16<br>_mm_add_epi8<br>_mm_add_pi32<br>_mm_add_pi16<br>_mm_add_pi8 |
| 減算 | -<br>-= | R = A - B<br>R -= A | _mm_sub_epi64<br>_mm_sub_epi32<br>_mm_sub_epi16<br>_mm_sub_epi8<br>_mm_sub_pi32<br>_mm_sub_pi16<br>_mm_sub_pi8 |

次の表は、右辺の各オペランドの符号が異なっている場合に、クラスを複数組み合わせさせたときの、加算と減算の戻り値を列挙したものです。この2個のオペランドは同じサイズでなければなりません。サイズが異なる場合は、型キャストを明示的に示す必要があります。

### 加算演算子と減算演算子の多重定義

| 戻り値       | 使用できる演算子 |    | 右辺のオペランド       |                |
|-----------|----------|----|----------------|----------------|
| R         | 加算       | 減算 | A              | B              |
| I64vec2 R | +        | -  | I[s u]64vec2 A | I[s u]64vec2 B |
| I32vec4 R | +        | -  | I[s u]32vec4 A | I[s u]32vec4 B |
| I32vec2 R | +        | -  | I[s u]32vec2 A | I[s u]32vec2 B |
| I16vec8 R | +        | -  | I[s u]16vec8 A | I[s u]16vec8 B |
| I16vec4 R | +        | -  | I[s u]16vec4 A | I[s u]16vec4 B |
| I8vec8 R  | +        | -  | I[s u]8vec8 A  | I[s u]8vec8 B  |
| I8vec16 R | +        | -  | I[s u]8vec2 A  | I[s u]8vec16 B |

次の表は、代入付きの加算演算子と減算演算子の各種オペランドについて、戻り値のデータ型を列挙したものです。戻り値のサイズと符号の有無は、左辺のオペランドによって決まります。右辺のオペランドは左辺のオペランドと同じサイズでなければなりません。サイズが異なる場合は、明示的な型キャストを行う必要があります。

## 乗算演算子の記号と対応する組込み関数

| 戻り値(R)          | 左辺(R)           | 加算 | 減算 | 右辺(A)           |
|-----------------|-----------------|----|----|-----------------|
| I[x]32vec4      | I[x]32vec2<br>R | += | -= | I[s u]32vec4 A; |
| I[x]32vec2<br>R | I[x]32vec2<br>R | += | -= | I[s u]32vec2 A; |
| I[x]16vec8      | I[x]16vec8      | += | -= | I[s u]16vec8 A; |
| I[x]16vec4      | I[x]16vec4      | += | -= | I[s u]16vec4 A; |
| I[x]8vec16      | I[x]8vec16      | += | -= | I[s u]8vec16 A; |
| I[x]8vec8       | I[x]8vec8       | += | -= | I[s u]8vec8 A;  |

### 乗算演算子

乗算演算子では、I[s|u]16vec4またはI[s|u]16vec8のいずれかのクラスのデータ型しか演算に使用できません。また、戻り値もそのいずれかのデータ型になります。次に例を示します。

#### 乗算演算子の構文の使用

```
/* Explicitly convert B to Is16vec4 */
Is16vec4 A, C;
Iu32vec2 B;
C = A * C;
C = A * (Is16vec4)B;
/* Return nearest common ancestor type, I16vec4 */
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B;
/* The mul_high and mul_add functions take Is16vec4 data
only */
Is16vec4 A,B,C,D;
C = mul_high(A,B);
D = mul_add(A,B);
```

#### 対応する組込み関数と乗算演算子

| 操作 | 記号 |    |                     | 構文の使用                                 | 組込み関数 |
|----|----|----|---------------------|---------------------------------------|-------|
| 乗算 | *  | += | R = A * B<br>R *= A | _mm_mullo_pi16<br><br>_mm_mullo_epi16 |       |

|  |  |          |     |                          |                                                |
|--|--|----------|-----|--------------------------|------------------------------------------------|
|  |  | mul_high | 非対応 | R =<br>mul_high(A,<br>B) | _mm_mulhi<br>_pi16<br><br>_mm_mulh<br>_i_epi16 |
|  |  | mul_add  | 非対応 | R =<br>mul_high(A,<br>B) | _mm_madd<br>_pi16<br><br>_mm_madd<br>_epi16    |

次の表に示すように、この乗算演算子の戻り値は常に、共通する親データ型のうち最も近いデータ型になります。2個のオペランドのサイズは16ビットでなければなりません。それ以外の場合は、型キャストを明示的に使用する必要があります。

## 乗算演算子の多重定義

| R          | 乗算       | A              | B              |
|------------|----------|----------------|----------------|
| I16vec4 R  | *        | I[s u]16vec4 A | I[s u]16vec4 B |
| I16vec8 R  | *        | I[s u]16vec8 A | I[s u]16vec8 B |
| Is16vec4 R | mul_add  | Is16vec4 A     | Is16vec4 B     |
| Is16vec8   | mul_add  | Is16vec8 A     | Is16vec8 B     |
| Is32vec2 R | mul_high | Is16vec4 A     | Is16vec4 B     |
| Is32vec4 R | mul_high | s16vec8 A      | Is16vec8 B     |

次の表は、代入付き乗算演算子を用いた場合の戻り値のデータ型を列挙したものです。オペランドのサイズはすべて16ビットでなければなりません。オペランドのサイズが正しくない場合は、明示的な型キャストを使用する必要があります。

## 代入付き乗算

| 戻り値(R)     | 左辺(R)      | 乗算 | 右辺(A)           |
|------------|------------|----|-----------------|
| I[x]16vec8 | I[x]16vec8 | *= | I[s u]16vec8 A; |
| I[x]16vec4 | I[x]16vec4 | *= | I[s u]16vec4 A; |

## シフト演算子

右シフトの引数は、整数値かIvec値であれば何でもかまわず、暗黙的にM64データ型に変換されます。<<という演算子の1番目のオペランドまたは左側のオペランドには、I[s|u]8vec[8|16]以外のどの型でも使用できます。

### シフト演算子の構文の使用例

```
/* Automatic size and sign conversion */
Is16vec4 A, C;
Iu32vec2 B;
C = A;
/* A&B returns I16vec4, which must be cast to Iu16vec4
```

```

to ensure logical shift, not arithmetic shift */
Is16vec4 A, C;
Iu16vec4 B, R;
R = (Iu16vec4)(A & B) C;
/* A&B returns I16vec4, which must be cast to Is16vec4
to ensure arithmetic shift, not logical shift */
R = (Is16vec4)(A & B) C;

```

### シフト演算子と対応する組込み関数

| 操作   | 記号       | 構文の使用                 | 組込み関数                                                                                                                                                             |
|------|----------|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 左シフト | <<<br>&= | R = A << B<br>R &= A  | _mm_sll_si64<br>_mm_slli_si64<br>_mm_sll_pi32<br>_mm_slli_pi32<br>_mm_sll_pi16<br>_mm_slli_pi16                                                                   |
| 右シフト | >>       | R = A >> B<br>R >>= A | _mm_srl_si64<br>_mm_srli_si64<br>_mm_srl_pi32<br>_mm_srli_pi32<br>_mm_srl_pi16<br>_mm_srli_pi16<br>_mm_sra_pi32<br>_mm_srai_pi32<br>_mm_sra_pi16<br>_mm_srai_pi16 |

符号付きのデータ型を右にシフトするときは算術演算シフトを使用します。符号なしクラスおよび中間クラスの場合はすべて、論理シフトが使用されます。次の表は、最初の引数の型によって戻り値の型がどう決まるかを示したものです。

### シフト演算子の多重定義

| 操作 | R        | 右シフト |     | 左シフト |     | A          | B                  |
|----|----------|------|-----|------|-----|------------|--------------------|
| 論理 | I64vec1  | >>   | >>= | <<   | <<= | I64vec1 A; | I64vec1 B;         |
| 論理 | I32vec2  | >>   | >>= | <<   | <<= | I32vec2 A  | I32vec2 B;         |
| 算術 | Is32vec2 | >>   | >>= | <<   | <<= | Is32vec2 A | I[s u][N]vec[N] B; |
| 論理 | Iu32vec2 | >>   | >>= | <<   | <<= | Iu32vec2 A | I[s u][N]vec[N] B; |
| 論理 | I16vec4  | >>   | >>= | <<   | <<= | I16vec4 A  | I16vec4 B          |
| 算術 | Is16vec4 | >>   | >>= | <<   | <<= | Is16vec4 A | I[s u][N]vec[N] B; |
| 論理 | Iu16vec4 | >>   | >>= | <<   | <<= | Iu16vec4 A | I[s u][N]vec[N] B; |

### 比較演算子

「=」および「≠」の比較については、オペランドは符号が異なってもかまいませんが、サイ

ズは同じでなければなりません。「<」および「>」の比較については、オペランドは符号もサイズも同じでなければなりません。

### 比較演算子の構文の使用例

```
/* The nearest common ancestor is returned for compare
for equal/not-equal operations */
Iu8vec8 A;
Is8vec8 B;
I8vec8 C;
C = cmpneq(A,B);
/* Type cast needed for different-sized elements for
equal/not-equal comparisons */
Iu8vec8 A, C;
Is16vec4 B;
C = cmpeq(A, (Iu8vec8)B);
/* Type cast needed for sign or size differences for
less-than and greater-than comparisons */
Iu16vec4 A;
Is16vec4 B, C;
C = cmpge((Is16vec4)A,B);
C = cmpgt(B,C);
```

### 比較演算子と対応する組込み関数

| 比較の条件               | 演算子    | 構文                     | 組込み関数                                                                |
|---------------------|--------|------------------------|----------------------------------------------------------------------|
| 等しい                 | cmpeq  | R =<br>cmpeq(A,<br>B)  | _mm_cmpeq_pi32<br>_mm_cmpeq_pi16<br>_mm_cmpeq_pi8                    |
| 等しくない               | cmpneq | R =<br>cmpneq(A,<br>B) | _mm_cmpeq_pi32<br>_mm_cmpeq_pi16<br>_mm_cmpeq_pi8<br>_mm_andnot_si64 |
| より大きい               | cmpgt  | R =<br>cmpgt(A, B)     | _mm_cmpgt_pi32<br>_mm_cmpgt_pi16<br>_mm_cmpgt_pi8                    |
| より大きい<br>または等し<br>い | cmpge  | R =<br>cmpge(A,<br>B)  | _mm_cmpgt_pi32<br>_mm_cmpgt_pi16<br>_mm_cmpgt_pi8<br>_mm_andnot_si64 |
| より小さい               | cmplt  | R =<br>cmplt(A, B)     | _mm_cmpgt_pi32<br>_mm_cmpgt_pi16<br>_mm_cmpgt_pi8                    |
| より小さい<br>または等し<br>い | cmple  | R =<br>cmple(A, B)     | _mm_cmpgt_pi32<br>_mm_cmpgt_pi16<br>_mm_cmpgt_pi8<br>_mm_andnot_si64 |

比較演算子のオペランドは、そのサイズも符号も、次の「比較演算子の多重定義」の表に従っていなければなりません。



## 比較演算子の多重定義

| R         | 比較    | A              | B              |
|-----------|-------|----------------|----------------|
| I32vec2 R | cmpeq | I[s u]32vec2 B | I[s u]32vec2 B |
| I16vec4 R | cmpne | I[s u]16vec4 B | I[s u]16vec4 B |
| I8vec8 R  |       | I[s u]8vec8 B  | I[s u]8vec8 B  |
| I32vec2 R | cmpgt | Is32vec2 B     | Is32vec2 B     |
| I16vec4 R | cmpge | Is16vec4 B     | Is16vec4 B     |
| I8vec8 R  | cmplt | Is8vec8 B      | Is8vec8 B      |
|           | cmple |                |                |

## 条件付き選択演算子

条件付き選択演算子の場合は、3番目と4番目のオペランドによって戻り値のデータ型が決まります。3番目と4番目のオペランドがサイズが同じで符号が異なる場合、戻り値は共通の親データ型のうち最も近いデータ型になります。

### 条件付き選択演算子の構文の使用例

```
/* Return the nearest common ancestor data type if third
and fourth operands are of the same size, but different signs
*/
```

```
I16vec4 R = select_neq(Is16vec4, Is16vec4, Is16vec4,
Iu16vec4);
```

```
/* Conditional Select for Equality */
```

```
R0 := (A0 == B0) ? C0 : D0;
```

```
R1 := (A1 == B1) ? C1 : D1;
```

```
R2 := (A2 == B2) ? C2 : D2;
```

```
R3 := (A3 == B3) ? C3 : D3;
```

```
/* Conditional Select for Inequality */
```

```
R0 := (A0 == B0) ? C0 : D0;
```

```
R1 := (A1 == B1) ? C1 : D1;
```

```
R2 := (A2 != B2) ? C2 : D2;
```

```
R3 := (A3 != B3) ? C3 : D3;
```

### 条件付き選択演算子と対応する組込み関数

| 条件付き選択の条件 | 演算子       | 構文                        | 対応する組込み関数                                         | その他の組込み関数(すべてに適用)                              |
|-----------|-----------|---------------------------|---------------------------------------------------|------------------------------------------------|
| =         | select_eq | R = select_eq(A, B, C, D) | _mm_cmpeq_pi32<br>_mm_cmpeq_pi16<br>_mm_cmpeq_pi8 | _mm_and_si64<br>_mm_or_si64<br>_mm_andnot_si64 |

|                      |            |                                   |                                                               |  |
|----------------------|------------|-----------------------------------|---------------------------------------------------------------|--|
| ≠                    | select_neq | R =<br>select_neq<br>(A, B, C, D) | _mm_cmpeq_<br>pi32<br>_mm_cmpeq_<br>pi16<br>_mm_cmpeq_<br>pi8 |  |
| より大きい                | select_gt  | R =<br>select_gt(<br>A, B, C, D)  | _mm_cmpgt_<br>pi32<br>_mm_cmpgt_<br>pi16<br>_mm_cmpgt_<br>pi8 |  |
| より大きい<br>or Equal To | select_ge  | R =<br>select_gt(<br>A, B, C, D)  | _mm_cmpge_<br>pi32<br>_mm_cmpge_<br>pi16<br>_mm_cmpge_<br>pi8 |  |
| <                    | select_lt  | R =<br>select_lt(<br>A, B, C, D)  | _mm_cmplt_<br>pi32<br>_mm_cmplt_<br>pi16<br>_mm_cmplt_<br>pi8 |  |
| <<br>or Equal To     | select_le  | R =<br>select_le(<br>A, B, C, D)  | _mm_cmple_<br>pi32<br>_mm_cmple_<br>pi16<br>_mm_cmple_<br>pi8 |  |

条件付き選択演算のオペランドはすべて同じサイズでなければなりません。戻り値のデータ型は、オペランドCおよびDに共通する親データ型の中でも最も近いデータ型になります。「>」および「<」という比較を用いて条件付き選択演算を行う場合は、1番目と2番目のオペランドの符号は次の表に従わなければなりません。

## 条件付き選択演算子の多重定義

| R         | 比較条件                   | A, B         | C            | D            |
|-----------|------------------------|--------------|--------------|--------------|
| I32vec2 R | select_eq<br>select_ne | I[s u]32vec2 | I[s u]32vec2 | I[s u]32vec2 |
| I16vec4 R |                        | I[s u]16vec4 | I[s u]16vec4 | I[s u]16vec4 |
| I8vec8 R  |                        | I[s u]8vec8  | I[s u]8vec8  | I[s u]8vec8  |
| I32vec2 R | select_gt              | Is32vec2     | Is32vec2     | Is32vec2     |

|           |                                     |          |          |          |
|-----------|-------------------------------------|----------|----------|----------|
|           | select_ge<br>select_lt<br>select_le |          |          |          |
| I16vec4 R |                                     | Is16vec4 | Is16vec4 | Is16vec4 |
| I8vec8 R  |                                     | Is8vec8  | Is8vec8  | Is8vec8  |

次の表は、任意の要素数に対するR0からR7までの戻り値の対応表です。戻り値の個数が3個以下のときも、これと同じ戻り値対応表を適用します。

## 条件付き選択演算子の戻り値対応表

| 戻り値  | オペランドA、B |          |    |   |    |   |    | オペランドC、D |            |
|------|----------|----------|----|---|----|---|----|----------|------------|
|      | A0       | 使用できる演算子 |    |   |    |   | B0 |          |            |
| R0:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C0 : D0; |
| R1:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C1 : D1; |
| R2:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C2 : D2; |
| R3:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C3 : D3; |
| R4:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C4 : D4; |
| R5:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C5 : D5; |
| R6:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C6 : D6; |
| R7:= | A0       | ==       | != | > | >= | < | <= | B0       | ? C7 : D7; |

## デバッグ

デバッグ演算は、MMX® テクノロジ命令についてはどのコンパイラ関数にも対応関係はありません。単にプログラムのデバッグに使用される演算にすぎません。この演算を使用するとパフォーマンスが下がる場合がありますので、デバッグの目的以外には使用しないでください。

## 出力

4個の32ビット値Aが出力バッファに格納され、次の形式で出力します(デフォルトでは10進数形式)。

```
cout << Is32vec4 A;
cout << Iu32vec4 A;
cout << hex << Iu32vec4 A; /* print in hex format */
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"
対応する組込み関数: なし
```

2個の32ビット値Aが出力バッファに格納され、次の形式で出力します(デフォルトでは10進数形式)。

```
cout << Is32vec2 A;
cout << Iu32vec2 A;
cout << hex << Iu32vec2 A; /* print in hex format */
```

```
"[1]:A1 [0]:A0"
```

対応する組込み関数: なし

8個の16ビット値Aが出力バッファに格納され、次の形式で出力します(デフォルトでは10進数形式)。

```
cout << Is16vec8 A;  
cout << Iu16vec8 A;  
cout << hex << Iu16vec8 A; /* print in hex format */  
"[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0"  
対応する組込み関数: なし
```

4個の16ビット値Aが出力バッファに格納され、次の形式で出力します(デフォルトでは10進数形式)。

```
cout << Is16vec4 A;  
cout << Iu16vec4 A;  
cout << hex << Iu16vec4 A; /* print in hex format */  
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"  
対応する組込み関数: なし
```

16個の8ビット値Aが出力バッファに格納され、次の形式で出力します(デフォルトでは10進数形式)。

```
cout << Is8vec16 A;cout << Iu8vec16 A;cout << hex <<  
Iu8vec8 A;  
/* print in hex format instead of decimal*/  
"[15]:A15 [14]:A14 [13]:A13 [12]:A12 [11]:A11 [10]:A10  
[9]:A9 [8]:A8 [7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2  
[1]:A1 [0]:A0"  
対応する組込み関数: なし
```

8個の8ビット値Aが出力バッファに格納され、次の形式で出力します(デフォルトでは10進数形式)。

```
cout << Is8vec8 A; cout << Iu8vec8 A;cout << hex << Iu8vec8  
A;  
/* print in hex format instead of decimal*/  
"[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0"  
対応する組込み関数: なし
```

## 要素アクセス演算子

```
int R = Is64vec2 A[i];  
unsigned int R = Iu64vec2 A[i];  
int R = Is32vec4 A[i];  
unsigned int R = Iu32vec4 A[i];  
int R = Is32vec2 A[i];
```

```

unsigned int R = Iu32vec2 A[i];
short R = Is16vec8 A[i];
unsigned short R = Iu16vec8 A[i];
short R = Is16vec4 A[i];
unsigned short R = Iu16vec4 A[i];
signed char R = Is8vec16 A[i];
unsigned char R = Iu8vec16 A[i];
signed char R = Is8vec8 A[i];
unsigned char R = Iu8vec8 A[i];

```

Aの要素iにアクセスして読み取ります。DEBUGが有効もなっているときにユーザがA以外の要素にアクセスしようとする、診断メッセージが出力され、プログラムが途中で終了します。

対応する組込み関数: なし

## 要素代入演算子

```

Is64vec2 A[i] = int R;
Is32vec4 A[i] = int R;
Iu32vec4 A[i] = unsigned int R;
Is32vec2 A[i] = int R;
Iu32vec2 A[i] = unsigned int R;
Is16vec8 A[i] = short R;
Iu16vec8 A[i] = unsigned short R;
Is16vec4 A[i] = short R;
Iu16vec4 A[i] = unsigned short R;
Is8vec16 A[i] = signed char R;
Iu8vec16 A[i] = unsigned char R;
Is8vec8 A[i] = signed char R;
Iu8vec8 A[i] = unsigned char R;

```

Aの要素iにRを代入します。DEBUGが有効になっているときにユーザがA以外の要素に何らかの値を代入しようとする、診断メッセージが表示され、プログラムは途中で終了します。

対応する組込み関数: なし

## アンパック演算子

```

I364vec2 unpack_high(I64vec2 A, I64vec2 B)
Is64vec2 unpack_high(Is64vec2 A, Is64vec2 B)
Iu64vec2 unpack_high(Iu64vec2 A, Iu64vec2 B)

```

Aの上位半分から取り出した64ビット値を、Bの上位半分から取り出した64ビット値とインターリーブします。

R0 = A1;

R1 = B1;

対応する組込み関数: `_mm_unpackhi_epi64`

`I32vec4 unpack_high(I32vec4 A, I32vec4 B)`

`Is32vec4 unpack_high(Is32vec4 A, Is32vec4 B)`

`Iu32vec4 unpack_high(Iu32vec4 A, Iu32vec4 B)`

Aの上位半分から取り出した2個の32ビット値を、Bの上位半分から取り出した2個の32ビット値とインターリーブします。

R0 = A1;

R1 = B1;

R2 = A2;

R3 = B2;

対応する組込み関数: `_mm_unpackhi_epi32`

`I32vec2 unpack_high(I32vec2 A, I32vec2 B)`

`Is32vec2 unpack_high(Is32vec2 A, Is32vec2 B)`

`Iu32vec2 unpack_high(Iu32vec2 A, Iu32vec2 B)`

Aの上位半分から取り出した32ビット値を、Bの上位半分から取り出した32ビット値とインターリーブします。

R0 = A1;

R1 = B1;

対応する組込み関数: `_mm_unpackhi_pi32`

`I16vec8 unpack_high(I16vec8 A, I16vec8 B)`

`Is16vec8 unpack_high(Is16vec8 A, Is16vec8 B)`

`Iu16vec8 unpack_high(Iu16vec8 A, Iu16vec8 B)`

Aの上位半分から取り出した4個の16ビット値を、Bの上位半分から取り出した4個の16ビット値とインターリーブします。

R0 = A2;

R1 = B2; R2 = A3;

R3 = B3;

対応する組込み関数: `_mm_unpackhi_epi16`

`I16vec4 unpack_high(I16vec4 A, I16vec4 B)`

`Is16vec4 unpack_high(Is16vec4 A, Is16vec4 B)`

`Iu16vec4 unpack_high(Iu16vec4 A, Iu16vec4 B)`

Aの上位半分から取り出した2個の16ビット値を、Bの上位半分から取り出した2個の16ビット値とインターリーブします。

R0 = A2; R1 = B2;

R2 = A3; R3 = B3;

対応する組込み関数: `_mm_unpackhi_pi16`

`I8vec8 unpack_high(I8vec8 A, I8vec8 B)`

`Is8vec8 unpack_high(Is8vec8 A, I8vec8 B)`

`Iu8vec8 unpack_high(Iu8vec8 A, I8vec8 B)`

Aの上位半分から取り出した4個の8ビット値を、Bの上位半分から取り出した4個の8ビット値とインターリーブします。

R0 = A4;

R1 = B4;

R2 = A5;

R3 = B5;

R4 = A6;

R5 = B6;

R6 = A7;

R7 = B7;

対応する組込み関数: `_mm_unpackhi_pi8`

`I8vec16 unpack_high(I8vec16 A, I8vec16 B)`

`Is8vec16 unpack_high(Is8vec16 A, I8vec16 B)`

`Iu8vec16 unpack_high(Iu8vec16 A, I8vec16 B)`

Aの上位半分から取り出した8個の8ビット値を、Bの上位半分から取り出した8個の8ビット値とインターリーブします。

R0 = A8;

R1 = B8;

R2 = A9;

R3 = B9;

R4 = A10;

R5 = B10;

R6 = A11;

R7 = B11;

R8 = A12;

R8 = B12;

R2 = A13;

R3 = B13;

R4 = A14;

R5 = B14;

R6 = A15;

R7 = B15;

対応する組込み関数: `_mm_unpackhi_epi16`

Aの下位半分から取り出した32ビット値を、Bの下位半分から取り出した32ビット値とインターリーブします。

R0 = A0;

R1 = B0;

対応する組込み関数: `_mm_unpacklo_epi32`

`I64vec2 unpack_low(I64vec2 A, I64vec2 B);`

`Is64vec2 unpack_low(Is64vec2 A, Is64vec2 B);`

`Iu64vec2 unpack_low(Iu64vec2 A, Iu64vec2 B);`

Aの下位半分から取り出した64ビット値を、Bの下位半分から取り出した64ビット値とインターリーブします。

R0 = A0;

R1 = B0;

R2 = A1;

R3 = B1;

対応する組込み関数: `_mm_unpacklo_epi32`

`I32vec4 unpack_low(I32vec4 A, I32vec4 B);`

`Is32vec4 unpack_low(Is32vec4 A, Is32vec4 B);`

`Iu32vec4 unpack_low(Iu32vec4 A, Iu32vec4 B);`

Aの下位半分から取り出した2個の32ビット値を、Aの下位半分から取り出した2個の32ビット値とインターリーブします。

R0 = A0; R1 = B0;

R2 = A1; R3 = B1;

対応する組込み関数: `_mm_unpacklo_epi32`

`I32vec2 unpack_low(I32vec2 A, I32vec2 B);`

`Is32vec2 unpack_low(Is32vec2 A, Is32vec2 B);`

`Iu32vec2 unpack_low(Iu32vec2 A, Iu32vec2 B);`



Aの下位半分から取り出した32ビット値を、Bの下位半分から取り出した32ビット値とインターリーブします。

R0 = A0;

R1 = B0;

対応する組込み関数: `_mm_unpacklo_pi32`

`I16vec8 unpack_low(I16vec8 A, I16vec8 B);`

`Is16vec8 unpack_low(Is16vec8 A, Is16vec8 B);`

`Iu16vec8 unpack_low(Iu16vec8 A, Iu16vec8 B);`

Aの下位半分から取り出した2個の16ビット値を、Aの下位半分から取り出した2個の16ビット値とインターリーブします。

R0 = A0;

R1 = B0;

R2 = A1;

R3 = B1;

R4 = A2;

R5 = B2;

R6 = A3;

R7 = B3;

対応する組込み関数: `_mm_unpacklo_epi16`

`I16vec4 unpack_low(I16vec4 A, I16vec4 B);`

`Is16vec4 unpack_low(Is16vec4 A, Is16vec4 B);`

`Iu16vec4 unpack_low(Iu16vec4 A, Iu16vec4 B);`

Aの下位半分から取り出した2個の16ビット値を、Aの下位半分から取り出した2個の16ビット値とインターリーブします。

R0 = A0;

R1 = B0;

R2 = A1;

R3 = B1;

対応する組込み関数: `_mm_unpacklo_pi16`

`I8vec16 unpack_low(I8vec16 A, I8vec16 B);`

`Is8vec16 unpack_low(Is8vec16 A, Is8vec16 B);`

`Iu8vec16 unpack_low(Iu8vec16 A, Iu8vec16 B);`

Aの下位半分から取り出した4個の8ビット値を、Bの下位半分から取り出した4個の8ビット値

とインターリーブします。

```
R0 = A0;  
R1 = B0;  
R2 = A1;  
R3 = B1;  
R4 = A2;  
R5 = B2;  
R6 = A3;  
R7 = B3;  
R8 = A4;  
R9 = B4;  
R10 = A5;  
R11 = B5;  
R12 = A6;  
R13 = B6;  
R14 = A7;  
R15 = B7;
```

対応する組込み関数: `_mm_unpacklo_epi8`

```
I8vec8 unpack_low(I8vec8 A, I8vec8 B);  
Is8vec8 unpack_low(Is8vec8 A, Is8vec8 B);  
Iu8vec8 unpack_low(Iu8vec8 A, Iu8vec8 B);
```

Aの下位半分から取り出した4個の8ビット値を、Bの下位半分から取り出した4個の8ビット値とインターリーブします。

```
R0 = A0;  
R1 = B0;  
R2 = A1;  
R3 = B1;  
R4 = A2;  
R5 = B2;  
R6 = A3;  
R7 = B3;
```

対応する組込み関数: `_mm_unpacklo_pi8`

## パック演算子

```
Is16vec8 pack_sat(Is32vec2 A, Is32vec2 B);
```

AとBに含まれている8個の32ビット値を8個の16ビット値(符号付き、飽和あり)にパックします。

対応する組込み関数: `_mm_packs_epi32`

`Is16vec4 pack_sat(Is32vec2 A, Is32vec2 B);`

AとBに含まれている4個の32ビット値を4個の16ビット値(符号付き、飽和あり)にパックします。

対応する組込み関数: `_mm_packs_pi32`

`Is8vec16 pack_sat(Is16vec4 A, Is16vec4 B);`

AとBに含まれている16個の16ビット値を16個の8ビット値(符号付き、飽和あり)にパックします。

対応する組込み関数: `_mm_packs_epi16`

`Is8vec8 pack_sat(Is16vec4 A, Is16vec4 B);`

AとBに含まれている8個の16ビット値を8個の8ビット値(符号付き、飽和あり)にパックします。

対応する組込み関数: `_mm_packs_pi16`

`Iu8vec16 packu_sat(Is16vec4 A, Is16vec4 B);`

AとBに含まれている16個の16ビット値を16個の8ビット値(符号なし、飽和あり)にパックします。

対応する組込み関数: `_mm_packus_epi16`

`Iu8vec8 packu_sat(Is16vec4 A, Is16vec4 B);`

AとBに含まれている8個の16ビット値を8個の8ビット値(符号なし、飽和あり)にパックします。

対応する組込み関数: `_mm_packs_pu16`

## MMX® テクノロジ・ステート消去演算子

`void empty(void);`

MMX® テクノロジ・レジスタを空にして MMX テクノロジ・ステートを消去します。EMMS命令の組込み関数の使用方法に関するガイドラインをお読みください。

対応する組込み関数: `_mm_empty`

## ストリーミングSIMD拡張命令用の整数組込み関数



次の機能を使用する場合は、ヘッダファイル `fvec.h` をインクルードする必要があります。

`Is16vec4 simd_max(Is16vec4 A, Is16vec4 B);`

AとBに含まれているそれぞれの符号付き整数ワードの中で、要素単位で見たときの最大値を計算します。

対応する組込み関数: `_mm_max_pi16`

`Is16vec4 simd_min(Is16vec4 A, Is16vec4 B);`

AとBに含まれているそれぞれの符号付き整数ワードの中で、要素単位で見たときの最小値を計算します。

対応する組込み関数: `_mm_min_pi16`

`Iu8vec8 simd_max(Iu8vec8 A, Iu8vec8 B);`

AとBに含まれているそれぞれの符号なしバイトの中で、要素単位で見たときの最大値を計算します。

対応する組込み関数: `_mm_max_pu8`

`Iu8vec8 simd_min(Iu8vec8 A, Iu8vec8 B);`

Aと Bに含まれているそれぞれの符号なしバイトの中で、要素単位で見たときの最小値を計算します。

対応する組込み関数: `_mm_min_pu8`

`int move_mask(I8vec8 A);`

Aに含まれているすべてのバイトの最上位ビットから8ビットマスクを1個作成します。

対応する組込み関数: `_mm_movemask_pi8`

`void mask_move(I8vec8 A, I8vec8 B, signed char *p) ;`

条件に従って、Aのバイト要素をいくつかアドレスpにストアします。選択子B に含まれている各バイトの上位ビットによって、A内の対応バイトがストアされるかされないかが決まります。

対応する組込み関数: `_mm_maskmove_si64`

`void store_nta(__m64 *p, M64 A) ;`

当該キャッシュ・データに影響を与えることなく、Aに含まれているデータをアドレスpにストアします。A はIvec型であれば何でもかまいません。

対応する組込み関数: `_mm_stream_pi`

`Iu8vec8 simd_avg(Iu8vec8 A, Iu8vec8 B);`

AとBに含まれているそれぞれの符号なし8ビット整数すべてについて、要素単位で見たときの平均値を計算します。

対応する組込み関数: `_mm_avg_pu8`

`Iu16vec4 simd_avg(Iu16vec4 A, Iu16vec4 B);`

AとBに含まれているそれぞれの符号なし16ビット整数すべてについて、要素単位で見たときの平均値を計算します。

対応する組込み関数: `_mm_avg_pu16`

### 変換(Fvec $\longleftrightarrow$ Ivec)

倍精度浮動小数点値Aの下位側を、切り捨てモードで1個の32ビット整数に変換します。

```
int F64vec2ToInt(F64vec4 A);
```

```
r := (int)A0;
```

4個の浮動小数点値 A を、2個の最下位倍精度浮動小数点値に変換します。

```
F64vec2 F32vec4ToF64vec2(F32vec4 A);
```

```
r0 := (double)A0;
```

```
r1 := (double)A1;
```

2個の倍精度浮動小数点値Aを、2個の単精度浮動小数点値に変換します。

```
F32vec4 F64vec2ToF32vec4(F64vec2 A);
```

```
r0 := (float)A0;
```

```
r1 := (float)A1;
```

Bに含まれている符号付き整数を、1個の倍精度浮動小数点値に変換し、その上位側の倍精度値を Aから演算結果に渡します。

```
F64vec2 InttoF64vec2(F64vec2 A, int B);
```

```
r0 := (double)B;
```

```
r1 := A1;
```

浮動小数点値Aの下位側を、切り捨てモードで1個の32ビット整数に変換します。

```
int F32vec4ToInt(F32vec4 A);
```

```
r := (int)A0;
```

浮動小数点値Aの下位側の2個を、切り捨てモードで2個の32ビット整数に変換し、その整数をパックド形式で返します。

```
Is32vec2 F32vec4ToIs32vec2(F32vec4 A);
```

```
r0 := (int)A0;
```

```
r1 := (int)A1;
```

32ビット整数値Bを、1個の浮動小数点値に変換します。その上位3個の浮動小数点値はAから渡されます。

```
F32vec4 IntToF32vec4(F32vec4 A, int B);
```

```
r0 := (float)B;
```

```
r1 := A1;
```

```
r2 := A2;
```

```
r3 := A3;
```

Bの中にパックド形式で入っている2個の32ビット整数値を、2個の浮動小数点値に変換します。

その上位2個の浮動小数点値はAから渡されます。

```
F32vec4 Is32vec2ToF32vec4(F32vec4 A, Is32vec2 B);
```

```
r0 := (float)B0;
```

```
r1 := (float)B1;
```

```
r2 := A2;
```

```
r3 := A3;
```

# 浮動小数点ベクトルクラス

## 浮動小数点ベクトルクラスの概要

浮動小数点ベクトル・クラスである F64vec2、F32vec4、および F32vec1 が SIMD 演算へのインターフェイスとなります。このクラスの仕様は次のとおりです。

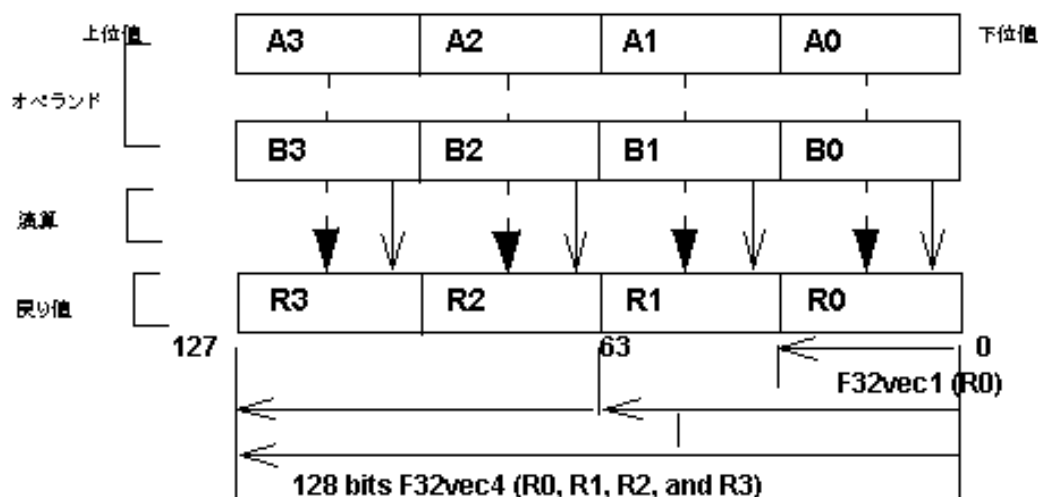
F64vec2 A(double x, double y);

F32vec4 A(float z, float y, float x, float w);

F32vec1 B(float w);

パックド浮動小数点の入力値は、下図に示すとおり最下位値を一番右側にして表現します。

### 単精度浮動小数点の諸要素



F32vec4は、パックド単精度浮動小数点値を4個(R0、R1、R2、およびR3)返します。  
F32vec2は、単精度浮動小数点値を1個(R0)返します。

## Fvecの表記法

本書では、構文と戻り値に次の表記法を使用しています。

### Fvecクラスの構文の表記法

Fvecクラスの構文には次の表記を使用します。

[Fvec\_Class] R = [Fvec\_Class] A [operator] [Ivec\_Class] B;

例 1: F64vec2 R = F64vec2 A & F64vec2 B;

[Fvec\_Class] R = [operator] ([Fvec\_Class] A, [Fvec\_Class] B);

例 2: F64vec2 R = andnot(F64vec2 A, F64vec2 B);

[Fvec\_Class] R [operator] = [Fvec\_Class] A;

例 3: F64vec2 R &= F64vec2 A;

各アイテムの意味は次のとおりです。

[operator] は演算子(例えば&、|、^)

[Fvec\_Class] は、Fvecクラスのいずれか(F64vec2、F32vec4、F32vec1)  
R、A、Bは、それぞれ宣言済みのFvec変数

## 戻り値の表記法

Fvecクラスはパックド要素をいくつか含んでいるため、通常その戻り値は下記の表「戻り値表記の対応表」に示した規則に従います。F32vec4は4個の単精度浮動小数点値(R0,R1,R2,R3)を返します。F64vec2は2個の倍精度浮動小数点値を返し、F32vec1は最下位単精度浮動小数点値(R0)を返します。

### 戻り値表記の対応表

| 例 1:              | 例 2:                      | 例 3:      | F32vec4 | F64vec2 | F32vec1 |
|-------------------|---------------------------|-----------|---------|---------|---------|
| R0 := A0<br>& B0; | R0 := A0<br>andnot<br>B0; | R0 &= A0; | x       | x       | x       |
| R1 := A1<br>& B1; | R1 := A1<br>andnot<br>B1; | R1 &= A1; | x       | x       | N/A     |
| R2 := A2<br>& B2; | R2 := A2<br>andnot<br>B2; | R2 &= A2; | x       | N/A     | N/A     |
| R3 := A3<br>& B3  | R3 := A3<br>andnot<br>B3; | R3 &= A3; | x       | N/A     | N/A     |

## データのアライメント

ストリーミングSIMD拡張命令を使用してメモリを操作する場合は、できる限り、16バイトでアライメントされたデータで行ってください。

F32vec4 およびF64vec2 オブジェクトの各変数は、デフォルトで正しくアライメントされています。浮動小数点配列は自動的にアライメントされないので注意してください。16バイトでアライメントするには、\_\_declspec を次のように使用します。

```
__declspec( align(16) ) float A[4];
```

## 変換

```
__m128d mm = A & B; /* where A,B are F64vec2 object variables */
```

```
__m128 mm = A & B; /* where A,B are F32vec4 object variables */
```

```
__m128 mm = A & B; /* where A,B are F32vec1 object variables */
```

Fvecオブジェクトのすべての変数は\_\_m128データ型に暗黙的に変換できます。例えば、F32vec4オブジェクトもF32vec1オブジェクトも、その各変数に対して実行された計算結果は\_\_m128データ型に変換できます。

## コンストラクタと初期化

次の表に、FvecクラスでF32vecオブジェクトを作成して初期化する方法を示します。



## Fvecクラスのコンストラクタと初期化

| 例                                                                                                                  | 組み込み関数         | 戻り値                                              |
|--------------------------------------------------------------------------------------------------------------------|----------------|--------------------------------------------------|
| <b>コンストラクタの宣言</b>                                                                                                  |                |                                                  |
| F64vec2 A;<br>F32vec4 B;<br>F32vec1 C;                                                                             | N/A            | N/A                                              |
| <b>__m128オブジェクトの初期化</b>                                                                                            |                |                                                  |
| F64vec2 A(__m128d mm);<br>F32vec4 B(__m128 mm);<br>F32vec1 C(__m128 mm);                                           | N/A            | N/A                                              |
| <b>double型の初期化</b>                                                                                                 |                |                                                  |
| /* Initializes two doubles. */<br>F64vec2 A(double d0, double d1);<br>F64vec2 A = F64vec2(double d0, double d1);   | _mm_set_pd     | A0 := d0;<br>A1 := d1;                           |
| F64vec2 A(doubled0);<br>/* Initializes both return values with the same double precision value */.                 | _mm_set1_pd    | A0 := d0;<br>A1 := d0;                           |
| <b>float型の初期化</b>                                                                                                  |                |                                                  |
| F32vec4 A(float f3, float f2, float f1, float f0);<br>F32vec4 A = F32vec4(float f3, float f2, float f1, float f0); | _mm_set_ps     | A0 := f0;<br>A1 := f1;<br>A2 := f2;<br>A3 := f3; |
| F32vec4 A(float f0);<br>/* Initializes all return values with the same floatingpoint value. */                     | _mm_set1_ps    | A0 := f0;<br>A1 := f0;<br>A2 := f0;<br>A3 := f0; |
| F32vec4 A(doubled0);<br>/* Initialize all return values with the same double-precision value. */                   | _mm_set1_ps(d) | A0 := d0;<br>A1 := d0;<br>A2 := d0;<br>A3 := d0; |
| F32vec1 A(doubled0);                                                                                               | mm_set_ss(d)   | A0 := d0;                                        |

|                                                                                                     |                |                                               |
|-----------------------------------------------------------------------------------------------------|----------------|-----------------------------------------------|
| /* Initializes the lowest value of A with d0 and the other values with 0.*/                         |                | A1 := 0;<br>A2 := 0;<br>A3 := 0;              |
| F32vec1 B(float f0);<br>/* Initializes the lowest value of B with f0 and the other values with 0.*/ | _mm_set_ss     | B0 := f0;<br>B1 := 0;<br>B2 := 0;<br>B3 := 0; |
| F32vec1 B(int I);<br>/* Initializes the lowest value of B with f0, other values are undefined.*/    | _mm_cvtsi32_ss | B0 := f0;<br>B1 := {}<br>B2 := {}<br>B3 := {} |

## 算術演算子

次の表は、Fvecクラスの算術演算子と一般的な構文を列挙したものです。この算術演算子はそれぞれ「標準演算」と「高度な演算」に分かれています。この2つについては本節の後半で詳しく解説します。

### Fvec算術演算子

| 分類    | 操作                      | 演算子               | 一般的な構文                               |
|-------|-------------------------|-------------------|--------------------------------------|
| 標準    | 加算                      | +<br>+=           | R = A + B;<br>R += A;                |
|       | 減算                      | -<br>-=           | R = A - B;<br>R -= A;                |
|       | 乗算                      | *<br>*=           | R = A * B;<br>R *= A;                |
|       | 除算                      | /<br>/=           | R = A / B;<br>R /= A;                |
| 高度な演算 | 平方根                     | sqrt              | R = sqrt(A);                         |
|       | 逆数<br>(ニュートン・ラフソン法)     | rcp<br>rcp_nr     | R = rcp(A);<br>R = rcp_nr(A);        |
|       | 平方根の逆数<br>(ニュートン・ラフソン法) | rsqrt<br>rsqrt_nr | R = rsqrt(A);<br>R =<br>rsqrt_nr(A); |

## 「標準算術演算子」の使用

次の2つの表は、標準算術演算子のクラスごとに戻り値を列挙したものです。「戻り値の表記法」節の前半に述べた構文が使われます。

## 標準算術演算の戻り値の対応表

| R    | A  | 演算子 |   |   |   | B  | F32vec4 | F64vec2 | F32vec1 |
|------|----|-----|---|---|---|----|---------|---------|---------|
| R0:= | A0 | +   | - | * | / | B0 |         |         |         |
| R1:= | A1 | +   | - | * | / | B1 |         |         | 非対応     |
| R2:= | A2 | +   | - | * | / | B2 |         | 非対応     | 非対応     |
| R3:= | A3 | +   | - | * | / | B3 |         | 非対応     | 非対応     |

## 代入付き算術演算の戻り値の対応表

| R    | 演算子 |    |    |    | A  | F32vec4 | F64vec2 | F32vec1 |
|------|-----|----|----|----|----|---------|---------|---------|
| R0:= | +=  | -= | *= | /= | A0 |         |         |         |
| R1:= | +=  | -= | *= | /= | A1 |         |         | 非対応     |
| R2:= | +=  | -= | *= | /= | A2 |         | 非対応     | 非対応     |
| R3:= | +=  | -= | *= | /= | A3 |         | 非対応     | 非対応     |

次の表は、標準算術演算子の構文と組み込み関数を列挙したものです。

## Fvecクラスの標準算術演算

| 操作 | 戻り値       | 構文の使用例                                                                 | 組み込み関数     |
|----|-----------|------------------------------------------------------------------------|------------|
| 加算 | floatが4個  | F32vec4 R =<br>F32vec4 A +<br>F32vec4 B;<br>F32vec4 R +=<br>F32vec4 A; | _mm_add_ps |
|    | doubleが2個 | F64vec2 R =<br>F64vec2 A +<br>F32vec2 B;<br>F64vec2 R +=<br>F64vec2 A; | _mm_add_pd |
|    | floatが1個  | F32vec1 R =<br>F32vec1 A +<br>F32vec1 B;<br>F32vec1 R +=<br>F32vec1 A; | _mm_add_ss |
| 減算 | floatが4個  | F32vec4 R =<br>F32vec4 A -<br>F32vec4 B;<br>F32vec4 R -=<br>F32vec4 A; | _mm_sub_ps |
|    | doubleが2個 | F64vec2 R =<br>F64vec2 A -<br>F32vec2 B;                               | _mm_sub_pd |

|    |           |                                                                        |            |
|----|-----------|------------------------------------------------------------------------|------------|
|    |           | F64vec2 R -=<br>F64vec2 A;                                             |            |
|    | floatが1個  | F32vec1 R =<br>F32vec1 A -<br>F32vec1 B;<br>F32vec1 R -=<br>F32vec1 A; | _mm_sub_ss |
| 乗算 | floatが4個  | F32vec4 R =<br>F32vec4 A *<br>F32vec4 B;<br>F32vec4 R *=<br>F32vec4 A; | _mm_mul_ps |
|    | doubleが2個 | F64vec2 R =<br>F64vec2 A *<br>F64vec2 B;<br>F64vec2 R *=<br>F64vec2 A; | _mm_mul_pd |
|    | floatが1個  | F32vec1 R =<br>F32vec1 A *<br>F32vec1 B;<br>F32vec1 R *=<br>F32vec1 A; | _mm_mul_ss |
| 除算 | floatが4個  | F32vec4 R =<br>F32vec4 A /<br>F32vec4 B;<br>F32vec4 R /=<br>F32vec4 A; | _mm_div_ps |
|    | doubleが2個 | F64vec2 R =<br>F64vec2 A /<br>F64vec2 B;<br>F64vec2 R /=<br>F64vec2 A; | _mm_div_pd |
|    | floatが1個  | F32vec1 R =<br>F32vec1 A /<br>F32vec1 B;<br>F32vec1 R /=<br>F32vec1 A; | _mm_div_ss |

## 「高度な算術演算子」の使用

次の表は、高度な算術演算子のクラスごとの戻り値を列挙したものです。各戻り値は、「戻り値の表記法」の節の前半に述べた表記法に従っています。

### 高度な算術演算子の戻り値の対応表

| R | 演算子 | A | F32vec1 | F64vec2 | F32vec1 |
|---|-----|---|---------|---------|---------|
|---|-----|---|---------|---------|---------|

|      |                |     |       |                     |          |    |     |     |     |
|------|----------------|-----|-------|---------------------|----------|----|-----|-----|-----|
| R0:= | sqrt           | rcp | rsqrt | rcp_nr              | rsqrt_nr | A0 |     |     |     |
| R1:= | sqrt           | rcp | rsqrt | rcp_nr              | rsqrt_nr | A1 |     |     | 非対応 |
| R2:= | sqrt           | rcp | rsqrt | rcp_nr              | rsqrt_nr | A2 |     | 非対応 | 非対応 |
| R3:= | sqrt           | rcp | rsqrt | rcp_nr              | rsqrt_nr | A3 |     | 非対応 | 非対応 |
| f := | add_horizontal |     |       | (A0 + A1 + A2 + A3) |          |    |     | 非対応 | 非対応 |
| d := | add_horizontal |     |       | (A0 + A1)           |          |    | 非対応 |     | 非対応 |

次の表に、高度な算術演算子の例をいくつか示します。

## Fvecクラス用の高度な算術演算

| 戻り値             | 構文の使用例                         | 組込み関数                                                |
|-----------------|--------------------------------|------------------------------------------------------|
| 平方根             |                                |                                                      |
| floatが4個        | F32vec4 R = sqrt(F32vec4 A);   | _mm_sqrt_ps                                          |
| doubleが2個       | F64vec2 R = sqrt(F64vec2 A);   | _mm_sqrt_pd                                          |
| floatが1個        | F32vec1 R = sqrt(F32vec1 A);   | _mm_sqrt_ss                                          |
| 逆数              |                                |                                                      |
| floatが4個        | F32vec4 R = rcp(F32vec4 A);    | _mm_rcp_ps                                           |
| doubleが2個       | F64vec2 R = rcp(F64vec2 A);    | _mm_rcp_pd                                           |
| floatが1個        | F32vec1 R = rcp(F32vec1 A);    | _mm_rcp_ss                                           |
| 平方根の逆数          |                                |                                                      |
| floatが4個        | F32vec4 R = rsqrt(F32vec4 A);  | _mm_rsqrt_ps                                         |
| doubleが2個       | F64vec2 R = rsqrt(F64vec2 A);  | _mm_rsqrt_pd                                         |
| floatが1個        | F32vec1 R = rsqrt(F32vec1 A);  | _mm_rsqrt_ss                                         |
| 逆数(ニュートン・ラフソン法) |                                |                                                      |
| floatが4個        | F32vec4 R = rcp_nr(F32vec4 A); | _mm_sub_ps<br>_mm_add_ps<br>_mm_mul_ps<br>_mm_rcp_ps |
| doubleが2個       | F64vec2 R = rcp_nr(F64vec2 A); | _mm_sub_pd<br>_mm_add_pd<br>_mm_mul_pd               |

|                     |                                          |                                                     |
|---------------------|------------------------------------------|-----------------------------------------------------|
|                     |                                          | mm_rcp_pd                                           |
| floatが1個            | F32vec1 R =<br>rcp_nr(F32vec1 A);        | _mm_sub_ss<br>_mm_add_ss<br>_mm_mul_ss<br>mm_rcp_ss |
| 平方根の逆数(ニュートン・ラフソン法) |                                          |                                                     |
| floatが4個            | F32vec4 R =<br>rsqrt_nr(F32vec4 A);      | _mm_sub_pd<br>_mm_mul_pd<br>mm_rsqrt_ps             |
| doubleが2個           | F64vec2 R =<br>rsqrt_nr(F64vec2 A);      | _mm_sub_pd<br>_mm_mul_pd<br>mm_rsqrt_pd             |
| floatが1個            | F32vec1 R =<br>rsqrt_nr(F32vec1 A);      | _mm_sub_ss<br>_mm_mul_ss<br>mm_rsqrt_ss             |
| 水平加算                |                                          |                                                     |
| floatが1個            | float f =<br>add_horizontal(F32vec4 A);  | _mm_add_ss<br>mm_shuffle_ss                         |
| doubleが1個           | double d =<br>add_horizontal(F64vec2 A); | _mm_add_sd<br>mm_shuffle_sd                         |

## 最小値演算子と最大値演算子

F64vec2 R = simd\_min(F64vec2 A, F64vec2 B)

2個の倍精度浮動小数点値A、Bの中での最小値を計算します。

R0 := min(A0,B0);

R1 := min(A1,B1);

対応する組込み関数: \_mm\_min\_pd

F32vec4 R = simd\_min(F32vec4 A, F32vec4 B)

4個の単精度浮動小数点値A、Bの中での最小値を計算します。

R0 := min(A0,B0);

R1 := min(A1,B1);

R2 := min(A2,B2);

R3 := min(A3,B3);

対応する組込み関数: \_mm\_min\_ps

F32vec1 R = simd\_min(F32vec1 A, F32vec1 B)

最下位単精度浮動小数点値A、Bの中での最小値を計算します。

R0 := min(A0,B0);

対応する組込み関数: `_mm_min_ss`

F64vec2 `simd_max`(F64vec2 A, F64vec2 B)

2個の倍精度浮動小数点値A、Bの中での最大値を計算します。

R0 := max(A0,B0);

R1 := max(A1,B1);

対応する組込み関数: `_mm_max_pd`

F32vec4 R = `simd_max`(F32vec4 A, F32vec4 B)

4個の単精度浮動小数点値A、Bの中での最大値を計算します。

R0 := max(A0,B0);

R1 := max(A1,B1);

R2 := max(A2,B2);

R3 := max(A3,B3);

対応する組込み関数: `_mm_max_ps`

F32vec1 `simd_max`(F32vec1 A, F32vec1 B)

最下位単精度浮動小数点値A、Bの中での最大値を計算します。

R0 := max(A0,B0);

対応する組込み関数: `_mm_max_ss`

## 論理演算子

下の「Fvec論理演算子の戻り値の対応表」は、Fvecクラスの論理演算子とその一般的な構文とを列挙したものです。F32vec1クラスの論理演算子のときは下位32ビットだけを使用します。

### Fvec論理演算子の戻り値の対応表

| ビット単位演算 | 演算子     | 一般的な構文                |
|---------|---------|-----------------------|
| AND     | &<br>&= | R = A & B;<br>R &= A; |
| OR      | <br> =  | R = A   B;<br>R  = A; |
| XOR     | ^<br>^= | R = A ^ B;<br>R ^= A; |
| andnot  | andnot  | R = andnot(A);        |

次の表は、論理演算子の標準的な構文と、対応する組込み関数を示したものです。

F32vec1クラスについては、対応するスカラー組込み関数がなく、パックドベクトル組込み関数の下位32ビットにアクセスするのに注意してください。

## Fvecクラスの論理演算

| 操作  | 戻り値       | 構文の使用例                                                               | 組込み関数      |
|-----|-----------|----------------------------------------------------------------------|------------|
| AND | floatが4個  | F32vec4 R = F32vec4 A<br>& F32vec4 B;<br>F32vec4 R &=<br>F32vec4 A;  | _mm_and_ps |
|     | doubleが2個 | F64vec2 R = F64vec2 A<br>& F32vec2 B;<br>F64vec2 R &=<br>F64vec2 A;  | _mm_and_pd |
|     | floatが1個  | F32vec1 R = F32vec1 A<br>& F32vec1 B;<br>F32vec1 R &=<br>F32vec1 A;  | _mm_and_ps |
| OR  | floatが4個  | F32vec4 R = F32vec4 A<br>  F32vec4 B;<br>F32vec4 R  =<br>F32vec4 A;  | _mm_or_ps  |
|     | doubleが2個 | F64vec2 R = F64vec2 A<br>  F32vec2 B;<br>F64vec2 R  =<br>F64vec2 A;  | _mm_or_pd  |
|     | floatが1個  | F32vec1 R = F32vec1 A<br>  F32vec1 B;<br>F32vec1 R  =<br>F32vec1 A;  | _mm_or_ps  |
| XOR | floatが4個  | F32vec4 R = F32vec4 A<br>^ F32vec4 B;<br>F32vec4 R ^=<br>F32vec4 A;  | _mm_xor_ps |
|     | doubleが2個 | F64vec2 R = F64vec2 A<br>^ F364vec2 B;<br>F64vec2 R ^=<br>F64vec2 A; | _mm_xor_pd |
|     | floatが1個  | F32vec1 R = F32vec1 A<br>^ F32vec1 B;<br>F32vec1 R ^=<br>F32vec1 A;  | _mm_xor_ps |



|        |           |                                                  |               |
|--------|-----------|--------------------------------------------------|---------------|
| ANDNOT | doubleが2個 | F64vec2 R =<br>andnot(F64vec2 A,<br>F64vec2 B) ; | _mm_andnot_pd |
|--------|-----------|--------------------------------------------------|---------------|

## 比較演算子

この節では、単精度浮動小数点値であるAとBを比較する演算子について説明します。Fvecクラスのオブジェクト同士を比較した場合は、比較されたオブジェクトと同じクラスで戻り値が返ってきます。

次の表は、Fvecクラスの比較演算子を示したものです。

### 比較演算子と対応する組み関数

| 比較の条件   | 演算子    | 構文               |
|---------|--------|------------------|
| 等しい     | cmpeq  | R = cmpeq(A, B)  |
| 等しくない   | cmpneq | R = cmpneq(A, B) |
| より大きい   | cmpgt  | R = cmpgt(A, B)  |
| ≥       | cmpge  | R = cmpge(A, B)  |
| より大きくない | cmpngt | R = cmpngt(A, B) |
| ≥ではない   | cmpnge | R = cmpnge(A, B) |
| より小さい   | cmplt  | R = cmplt(A, B)  |
| ≤       | cmple  | R = cmple(A, B)  |
| より小さくない | cmpnlt | R = cmpnlt(A, B) |
| ≤ではない   | cmpnle | R = cmpnle(A, B) |

## 比較演算子

比較した結果が真の場合、そのマスクは浮動小数点値ごとに0xffffffffにセットされます。同様に、偽の場合は0x00000000にセットされます。次の表は比較演算子のクラスごとの戻り値を示したものです。各戻り値の表記は、「戻り値の表記法」の節の前半に述べた表記法に従っています。

### 比較演算子の戻り値の対応表

| R    | A0          | 演算子                                                                    | B          | 真の場合       | 偽の場合       | F32vec4 | F64vec2 | F32vec1 |
|------|-------------|------------------------------------------------------------------------|------------|------------|------------|---------|---------|---------|
| R0:= | (A1<br>!(A1 | cmp[eq   lt  <br>le   gt   ge]<br>cmp[ne   nlt<br>  nle   ng  <br>nge] | B1)<br>B1) | 0xffffffff | 0x00000000 | X       | X       | X       |
| R1:= | (A1<br>!(A1 | cmp[eq   lt  <br>le   gt   ge]<br>cmp[ne   nlt                         | B2)<br>B2) | 0xffffffff | 0x00000000 | X       | X       | 非対応     |

|      |             |                                                                        |            |            |            |   |     |     |
|------|-------------|------------------------------------------------------------------------|------------|------------|------------|---|-----|-----|
|      |             | nle   ng   nge]                                                        |            |            |            |   |     |     |
| R2:= | (A1<br>!(A1 | cmp[eq   lt  <br>le   gt   ge]<br>cmp[ne   nlt<br>  nle   ng  <br>nge] | B3)<br>B3) | 0xffffffff | 0x00000000 | X | 非対応 | 非対応 |
| R3:= | A3          | cmp[eq   lt  <br>le   gt   ge]<br>cmp[ne   nlt<br>  nle   ng  <br>nge] | B3)<br>B3) | 0xffffffff | 0x00000000 | X | 非対応 | 非対応 |

次の表に、比較演算と組込み関数の例をいくつか示します。

## Fvecクラスの比較演算

| 戻り値        | 構文の使用例                            | 組込み関数         |
|------------|-----------------------------------|---------------|
| 「=」かどうかの比較 |                                   |               |
| floatが4個   | F32vec4 R =<br>cmpeq(F32vec4 A);  | _mm_cmpeq_ps  |
| doubleが2個  | F64vec2 R =<br>cmpeq(F64vec2 A);  | _mm_cmpeq_pd  |
| floatが1個   | F32vec1 R =<br>cmpeq(F32vec1 A);  | _mm_cmpeq_ss  |
| 「≠」かどうかの比較 |                                   |               |
| floatが4個   | F32vec4 R =<br>cmpneq(F32vec4 A); | _mm_cmpneq_ps |
| doubleが2個  | F64vec2 R =<br>cmpneq(F64vec2 A); | _mm_cmpneq_pd |
| floatが1個   | F32vec1 R =<br>cmpneq(F32vec1 A); | _mm_cmpneq_ss |
| 「<」かどうかの比較 |                                   |               |
| floatが4個   | F32vec4 R =<br>cmplt(F32vec4 A);  | _mm_cmplt_ps  |
| doubleが2個  | F64vec2 R =<br>cmplt(F64vec2 A);  | _mm_cmplt_pd  |
| floatが1個   | F32vec1 R =<br>cmplt(F32vec1 A);  | _mm_cmplt_ss  |
| 「≤」かどうかの比較 |                                   |               |
| floatが4個   | F32vec4 R =<br>cmple(F32vec4 A);  | _mm_cmple_ps  |
| doubleが2個  | F64vec2 R =                       | _mm_cmple_pd  |

|                |                                   |               |
|----------------|-----------------------------------|---------------|
|                | cmple(F64vec2 A);                 |               |
| floatが1個       | F32vec1 R =<br>cmple(F32vec1 A);  | _mm_cmple_pd  |
| 「>」かどうかの比較     |                                   |               |
| floatが4個       | F32vec4 R =<br>cmpgt(F32vec4 A);  | _mm_cmpgt_ps  |
| doubleが2個      | F64vec2 R =<br>cmpgt(F32vec42 A); | _mm_cmpgt_pd  |
| floatが1個       | F32vec1 R =<br>cmpgt(F32vec1 A);  | _mm_cmpgt_ss  |
| 「≥」かどうかの比較     |                                   |               |
| floatが4個       | F32vec4 R =<br>cmpge(F32vec4 A);  | _mm_cmpge_ps  |
| doubleが2個      | F64vec2 R =<br>cmpge(F64vec2 A);  | _mm_cmpge_pd  |
| floatが1個       | F32vec1 R =<br>cmpge(F32vec1 A);  | _mm_cmpge_ss  |
| 「<ではない」かどうかの比較 |                                   |               |
| floatが4個       | F32vec4 R =<br>cmpnlt(F32vec4 A); | _mm_cmpnlt_ps |
| doubleが2個      | F64vec2 R =<br>cmpnlt(F64vec2 A); | _mm_cmpnlt_pd |
| floatが1個       | F32vec1 R =<br>cmpnlt(F32vec1 A); | _mm_cmpnlt_ss |
| 「≤ではない」かどうかの比較 |                                   |               |
| floatが4個       | F32vec4 R =<br>cmpnle(F32vec4 A); | _mm_cmpnle_ps |
| doubleが2個      | F64vec2 R =<br>cmpnle(F64vec2 A); | _mm_cmpnle_pd |
| floatが1個       | F32vec1 R =<br>cmpnle(F32vec1 A); | _mm_cmpnle_ss |
| 「>ではない」かどうかの比較 |                                   |               |
| floatが4個       | F32vec4 R =<br>cmpngt(F32vec4 A); | _mm_cmpngt_ps |
| doubleが2個      | F64vec2 R =<br>cmpngt(F64vec2 A); | _mm_cmpngt_pd |

|                       |                                   |               |
|-----------------------|-----------------------------------|---------------|
| floatが1個              | F32vec1 R =<br>cmpngt(F32vec1 A); | _mm_cmpngt_ss |
| 「 $\geq$ ではない」かどうかの比較 |                                   |               |
| floatが4個              | F32vec4 R =<br>cmpnge(F32vec4 A); | _mm_cmpnge_ps |
| doubleが2個             | F64vec2 R =<br>cmpnge(F64vec2 A); | _mm_cmpnge_pd |
| floatが1個              | F32vec1 R =<br>cmpnge(F32vec1 A); | _mm_cmpnge_ss |

## Fvecクラスの条件付き選択演算子

各条件付き関数は、単精度浮動小数点値であるAとBを比較します。パラメータC、Dは戻り値に使用されます。Fvecクラスのオブジェクト同士を比較したときの戻り値のデータ型は、比較したオブジェクトと同じデータ型になります。

### Fvecクラスの条件付き選択演算子

| 条件付き選択の条件   | 演算子        | 構文                   |
|-------------|------------|----------------------|
| 等しい         | select_eq  | R = select_eq(A, B)  |
| 等しくない       | select_neq | R = select_neq(A, B) |
| より大きい       | select_gt  | R = select_gt(A, B)  |
| $\geq$      | select_ge  | R = select_ge(A, B)  |
| より大きくない     | select_gt  | R = select_gt(A, B)  |
| $\geq$ ではない | select_ge  | R = select_ge(A, B)  |
| より小さい       | select_lt  | R = select_lt(A, B)  |
| $\leq$      | select_le  | R = select_le(A, B)  |
| より小さくない     | select_nlt | R = select_nlt(A, B) |
| $\leq$ ではない | select_nle | R = select_nle(A, B) |

## 条件付き選択演算子の使用

条件付き選択演算子では、比較結果が真の場合は戻り値がCに格納され、偽の場合はDに格納されます。次の表は、条件付き選択演算子のクラスごとの戻り値を列挙したものです。各戻り値は、先に説明した「戻り値の表記法」に従っています。

### 比較演算子の戻り値の対応表

| R    | A0          | 演算子                                   | B          | C        | D        | F32vec4  | F64vec2  | F32vec1  |
|------|-------------|---------------------------------------|------------|----------|----------|----------|----------|----------|
| R0:= | (A1<br>!(A1 | select_[eq  <br>lt   le   gt  <br>ge] | B0)<br>B0) | C0<br>C0 | D0<br>D0 | <b>X</b> | <b>X</b> | <b>X</b> |

|      |             |                                                                                   |            |          |          |   |     |     |
|------|-------------|-----------------------------------------------------------------------------------|------------|----------|----------|---|-----|-----|
|      |             | select_[ne  <br>nlt   nle   ng<br>  nge]                                          |            |          |          |   |     |     |
| R1:= | (A2<br>!(A2 | select_[eq  <br>lt   le   gt  <br>ge]<br>select_[ne  <br>nlt   nle   ng<br>  nge] | B1)<br>B1) | C1<br>C1 | D1<br>D1 | X | X   | 非対応 |
| R2:= | (A2<br>!(A2 | select_[eq  <br>lt   le   gt  <br>ge]<br>select_[ne  <br>nlt   nle   ng<br>  nge] | B2)<br>B2) | C2<br>C2 | D2<br>D2 | X | 非対応 | 非対応 |
| R3:= | (A3<br>!(A3 | select_[eq  <br>lt   le   gt  <br>ge]<br>select_[ne  <br>nlt   nle   ng<br>  nge] | B3)<br>B3) | C3<br>C3 | D3<br>D3 | X | 非対応 | 非対応 |

次の表に、条件付き選択演算および対応する組込み関数の例をいくつか示します。

## Fvecクラスの条件付き選択演算

| 戻り値        | 構文の使用例                                | 組込み関数         |
|------------|---------------------------------------|---------------|
| 「=」かどうかの比較 |                                       |               |
| floatが4個   | F32vec4 R =<br>select_eq(F32vec4 A);  | _mm_cmpeq_ps  |
| doubleが2個  | F64vec2 R =<br>select_eq(F64vec2 A);  | _mm_cmpeq_pd  |
| floatが1個   | F32vec1 R =<br>select_eq(F32vec1 A);  | _mm_cmpeq_ss  |
| 「≠」かどうかの比較 |                                       |               |
| floatが4個   | F32vec4 R =<br>select_neq(F32vec4 A); | _mm_cmpneq_ps |
| doubleが2個  | F64vec2 R =<br>select_neq(F64vec2 A); | _mm_cmpneq_pd |
| floatが1個   | F32vec1 R =<br>select_neq(F32vec1 A); | _mm_cmpneq_ss |
| 「<」かどうかの比較 |                                       |               |
| floatが4個   | F32vec4 R =<br>select_lt(F32vec4 A);  | _mm_cmplt_ps  |

|                       |                                       |               |
|-----------------------|---------------------------------------|---------------|
| doubleが2個             | F64vec2 R =<br>select_lt(F64vec2 A);  | _mm_cmplt_pd  |
| floatが1個              | F32vec1 R =<br>select_lt(F32vec1 A);  | _mm_cmplt_ss  |
| 「 $\leq$ 」かどうかの比較     |                                       |               |
| floatが4個              | F32vec4 R =<br>select_le(F32vec4 A);  | _mm_cmple_ps  |
| doubleが2個             | F64vec2 R =<br>select_le(F64vec2 A);  | _mm_cmple_pd  |
| floatが1個              | F32vec1 R =<br>select_le(F32vec1 A);  | _mm_cmple_ss  |
| 「 $>$ 」かどうかの比較        |                                       |               |
| floatが4個              | F32vec4 R =<br>select_gt(F32vec4 A);  | _mm_cmpgt_ps  |
| doubleが2個             | F64vec2 R =<br>select_gt(F64vec2 A);  | _mm_cmpgt_pd  |
| floatが1個              | F32vec1 R =<br>select_gt(F32vec1 A);  | _mm_cmpgt_ss  |
| 「 $\geq$ 」かどうかの比較     |                                       |               |
| floatが4個              | F32vec1 R =<br>select_ge(F32vec4 A);  | _mm_cmpge_ps  |
| doubleが2個             | F64vec2 R =<br>select_ge(F64vec2 A);  | _mm_cmpge_pd  |
| floatが1個              | F32vec1 R =<br>select_ge(F32vec1 A);  | _mm_cmpge_ss  |
| 「 $<$ ではない」かどうかの比較    |                                       |               |
| floatが4個              | F32vec1 R =<br>select_nlt(F32vec4 A); | _mm_cmpnlt_ps |
| doubleが2個             | F64vec2 R =<br>select_nlt(F64vec2 A); | _mm_cmpnlt_pd |
| floatが1個              | F32vec1 R =<br>select_nlt(F32vec1 A); | _mm_cmpnlt_ss |
| 「 $\leq$ ではない」かどうかの比較 |                                       |               |
| floatが4個              | F32vec1 R =<br>select_nle(F32vec4 A); | _mm_cmpnle_ps |
| doubleが2個             | F64vec2 R =<br>select_nle(F64vec2 A); | _mm_cmpnle_pd |

|                |                                       |               |
|----------------|---------------------------------------|---------------|
| floatが1個       | F32vec1 R =<br>select_nle(F32vec1 A); | _mm_cmpnle_ss |
| 「>ではない」かどうかの比較 |                                       |               |
| floatが4個       | F32vec1 R =<br>select_ngt(F32vec4 A); | _mm_cmpngt_ps |
| doubleが2個      | F64vec2 R =<br>select_ngt(F64vec2 A); | _mm_cmpngt_pd |
| floatが1個       | F32vec1 R =<br>select_ngt(F32vec1 A); | _mm_cmpngt_ss |
| 「≥ではない」かどうかの比較 |                                       |               |
| floatが4個       | F32vec1 R =<br>select_nge(F32vec4 A); | _mm_cmpnge_ps |
| doubleが2個      | F64vec2 R =<br>select_nge(F64vec2 A); | _mm_cmpnge_pd |
| floatが1個       | F32vec1 R =<br>select_nge(F32vec1 A); | _mm_cmpnge_ss |

## キャッシュ操作

```
void store_nta(double *p, F64vec2 A);
```

Aという、すぐには参照しない2個の倍精度浮動小数点値をストアします。16バイトでアライメントされたアドレスが1つ必要です。

対応する組込み関数: `_mm_stream_pd`

```
void store_nta(float *p, F32vec4 A);
```

Aという、すぐには参照しない4個の単精度浮動小数点値をストアします。16バイトでアライメントされたアドレスが1つ必要です。

対応する組込み関数: `_mm_stream_ps`

## デバッグ

デバッグ演算は、MMX® テクノロジ命令やストリーミングSIMD拡張命令にも、どのコンパイラ組込み関数とも対応関係はありません。単にプログラムのデバッグに使用される演算にすぎません。この演算を使用するとパフォーマンスが下がる場合がありますので、デバッグの目的以外には使用しないでください。

## 出力演算

2個の倍精度浮動小数点値Aが出力バッファに格納され、次のように10進数形式で出力されます。

```
cout << F64vec2 A;  
"[1]:A1 [0]:A0"  
対応する組込み関数:なし
```

4個の単精度浮動小数点値Aが出力バッファに格納され、次のように10進数形式で出力されます。

```
cout << F32vec4 A;  
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"  
対応する組込み関数:なし
```

最下位単精度浮動小数点値Aが出力バッファに格納され出力されます。

```
cout << F32vec1 A;  
対応する組込み関数:なし
```

## 要素アクセス演算

```
double d = F64vec2 A[int i]
```

対応する浮動小数点値を変更することなく、2個の倍精度浮動小数点値Aのうちの1個を読み取ります。iの許容値は0と1です。次に例を示します。

DEBUGが有効になっている場合、iが許容値(0または1)のいずれでもないときは、診断メッセージが出力され、プログラムは途中で終了します。

```
double d = F64vec2 A[1];  
対応する組込み関数:なし
```

対応する浮動小数点値を変更せずに、4個の単精度浮動小数点値Aのうちの1個を読み取ります。iの許容値は0、1、2、3です。次に例を示します。

```
float f = F32vec4 A[int i]
```

DEBUGが有効になっている場合、iが許容値(0～3)のいずれでもないときは、診断メッセージが出力され、プログラムは途中で終了します。

```
float f = F32vec4 A[2];  
対応する組込み関数:なし
```

## 要素代入演算

```
F64vec4 A[int i] = double d;
```

2個の倍精度浮動小数点値Aのうちの1個を変更します。整数iの許容値は0と1です。次に例を示します。

```
F32vec4 A[1] = double d;  
F32vec4 A[int i] = float f;
```

4個の単精度浮動小数点値Aのうちの1個を変更します。整数iの許容値は0、1、2、3です。次に例を示します。

```
DEBUGが有効になっている場合、整数iが許容値(0～3)のいずれでもないときは、診断メッセージが出力され、プログラムは途中で終了します。  
F32vec4 A[3] = float f;
```



対応する組込み関数: なし

## ロード演算子とストア演算子

```
void loadu(F64vec2 A, double *p)
```

2個の倍精度浮動小数点値をロードし、それらを2個の浮動小数点値Aにコピーします。アライメントは合っていないてもかまいません。

対応する組込み関数: `_mm_loadu_pd`

```
void storeu(float *p, F64vec2 A);
```

2個の倍精度浮動小数点値Aをストアします。アライメントは合っていないてもかまいません。

対応する組込み関数: `_mm_storeu_pd`

```
void loadu(F32vec4 A, double *p)
```

4個の単精度浮動小数点値をロードし、それらを4個の浮動小数点値Aにコピーします。アライメントは合っていないてもかまいません。

対応する組込み関数: `_mm_loadu_ps`

```
void storeu(float *p, F32vec4 A);
```

4個の単精度浮動小数点値Aをストアします。アライメントは合っていないてもかまいません。

対応する組込み関数: `_mm_storeu_ps`

## Fvec演算子のアンパック演算子

```
F64vec2 R = unpack_low(F64vec2 A, F64vec2 B);
```

AとBの中から、下位側にある倍精度浮動小数点値を取り出してインターリーブします。

対応する組込み関数: `_mm_unpacklo_pd(a, b)`

```
F64vec2 R = unpack_high(F64vec2 A, F64vec2 B);
```

AとBの中から、上位側にある倍精度浮動小数点値を取り出してインターリーブします。

対応する組込み関数: `_mm_unpackhi_pd(a, b)`

```
F32vec4 R = unpack_low(F32vec4 A, F32vec4 B);
```

AとBの中から、下位側にある2個の単精度浮動小数点値を取り出してインターリーブします。

対応する組込み関数: `_mm_unpacklo_ps(a, b)`

```
F32vec4 R = unpack_high(F32vec4 A, F32vec4 B);
```

AとBの中から、上位側にある2個の単精度浮動小数点値を取り出してインターリーブします。

対応する組込み関数: `_mm_unpackhi_ps(a, b)`

## マスク移動演算子

int i = move\_mask(F64vec2 A)

2個の倍精度浮動小数点値Aの最上位ビットから2ビットマスクを1個作成します。次に例を示します。

i := sign(a1)<<1 | sign(a0)<<0

対応する組込み関数: \_mm\_movemask\_pd

int i = move\_mask(F32vec4 A)

4個の単精度浮動小数点値Aの最上位ビットから4ビットマスクを1個作成します。次に例を示します。

i := sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)<<0

対応する組込み関数: \_mm\_movemask\_ps

## 各種クラスのクイック・リファレンス

この付録には、SIMD演算用のインテル® C++ クラス・ライブラリのクラスごとに、クラス、機能、および対応する組込み関数のそれぞれを列挙した表がいくつか記載されています。次の表は、C++ の各SIMDクラスには実装されていないインテル C++ コンパイラ組込み関数をすべてリストしたものです。

### 論理演算子: 対応する組込み関数とクラス

| 演算子    | 対応する組込み関数     | l128vec1, l64vec2, l32vec4, l16vec8, l8vec16 | l64vec, l32vec, l16vec, l8vec8 | F64vec2 | F32vec4 | F32vec1 |
|--------|---------------|----------------------------------------------|--------------------------------|---------|---------|---------|
| &, &=  | mm_and [x]    | si128                                        | si64                           | pd      | ps      | ps      |
| l,  =  | mm_or [x]     | si128                                        | si64                           | pd      | ps      | ps      |
| ^, ^=  | mm_xor [x]    | si128                                        | si64                           | pd      | ps      | ps      |
| Andnot | mm_andnot [x] | si128                                        | si64                           | pd      | N/A     | N/A     |

### 算術演算子: 対応する組込み関数とクラス

| 演算子   | 対応する組込み関数  | l64vec2 | l32vec4 | l16vec8 | l8vec16 | l32vec2 | l16vec4 | l8vec8 | F64vec2 | F32vec4 | F32vec1 |
|-------|------------|---------|---------|---------|---------|---------|---------|--------|---------|---------|---------|
| +, += | mm_add [x] | epi64   | epi32   | epi16   | epi8    | pi32    | pi16    | pi8    | pd      | ps      | ss      |
| -, -= | mm_sub [x] | epi64   | epi32   | epi16   | epi8    | pi32    | pi16    | pi8    | pd      | ps      | ss      |
| *, *= | mm_mul [x] | N/A     | N/A     | epi16   | N/A     | N/A     | pi16    | N/A    | pd      | ps      | ss      |

|          |                                                                                          |     |     |       |     |     |      |     |     |     |     |
|----------|------------------------------------------------------------------------------------------|-----|-----|-------|-----|-----|------|-----|-----|-----|-----|
|          | ]                                                                                        |     |     |       |     |     |      |     |     |     |     |
| /,/=     | $\text{mm\_div\_x}$                                                                      | N/A | N/A | N/A   | N/A | N/A | N/A  | N/A | pd  | ps  | ss  |
| mul_high | $\text{mm\_mul\_hi\_x}$                                                                  | N/A | N/A | epi16 | N/A | N/A | pi16 | N/A | N/A | N/A | N/A |
| mul_add  | $\text{mm\_mad\_x}$                                                                      | N/A | N/A | epi16 | N/A | N/A | pi16 | N/A | N/A | N/A | N/A |
| sqrt     | $\text{mm\_sqrt\_x}$                                                                     | N/A | N/A | N/A   | N/A | N/A | N/A  | N/A | pd  | ps  | ss  |
| rcp      | $\text{mm\_rcp\_x}$                                                                      | N/A | N/A | N/A   | N/A | N/A | N/A  | N/A | pd  | ps  | ss  |
| rcp_nr   | $\text{mm\_rcp\_x}$<br>$\text{mm\_add\_x}$<br>$\text{mm\_sub\_x}$<br>$\text{mm\_mul\_x}$ | N/A | N/A | N/A   | N/A | N/A | N/A  | N/A | pd  | ps  | ss  |
| rsqrt    | $\text{mm\_rsqrt\_x}$                                                                    | N/A | N/A | N/A   | N/A | N/A | N/A  | N/A | pd  | ps  | ss  |
| rsqrt_nr | $\text{mm\_rsqrt\_x}$<br>$\text{mm\_sub\_x}$<br>$\text{mm\_mul\_x}$                      | N/A | N/A | N/A   | N/A | N/A | N/A  | N/A | pd  | ps  | ss  |

## シフト演算子:対応する組込み関数とクラス

| 演算子     | 対応する組込み関数                                                                                | l128vec1                 | l64vec2                      | l32vec4                          | l16vec8                          | l8vec16                  | l64vec1                    | l32vec2                      | l16vec4                      | l8vec8                   |
|---------|------------------------------------------------------------------------------------------|--------------------------|------------------------------|----------------------------------|----------------------------------|--------------------------|----------------------------|------------------------------|------------------------------|--------------------------|
| >>, >>= | $\text{mm\_srl\_x}$<br>$\text{mm\_srl\_x}$<br>$\text{mm\_sra\_x}$<br>$\text{mm\_sra\_x}$ | N/A<br>N/A<br>N/A<br>N/A | epi64<br>epi64<br>N/A<br>N/A | epi32<br>epi32<br>epi32<br>epi32 | epi16<br>epi16<br>epi16<br>epi16 | N/A<br>N/A<br>N/A<br>N/A | si64<br>si64<br>N/A<br>N/A | pi32<br>pi32<br>pi32<br>pi32 | pi16<br>pi16<br>pi16<br>pi16 | N/A<br>N/A<br>N/A<br>N/A |
| <<, <<= | $\text{mm\_sll\_x}$                                                                      | N/A<br>N/A               | epi64<br>4                   | epi32<br>2                       | epi16<br>6                       | N/A<br>N/A               | si64<br>si64               | pi32<br>pi32                 | pi16<br>pi16                 | N/A<br>N/A               |

|  |                                |  |           |           |           |  |  |  |  |  |
|--|--------------------------------|--|-----------|-----------|-----------|--|--|--|--|--|
|  | <code>_mm_sl<br/>li [x]</code> |  | epi6<br>4 | epi3<br>2 | epi1<br>6 |  |  |  |  |  |
|--|--------------------------------|--|-----------|-----------|-----------|--|--|--|--|--|

## 比較演算子: 対応する組込み関数とクラス

| 演算子                                                                                         | 対応する<br>組込み関数                                                 | i32ve<br>c4            | i16ve<br>c8            | i8vec<br>16       | i32ve<br>c2  | i16ve<br>c4  | i8vec<br>8  | F64ve<br>c2 | F32ve<br>c4 | F32ve<br>c1 |
|---------------------------------------------------------------------------------------------|---------------------------------------------------------------|------------------------|------------------------|-------------------|--------------|--------------|-------------|-------------|-------------|-------------|
| <code>cmpeq</code>                                                                          | <code>_mm_cmp<br/>eq [x]</code>                               | epi3<br>2              | epi1<br>6              | epi8              | pi32         | pi16         | pi8         | pd          | ps          | ss          |
| <code>cmpneq</code>                                                                         | <code>_mm_cmp<br/>eq [x]<br/>_mm_and<br/>not [y]<br/>*</code> | epi3<br>2<br>si12<br>8 | epi1<br>6<br>si12<br>8 | epi8<br>si12<br>8 | pi32<br>si64 | pi16<br>si64 | pi8<br>si64 | pd          | ps          | ss          |
| <code>cmpgt</code>                                                                          | <code>_mm_cmp<br/>gt [x]</code>                               | epi3<br>2              | epi1<br>6              | epi8              | pi32         | pi16         | pi8         | pd          | ps          | ss          |
| <code>cmpge</code>                                                                          | <code>_mm_cmp<br/>ge [x]<br/>_mm_and<br/>not [y]<br/>*</code> | epi3<br>2<br>si12<br>8 | epi1<br>6<br>si12<br>8 | epi8<br>si12<br>8 | pi32<br>si64 | pi16<br>si64 | pi8<br>si64 | pd          | ps          | ss          |
| <code>cmplt</code>                                                                          | <code>_mm_cmp<br/>lt [x]</code>                               | epi3<br>2              | epi1<br>6              | epi8              | pi32         | pi16         | pi8         | pd          | ps          | ss          |
| <code>cmple</code>                                                                          | <code>_mm_cmp<br/>le [x]<br/>_mm_and<br/>not [y]<br/>*</code> | epi3<br>2<br>si12<br>8 | epi1<br>6<br>si12<br>8 | epi8<br>si12<br>8 | pi32<br>si64 | pi16<br>si64 | pi8<br>si64 | pd          | ps          | ss          |
| <code>cmpngt</code>                                                                         | <code>_mm_cmp<br/>ngt [x]</code>                              | epi3<br>2              | epi1<br>6              | epi8              | pi32         | pi16         | pi8         | pd          | ps          | ss          |
| <code>cmpnge</code>                                                                         | <code>_mm_cmp<br/>nge [x]</code>                              | N/A                    | N/A                    | N/A               | N/A          | N/A          | N/A         | pd          | ps          | ss          |
| <code>cmnpnlt</code>                                                                        | <code>_mm_cmp<br/>nlt [x]</code>                              | N/A                    | N/A                    | N/A               | N/A          | N/A          | N/A         | pd          | ps          | ss          |
| <code>cmnpnle</code>                                                                        | <code>_mm_cmp<br/>nle [x]</code>                              | N/A                    | N/A                    | N/A               | N/A          | N/A          | N/A         | pd          | ps          | ss          |
| * 組込み関<br>数<br><code>_mm_andn<br/>ot [y]</code> は<br>fvecクラス<br>には適用され<br>ないので注意<br>してください。 |                                                               |                        |                        |                   |              |              |             |             |             |             |

## 条件付き選択演算子:対応する組込み関数とクラス

| 演算子            | 対応する<br>組込み関<br>数                                                                     | I32vec<br>4                                                   | I16vec<br>8                                                   | I8vec1<br>6                                 | I32vec<br>2                  | I16vec<br>4                  | I8vec8                      | F64ve<br>c2 | F32ve<br>c4 | F32ve<br>c1 |
|----------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------|---------------------------------------------------------------|---------------------------------------------|------------------------------|------------------------------|-----------------------------|-------------|-------------|-------------|
| select<br>_eq  | _mm_cm<br>peq_[x<br>]<br>_mm_an<br>d_[y]<br>_mm_an<br>dnot_<br>[y]*<br>_mm_or<br>_[y] | epi3<br>2<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi1<br>6<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | pi32<br>si64<br>si64<br>si64 | pi16<br>si64<br>si64<br>si64 | pi8<br>si64<br>si64<br>si64 | pd          | ps          | ss          |
| select<br>_neq | _mm_cm<br>peq_[x<br>]<br>_mm_an<br>d_[y]<br>_mm_an<br>dnot_<br>[y]*<br>_mm_or<br>_[y] | epi3<br>2<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi1<br>6<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | pi32<br>si64<br>si64<br>si64 | pi16<br>si64<br>si64<br>si64 | pi8<br>si64<br>si64<br>si64 | pd          | ps          | ss          |
| select<br>_gt  | _mm_cm<br>pgt_[x<br>]<br>_mm_an<br>d_[y]<br>_mm_an<br>dnot_<br>[y]*<br>_mm_or<br>_[y] | epi3<br>2<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi1<br>6<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | pi32<br>si64<br>si64<br>si64 | pi16<br>si64<br>si64<br>si64 | pi8<br>si64<br>si64<br>si64 | pd          | ps          | ss          |
| select<br>_ge  | _mm_cm<br>pge_[x<br>]<br>_mm_an<br>d_[y]<br>_mm_an<br>dnot_<br>[y]*<br>_mm_or<br>_[y] | epi3<br>2<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi1<br>6<br>si12<br>8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | pi32<br>si64<br>si64<br>si64 | pi16<br>si64<br>si64<br>si64 | pi8<br>si64<br>si64<br>si64 | pd          | ps          | ss          |

|                                                                                          |                                                                                        |                                                  |                                                  |                                             |                              |                              |                             |    |    |    |
|------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|--------------------------------------------------|--------------------------------------------------|---------------------------------------------|------------------------------|------------------------------|-----------------------------|----|----|----|
| select<br>_lt                                                                            | _mm_cm<br>plt_[x<br>]<br>_mm_an<br>d_[y]<br>_mm_an<br>dnot_<br>[y] *<br>_mm_or<br>_[y] | epi3<br>2<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi1<br>6<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | pi32<br>si64<br>si64<br>si64 | pi16<br>si64<br>si64<br>si64 | pi8<br>si64<br>si64<br>si64 | pd | ps | ss |
| select<br>_le                                                                            | _mm_cm<br>ple_[x<br>]<br>_mm_an<br>d_[y]<br>_mm_an<br>dnot_<br>[y] *<br>_mm_or<br>_[y] | epi3<br>2<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi1<br>6<br>si12<br>8<br>si12<br>8<br>si12<br>8 | epi8<br>si12<br>8<br>si12<br>8<br>si12<br>8 | pi32<br>si64<br>si64<br>si64 | pi16<br>si64<br>si64<br>si64 | pi8<br>si64<br>si64<br>si64 | pd | ps | ss |
| select<br>_ngt                                                                           | _mm_cm<br>pgt_[x<br>]                                                                  | N/A                                              | N/A                                              | N/A                                         | N/A                          | N/A                          | N/A                         | pd | ps | ss |
| select<br>_nge                                                                           | _mm_cm<br>pge_[x<br>]                                                                  | N/A                                              | N/A                                              | N/A                                         | N/A                          | N/A                          | N/A                         | pd | ps | ss |
| select<br>_nlt                                                                           | _mm_cm<br>plt_[x<br>]                                                                  | N/A                                              | N/A                                              | N/A                                         | N/A                          | N/A                          | N/A                         | pd | ps | ss |
| select<br>_nle                                                                           | _mm_cm<br>ple_[x<br>]                                                                  | N/A                                              | N/A                                              | N/A                                         | N/A                          | N/A                          | N/A                         | pd | ps | ss |
| * 組込み<br>関数<br>_mm_an<br>dnot_<br>[y]は<br>fvecク<br>ラスには<br>適用され<br>ないので<br>注意してく<br>ださい。 |                                                                                        |                                                  |                                                  |                                             |                              |                              |                             |    |    |    |



## パック演算子とアンパック演算子:対応する組込み関数とクラス

| 演算子         | 対応する組込み関数            | I64vec2 | I32vec4 | I16vec8 | I8vec16 | I32vec2 | I16vec4 | I8vec8 | F64vec2 | F32vec4 | F32vec1 |
|-------------|----------------------|---------|---------|---------|---------|---------|---------|--------|---------|---------|---------|
| unpack_high | _mm_unpackhi_ps [x]  | epi64   | epi32   | epi16   | epi8    | pi32    | pi16    | pi8    | pd      | ps      | N/A     |
| unpack_low  | _mm_unpacklo_ps [x]  | epi64   | epi32   | epi16   | epi8    | pi32    | pi16    | pi8    | pd      | ps      | N/A     |
| pack_sat    | _mm_packss_epi32 [x] | N/A     | epi32   | epi16   | N/A     | pi32    | pi16    | N/A    | N/A     | N/A     | N/A     |
| pack_u_sat  | _mm_packus_epi16 [x] | N/A     | N/A     | epi16   | N/A     | N/A     | pu16    | N/A    | N/A     | N/A     | N/A     |
| sat_add     | _mm_adds_epi64 [x]   | N/A     | N/A     | epi16   | epi8    | N/A     | pi16    | pi8    | pd      | ps      | ss      |
| sat_sub     | _mm_subs_epi64 [x]   | N/A     | N/A     | epi16   | epi8    | N/A     | pi16    | pi8    | pi16    | pi8     | pd      |

## 変換演算子:対応する組込み関数とクラス

| 演算子              | 対応する組込み関数      |
|------------------|----------------|
| F64vec2ToInt     | mm_cvttsd_si32 |
| F32vec4ToF64vec2 | mm_cvtps_pd    |
| F64vec2ToF32vec4 | mm_cvtpd_ps    |
| IntToF64vec2     | mm_cvtsi32_sd  |
| F32vec4ToInt     | mm_cvtt_ss2si  |
| F32vec4ToI32vec2 | mm_cvttps_pi32 |
| IntToF32vec4     | mm_cvtsi32_ss  |
| I32vec2ToF32vec4 | mm_cvtpi32_ps  |



## プログラミング例

次のサンプル・プログラムは、F32vec4クラスを使用して20個の要素から成る浮動小数点配列のすべての要素の平均値を計算するものです。このコードは、AvgClass.cppのファイルにもサンプル・プログラムとして含まれています。

```
// Include Streaming SIMD Extension Class

Definitions
#include <fvec.h>

// Shuffle any 2 single precision floating point from a
// into low 2 SP FP and shuffle any 2 SP FP from b
// into high 2 SP FP of destination

#define SHUFFLE(a,b,i) (F32vec4)_mm_shuffle_ps(a,b,i)
#include <stdio.h>
#define SIZE 20

// Global variables
float result;
_MM_ALIGN 16 float array[SIZE];

//*****
// Function: Add20ArrayElements
// Add all the elements of a 20 element array
//*****

void Add20ArrayElements (F32vec4 *array, float *result)
{
    F32vec4 vec0, vec1;
    vec0 = _mm_load_ps ((float *) array);

    // Load array's first 4 floats

    //*****
    // Add all elements of the array, 4 elements at a time
    //*****
    *

    vec0 += array[1]; // Add elements 5-8
```

```

vec0 += array[2]; // Add elements 9-12
vec0 += array[3]; // Add elements 13-16
vec0 += array[4]; // Add elements 17-20

//*****
// There are now 4 partial sums. Add the 2 lowers to the
// 2 raises,
// then add those 2 results together
//*****

vec1 = SHUFFLE(vec1, vec0, 0x40);
vec0 += vec1;
vec1 = SHUFFLE(vec1, vec0, 0x30);
vec0 += vec1;
vec0 = SHUFFLE(vec0, vec0, 2);
_mm_store_ss (result, vec0); // Store the final sum
}

void main(int argc, char *argv[])
{
int i;

// Initialize the array
for (i=0; i < SIZE; i++)
{
array[i] = (float) i;
}

// Call function to add all array elements
Add20ArrayElements(array, &result);

// Print average array element value
printf ("Average of all array values = %f¥n", result/20.);
printf ("The correct answer is %f¥n¥n¥n", 9.5);
}

```

## 浮動小数点演算の精度の制限

-mpオプションは、宣言された精度を維持しながら、浮動小数点演算を ANSI および IEEE 標準にできるだけ準拠するために、最適化を制限します。

このオプションは、ほとんどのプログラムのパフォーマンスに不利に働きます。目的のアプリケーションにこのオプションが必要かどうかよくわからない場合は、このオプションを指定した場合と指定しない場合で実際にプログラムをコンパイルし、実行して、パフォーマンスと精度に対する効果を評価してみてください。

このオプションを指定すると、プログラムのコンパイルに次のような影響が出ます。

- 浮動小数点型として宣言されたユーザ変数はレジスタに割り当てられません。
- 浮動小数点演算の比較は、NaN (非数)の動作以外はIEEE 754に準拠します。
- 演算は、コード内で指定したとおりに実行します。例えば、除算が「逆数の乗算」に変更されることはありません。
- コンパイラは関連付けを変更せずに、指定した順序で浮動小数点演算を実行します。
- コンパイラは浮動小数点値に関する定数の畳込みによる最適化を行いません。定数の畳込みとは、1を掛けたり、1で割ったり、0の足し引きをしたりといった計算を省くことです。定数の畳込みを実行しないので、例えば、0.0の足し算についても記述したとおりに実行します。また、コンパイル時の浮動小数点演算も実行しません。これは、浮動小数点例外についてもその状態を変えないようにするためです。
- IA-32 システムでは、スピルされる式は、64ビット(倍精度)ではなく、必ず80ビット(拡張精度)としてスピルされます。浮動小数点演算は、IEEE 754 に準拠します。-O0 を指定しない場合は、精度の追加ビットは必ずしも変数が再利用する前に丸められるわけではありません。
- -xK、-xW、-axK、-axWのうちいずれかのオプションによりベクトル化を有効にしても、リダクション・ループ(ドット積を計算するループ)と、精度の種類がいくつか混在しているループは、ベクトル化されません。

## インクルード・ファイルの検索

デフォルトでは、コンパイラは、INCLUDE 環境変数に指定したディレクトリの中で、標準のインクルード・ファイルを検索します。インクルード・ファイルの格納場所は設定ファイルの中に指定できます。

## インクルード・ディレクトリの指定方法

インクルード・ファイルの検索先ディレクトリを1個追加指定するには、-Idirectory オプションを使用します。検索先ディレクトリを複数指定するときは、-Idirectory コマンドを複数指定する必要があります。インクルードしたファイルは、#include プリプロセッサ・ディレクティブでプログラムに取り込まれます。コンパイラはインクルード・ファイルを検索するとき、次の順番で各ディレクトリを見ます。

- include が含まれるソースファイル・ディレクトリ
- -I オプションで指定されるディレクトリ
- INCLUDE 環境変数に指定されるディレクトリ

## インクルード・ディレクトリの削除方法

INCLUDE 環境変数に指定しているデフォルトのパスの検索をコンパイラに禁止するには、  
-X オプションを使用します。

-X オプションと -I オプションを併用すると、インクルード・ファイルを検索する際のデフォルトのパスではなく別のパスを検索するようにコンパイラに命令できます。

例えば、デフォルトのパスの代わりに/alt/includeのパスを検索するようにコンパイラに命令するときは次のようにします。

- IA-32 システム: prompt>icc -X -I/alt/include newmain.c
- Itanium® ベース・システム: prompt>ecc -X -I/alt/include newmain.c

## ハイパー・スレッディング・テクノロジー

インテル® プロセッサは、1 クロックサイクルで複数の命令を実行できるスーパースカラーです。インテルのハイパー・スレッディング・テクノロジーは、1つの物理プロセッサを、2 つのスレッドを並列に実行できる 2 つの論理プロセッサに見せることによって、この機能を拡張します。ソフトウェア側からは、オペレーティング・システムおよびプログラムが、2 つの物理プロセッサ上でプロセスやスレッドが実行しているかのように、プロセスまたはスレッドをスケジューリングできます。マイクロアーキテクチャ側では、2 つの論理プロセッサからの命令が、1 つの物理プロセッサの共有リソースで同時に実行します。このようにして、全体のリソース稼働率を高めます。

操作環境は、2つのうちのいずれかの方法で、ハイパー・スレッディング・テクノロジーのハイパー・スレッディング・テクノロジーの利点を最大限に利用できます。

1. 各アプリケーションが同時にプロセッサ上で並列スレッドを実行し、アプリケーションのマルチスレッド化を実現します。データベース・エンジン、科学計算プログラムおよびマルチメディア・デザイン・ソフトウェアなどの高パフォーマンス・アプリケーションと並んで、Windows .Net\*およびXP\*のようなオペレーティング・システムは、現在マルチスレッド対応で、通常、デュアル/マルチ・プロセッサ環境で起動します。
2. システムが複数のアプリケーションを並列に実行するマルチタスク環境で、ハイパー・スレッディング・テクノロジーを使用できます。この場合、各アプリケーションは、同一のプロセッサ上で別々のスレッドとして実行することができ、実行単位の効率と全体のプラットフォームのパフォーマンスが向上されます。



# 索引

## C

C++ クラスとSIMD演算 .....361

## E

EMMSを使用する際のガイドライン .....222

    EMMS 命令 .....221

## F

FLEXlm\* 電子ライセンス .....23

Fvec クラスの条件付き選択演算子 .....407

Fvec 演算子のアンパック演算子 .....413

Fvecの表記法 .....394

## G

GNU\* インライン・アセンブリ .....332

## I

Itanium(R) 命令のネイティブ組込み関数 .....309

## M

MMX(R) テクノロジーの組込み関数 .....339

    Itanium(R) アーキテクチャの組込み関数 .....233

    MMX(R) ステート消去演算子 .....390

    シフト組込み関数 .....227

    パックド算術組込み関数 .....225

    一般的な組込み関数 .....222

    設定組込み関数 .....231

    比較組込み関数 .....230

    論理組込み関数 .....229

## P

PGO API サポートの概要 .....116

    インターバル・プロファイル・ダンプ .....117

    プロファイル情報のダンプ .....116

    プロファイル情報のダンプとリセット .....117

    環境変数 .....118

## S

SIMD ライブラリ .....361

SIMD 演算用のクラス・ライブラリ .....359

## あ

アライメントの合ったメモリブロックの割り当てと解放 .....331

アンパック演算子 .....384

## い

インクルード・ファイル .....68

    検索 .....422

|                               |          |
|-------------------------------|----------|
| <b>お</b>                      |          |
| オプション .....                   | 28       |
| オプションのクイック・リファレンス .....       | 32       |
| オプションの対応表 .....               | 55       |
| <b>く</b>                      |          |
| クラス・ライブラリ - 算術演算子 .....       | 397      |
| <b>こ</b>                      |          |
| コンストラクタと初期化 .....             | 396      |
| コンパイラの制限 .....                | 206      |
| コンパイル .....                   | 78       |
| インクルード・ファイル .....             | 68       |
| <b>検索</b> .....               | 422      |
| オプションのクイック・リファレンス・ガイド .....   | 32       |
| オプションの対応表 .....               | 55       |
| コマンドラインから .....               | 60       |
| コマンドラインからの make .....         | 61       |
| フェーズ .....                    | 64       |
| フローの制御 .....                  | 47       |
| 応答ファイル .....                  | 68       |
| 環境のカスタマイズ .....               | 66       |
| 出力の制御 .....                   | 48       |
| 設定ファイル .....                  | 67       |
| 代替ツールと代替パスの指定 .....           | 70       |
| 入力ファイル .....                  | 61       |
| <b>さ</b>                      |          |
| サポート .....                    | 22       |
| <b>し</b>                      |          |
| システムの要件 .....                 | 22       |
| シフト演算子 .....                  | 377      |
| シンボリック・デバッグ .....             | 85       |
| <b>す</b>                      |          |
| ストリーミングSIMD拡張命令 .....         | 235      |
| ストリーミングSIMD拡張命令2 .....        | 267      |
| Itanium(R) アーキテクチャの拡張命令 ..... | 262      |
| キャッシュ制御 .....                 | 252      |
| シャッフルを行うマクロ関数 .....           | 265      |
| ストア操作 .....                   | 251, 308 |
| ストリーミングSIMD拡張命令2 の設定操作 .....  | 305      |
| その他のオプション .....               | 283      |
| その他の組込み関数 .....               | 260      |
| ロード演算 .....                   | 305      |

|                                                     |          |
|-----------------------------------------------------|----------|
| ロード操作 .....                                         | 249      |
| 算術演算 .....                                          | 236      |
| 使用 .....                                            | 343, 348 |
| 整数メモリと初期化 .....                                     | 305      |
| 整数演算組込み関数 .....                                     | 253      |
| 整数論理演算 .....                                        | 290      |
| 設定操作 .....                                          | 250      |
| 代入演算子 .....                                         | 371      |
| 比較操作 .....                                          | 240      |
| 浮動小数点の組込み関数 .....                                   | 235      |
| 変換操作 .....                                          | 246, 297 |
| 論理演算 .....                                          | 239, 271 |
| <b>て</b>                                            |          |
| データのアライメント .....                                    | 395      |
| データ設定の監視 .....                                      | 79       |
| デバッグ .....                                          | 84       |
| オプション .....                                         | 48       |
| シンボリック・デバッグ .....                                   | 85       |
| デフォルト .....                                         | 63       |
| インライン展開 .....                                       | 332      |
| オプション .....                                         | 62       |
| ライブラリ .....                                         | 171      |
| <b>は</b>                                            |          |
| ハードウェアとソフトウェアの要件 .....                              | 359      |
| バック演算子 .....                                        | 389      |
| <b>ふ</b>                                            |          |
| プロシージャ間の最適化(IPO) .....                              | 102      |
| IPOオブジェクトからライブラリを作成する .....                         | 106      |
| インライン展開の条件 .....                                    | 109      |
| インライン展開の制御 .....                                    | 109      |
| マルチファイル .....                                       | 103      |
| Project Makefile を使用してマルチファイル IPO 実行ファイルを作成する ..... | 105      |
| マルチファイル IPO の実行ファイルを作成する .....                      | 104      |
| 効果を分析する .....                                       | 107      |
| ライブラリ関数のインライン展開 .....                               | 332      |
| 基本的なオプション .....                                     | 52       |
| 実際のオブジェクト・ファイル .....                                | 103      |
| プロセッサ・ディスパッチ .....                                  | 51       |
| -ax[i M K W] の専用コード .....                           | 98       |
| プロセッサと拡張機能サポートの対象 .....                             | 94       |
| 対象となるプロセッサ (IA-32 のみ) .....                         | 96       |



|                               |     |
|-------------------------------|-----|
| 対象となるプロセッサとディスパッチ・オプション ..... | 100 |
| 排他的な専用コード (IA-32 のみ) .....    | 97  |
| プロファイルに基づく最適化 (PGO) .....     | 110 |
| オプション .....                   | 52  |
| 環境変数 .....                    | 114 |
| 方法 .....                      | 110 |
| 例 .....                       | 112 |
| <b>へ</b>                      |     |
| ベクトル化 .....                   | 147 |
| データ依存性 .....                  | 150 |
| ループ .....                     | 151 |
| ストリップ・マイニングとクリーンアップ .....     | 153 |
| ループ本体内の文 .....                | 154 |
| 出口条件 .....                    | 152 |
| 主要プログラミング・ガイドライン .....        | 148 |
| 例 .....                       | 157 |
| <b>ま</b>                      |     |
| マクロ関数 .....                   | 267 |
| コントロール・レジスタを読み書きする .....      | 265 |
| シャッフル .....                   | 298 |
| ストリーミングSIMD拡張命令によるシャッフル ..... | 265 |
| マスク移動演算子 .....                | 413 |
| マトリックス乗算 .....                | 159 |
| <b>ら</b>                      |     |
| ライブラリ .....                   | 171 |
| 管理 .....                      | 173 |
| ライブラリ関数のインライン展開 .....         | 332 |
| <b>り</b>                      |     |
| リマーク・メッセージ .....              | 202 |
| リンク .....                     | 82  |
| <b>る</b>                      |     |
| ループのアンロール .....               | 120 |
| ループ変換 .....                   | 119 |
| <b>ろ</b>                      |     |
| ロードとストア演算子 .....              | 412 |
| ロックおよびアトミック操作に関連する組込み関数 ..... | 312 |
| <b>漢字</b>                     |     |
| 演算子の規則 .....                  | 369 |
| 応答ファイル .....                  | 68  |
| 加算と減算演算子 .....                | 374 |
| 環境変数 .....                    | 66  |

|                                |     |
|--------------------------------|-----|
| 構文解析のみを行う .....                | 85  |
| 高レベル最適化(HLO) .....             | 119 |
| オプション .....                    | 53  |
| 最小値と最大値の演算子 .....              | 401 |
| 最適化項目 .....                    | 88  |
| オプション .....                    | 51  |
| レポート .....                     | 53  |
| 制限 .....                       | 90  |
| 主要ファイル .....                   | 208 |
| Itanium(R) コンパイラの主要なファイル ..... | 208 |
| 乗算演算子 .....                    | 375 |
| 条件付き選択演算子 .....                | 379 |
| 新機能 .....                      | 21  |
| オプション .....                    | 28  |
| 診断およびメッセージ                     |     |
| lintの停止 .....                  | 201 |
| 言語診断 .....                     | 200 |
| 制限 .....                       | 202 |
| 停止 .....                       | 201 |
| 設定ファイル .....                   | 67  |
| 前処理 .....                      | 72  |
| オプション .....                    | 47  |
| のみ .....                       | 73  |
| マクロ .....                      | 74  |
| 組込み関数 .....                    | 211 |
| アライメントのサポート .....              | 330 |
| オペレーティング・システムに関連 .....         | 316 |
| すべての IA の関数 .....              | 335 |
| その他 .....                      | 219 |
| 命名と使用する構文 .....                | 214 |
| 代替ツールと代替パス .....               | 47  |
| 代替ツールと代替パスの指定 .....            | 70  |
| 適合性オプション .....                 | 50  |
| C の標準規格 .....                  | 86  |
| C++ の標準規格 .....                | 87  |
| 特徴と利点 .....                    | 21  |
| 入力ファイル .....                   | 61  |
| 比較演算子 .....                    | 404 |
| 表記法 .....                      | 24  |
| 浮動小数点の組込み関数 .....              | 235 |
| 浮動小数点ベクトルクラス .....             | 394 |

|                                          |     |
|------------------------------------------|-----|
| 浮動小数点演算の精度.....                          | 422 |
| 文献.....                                  | 26  |
| 文字列とブロックのコピー.....                        | 219 |
| 並列化.....                                 | 122 |
| 自動並列化.....                               | 142 |
| しきい値.....                                | 144 |
| 診断.....                                  | 143 |
| 変換.....                                  | 395 |
| 変換(Fvec $\longleftrightarrow$ Ivec)..... | 392 |
| 免責条項.....                                | 20  |
| 用語、規則、および構文.....                         | 368 |
| 論理演算子.....                               | 372 |