15 章
Silvermont† マイクロアーキテクチャーとソフトウェアの最適化

15.1 概要

本章では、Silvermont† マイクロアーキテクチャーの概要と、Silvermont マイクロアーキテクチャー・ベースの次世代インテル® Atom™ プロセッサー向けソフトウェアに固有のコーディング手法について説明する。本章で紹介するソフトウェアの最適化に関する推奨事項は Silvermont マイクロアーキテクチャーに固有のものであり、汎用の x86 コーディング・スタイルに加えてこれらを考慮すべきである。

15.1.1 次世代インテル® Atom™ プロセッサー・ファミリー

次世代インテル® Atom™ プロセッサー・ファミリーは Silvermont マイクロアーキテクチャー・ベースであり、インテルの 22nm プロセス・テクノロジーを採用する幅広いコンピューター・デバイスで利用できる。インテル® 64 アーキテクチャーと IA-32 アーキテクチャーのサポートに加えて、Silvermont マイクロアーキテクチャーでは主に次の点が拡張されている。

- 整数命令のアウトオブオーダー実行、および非整数命令とメモリー命令間の実行順序の切り離し。対照的に、前世代のインテル® Atom™ マイクロアーキテクチャー（第 14 章を参照）ではインオーダー実行が厳守され、命令レベルの並列性が制限されていた。
- 非ブロッキング命令における複数の未処理ミスの許容（8 回まで）。前世代のプロセッサーでは、1 つのメモリー命令 で問題が発生すると（例えば、キャッシュミスなど）、その問題が解決されるまで後続のすべての命令がストールしたが、新しいマイクロアーキテクチャーでは最大 8 つの未処理参照が許容される。
- 2 コアのモジュラーシステム設計。フロントサイド・バスの代わりにポイントツーポイントのインターフェイスを使って、新しい内蔵メモリー・コントローラーに接続された L2 キャッシュを共有する。
- 命令セットの拡張。インテル® SSE4.1、SSE4.2、AESNI、PCLMULQDQ が追加されており、32nm プロセス・テクノロジーを採用した第 1 世代インテル® Core™ プロセッサーと互換性がある。

15.2 Silvermont マイクロアーキテクチャー

図 15-1 に Silvermont マイクロアーキテクチャーのブロック図を示す。シングルスレッドのパフォーマンスを向上するため、メモリークラスターと実行クラスターの設計が大幅に見直されている一方、これまでと同様に、小さなフォームファクターで低消費電力を実現する取り組みが行われている。各パイプラインには、リザベーション・ステーションと呼ばれる専用のスケジューリング・キューがある。浮動小数点命令とメモリー命令はそれぞれのキューからプログラム順にスケジュールされ、整数命令はそれぞれのキューからアウトオブオーダーでスケジュールされる。

これは、整数命令がインオーダー実行であった前世代とは対照的である。アウトオブオーダー・スケジューリングにより、これらの命令ではソースやリソースが利用できない場合に発生するストールを許容することができる。メモリー命令は、アドレスの生成 (AGEN) をインオーダーで行い、スケジューリング・キューからインオーダーでスケジュールしなければならないが、実行はアウトオブオーダーで行うことができる。

(SIMD 整数、SIMD 浮動小数点、x87 浮動小数点を含む) 非整数命令も、それぞれのスケジューリング・キューからプログラム順にスケジュールされるが、これらは個別のスケジューリング・キューなので、ほかのスケジューリング・キューにある命令とは切り離して実行することができる。

† 開発コード名

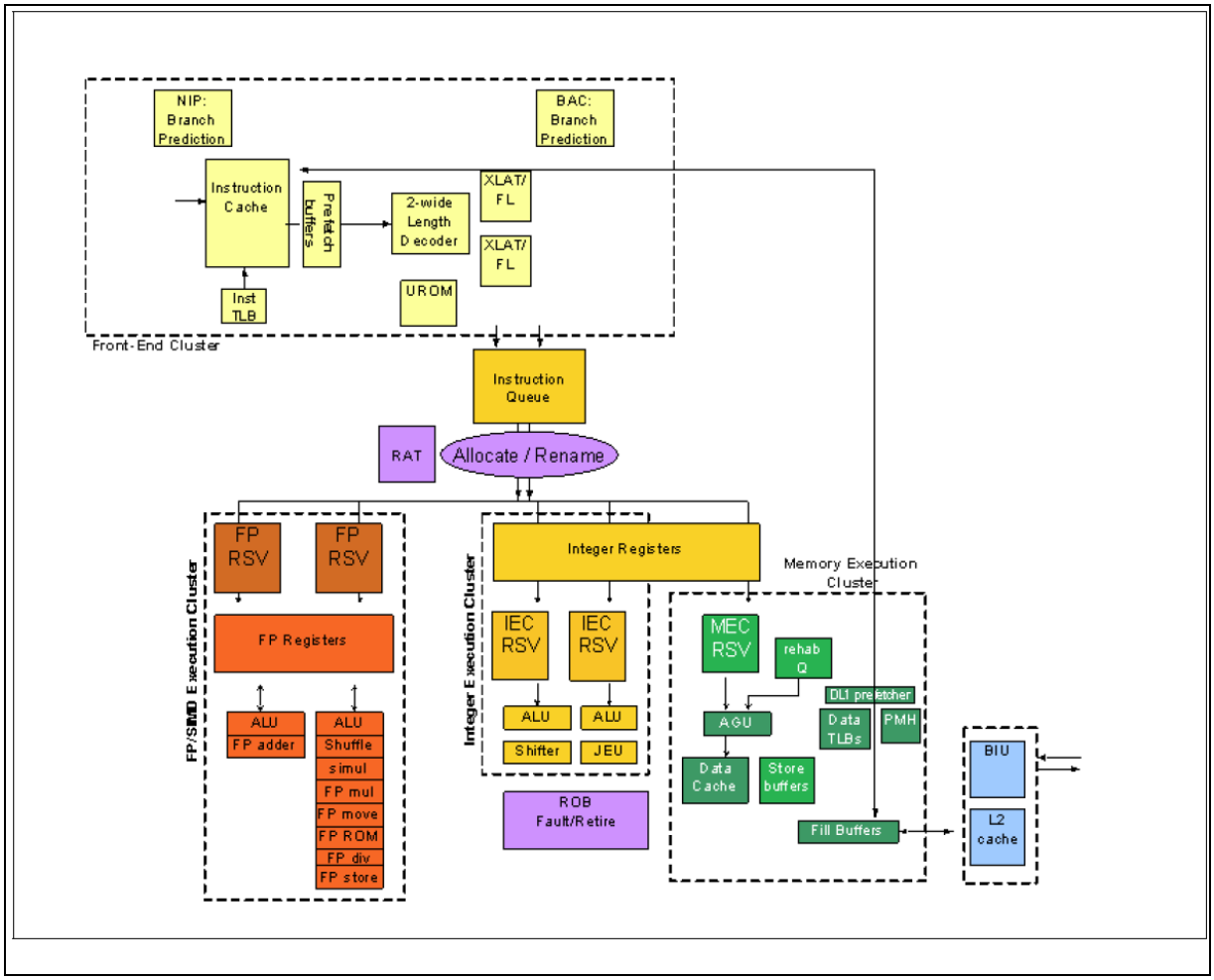

図 15-1 Silvermont マイクロアーキテクチャーのパイプライン

Silvermont マイクロアーキテクチャーは、アウトオブオーダー・スケジューリングにより、多様なフォームファクターの（例えば、携帯電話、タブレットからマイクロサーバーにわたる）プラットフォーム・パフォーマンスを最大限に引き出し、消費電力と面積コストを最小限に抑えるように設計されている（つまり、パフォーマンス/電力/コスト効率を最大化している）。共有 L2 キャッシュを装備したマルチコア・アーキテクチャーを採用しているため、インテル® ハイパースレッディング・テクノロジーはサポートされていない。クラスターレベルの機能については、この節の後半で説明する。

図 15-1 に薄黄色で示されているフロント・エンド・クラスター（FEC）は、同時に 2 命令を処理できるデコード・パイプラインであり、消費電力が最適化されている。FEC はメモリーから命令をフェッチしデコードする。このとき、命令キャッシュからのプリデコード情報を利用することで、コストのかかる命令長の検出をデコード時に行わないようにしている。フロントエンドには分岐ターゲットバッファー（BTB）と高度な分岐予測ハードウェアがある。

フロントエンドは、図 15-1 に薄紫色で示されているアロケート、リネーム、リタイア（ARR）クラスターを介して実行ユニットに接続されている。ARR は、FEC からマイクロオペレーション（uOP）を受け取り、リソースチェックを行う。レジスター・エイリアス・テーブル（RAT）は、論理レジスターから物理レジスターへのリネームを行う。リオーダーバッファー（ROB）は、プログラム順に操作を並べ替えて実行（リタイア）し、割り込み、例外、アシスト時には実行を停止して、マイクロコードに対するプログラム制御を実行する。

Silvermont マイクロアーキテクチャーは分散スケジューリングを採用しているため、リネーム処理後にマイクロオペレーション（uOP）はさまざまなクラスター（IEC: 整数実行クラスター、MEC: メモリー実行クラスター、FPC: 浮動小数点クラスター）に送られ、スケジューリングされる（図 15-1 では FP RSV、IEC RSV、MEC RSV として示されている）。

FPC RSV と IEC RSV は 2 セット（各ポートに 1 つずつ）あり、MEC RSV は 1 セットある。各 RSV は、ARR クラスターからサイクルごとに最大 2 マイクロオペレーション（uOP）を受け取り、実行準備が整ったものから実行ユニットへディスパッチする。

分散型リザベーション・ステーションの概念をサポートするため、整数実行を要求する load-op（ロード-実行）型や load-op-store（ロード - 実行 - ストア）型のマクロ命令は、MEC RSV に送られるメモリー操作と、IEC RSV に送られる整数実行操作に分ける必要がある。IEC スケジューラーは、各 IEC RSV から実行準備が整っている最も古い命令を選択する。一方、MEC スケジューラーと FPC スケジューラーは、それぞれの RSV から最も古い命令を選択する。MEC クラスターと FPC クラスターはインオーダー・スケジューラーを採用しているが、FPC RSV の新しい命令は、別の FPC RSV や IEC RSV、MEC RSV にあるより古い命令よりも前に実行できる。

表 15-1 に示すとおり、各実行ポートには固有の機能ユニットがあり、図 15-1 では IEC がオレンジ、MEC が緑、FPC が赤で示されている。前世代のインテル® Atom™ マイクロアーキテクチャーと比べると、Silvermont マイクロアーキテクチャーでは IEC に整数乗算ユニット（IMUL）が追加されている。

表 15-1 Silvermont マイクロアーキテクチャーの機能ユニットの割り当て

	ポート 0	ポート 1
IEC	ALU0、シフト/ローテートユニット、LEA（インデックスなし）	ALU1、ビット処理ユニット、ジャンプユニット、IMUL、POPCNT、CRC32、LEA[1]
FPC	SIMD ALU、SIMD シフト/シャッフルユニット、SIMD FP 乗算/除算/変換ユニット、STTNI/AESNI/PCLMULQDQ ユニット、RCP/RSQRT ユニット、F2I 変換ユニット	SIMD ALU、SIMD FP 加算器、F2I 変換ユニット
MEC	ロード/ストア	

メモリー実行クラスター（MEC）（図 15-1 に緑色で示されている）は、32 ビットと 36 ビットの物理アドレスモードをサポートする。Silvermont マイクロアーキテクチャーは、2 つのレベルからなるデータ TLB を実装しており、スモールページとラージページ（2MB または 4MB）をサポートしている。第 1 レベルのマイクロ TLB（uTLB）は小さく、より大きな第 2 レベルの TLB（DTLB）にバックアップされる。命令 TLB ミスとデータ TLB ミスはどちらもハードウェア・ページ・ウォーカーによって処理される。

MEC には、すべてのロードとストアのスケジューリングを行う MEC RSV もある。ロード命令とストア命令は、後のパイプラインでメモリーを並べ替えなくても済むように、プログラム順にアドレス生成処理が行われる。そのため、不明なアドレスによって新しいメモリー命令がストールする。（uTLB ミスやリソースが利用できないなどの）問題が発生したメモリー操作は RehabQ（修復キュー）と呼ばれる別のキューに配置されるため、後続の命令をすべてストールする代わりに、（問題が発生していない）より新しい命令の実行を継続できる。問題が発生した命令は、問題が解決した後に RehabQ から再発行される。Silvermont マイクロアーキテクチャーでは、データ・キャッシュ・ミスは 8 回までブロックされないため、ロードミスはそれほど問題と見なされない。

バスクラスター（BIU）の L2 キャッシュは、プロセッサー・コア外部とのすべての通信を処理する。この L2 キャッシュは最大 1MB で、前世代のインテル® Atom™ マイクロアーキテクチャーと比べるとレイテンシーが向上している。前世代のインテル® Atom™ プロセッサーのフロントサイド・バスに代わり、最適化された新しいメモリー・コントローラーに接続するイントラダイ・インターコネクト（IDI）ファブリックが採用されている。BIU には L2 データ・プリフェッチャーも装備される。

新しいコアレベルのマルチプロセシング（CMP）システム構成では、2 つのプロセッサー・コアが 1 つの BIU に要求を送り、コア間の多重化は BIU によって処理される。この基本 CMP モジュールを複製してクアッドコア構成を作成したり、1 コアのみにしてシングルコア構成を作成できる。

15.2.1 整数パイプライン

ロードのパイプライン・ステージがほかの整数パイプラインとインライン化されなくなったため、ロードを伴わない操作の実行を高速化し、分岐予測のペナルティーが前世代のインテル® Atom™ プロセッサーよりも 3 サイクル少なくなっている。フロントエンドのパイプライン・ステージは前世代のインテル® Atom™ プロセッサーと同じで、フェッチに 3 サイクル、デコードに 3 サイクルかかる。ARR パイプステージは、アウトオブオーダー・アロケーションとレジスターのリネームを行い、必要に応じてマイクロオペレーション（uOP）を分割し、各リザベーション・ステーションへ送る。RSV ステージでは、各リザベーション・ステーションがそれぞれ

[1]有効なインデックスとディスプレースメントを持つ LEA は複数のマイクロオペレーション（uOP）に分けられ、両方のポートを使用する。有効なインデックスを持つ LEA はポート 1 で実行される。

のスケジューリングを行う。実行パイプラインは前世代のインテル® Atom™ プロセッサーによく似ている。マイクロオペレーション (uOP) のすべての部分の操作が完了すると、ROB がインオーダーで最終処理を行う。

15.2.2 浮動小数点パイプライン

INT パイプラインよりも FP パイプラインのほうが長く、命令に応じて 1 ～ 5 の実行ステージがある。ほかのインテル® マイクロアーキテクチャーと同様に、Silvermont マイクロアーキテクチャーでもハイパフォーマンスを達成するため、FP アシスト (特定の浮動小数点操作を実行パイプラインでネイティブに処理できず、マイクロコードで実行しなければならない場合) の数を最小限に抑える必要がある。そのため、可能な場合は例外をマスクし、DAZ (デノーマルをゼロとして扱う) フラグと FTZ (ゼロフラッシュ) フラグを設定して実行する。

前述のように、各 FPC RSV において命令はインオーダーでスケジュールされるが、RSV 間でアウトオブオーダーになってもかまわない。

15.2 Silvermont マイクロアーキテクチャーにおけるコーディングの推奨事項

15.3.1 フロントエンドの最適化

15.3.1.1 命令デコーダー

一部の命令は、1 マイクロオペレーション (uOP) にデコードするには複雑すぎるため、複数のマイクロオペレーション (uOP) にデコードされ、完了したマイクロオペレーション (uOP) を特定するためにマイクロコード・シーケンサー ROM (MSROM) のルックアップが必要になる。MSROM ルックアップが必要な命令については、付録 A のレイテンシー/スループットの表を参照のこと。

Silvermont マイクロアーキテクチャーでは、MSROM を必要とする命令の数は非常に少なくなっている。パックド倍精度 SIMD 命令やアライメントされていないロード/ストアなどの主要な命令はマイクロコード化されない。

マイクロコード・フローは、できるだけ回避することが推奨される。

チューニングの推奨事項 1: perfmon カウンター MS_DECODED.MS_ENTRY で MSROM が必要な命令の数が分かる (すべてのアシストとフォルトが含まれる)。

アセンブリー/コンパイラー・コーディング規則 1 (影響 M、一般性 M): 命令長をできるだけ短くすることで、プリデコード・ビットを効率良く再利用できる。

チューニングの推奨事項 2: perfmon カウンター DECODE_RESTRICTION.PREDECODE_WRONG で、プリデコード・ビットが正しくないためデコードの制限によって命令デコードのスループットが低下した回数が分かる。

15.3.1.2 フロントエンドの IPC が高い場合の考慮事項

一般に、サイクルあたりの命令数 (IPC) が高く (>1 に) なるまで、フロントエンドがパフォーマンスを制限することはない。

デコーダーでサイクルあたり 2 命令を処理するには、次のデコードの制限に従う必要がある。

- MSROM 命令はできるだけ回避する。例えば、メモリーバージョンの PUSH と CALL の代わりに、レジスターへロードし、レジスターバージョンの PUSH と CALL を実行する。
- 一緒にデコードされる命令ペアの長さの合計は 16 バイト未満に、最初の命令の長さは 8 バイト以下にする。Silvermont マイクロアーキテクチャーでは、命令が 8 バイトを超えるとサイクルあたり 1 命令しかデコードできない。
- 命令のプリフィクス + エスケープは 3 バイトを超えないようにする。3 バイトを超えると 3 サイクルのペナルティーが発生する。
- Silvermont マイクロアーキテクチャーは、同じサイクルで 2 つの分岐をデコードできない。例えば、分岐しない条件分岐がデコーダー 0 にあり、条件なしのジャンプがデコーダー 1 にある場合、条件なしのジャンプで 3 サイクルのペナルティーが発生する。

前世代と異なり、Silvermont マイクロアーキテクチャーでは、同じサイクルで 2 つの x87 命令をデコードしても 2 サイクルのペナルティーは発生しない。分岐デコーダーの制限も緩和されている。前世代のインテル® Atom™ プロセッサーでは、デコーダー 0 で条件分岐または間接分岐の次の命令のデコードに 2 サイクルのペナルティーが発生した。Silvermont マイクロアー

キテクチャーでは、デコーダー 0 で条件分岐または間接分岐の次の命令をペナルティーなしでデコードできる。ただし、(デコーダー 1 にある) 次の命令が分岐の場合は、その分岐命令で 3 サイクルのペナルティーが発生する。

アセンブリー/コンパイラー・コーディング規則 2 (影響 MH、一般性 H): サイクルあたり 2 命令のスループットを達成するため、次の命令の使用はできるだけ控える: (i) MSROM を使用する命令、(ii) プリフィクス+エスケープが 3 バイトを超える命令、(iii) 長さが 8 バイトを超える命令、(iv) 連続する分岐命令。

プリフィクスが多すぎてスループットが低下する例として、次のように PCLMULQDQ 命令で REX プリフィクスを使用する場合が挙げられる。

```
PCLMULQDQ 66 0F 3A 44 C7 01    pclmulqdq xmm0, xmm7, 0x1
```

オペコードバイト 44 の前にある 66 と 0F 3A は必要なプリフィクスである。XMM レジスターのいずれか (XMM8-15) が参照される場合は、次のようにREX プリフィクスも必要になる。

```
PCLMULQDQ 66 41 0F 3A 44 C0 01 pclmulqdq xmm0, xmm8, 0x1
```

66 と 0F 3A の間に 41 が追加されていることが分かる。

この 4 つ目のプリフィクスにより、デコードで 3 サイクルの遅延が生じる。さらに、このプリフィクスは命令をデコーダー 0 でデコードすることを強制するため、命令がデコーダー 1 で開始された場合、デコーダー 0 へ切り替えるのに 3 サイクルかかり、ペナルティーはさらに大きくなる (デコーダーのペナルティーは合計 6 サイクルになる)。そのため、ハイパフォーマンスなアセンブリーを記述するには、これらを考慮したほうが良い。これらのケースは頻繁に発生しなければ、成立分岐ターゲットや MS エントリーポイントによってあらかじめデコーダー 0 へアライメントしたほうが良い。NOP 命令は、パイプラインのほかのリソースを消費するため、NOP の挿入は最終手段として行うべきである。MS エントリーポイントも、デコーダー 1 で開始した場合 3 サイクルのペナルティーが発生するため、同様のアライメントが必要である。

フロントエンドにおけるもう 1 つの重要なパフォーマンスの考慮事項は分岐予測である。64 ビットのアプリケーションでは、分岐ターゲットが分岐から 4GB 以上離れた場所にあると分岐予測のパフォーマンスが低下する。これは、アプリケーションが共有ライブラリーに分割される場合に発生しやすい。静的にビルドするとコードの局所性が向上し、LTO でビルドするとパフォーマンスがさらに向上する。

15.3.1.3 ループアンロールおよびループストリーム検出器

Silvermont マイクロアーキテクチャーは、バックエンドにデコード済みのマイクロオペレーション (uOP) を提供するループストリーム検出器 (LSD) を備えている。これは、パフォーマンスと消費電力において利点をもたらす。LSD を利用することで、プリフィクス+エスケープのバイト数や命令の長さなどのフロントエンドの制限が排除される。

ループのオーバーヘッドを減らし、独立したループ反復の作業量を増やす 1 つの方法として、ソフトウェアによるループアンロールを利用できる。ただし、ループアンロールは利点をもたらす一方、パフォーマンスを低下させる恐れもあるため、慎重に使用しなければならない。パフォーマンスの低下は、コードサイズが大きくなったり、BTB およびレジスターの負荷が増えることで生じる。また、ループアンロールにより、ループサイズが LSD の上限を超える可能性があるため、ループが LSD に収まるようにループサイズを 29 命令未満に抑える対策が必要である。

ユーザー/ソース・コーディング規則 1 (影響 M、一般性 M): 反復数の多いショートループでループアンロールを利用する場合は、反復あたりの命令数を 29 未満に抑える。

チューニングの推奨事項 3: perfmon カウンター BACLEARS.ANY で、ループアンロールにより負荷が大きくなりすぎていないかを確認できる。また、perfmon カウンター ICACHE.MISSES で、ループアンロールにより命令フットプリントに大きな悪影響が生じていないかを確認できる。

15.3.2 実行コアの最適化

15.3.2.1 スケジューリング

Silvermont マイクロアーキテクチャーでは、整数命令でアウトオブオーダー実行が導入されているため、前世代と比べると命令の実行順序が緩和されている。FP 命令には専用のリザベーション・ステーションが 2 つあるが、互いにインオーダーで実行される。メモリー命令もインオーダーで発行されるが、修復キュー (Rehab Queue) が追加されているため、アウトオブオーダーで完了することができ、メモリーシステムの遅延によって実行が妨げられることはない。

チューニングの推奨事項 4: perfmon カウンター NO_ALLOC_CYCLE.ANY からバックエンドのパフォーマンス・ボトルネックが分かる。このカウンター値にはメモリーシステムの遅延や実行の遅延などが含まれる。

15.3.2.2 アドレス生成

前世代のインテル® Atom™ マイクロアーキテクチャーのアドレス生成の制限は、Silvermont マイクロアーキテクチャーでは解決されている。そのため、LEA 命令と ADD 命令のどちらを使ってアドレスを生成しても、その効果は同じである。前世代のインテル® Atom™ マイクロアーキテクチャーでは LEA によるアドレス生成が推奨されていた。

経験則上、SCALE を使用するか、有効なインデックスやディスプレースメントを持つ LEA を非破壊デスティネーション（特にスタックオフセット）に使用し、そうでない場合は ADD を使用すると良い。

15.3.2.3 FP 乗算-加算-ストアの実行

Silvermont マイクロアーキテクチャーでは、異なるポートで実行する FP 算術命令は互いにアウトオブオーダーで実行できる。そのため、アンロールされたループで乗算結果を加算命令に供給し、その結果をストアする場合、ループの最後にストア命令をまとめることでパフォーマンスが向上する。この方法では、乗算命令と加算命令の実行をオーバーラップさせることができる。例 15-1 について考えてみる。

例 15-1 乗算-ストアポートの競合によってアンロールされたループはインオーダーで実行

命令	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
mulps, xmm1, xmm1	EX1	EX2	EX3	EX4	EX5													
addps xmm1, xmm1						EX1	EX2	EX3										
movaps mem, xmm1									EX1									
mulps, xmm2, xmm2										EX1	EX2	EX3	EX4	EX5				
addps xmm2, xmm2															EX1	EX2	EX3	
movaps mem, xmm2																		EX1

データの依存性により、加算命令は、対応する乗算命令が実行されるまで実行を開始できない。乗算命令とストア命令は、同じポートを使用するため、プログラム順に実行しなければならない。つまり、2 つ目の乗算命令は 1 つ目の乗算命令および加算命令と依存性がないにもかかわらず、実行を開始できない。次のように、ループの最後にストア命令をまとめることで、2 つ目の乗算命令を 1 つ目の乗算命令と並列に実行できる（乗算命令をオーバーラップさせると 1 サイクルのバブルが発生する）。

例 15-2 ストア命令をまとめることでバブルを排除し IPC を向上

命令	1	2	3	4	5	6	7	8	9	10	11
mulps, xmm1, xmm1	EX1	EX2	EX3	EX4	EX5						
addps xmm1, xmm1						EX1	EX2	EX3			
mulps, xmm2, xmm2		バブル	EX1	EX2	EX3	EX4	EX5				
addps xmm2, xmm2								EX1	EX2	EX3	
movaps mem, xmm1									EX1		
movaps mem, xmm2											EX1

15.3.2.4 整数乗算の実行

Silvermont マイクロアーキテクチャーには専用の整数乗算器がある。表 15-2 に各種 MUL/IMUL 命令のレイテンシーとマイクロオペレーション（uOP）の数を示す。

表 15-2 整数乗算命令のレイテンシー

入力サイズ	8 ビット		16 ビット		32 ビット		64 ビット	
出力サイズ/レイテンシー	サイズ	レイテンシー	サイズ	レイテンシー	サイズ	レイテンシー	サイズ	レイテンシー
imul/mul reg	16	5[u]	32	5[u]	64	4[u]	128	7[u]
imul/mul reg, reg			16	4[u]	32	3	64	4
imul/mul reg, reg, imm8			16	4[u]	32	3	64	4
imul/mul reg, reg, imm16/32			16	4[u]	32	3	64	4
u: マイクロコードを利用								

マイクロコードを利用する乗算形式は回避すべきである。また、この表から、imul, r32, rm32 形式の命令を実行する場合に最良のパフォーマンスが得られることが分かる。この形式の乗算命令は完全にパイプライン化されているが（1 サイクルで 1 つの結果を生成）、imul r64, r/m64 形式は 4 サイクルごとに 1 つの結果を生成する。

15.3.2.5 ゼロイディオム

XOR/PXOR/XORPS/XORPD 命令は、ソースレジスターとデスティネーション・レジスターが同じ場合（例: XOR eax, eax)、レジスター値をゼロに設定するのによく使用される。

同等の命令として MOV eax, 0x0 命令があるが、MOV エンコードのほうが XOR よりもコードバイトが大きくなるため、コンパイラーにとっては MOV よりもこれらの命令のほうが好ましい。

Silvermont マイクロアーキテクチャーには、これらのケースを認識し、アーキテクチャーのレジスターファイルでどちらのソースも有効としてマークする特別なハードウェア・サポートがある。どのような値であってもそれ自身と XOR することでゼロに設定できるため、これにより XOR を高速に実行できる。

このロジックは、PXOR、XORPS、XORPD でもサポートされる。

これらの命令では、64 ビット・オペランドはサポートされない。下位のアーキテクチャー・レジスターで 64 ビット・レジスターをゼロに設定することはできるが、REX.W ビットを避けなければならない。例えば、rax をゼロに設定するには、XOR rax, rax ではなく XOR eax, eax と指定する。同様に、r8 をゼロに設定するには、XOR r8, r8 ではなく XOR r8d, r8d と指定する。

15.3.2.6 フラグの使用

多くの命令には、フラグレジスターに格納される暗黙のデータがある。これらのデータは、条件移動（CMOVS)、分岐、さまざまな論理/算術演算（RCL など）といった幅広い命令で利用される。分岐条件としてよく使用される命令に比較命令（CMP）がある。CMP 命令に依存する分岐は次のサイクルで実行できる。ADD 命令や SUB 命令に依存する分岐でも同様のことが言える。

Silvermont マイクロアーキテクチャーでは、INC 命令と DEC 命令は一部のフラグのみ設定するため、フラグをマージする追加のマイクロオペレーション（uOP）が必要になる。そのため、INC 命令や DEC 命令に依存する分岐には 1 サイクルのペナルティーが伴う。このペナルティーは、INC 命令や DEC 命令に直接依存する分岐にのみ適用される。

アセンブリー/コンパイラー・コーディング規則 3 (影響 M、一般性 M): 分岐条件には、INC/DEC 命令ではなく CMP/ADD/SUB 命令をできるだけ使用する。

15.3.2.7 命令の選択

表 15-3 に、Silvermont マイクロアーキテクチャーの浮動小数点操作と SIMD 整数操作のレイテンシーを示す。「スループット」は、サイクルごとに実行を開始できる命令数を表している（例えば、1/4 は 4 サイクルごとに 1 命令を開始できることを示す)。

表 15-3 浮動小数点と SIMD 整数のレイテンシー

	レイテンシー	スループット*
SIMD 整数 ALU		
128 ビット ALU/論理演算/MOVE	1	2/1
64 ビット ALU/論理演算/MOVE	1	2/1
SIMD 整数シフト		
128 ビット	1	1/1
64 ビット	1	1/1
SIMD シャッフル		
128 ビット	1	1/1
64 ビット	1	1/1
SIMD 整数乗算器		
128 ビット	5	1/2

64 ビット	4	1/1
FP 加算器		
x87 (FADD)	3	1/1
スカラー (ADDSD、ADDSS)	3	1/1
パックド (ADDPD、ADDPS)	4	1/2
FP 乗算器		
x87 (FMUL)	5	1/2
スカラー単精度 (MULSS)	4	1/1
スカラー倍精度 (MULSD)	5	1/2
パックド単精度 (MULPS)	5	1/2
パックド倍精度 (MULPD)	7	1/4
変換器		
CVTDQ2PD、CVTDQ2PS、CVTPD2DQ、CVTPD2PI、CVTPD2PS、CVTPI2PD、CVTPS2DQ、CVTPS2PD、CVTTPD2DQ、CVTPD2PI、CVTPS2DQ	5	1/2
CVTPI2PS、CVTPS2PI、CVTSD2SI、CVTSD2SS、CVTSI2SD、CVTSI2SS、CVTSS2SD、CVTSS2SI、CVTTPS2PI、CVTTSD2SI、CVTTSS2SI	4	1/1
FP 除算器		
x87 FDIV (拡張精度)	39	1/39
x87 FDIV (倍精度)	34	1/34
x87 FDIV (単精度)	19	1/19
スカラー単精度 (DIVSS)	19	1/17
スカラー倍精度 (DIVSD)	34	1/32
パックド単精度 (DIVPS)	39	1/39
パックド倍精度 (DIVPD)	69	1/69
* スループットの値 "m/n" は、'n' サイクルごとに 'm' マイクロオペレーション (uOP) をディスパッチできることを示す。		

スカラー単精度乗算はほかの FP 操作よりも 1 サイクル速く、パックド倍精度操作はスカラー倍精度操作よりもレイテンシーがわずかに長く、スループットが小さい。

アセンブリー/コンパイラー・コーディング規則 4 (影響 M、一般性 M): x87 浮動小数点命令よりも SSE 浮動小数点命令を利用したほうが良い。

アセンブリー/コンパイラー・コーディング規則 5 (影響 MH、一般性 M): (できるだけ) 例外をマスクして、DAZ フラグと FTZ フラグを設定して実行する。

チューニングの推奨事項 5: perfmon カウンター MACHINE_CLEARS.FP_ASSIST で、浮動小数点例外がプログラムのパフォーマンスに影響しているかどうかを確認できる。

整数除算操作のレイテンシーは、入力値とデータサイズにより大きく異なる (これは、さまざまなインテル® マイクロアーキテクチャーで共通である)。表 15-4 と表15-5 に除算命令のレイテンシーの範囲を示す。

表 15-4 符号なし整数除算操作のレイテンシー

	被除数	除数	商	剰余	サイクル数
DIV r8	AX	r8	AL	AH	25
DIV r16	DX:AX	r16	AX	DX	26-30
DIV r32	EDX:EAX	r32	EAX	EDX	26-38
DIV r64	RDX:RAX	r64	RAX	RDX	38-123

表 15-5 符号付き整数除算操作のレイテンシー

	被除数	除数	商	剰余	サイクル数
IDIV r8	AX	r8	AL	AH	34
IDIV r16	DX:AX	r16	AX	DX	35-40
IDIV r32	EDX:EAX	r32	EAX	EDX	35-47
IDIV r64	RDX:RAX	r64	RAX	RDX	49-135

ユーザー/ソース・コーディング規則 2 (影響 M、一般性 L): 除算は本当に必要な場合のみ利用し、最も効率良く実行できるように正しいデータサイズと符号を使用する。

チューニングの推奨事項 6: perfmon カウンター UOPS_RETIRED.DIV と CYCLES_DIV_BUSY.ANY で、除算がプログラムのボトルネックになっているかどうかを確認できる。

アライメントされている配列からアライメントされていないパックド単精度のグループを取得する場合、MOVUPS よりも PALIGNR のほうが推奨される。例えば、load A[x+y+3:x+y] について考えてみる。ここで x と y はループ変数である。この場合、(x+y で MOVUPS を使用するよりも) x+y を計算して 4 の倍数に切り下げ、MOVAPS と PALIGNR で要素を取得したほうが良い。この方法は時間がかかるように見えるが、整数操作は FP 操作と並列に実行できる。また、約 6 サイクルのコストを伴う MOVUPS によるライン分割を回避することもできる。

ユーザー/ソース・コーディング規則 3 (影響 M、一般性 M): パックド単精度要素の取得には PALIGNR を使用する。

15.3.3 メモリーアクセスの最適化

15.3.3.1 メモリー操作の再発行/スリープの原因

メモリークラスターは、次のような状況でメモリー命令を RehabQ (修復キュー) に追加する。

- ロードのブロック
- ロード/ストアの分割
- ロック

- TLB ミス
- 不明なアドレス
- ストアが多すぎる

チューニングの推奨事項 7: perfmon カウンター REHABQ で、再発行によるアプリケーション・パフォーマンスへの影響を確認できる。

15.3.3.2 ストア・フォワーディング

Silvermont マイクロアーキテクチャーでは、前世代と比べてストア・フォワーディングが大幅に改善されている。次の条件を満たす場合、先行するストア操作命令から後続のロード命令にデータを転送できる。

- ストア操作とロード操作の開始アドレスが同じである。
- ロード操作の幅がストア操作の幅以下である。
- ストア操作またはロード操作でキャッシュラインの分割が発生しない。

これらの条件のいずれかが満たされない場合、ロードはブロックされ、再発行のために RehabQ に追加される。

以下のガイドラインに従って、ストア・フォワーディングの問題を排除/回避できる（推奨順に示す）。

- メモリーの代わりにレジスターを使用する。
- できるだけ早くストア操作を実行する（ストアはロードよりも後のパイプライン・ステージで実行されるため、ロードよりもかなり先行して実行する必要がある）。

Silvermont マイクロアーキテクチャーでは、前世代のインテル® Atom™ マイクロプロセッサーと比べて、ストア・フォワーディングに 3 サイクル追加される（つまり、ストアが n サイクルで実行されると、ロードは n+3 サイクルで実行される）。

15.3.3.3 PrefetchW 命令

Silvermont マイクロアーキテクチャーは PrefetchW 命令（0f 0d /1）をサポートしている。この命令は、RFO (read-for-ownership) 要求で指定したラインをキャッシュにプリフェッチするようにハードウェアを支援する。この命令を使用すると、後続のストアはラインがプリフェッチされていない場合や、別の命令でプリフェッチされた場合よりも、そのラインへの操作を速く完了できる。すべてのプリフェッチ命令は、正しく使用しないとパフォーマンスの低下につながる可能性があるため、PrefetchW を含め、プリフェッチ命令を使用する場合は実際にパフォーマンスが向上するように慎重に利用すべきである。命令オペコード 0f 0d /0 は引き続き NOP として処理され、指定されたラインはプリフェッチされない。

15.3.3.4 キャッシュラインの分割とアライメント

キャッシュラインの分割は、ロード命令とストア命令で利用可能な帯域幅を減らすため、できるだけ回避すべきである。

チューニングの推奨事項 8: perfmon カウンター REHABQ.ST_SPLIT と REHABQ.LD_SPLIT で、複数のキャッシュラインにまたがる操作とその数が分かる。

アライメントされたアクセスが推奨されるが、Silvermont マイクロアーキテクチャーには、アライメントされていないアクセスに対するハードウェア・サポートがある。そのため、前世代のインテル® Atom™ プロセッサーとは対照的に、MOVUPS/MOVUPD/MOVDQU 命令はすべて 1 つのマイクロオペレーション（uOP）命令である。

15.3.3.5 セグメントベース

Silvermont マイクロアーキテクチャーの AGU は、セグメントベースが 0 であると想定している。ほとんどの場合は問題ないが、ゼロ以外のセグメントベース（NZB）を使用しなければならない場合は、可能な限りセグメントベースをキャッシュライン（0x40）境界にアライメントする必要がある。NZB アドレスの生成には 1 サイクルのペナルティーが伴う。

15.3.3.6 コピーと文字列のコピー

通常、memcpy/memset ルーチンを含むライブラリーがコンパイラーによって提供される。これらのライブラリーは優れたパフォーマンスをもたらし、コードサイズとアライメントの問題にも対応している。

memcpy/memset 操作は、最適なバイト/ダブルワード単位のアライメントされた操作に分割した REP MOVS/STOS 命令で対応できる。これは、ほとんどの場合に汎用メモリーコピー/セットのソリューションとして利用できる。REP MOVS/STOS 命令には固有のオーバーヘッドがある。REP STOS は複数のキャッシュラインにまたがる長い文字列に対応できるが、REP MOVS はできない。これは、ソースとデスティネーション間のアライメントの一致が複雑であるためである。

特定のメモリーコピー/セットにおいて SIMD 命令を使用するマクロコード・シーケンスは、アライメント、バッファー長、バッファー内のキャッシュの有無に応じて、ある程度のパフォーマンスの向上をもたらす（約 12 サイクル）。ただし、複数のキャッシュラインにまたがる大きなメモリーのコピーは例外である。よく考慮されたマクロコードはキャッシュラインの分割を回避し、REP MOV のパフォーマンスを大幅に向上する。

Silvermont マイクロアーキテクチャー・ベースのプロセッサーは、REP MOVSB と STOSB の拡張操作（ERMSB）をサポートしている。MOVSB と STOSB を使用する REP 文字列操作は、メモリーのコピー/セット操作などのよくある状況において、最小コードサイズで柔軟かつハイパフォーマンスな REP 文字列操作を提供する。拡張 MOVSB/STOSB 操作をサポートするプロセッサーは、次のように CPUID 機能フラグで検出できる。
CPUID:(EAX=7H, ECX=0H):EBX.ERMSB[bit 9] = 1

汎用性のある実装（将来の実装を含む）で動作する単純なデフォルトの文字列コピー/セットルーチンが必要な場合は、ERMSB をサポートする実装で REP MOVSB または REP STOSB の使用を検討すべきである。これらの命令は、特定の実装では専用のコピー/セットルーチンよりも遅くなることがあるが、専用のコピー/セットルーチンは将来のプロセッサーで同じように動作せず、将来の拡張を利用できない恐れがある。REP MOVSB と REP STOSB は、将来のプロセッサーでもある程度のパフォーマンスが期待できる。

15.4 チューニング向けの一般的なパフォーマンス監視イベント

この節では、Silvermont マイクロアーキテクチャー・ベースのプロセッサーでサポートされているパフォーマンス監視イベントのうち、パフォーマンス・チューニングに役立つものを説明する。表 15-6 にパフォーマンス監視イベントの一覧を示す。イベントコードは IA32_PERFEVTSELx.EventSelect（ビット 7:0）に指定する値で、Umask 値は IA32_PERFEVTSELx.Umask（ビット 15:8）に指定する値である。

表 15-6 Silvermont マイクロアーキテクチャーのパフォーマンス・イベント

イベントコード	Umask 値	イベント名	定義	概要
03H	01H	REHABQ.LD_BLOCK_ST_FORWARD	ストアフォワードの制限によりブロックされたロード	アドレスの不一致により、先行するストアからデータを受け取れなかったリタイアしたロードの数をカウントする。
03H	02H	REHABQ.LD_BLOCK_STD_NOTREADY	ストアデータが利用できないためにブロックされたロード	データの転送は可能だが、ストアデータが利用できないためにストアフォワードが行われなかった回数をカウントする。
03H	04H	REHABQ.ST_SPLITS	複数のキャッシュライン境界にまたがるストア・マイクロオペレーション（uOP）	複数のキャッシュライン境界にまたがったリタイアしたストア命令の数をカウントする。
03H	08H	REHABQ.LD_SPLITS	複数のキャッシュライン境界にまたがるロード・マイクロオペレーション（uOP）	複数のキャッシュライン境界にまたがったリタイアしたロード命令の数をカウントする。
03H	10H	REHABQ.LOCK	ロック・セマンティクスを使用するマイクロオペレーション（uOP）	ロック・セマンティクスを使用するリタイアしたメモリー操作の数をカウントする。暗黙のロック命令（XCHG など）と明示的な LOCK プリフィクスを持つ命令（0xF0）が含まれる。
03H	20H	REHABQ.STA_FULL	ストア・アドレス・バッファーがフル	ストア・アドレス・バッファーが利用できないために遅延が発生したリタイアしたストア命令の数をカウントする。
03H	40H	REHABQ.ANY_LD	再発行されたロード・マイクロオペレーション	RehabQ から再発行されたロード・マイクロオペレーション（uOP）の数をカウントする。

			(uOP)	
03H	80H	REHABQ.ANY_ST	再発行されたストア・マイクロオペレーション (uOP)	RehabQ から再発行されたストア・マイクロオペレーション（uOP）の数をカウントする。
REHABQ は Silvermont マイクロアーキテクチャーの内部キューであり、何らかの理由で完了できないメモリー参照マイクロオペレーション（uOP）を保持する。REHABQ 内のマイクロオペレーション（uOP）は、再発行され処理が成功するまで REHABQ に残る。 マイクロオペレーション（uOP）が REHABQ に送られる原因となるボトルネックの例として、キャッシュラインの分割、ブロックされたストアフォワード、データが利用できないことが挙げられる。ロードまたはストアが REHABQ に送られる条件はこのほかにも多数ある。例えば、先行するストアのアドレスが不明な場合、アドレスが判明するまで後続のストアはすべて REHABQ に送られる。				
04H	01H	MEM_UOP_RETIRED.LD_DCU_MISS	DCU ミスになったリタイアしたロード	L1 データ・キャッシュ・ミスになったリタイアしたロード・マイクロオペレーション（uOP）の数をカウントする。プリフェッチ・ミスはカウントされない。
04H	02H	MEM_UOP_RETIRED.LD_L2_HIT	L2 でヒットしたリタイアしたロード	L2 でヒットしたリタイアしたロード・マイクロオペレーション（uOP）の数をカウントする。
04H	04H	MEM_UOP_RETIRED.LD_L2_MISS	L2 ミスになったリタイアしたロード	L2 ミスになったリタイアしたロード・マイクロオペレーション（uOP）の数をカウントする。
04H	08H	MEM_UOP_RETIRED.LD_DTLB_MISS	DTLB ミスになったロード	DTLB ミスになったリタイアしたロード操作の数をカウントする。
04H	10H	MEM_UOP_RETIRED.LD_UTLB_MISS	UTLB ミスになったロード	UTLB ミスになったリタイアしたロード操作の数をカウントする。
04H	20H	MEM_UOP_RETIRED.HITM	クロスコア/クロスモジュールの HITM	ほかのコアやほかのモジュールからデータを受け取ったリタイアしたロード操作の数をカウントする。
04H	40H	MEM_UOP_RETIRED.ANY_LD	すべてのロード	リタイアしたロード操作の数をカウントする。
04H	80H	MEM_UOP_RETIRED.ANY_ST	すべてのストア	リタイアしたストア操作の数をカウントする。
05H	01H	PAGE_WALKS.D_SIDE_CYCLES	データ・ページ・ウォークの継続期間（コアサイクル数）	ロードによるデータ・ページ・ウォークの継続期間中のサイクル数をカウントする。ページウォークの継続期間をページウォークの数で割ると、ページウォークの継続期間の平均が得られる。 Edge トリガービットはクリアする必要がある。ページウォークの数をカウントする場合は Edge を設定する。
05H	02H	PAGE_WALKS.I_SIDE_CYCLES	命令ページウォークの継続期間（コアサイクル数）	命令フェッチによる命令ページウォークの継続期間中のサイクル数をカウントする。ページウォークの継続期間をページウォークの数で割ると、ページウォークの継続期間の平均が得られる。 Edge トリガービットはクリアする必要がある。ページウォークの数をカウントする場合は Edge を設定する。
2EH	41H	LLC_RQSTS.MISS	L2 キャッシュ要求ミス	L2 キャッシュ参照の数と L2 キャッシュミスの数をカウントする。 L3 は、Silvermont マイクロアーキテクチャーではサポートされない。
2EH	4FH	LLC_RQSTS.ANY	このコアからの L2 キャッシュ要求	L2 キャッシュ内のキャッシュラインを参照するコアからの要求の数をカウントする。 L3 は、Silvermont マイクロアーキテクチャーではサポートされない。

30H	00H	L2_REJECT_XQ	拒否された XQ への L2 キャッシュ要求	XQ が一杯またはほぼ一杯 (IDI リンクからのバック・プレッシャーを示している可能性が高い) のため、L2 XQ に拒否されたデマンドとプリフェッチ・トランザクションの数をカウントする。XQ は、L2Q (キャッシュできない要求)、BBS (L2 ミス)、WOB (L2 ライトバック対象) からのトランザクションを拒否することがある。
メモリー参照は、L1 キャッシュミスが生じると L2 キュー (L2Q) に送られる。L2 キャッシュミスが起こると XQ に送られ、そこで IDI リンクを介してメモリーへ発行されるまで待機する。L2 はプロセッサー・コアのペアで共有されるため、1 つの L2Q が 2 つのコア間で共有される。同様に、L2Q と IDI リンクの間に配置されている 1 つの XQ が、プロセッサーのペアで共有される。 XQ が新しい要求を受け取る速度よりも、IDI リンクからの応答速度のほうが遅いと、XQ は一杯になる。L2_reject_XQ イベントは、XQ が一杯で L2 キューから XQ へ要求を移動できないこと、つまり、メモリーシステムのオーバーサブスクリプションを示す。				
31H	00H	CORE_REJECT	L2Q によって拒否されたデマンドと L1 プリフェッチャー要求	L2Q が一杯またはほぼ一杯 (L2Q からのバック・プレッシャーを示している可能性が高い) のため、L2Q に拒否されたデマンドと L1 プリフェッチャー要求の数をカウントする。XQ に直接送られ、XQ が一杯またはほぼ一杯 (IDI リンクからのバック・プレッシャーを示している可能性が高い) のため拒否された要求もカウントされる。L2Q は、コア間の公平性を維持したり、受け取る外部スヌープのアドレスが競合する場合にコアのダーティーデータの退避を遅らせるため、コアからのトランザクションを拒否する場合がある。拒否された L2 プリフェッチャー要求はカウントされない。
CORE_REJECT イベントは、コアからの要求を L2Q で受け付けられないことを示す。ただし、要求が L2Q によって拒否される理由はこのほかにもいくつかある。L2Q が一杯で要求を拒否する場合のほかに、ほかのコアとの公平性を維持するために、あるコアからの要求を拒否することがある。つまり、あるコアが共有の L2Q/キャッシュ/XQ/IDI リンクを占領しないように、L2Q に利用可能な領域があっても、そのコアの要求を拒否することがある。さらに、コアから L1 キャッシュのダーティーデータの退避が要求された場合、退避によって L2Q 内の保留中の要求と競合が発生しないようにハードウェアは保証しなければならない (保留中の要求には外部スヌープも含まれる)。競合が発生すると、ダーティーデータの退避要求は、L2Q に利用可能な領域があっても拒否されることがある。 そのため、L2_REJECT_XQ イベントは 2 つのコアからのメモリー要求速度がメモリーの応答速度を超えているかどうかを示すのに対して、CORE_REJECT イベントは L2Q への要求速度が XQ からの応答速度を超えているかどうか、L2Q への要求速度が L2 からの応答速度を超えているかどうか、あるいはあるコアがほかのコアよりも多くの応答を L2Q から要求しているかどうか、そして、ダーティーデータの退避と保留中の要求との間に競合があるかどうかを示す。 つまり、L2_REJECT_XQ イベントはメモリー・オーバーサブスクリプションを示し、CORE_REJECT イベントは次のいずれかを示す: (1) メモリー・オーバーサブスクリプション、(2) L2 オーバーサブスクリプション、(3) コア間の公平性を維持するためにあるコアからの要求を拒否、(4) ダーティーデータの退避と保留中の要求との間の競合。				
3CH	00H	CPU_CLK_UNHALTED.CORE_P	コアが停止されなかったコアサイクル数	コアが停止状態でなかったコアサイクル数をカウントする。HLT 命令を実行中、コアは停止状態になる。モバイルシステムでは、コア周波数はそのときどきで変わる可能性がある。そのため、このイベントの比率も変わる可能性がある。
なし	01H	CPU_CLK_UNHALTED.CORE	リタイアした命令数	固定カウンター 1 を使って、CPU_CLK_UNHALTED.CORE_P と同じ状態をカウントする。
3CH	01H	CPU_CLK_UNHALTED.REF_P	コアが停止されなかった参照サイクル数	コアが停止状態でなかった参照サイクル数をカウントする。HLT 命令を実行中、コアは停止状態になる。 モバイルシステムでは、コア周波数はそのときどきで変わる可能性がある。このイベントはコア周波数の変動に影響されず、コアが常に最大周波数で実行しているかのようにカウントする。

なし	02H	CPU_CLK_UNHALTED.REF_	リタイアした命令数	固定カウンター 2 を使って、CPU_CLK_UNHALTED.REF_P と同じ状態をカウントする。
80H	01H	ICACHE.HIT	命令キャッシュからの命令フェッチ数	命令キャッシュからのすべての命令フェッチの数をカウントする。
80H	02H	ICACHE.MISSES	命令キャッシュミス	命令キャッシュミスまたはメモリー要求が生じたすべての命令フェッチの数をカウントする。キャッシュできないフェッチを含む。命令フェッチミスは、フェッチされるまでサイクルごとにカウントされるのではなく、一度のみカウントされる。
80H	03H	ICACHE.ACCESSES	命令フェッチ数	キャッシュできないフェッチを含め、すべての命令フェッチをカウントする。
B7H	01H	OFFCORE_RESPONSE_0	「オフコア応答イベント」を参照のこと。	要求タイプと応答の指定に MSR_OFFCORE_RESP0 が必要。
B7H	02H	OFFCORE_RESPONSE_1	「オフコア応答イベント」を参照のこと。	要求タイプと応答の指定に MSR_OFFCORE_RESP1 が必要。
C0H	00H	INST_RETIRED.ANY_P	リタイアした命令数 (PEBS は IA32_PMC0 でサポート)	実行をリタイアした命令数をカウントする。複数のマイクロオペレーション (uOP) から成る命令の場合、その命令の最後のマイクロオペレーション (uOP) のリタイアの数をカウントする。ハードウェア割り込み中、トラップ中、そして割り込みハンドラー内でもカウントが続行される。
なし	00H	INST_RETIRED.ANY	リタイアした命令数	固定カウンター 0 を使って、INST_RETIRED.ANY_P と同じ状態をカウントする。
C2H	01H	UOPS_RETIRED.MS	リタイアした MSROM マイクロオペレーション (uOP) の数	MSROM から供給されたリタイアしたマイクロオペレーション (uOP) の数をカウントする。
C2H	02H	UOPS_RETIRED.X87	リタイアした X87 マイクロオペレーション (uOP) の数	x87 ハードウェアを使用したリタイアしたマイクロオペレーション (uOP) の数をカウントする。
C2H	04H	UOPS_RETIRED.MUL	リタイアした MUL マイクロオペレーション (uOP) の数	MUL ハードウェアを使用したリタイアしたマイクロオペレーション (uOP) の数をカウントする。
C2H	08H	UOPS_RETIRED.DIV	リタイアした DIV マイクロオペレーション (uOP) の数	DIV ハードウェアを使用したリタイアしたマイクロオペレーション (uOP) の数をカウントする。
C2H	10H	UOPS_RETIRED.ANY	リタイアしたマイクロオペレーション (uOP) の数	リタイアしたマイクロオペレーション (uOP) の数をカウントする。
プロセッサーは、複雑なマクロ命令を単純なマイクロオペレーション (uOP) のシーケンスにデコードする。ほとんどの命令は 1 つまたは 2 つのマイクロオペレーション (uOP) にデコードされる。リピート命令、浮動小数点超越命令、アシストなど、一部の命令はより長いシーケンスにデコードされる。一部のケースでは、マイクロオペレーション (uOP) のシーケンスが融合されたり、命令全体が 1 つのマイクロオペレーション (uOP) に融合される。リタイアした融合された命令とリタイアした融合されていない命令は、ほかの UOPS_RETIRED イベントから区別できる。				
C3H	01H	MACHINE_CLEARS.SMC	自己修正コードの検出回数	プログラムがコードセクションへの書き込みを行った回数をカウントする。自己修正コードは、すべてのインテル® アーキテクチャー・プロセッサーにおいて重大なペナルティーにつながる。

C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	メモリーの順序付けによるストール数	メモリーの順序付けの問題により、パイプラインがクリアされた回数をカウントする。
C3H	04H	MACHINE_CLEARS.FP_ASSIST	FP アシストによるストール数	アシストが必要な FP 操作により、パイプラインがストールされた回数をカウントする。
C3H	08H	MACHINE_CLEARS.ANY	すべてのストール数	SMC、MO、FP アシストなど、何らかの理由でパイプラインがストールされた回数をカウントする。
マシンクリアは、(割り込み、トラップ、フォルトの受け取りを含む) 多くの条件によって生じる。これらの条件 (MO、SMC、FP、ほか) はすべて ANY イベントで取得できる。さらに、いくつかの条件 (SMC、MO、FP) については個別にカウントできる。ただし、SMC、MO、FP マシンクリアの合計が、必ずしも ANY の数に等しくなるとは限らない。 FP アシスト: ほとんどの場合、浮動小数点実行ユニットは適切に正しい結果を生成できるが、まれに支援を必要とすることがある。その場合、対象命令に対してマシンクリアがアサートされる。(前述のとおり) マシンクリアが行われると、マシンのフロントエンドは要求された FP 命令を特定するための命令を送り、正しい FP 結果を生成できるように追加処理を行う (例えば、結果が浮動小数点デノーマルの場合、ハードウェアは IEEE に準拠するように正しく丸められた結果を生成するため、支援が必要になることがある)。 SMC (自己修正コード): SMC は、実行中の命令が変更されている恐れがある場合に起こる。例えば、実行中の命令の後続の命令ストリームに対する変更を含むコードを記述した場合などが挙げられる。Silvermont マイクロアーキテクチャーでは、アライメントされた 1K 領域内でこの検出が行われる。 実行中の場所から 1K 以内のメモリー位置へ書き込みを行うと、命令が変更された恐れがあると判断され、マシンクリアがアサートされる可能性がある。マシンクリアはストア・パイプラインを空にするため、フロントエンドの再起動時に (変更後の) 正しい命令が実行される。				
MO (メモリーオーダー): MO マシンクリアは、スヌープ要求時にメモリーオーダーが保持されるかどうか不明な場合に発生する。例えば、プログラム順では 1 つ目はアドレス X へロードし、2 つ目はアドレス Y へロードする、連続する 2 つのロードがあるとする。どちらのロードも発行済みの場合、Y へのロードが先に完了すると、X へのロードが待機中のまま、Y へのロードに依存するすべての操作は Y に読み込まれたデータを使用することになる。同時に、別のプロセッサーが同じアドレス Y へ書き込みを行うと、アドレス Y に対するスヌープが発生する。 これは、Y に古い値がロードされる一方、X へのロードが完了しないという問題につながる。ほかのプロセッサーはロードが異なる順序で行われたこと検知し、アドレス Y へのストアから最新の値を取得しない。アドレス Y へのストア後のデータを利用できるようにするには、アドレス Y へのロードからすべてをやり直す必要がある。このメモリーオーダーの問題は、アドレス X へのロードが完了していないことが原因で生じているため、保留中の読み込みがない場合、アドレス Y へのロードをやり直す必要はない。				
C4H	00H	BR_INST_RETIRED.ANY	リタイアした分岐命令の数	リタイアした分岐命令の数をカウントする。
C4H	7EH	BR_INST_RETIRED.JCC	リタイアした条件付きジャンプ分岐命令の数	リタイアした条件付きジャンプ分岐命令の数をカウントする。
C4H	BFH	BR_INST_RETIRED.FAR	リタイアした遠くへの分岐命令の数	リタイアした遠くへの分岐命令の数をカウントする。
C4H	EBH	BR_INST_RETIRED.NON_RETURN_IND	リタイアした近くへの間接ジャンプまたは呼び出し命令の数	リタイアした近くへの間接ジャンプまたは近くへの間接呼び出し分岐命令の数をカウントする。
C4H	F7H	BR_INST_RETIRED.RETURN	リタイアした近くへのリターン命令の数	リタイアした近くへの RET 分岐命令の数をカウントする。
C4H	F9H	BR_INST_RETIRED.CALL	リタイアした近くへの呼び出し命令の数	リタイアした近くへの CALL 分岐命令の数をカウントする。
C4H	FBH	BR_INST_RETIRED.IND_CALL	リタイアした近くへの間接呼び出し命令の数	リタイアした近くへの間接 CALL 分岐命令の数をカウントする。
C4H	FDH	BR_INST_RETIRED.REL_CALL	リタイアした近くへの相対呼び出し命令の数	リタイアした近くへの相対 CALL 分岐命令の数をカウントする。

C4H	FEH	BR_INST_RETIRED.TAKEN_JCC	リタイアした分岐すると予測された条件付きジャンプの数	リタイアした分岐すると予測された条件付きジャンプ分岐命令の数をカウントする。
C5H	00H	BR_MISP_INST_RETIRED.ANY	リタイアした予測ミスした分岐命令の数	リタイアした予測ミスした分岐命令の数をカウントする。
C5H	7EH	BR_MISP_INST_RETIRED.JCC	リタイアした予測ミスした条件付きジャンプの数	リタイアした予測ミスした条件付きジャンプ分岐命令の数をカウントする。
C5H	BFH	BR_MISP_INST_RETIRED.FAR	リタイアした予測ミスした遠くへの分岐命令の数	リタイアした予測ミスした遠くへの分岐命令の数をカウントする。
C5H	EBH	BR_MISP_INST_RETIRED.NON_RETURN_IND	リタイアした予測ミスした近くへの間接ジャンプまたは呼び出し命令の数	リタイアした予測ミスした近くへの間接ジャンプまたは近くへの間接 CALL 分岐命令の数をカウントする。
C5H	F7H	BR_MISP_INST_RETIRED.RETURN	リタイアした予測ミスした近くへのリターン命令の数	リタイアした予測ミスした近くへの RET 分岐命令の数をカウントする。
C5H	F9H	BR_MISP_INST_RETIRED.CALL	リタイアした予測ミスした近くへの呼び出し命令の数	リタイアした予測ミスした近くへの CALL 分岐命令の数をカウントする。
C5H	FBH	BR_MISP_INST_RETIRED.IND_CALL	リタイアした予測ミスした近くへの間接呼び出し命令の数	リタイアした予測ミスした近くへの間接 CALL 分岐命令の数をカウントする。
C5H	FDH	BR_MISP_INST_RETIRED.REL_CALL	リタイアした予測ミスした近くへの相対呼び出し命令の数	リタイアした予測ミスした近くへの相対 CALL 分岐命令の数をカウントする。
C5H	FEH	BR_MISP_INST_RETIRED.TAKEN_JCC	分岐すると予測されたが分岐しなかったリタイアした条件付きジャンプの数	分岐すると予測されたが分岐しなかったリタイアした条件付きジャンプ分岐命令の数をカウントする。
CAH	3FH	NO_ALLOC_CYCLES.ANY	フロントエンドから命令が供給されなかったサイクル数	何らかの理由でフロントエンドから命令が供給されなかったサイクルの数をカウントする。
CAH	50H	NO_ALLOC_CYCLES.NOT_DELIVERED	フロントエンドから命令が供給されなかったがバックエンドがストールしなかったサイクル数	何らかの理由でフロントエンドから命令が供給されなかったが、バックエンドがストールしなかったサイクルの数をカウントする。
フロントエンドは命令をフェッチし、マイクロオペレーション（uOP）へデコードして、バックエンドによって処理されるマイクロオペレーション・キューに配置する。バックエンドはキューからマイクロオペレーション（uOP）を取得して、必要なリソースを割り当てる。すべてのリソースの準備が完了するとマイクロオペレーション（uOP）が実行される。バックエンドでフロントエンドからのマイクロオペレーション（uOP）の受け入れ準備ができていない場合は、フロントエンドのボトルネックとしてカウントしない。しかし、バックエンドでボトルネックが発生した場合は常に、アロケーション・ユニットがストールし、最終的にフロントエンドはバックエンドの準備が完了するまで待機しなければならない。このイベントは、バックエンドがマイクロオペレーション（uOP）を要求し、フロントエンドが供給できない場合のサイクル数のみカウントする。				
CBH	01H	RS_FULL_STALL.MEC	MEC RS がフル	MEC クラスターの RS が一杯でアロケーション・パイプラインがストールしたサイクル数をカウントする。
CBH	02H	RS_FULL_STALL.IEC_PORT0	ポート 0 の IEC RS がフル	ポート 0 の整数クラスターの RS が一杯でアロケーション・パイプラインがストールしたサイクル数をカウントする。

CBH	04H	RS_FULL_STALL.IEC_PORT1	ポート 1 の IEC RS がフル	ポート 1 の整数クラスターの RS が一杯でアロケーション・パイプラインがストールしたサイクル数をカウントする。
CBH	08H	RS_FULL_STALL.FPC_PORT0	ポート 0 の FPC RS がフル	ポート 0 の FP クラスターの RS が一杯でアロケーション・パイプラインがストールしたサイクル数をカウントする。
CBH	10H	RS_FULL_STALL.FPC_PORT1	ポート 1 の FPC RS がフル	ポート 1 の FP クラスターの RS が一杯でアロケーション・パイプラインがストールしたサイクル数をカウントする。
CBH	1FH	RS_FULL_STALL.ANY	いずれかの RS がフル	いずれかの RS が一杯でアロケーション・パイプラインがストールしたサイクル数をカウントする。
Silvermont マイクロアーキテクチャーには、マイクロオペレーション（uOP）をフロントエンドからバックエンドへ移動するアロケーション・パイプライン（RAT と呼ばれる）がある。アロケーション・パイプラインの最後で、マイクロオペレーション（uOP）は 6 つのリザベーション・ステーション（RS）のいずれかに書き込まれる。各 RS には特定の実行（またはメモリー）クラスターへ送られるマイクロオペレーション（uOP）が格納されている。各 RS の容量は決まっており、マイクロオペレーション（uOP）を実行クラスターへ送ることができない場合、それらは蓄積される。RS が一杯になるよくある原因として、除算などのレイテンシーの長いマイクロオペレーション（uOP）の実行、依存関係によりマイクロオペレーション（uOP）をスケジュールできない場合、メモリー参照が多すぎる場合が挙げられる。RS が一杯になるとマイクロオペレーション（uOP）を受け付けられなくなり、アロケーション・パイプラインがストールする。RS_FULL_STALL.ANY イベントは、いずれかの RS が一杯でアロケーションがストールされたサイクルにのみアサートされる（つまり、アロケーション・パイプラインがストールしても RS が一杯でない場合、RS_FULL_STALL.ANY イベントはそのサイクルをカウントしない）。サブイベントから、どの RS によってアロケーションがストールされたのかが分かる。				
CDH	01H	CYCLES_DIV_BUSY.ANY	除算器がビジー	除算器がビジー状態だったサイクル数をカウントする。
このイベントは、除算器がディスパッチ済みマイクロオペレーション（uOP）の処理中で、新しい除算マイクロオペレーション（uOP）を受け付けることができないサイクル数をカウントする。RS から除算器への供給を待機しているほかの除算マイクロオペレーション（uOP）があるかどうかは関係ない。また、除算処理中のサイクルは RS が空でもカウントされる。				
E6H	01H	BACLEARS.ANY	すべての分岐にアサートされた BACLEARS の数	すべての分岐の BACLEARS の数をカウントする。
E6H	02H	BACLEARS.INDIRECT	間接分岐にアサートされた BACLEARS の数	間接分岐の BACLEARS の数をカウントする。
E6H	04H	BACLEARS.UNCOND	無条件分岐にアサートされた BACLEARS の数	無条件分岐の BACLEARS の数をカウントする。
E6H	08H	BACLEARS.RETURN	リターン分岐にアサートされた BACLEARS の数	リターン分岐の BACLEARS の数をカウントする。
E6H	10H	BACLEARS.COND	条件付き分岐にアサートされた BACLEARS の数	条件付き分岐の BACLEARS の数をカウントする。
E7H	01H	MS_DECODED.MS_ENTRY	MS デコードによって開始された回数	MSROM によってマイクロオペレーション（uOP）のフローが開始された回数をカウントする。
E9H	01H	DECODE_RESTRICTION.PREDECODE_WRONG	命令長の予測ミスによる遅延回数	プリデコード・キャッシュからの命令長の予測が正しくなかった回数をカウントする。
命令によりバイト数は異なる。プロセッサーは、フロントエンドが命令の開始アドレスと終了アドレスを把握できるように命令長（バイト数）の予測を試みる。予測ミスした場合、プロセッサーは適切にデコードできるように正しい命令長を特定しなければならず、固定サイクル数のペナルティーが生じる。				

15.4.1 オフコア応答イベント

イベントコード 0B7H は、MSR_OFFCORE_RSP0 (アドレス 0x1A6) とアンマスク値 01H または MSR_OFFCORE_RSP1 (アドレス 0x1A7) と UMask 値 02H の組み合わせにより、オフコア応答の監視をサポートする。表 15-7 にイベントコード、UMask 値、IA32_PMCx を使ってオフコア関連のイベントをカウントするため追加で設定しなければならないオフコア構成 MSR の一覧を示す。

表 15-7 オフコア応答イベントのエンコーディング

カウンター	イベントコード	UMask	必要なオフコア応答 MSR
PMC0-3	0xB7	0x01	MSR_OFFCORE_RSP0 (アドレス 0x1A6)
PMC0-3	0xB7	0x02	MSR_OFFCORE_RSP1 (アドレス 0x1A7)

MSR_OFFCORE_RSP0 と MSR_OFFCORE_RSP1 のレイアウトを図 15-2 と図 15-3 に示す。ビット 15:0 はアンコアへのトランザクション要求タイプ、ビット 30:16 は供給元情報、ビット 37:31 はスヌープ応答情報を指定する。

さらに、MSR_OFFCORE_RSP0 には、2 つのプログラム可能なカウンターを同時に使ってオフコア・トランザクション要求の平均レイテンシーを測定できるビット 38 がある。詳細は、「15.4.2 平均オフコア要求レイテンシーの測定」を参照のこと。

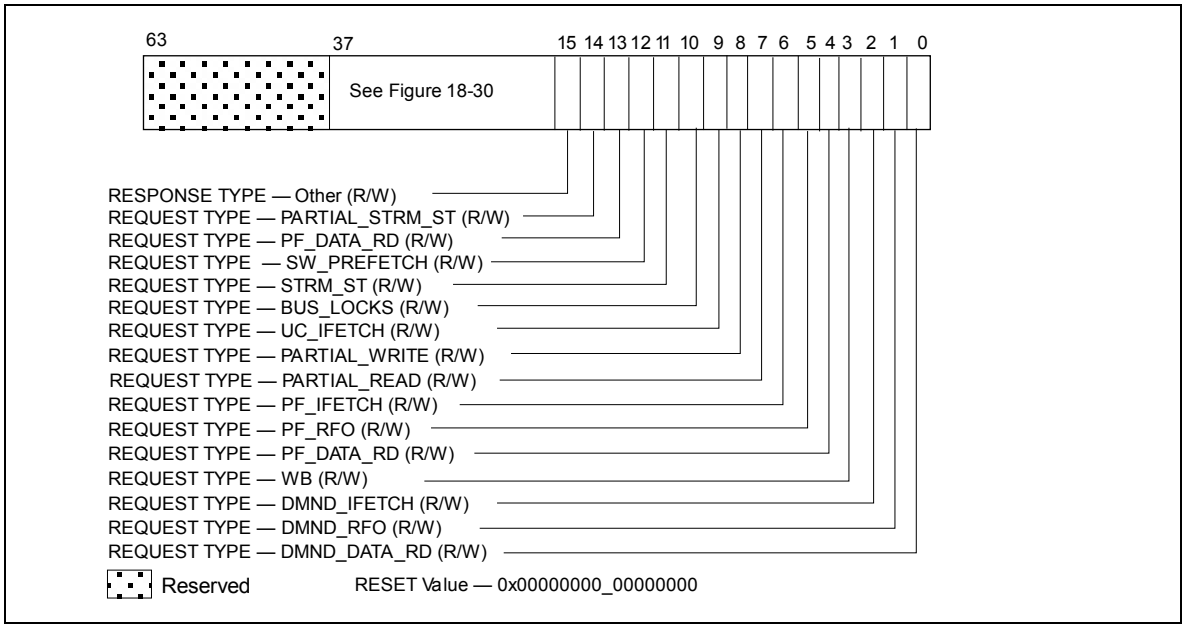

図 15-2 MSR_OFFCORE_RSPx の要求タイプフィールド

表 15-8 MSR_OFFCORE_RSPx の要求タイプフィールド定義

ビット名	オフセット	説明
DMND_DATA_RD	0	(R/W)。全体および部分的なキャッシュラインのデータ読み取り要求と DCU プリフェッチの数、およびページ・テーブル・エントリーのキャッシュラインのデータ読み取り要求の数をカウントする。L2 データの読み取りプリフェッチまたは命令フェッチはカウントされない。
DMND_RFO	1	(R/W)。データ・キャッシュラインへの書き込みによって生成される RFO (read-for-ownership) 要求と DCU プリフェッチの数をカウントする。L2 RFO プリフェッチはカウントされない。
DMND_IFETCH	2	(R/W)。キャッシュラインの読み取り要求と DCU プリフェッチ命令の数をカウントする。L2 コードの読み取りプリフェッチはカウントされない。

WB	3	(R/W)。ライトバック (Modified 状態から Exclusive 状態になる) トランザクション数をカウントする。
PF_DATA_RD	4	(R/W)。L2 プリフェッチャーによって生成されるデータ・キャッシュラインの読み取りの数をカウントする。
PF_RFO	5	(R/W)。L2 プリフェッチャーによって生成される RFO 要求の数をカウントする。
PF_IFETCH	6	(R/W)。L2 プリフェッチャーによって生成されるコード読み取りの数をカウントする。
PARTIAL_READ	7	(R/W)。(UC と WC を含む) 部分的なキャッシュラインの読み取り要求の数をカウントする。
PARTIAL_WRITE	8	(R/W)。(UC、WT、WP を含む) 部分的なキャッシュラインへの書き込みを行う RFO 要求の数をカウントする。
UC_IFETCH	9	(R/W)。UC 命令フェッチの数をカウントする。
BUS_LOCKS	10	(R/W)。バスロック要求とロック分割要求の数をカウントする。
STRM_ST	11	(R/W)。ストリーミング・ストア要求の数をカウントする。
SW_PREFETCH	12	(R/W)。ソフトウェア・プリフェッチ要求の数をカウントする。
PF_DATA_RD	13	(R/W)。DCU ハードウェア・プリフェッチ・データ読み取り要求の数をカウントする。
PARTIAL_STRM_ST	14	(R/W)。ストリーミング・ストア要求の数をカウントする。
OTHER	15	(R/W)。IDI に関連するそのほかの要求 (I/O を含む) をカウントする。

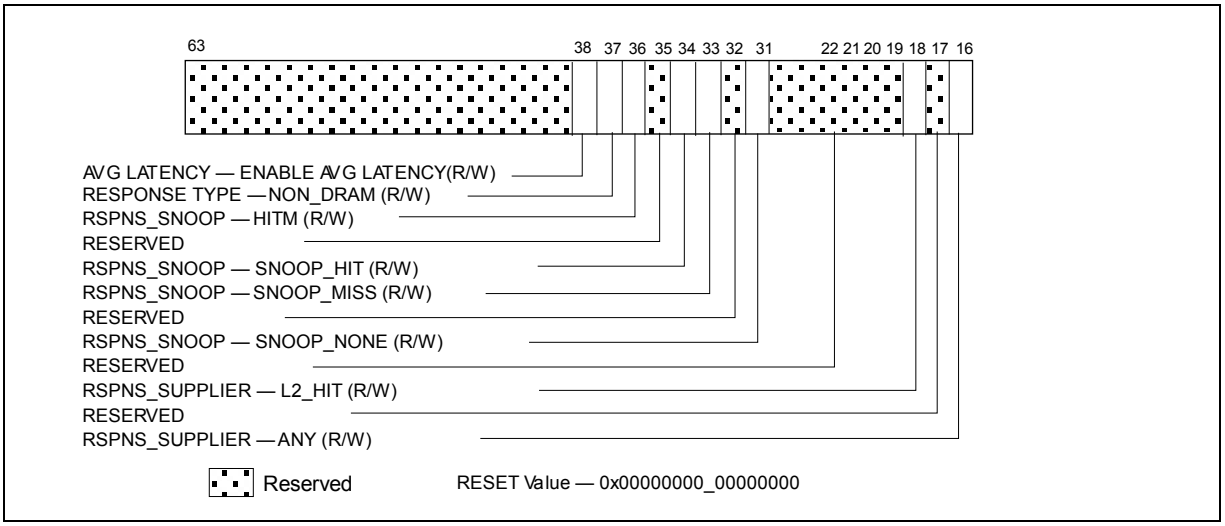

図 15-3 MSR_OFFCORE_RSPx の応答供給元情報フィールドとスヌープ情報フィールド

この追加レジスターを利用するには、ソフトウェアで少なくとも 1 つの要求タイプビットと有効な応答タイプパターンを設定する必要がある。そうでないと、イベント数はゼロになる。複数の要求タイプビットと応答タイプビットを設定すると、さまざまなクラスのオフコア応答イベントを取得できる。MSR_OFFCORE_RSPx では、エージェント・ソフトウェアにより上記のガイドラインを満たす多数の組み合わせが可能だが、すべての組み合わせが意味のあるデータを生成するとは限らない。

表 15-9 MSR_OFFCORE_RSP_x の応答供給元情報フィールド定義

サブタイプ	ビット名	オフセット	説明
共通	Any	16	(R/W)。全応答タイプのすべての値を検出する。
供給元情報	予約済み	17	予約済み。
	L2_HIT	18	(R/W)。M/E/S いずれかの状態の L2 キャッシュ参照のヒット数をカウントする。
	予約済み	30:19	予約済み。

オフコア応答フィルターを指定するには、ソフトウェアにより要求フィールドと要求タイプフィールドで適切なビットを設定する必要がある。有効な要求タイプは、予約済みでないビット 15:0 の少なくとも 1 つが設定されていなければならない。有効な応答タイプは、次の式の値が非ゼロにならなければならない。

ANY | [(すべての供給元情報ビットの 'OR') & (すべてのスヌープ情報ビットの 'OR')]

"ANY" ビットが設定されている場合、供給元情報ビットとスヌープ情報ビットは無視される。

表 15-10 MSR_OFFCORE_RSPx のスヌープ情報フィールド定義

サブタイプ	ビット名	オフセット	説明
スヌープ情報	SNP_NONE	31	(R/W)。スヌープ関連情報の詳細を出力しない。
	予約済み	32	予約済み。
	SNOOP_MISS	33	(R/W)。L2 ミス時のスヌープミスの数をカウントする。
	SNOOP_HIT	34	(R/W)。変更されたコピーがないほかのモジュールでヒットしたスヌープ数をカウントする。
	予約済み	35	予約済み。
	HITM	36	(R/W)。変更されたコピーがほかのコアの L1 キャッシュで見つかったほかのモジュールでヒットしたスヌープ数をカウントする。
	NON_DRAM	37	(R/W)。ターゲットが非 DRAM システムアドレスの場合をカウントする。MMIO トランザクションを含む。
	AVG_LATENCY	38	(R/W)。ビット 15:0 で指定された要求タイプとすべての応答 (ビット 37:16 が 0 にクリアされる) のオフコア要求の重み付けされたサイクル数をカウントすることで、平均レイテンシーを測定する。 このビットは MSR_OFFCORE_RESP0 で利用できる。重み付けされたサイクル数は、指定された設定可能なカウンター IA32_PMCx で集計され、特定の要求の発生回数は別のプログラム可能なカウンターでカウントされる。

15.4.2 平均オフコア要求レイテンシーの測定

オフコア・トランザクション要求の平均レイテンシーは、MSR_OFFCORE_RSP0.[ビット 38] を設定し、MSR_OFFCORE_RSP0.[ビット 15:0] で任意の要求タイプを指定し、そして MSR_OFFCORE_RSP0.[ビット 37:16] を 0 に設定することで測定できる。

平均レイテンシーの測定が有効な場合、例えば IA32_PERFEVTSEL0.[ビット 15:0] = 0x01B7 とし、MSR_OFFCORE_RSP0 の値を選択すると、IA32_PMC0 は、指定されたトランザクション要求タイプのトランザクション要求の重み付けされたサイクルを集計する。同時に、IA32_PMC1 は、指定されたタイプのトランザクション要求の発生回数を集計する。

15.5 命令レイテンシー

この節では、Silvermont マイクロアーキテクチャーのポート・バインディングとレイテンシーに関する情報を提供する。MSROMによるデコーダーの支援が必要な命令には、「MSROM」列に「Y」マークを入れてある (よりデコード効率の高い代替手段があれば、使用は最小限に抑えるべきである)。

表 15-11 Silvermont マイクロアーキテクチャーの命令レイテンシーとスループット

命令	スループット	レイテンシー	MSROM
DisplayFamily_DisplayModel	06_37H、06_4AH、06_4DH	06_37H、06_4AH、06_4DH	06_37H、06_4AH、06_4DH
ADC/SBB r32, imm8	2	2	N
ADC/SBB r32, r32	2	2	N
ADD/AND/CMP/OR/SUB/XOR/TEST r32, r32	0.5	1	N
ADDPD/ADDSUBPD/MAXPD/MINPD/SUBPD xmm, xmm	2	4	N
ADDPS/ADDSD/ADDSS/ADDSUBPS/SUBPS/SUBSD/SUBSS xmm, xmm	1	3	N
MAXPS/MAXSD/MAXSS/MINPS/MINSD/MINSS xmm, xmm	1	3	N
ANDNPD/ANDNPS/ANDPD/ANDPS/ORPD/ORPS/XORPD/XORPS xmm, xmm	0.5	1	N
AESDEC/AESDECLAST/AESENC/AESENCLAST/AESIMC/AESKEYGEN xmm, xmm	5	8	Y
BLENDVPD/BLENDVPS xmm, xmm	4	4	Y
BSF/BSR r32, r32	10	10	Y
BSWAP r32	1	1	N
BT/BTC/BTR/BTS r32, r32	1	1	N
CBW	4	4	Y
CDQ/CLC/CMC	1	1	N
CMOVxx r32; r32	1	2	N
CMPPD xmm, xmm, imm	2	4	N
CMPSD/CMPPS/CMPSS xmm, xmm, imm	1	3	N
CMPXCHG r32, r32	6	6	Y
(U)COMISD/(U)COMISS xmm, xmm;	1	1	N
CPUID	60	60	Y
CRC32 r32, r32	1	3	N
CVTDQ2PD/CVTDQ2PS/CVTPD2DQ/CVTPD2PS xmm, xmm	2	5	N
CVT(T)PD2PI/CVT(T)PI2PD xmm, xmm	2	2	N

CVT(T)PS2DQ/CVTPS2PD xmm, xmm;	2	5	N
CVT(T)SD2SS/CVTSS2SD xmm, xmm	1	4	N
CVTSI2SD xmm, r32	1	1	N
DEC/INC r32	1	1	N
DIV r8	25	25	Y
DIV r16	26-30	26-30	Y
DIV r32	26-38	26-38	Y
DIV r64	38-123	38-123	Y
DIVPD	27-69	27-69	Y
DIVPS	27-39	27-39	Y
DIVSD	11-32	13-34	N
DIVSS	11-17	13-19	N
DPPD xmm, xmm, imm	8	12	Y
DPPS xmm, xmm, imm	12	15	Y
EMMS	10	10	Y
EXTRACTPS	4	4	Y
F2XM1	88	88	Y
FABS/FCHS/FCOM/FXCH	1	1	N
FADD/FSUB	1	3	N
FCOS	168	168	Y
FDECSTP/FINCSTP	0.5	1	N
FDIV	39	39	Y
FLDZ	277	277	Y
FMUL	1	5	N
FPATAN/FYL2X/FYL2XP1	296	296	Y
FPTAN/FSINCOS	281	281	Y
FRNDINT	25	25	Y
FSCALE	74	74	Y
FSIN	150	150	Y
FSQRT	40	40	Y
HADDPD/HSUBPD xmm, xmm	5	6	Y
HADDPS/HSUBPS xmm, xmm	6	6	Y
IDIV r8	34	34	Y
IDIV r16	35-40	35-40	Y
IDIV r32	35-47	35-47	Y
IDIV r64	49-135	49-135	Y
IMUL r32, r32	1	3	N

INSERTPS	1	1	N
MASKMOVDQU	5	5	Y
MOVAPD/MOVAPS/MOVDQA/MOVDQU/MOVUPD/MOVUPS xmm, xmm;	0.5	1	N
MOVD r32, xmm; MOVD xmm, r32	1	1	N
MOVDDUP/MOVHLPS/MOVLHPS/MOVSHDUP/MOVSLDUP	1	1	N
MOVDQ2Q/MOVQ/MOVQ2DQ	0.5	1	N
MOVSD/MOVSS xmm, xmm;	0.5	1	N
MPSADBW	5	7	Y
MULPD	7	4	Y
MULPS; MULSD	5	2	N
MULSS	4	1	N
NEG/NOT r32	0.5	1	N
PACKSSDW/WB xmm, xmm; PACKUSWB xmm, xmm	1	1	N
PABSB/D/W xmm, xmm	0.5	1	N
PADDB/D/W xmm, xmm; PSUBB/D/W xmm, xmm	0.5	1	N
PADDQ/PSUBQ/PCMPEQQ xmm, xmm	4	4	Y
PADDSB/W; PADDUSB/W; PSUBSB/W; PSUBUSB/W	0.5	1	N
PALIGNR xmm, xmm	1	1	N
PAND/PANDN/POR/PXOR xmm, xmm	0.5	1	N
PAVGB/W xmm, xmm	0.5	1	N
PBLENDVB xmm, xmm	0.5	1	N
PCLMULQDQ xmm, xmm, imm	10	10	Y
PCMPEQB/D/W xmm, xmm	0.5	1	N
PCMPESTRI xmm, xmm, imm	21	21	Y
PCMPESTRM xmm, xmm, imm	17	17	Y
PCMPGTB/D/W xmm, xmm	0.5	1	N
PCMPGTQ/PHMINPOSUW xmm, xmm	2	5	N
PCMPISTRI xmm, xmm, imm	17	17	Y
PCMPISTRM xmm, xmm, imm	13	13	Y
PEXTRB/WD r32, xmm, imm	4	4	Y
PINSRB/WD xmm, r32, imm	1	1	N
PHADDD/PHSUBD xmm, xmm	6	6	Y
PHADDW/PHADDSW xmm, xmm	9	9	Y
PHSUBW/PHSUBSW xmm, xmm	9	9	Y
PMADDUBSW/PMADDWD/PMULHRSW/PSADBW xmm, xmm	2	5	N
PMAXSB/W/D xmm, xmm; PMAXUB/W/D xmm, xmm	0.5	1	N
PMINSB/W/D xmm, xmm; PMINUB/W/D xmm, xmm	0.5	1	N

PMOVMSKB r32, xmm	1	1	N
PMOVSXBW/BD/BQ/WD/WQ/DQ xmm, xmm	1	1	N
PMOVZXBW/BD/BQ/WD/WQ/DQ xmm, xmm	1	1	N
PMULDQ/PMULUDQ xmm, xmm	2	5	N
PMULHUW/PMULHW/PMULLW xmm, xmm	2	5	N
PMULLD xmm, xmm	11	11	Y
POPCNT r32, r32	1	3	N
PSHUFB xmm, xmm	5	5	Y
PSHUFD xmm, mem, imm	1	1	N
PSHUFHW; PSHUFLW; PSHUFW	1	1	N
PSIGNB/D/W xmm, xmm	1	1	N
PSLLDQ/PSRLDQ xmm, imm; SHUFPD/SHUFPS	1	1	N
PSLLD/Q/W xmm, xmm	2	2	N
PSRAD/W xmm, imm;	1	1	N
PSRAD/W xmm, xmm;	2	2	N
PSRLD/Q/W xmm, imm;	1	1	N
PSRLD/Q/W xmm, xmm	2	2	N
PUNPCKHBW/DQ/WD; PUNPCKLBW/DQ/WD	1	1	N
PUNPCKHQDQ; PUNPCKLQDQ	1	1	N
RCPPS/RSQRTPS	8	9	Y
RCPSS/RSQRTSS	1	4	N
RDTSC	30	30	Y
ROUNDPD/PS	2	5	N
ROUNDSD/SS	1	4	N
ROL; ROR; SAL; SAR; SHL; SHR	1	1	N
SHLD/SHRD r32, r32, imm	2	2	N
SHLD/SHRD r32, r32, CL	4	4	Y
SHUFPD/SHUFPS xmm, xmm, imm	1	1	N
SQRTPD/PS	26	26	N
SQRTSD/SS	11	13	N
TEST r32, r32	0.5	1	N
UNPCKHPD; UNPCKHPS; UNPCKLPD, UNPCKLPS	1	1	N
XADD r32, r32	5	5	Y
XCHG r32, r32	5	5	Y