多倍長整数算法の Pentium による最適化

梅谷 武

作成：2001-01-23 更新：2005-04-20

Pentium の最適化技術を使って多倍長整数クラスの高速化を試みる。
IMS:20010123001; NDC:007.64; keywords:Pentium, 最適化;

目　　次

1　ゲームのルール

1.1　Pentium

Pentium という言葉は、文献 [X5] における Pentium Plain を意味するものとします。MMX, Pro,　,　,　というような修飾子が付かないのですが、最初に出荷された Pentium、あるいは各修飾子固有のアーキテクチャに依存しないという二通りの意味をもたせることにします。この文書ではこの Pentium の最適化技術について述べますが、これは修飾子が付いた Pentium にも有効です。ただ、各 Pentium 固有の高速化機能を使っていないわけですから、そのような高速化よりは性能は落ちることになります。

1.2　対向可能整数命令

486 では 5 ステージのパイプラインアーキテクチャが採用されましたが、Pentium ではそのパイプラインが 2 つ並列に動作します。スーパースケーラという言葉はこのことを指しているようです。各パイプラインでは整数命令が 1 クロックで実行できますから、1 クロックで 2 つの命令が実行されることになります。2 つのパイプには U パイプ・V パイプという名前が付けられていて、連続した 2 つの命令が並列に実行されることを対向 (Pairing) と呼びます。連続する 2 つの整数命令は必ずしも対向になるわけではありません。対向となるには、

- 対向可能整数命令であること
- 対向可能条件を満たすこと

が必要です。このことから Pentium における最適化を次のように表現することができます。

定義 1.1 (Pentium における最適化) *Pentium* における最適化とは、命令列を可能な限り対向している整数命令列に置き換えることによって実行クロック数を最小化することをいう。

このことによって、同じ x86 であっても 486 以前と Pentium の最適化とはまったく違った様相を示すことになります。対向可能な整数命令の表を以下に示しますが、これでわかるようには使える命令の数はかなり限られてしまいます。ただ、このことによって人手による最適化作業が楽になっている面もあります。

表 1　対向可能整数命令表

オペコード	オペランド	クロック	U	V
nop	-	1	○	○
mov	r/m, r/m/i	1	○	○
mov(*1)	m, acc	1	○	○
push	r/i	1	○	○
pop	r	1	○	○
lea	r/m	1	○	○
add/sub/and/or/xor	r, r/i	1	○	○
add/sub/and/or/xor	r, m	2	○	○
add/sub/and/or/xor	m, r/i	3	○	○
adc/sbb	r, r/i	1	○	-
adc/sbb	r, m	2	○	-
adc/sbb	m, r/i	3	○	-
cmp	r, r/i	1	○	○
cmp	m, r/i	2	○	○
test	r, r	1	○	○
test	m, r	2	○	○
test	acc, i	1	○	○
inc/dec	r	1	○	○
inc/dec	m	3	○	○
shr/shl/sar/sal	r, i	1	○	-
shr/shl/sar/sal	m, i	3	○	-
ror/rol/rcr/rcl	r/m, 1	1/3	○	-
jmp/call(*2)	short/near	1	-	○
jcc(*2)	short/near	1/4/5/6	-	○

r=レジスタ, m=メモリ, i=即値, acc=アキュムレータ, クロックは最小値を意味する。

1) acc に書き込んだ場合と同一条件

2) 分岐予測を参照のこと

1.3 対向可能条件

連続する 2 つの命令は以下の条件が満たされたとき対向可能となります。

1. 最初の命令は U パイプで、二番目の命令は V パイプで対向可能であること。
2. 二番目の命令は最初の命令が書くレジスタを読み書きしないこと。
3. 規則 2 で部分レジスタはレジスタ全体として扱われる。
4. 規則 2 と 3 にかかわらず、フラグレジスタの一部に書き込む二つの命令は対向にできる。
5. 規則 2 にかかわらず、フラグに書き込む命令と条件ジャンプは対向にできる。
6. 次の命令の組合せは、両方がスタックポインタを変更するという事実にもかかわらず、対向にできる。
   PUSH + PUSH, PUSH + CALL, POP + POP
7. プリフィックスつき命令は、near 条件ジャンプを除いて U パイプでのみ実行可能である。
8. 変位と即値の両方を持たないこと。

1.4 番地生成インターロック

番地が一つ前のクロックサイクルで実行された命令の結果に依存する場合は、番地の計算のために 1 クロックサイクル余分に待たなければなりません。これは番地生成インターロックと呼ばれます。

1.5 不完全対向条件

対向する 2 つの命令が同時に実行されなかったり、時間的に一部だけオーバーラップしたりする状況があります。これを不完全な対向と呼びます。不完全な対向の両方の命令の実行が完了しないと、引き続く命令の実行は始まりません。以下に不完全な対向となる条件を示します。

1. 二番目の命令が番地生成インターロックを生ずるとき。
2. 2 つの命令はメモリの同じ DWORD を同時に読み書きするとき。
3. 規則 2 は 2 つの番地のビット 2〜4 が同じである場合に拡張される (キャッシュバンク競合)。DWORD の番地に対しては、これは 2 つの番地の差が 32 で割り切れてはならないことを意味する。
4. read/modify/write 命令が read/modify 命令または read/modify/write 命令と対向になるとき
5. 対向する 2 つの命令が両方とも、キャッシュミス、ミスアラインメント、または分岐予測ミスによって余分な時間がかかるとき、その対向は各命令単独よりは時間がかかるが 2 つの和よりは少ない。

1.6 分岐予測

Pentium は分岐予測機構を持ち、正しく予測できたときには、ジャンプ命令を 1 クロックで実行できます。失敗した場合は、通常は 3 クロック、対向で実行される条件ジャンプの場合は、4 クロックの追加クロックが必要になります。したがって、なるべく分岐予測が成功するような命令列にする必要があります。このための原則を次に示します。

1. 条件ジャンプはなるべくジャンプするように配置する。
2. 予測ミスの起きやすい分岐命令の後に実行される命令ペアに、分岐命令を入れないようにする。
3. コードの重要な部分では、サブルーチンを異なる場所から交互に呼んだりしないようにする。

1.7 LEA 命令

LEA 命令は、シフトと二回の加算とデータの移動を 1 クロックの対向可能命令だけでできます。

参考文献

x86

[X1] 蒲池 輝尚, *“はじめて読む 486”*, アスキー, 1994

[X2] インテル, *“Pentium プロセッサ・ファミリー アーキテクチャとプログラミング”*, インテル, 1993

[X3] 藤波 順久, アセンブラでの高速化

[X4] A. Fog(藤波 順久訳), *HOW TO OPTIMIZE FOR THE PENTIUM PROCESSOR*