Microsoft* Windows* 10 における新しい命令セットの利用

この記事は、インテル® デベロッパー・ゾーンに公開されている「Follow-Up: How does Microsoft Windows 10 Use New Instruction Sets?」の日本語参考訳です。

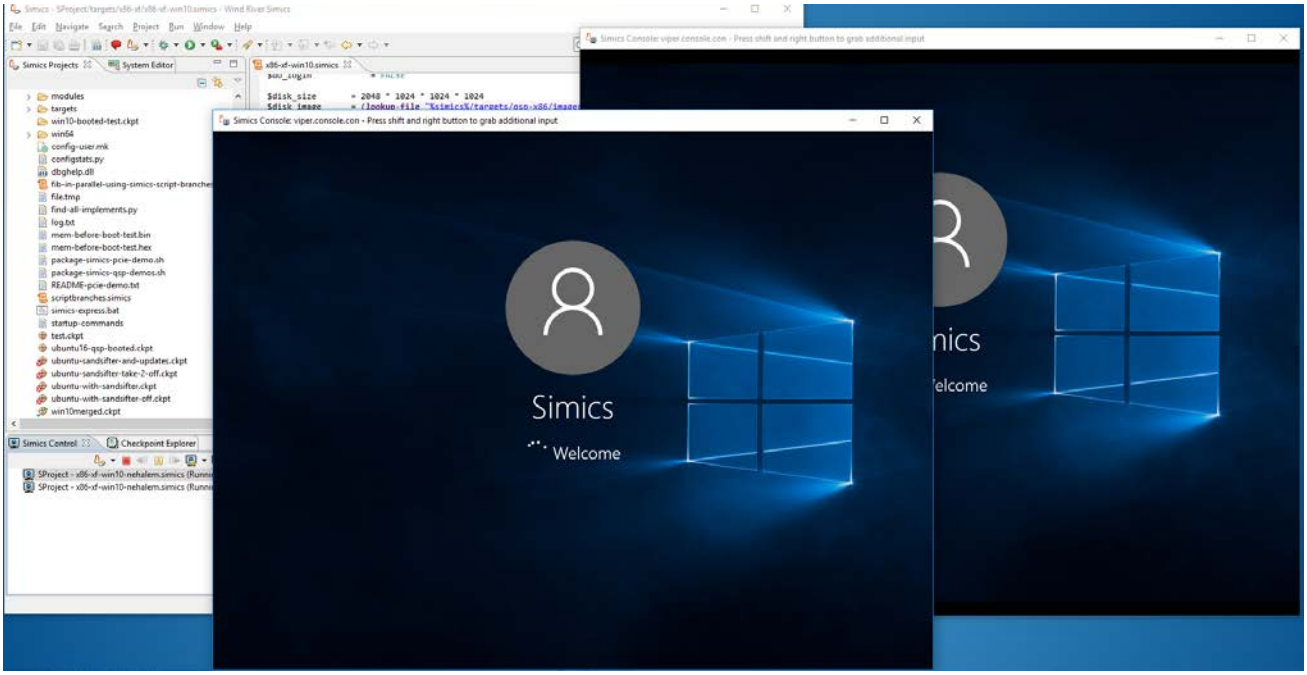
以前のブログ「ソフトウェアは実際に新しい命令セットを使用しているのか?」(英語) では、いくつかの異なる Linux* 環境で使用されている命令の種類と、プロセッサー・タイプの変更がそれらの環境にどのように影響するかに注目しました。その続きとして、ここでは Microsoft* Windows* 10 について検証します。さまざまな世代のプロセッサーで Windows* 10 の動作を調べ、それを Ubuntu* 16 と比較します。また、Ubuntu* のさまざまな利用シナリオにおける命令の使用についても考察します。

テスト環境のセットアップ

以前と同様に、Wind River* Simics* 仮想プラットフォーム・ツールの「汎用 PC」プラットフォームを使用して、2 つの異なるプロセッサー・モデルで実行します。1 つは、第 1 世代インテル® Core™ i7 プロセッサー (開発コード名 Nehalem) で、もう一方は、第 6 世代インテル® Core™ i7 プロセッサー (開発コード名 Skylake) です。

これらの異なるモデルを使用して、Windows* 10 イメージ (ビルド 1511) を起動します。前回と同様に、約 60 秒間でアイドル状態のデスクトップが起動されました。

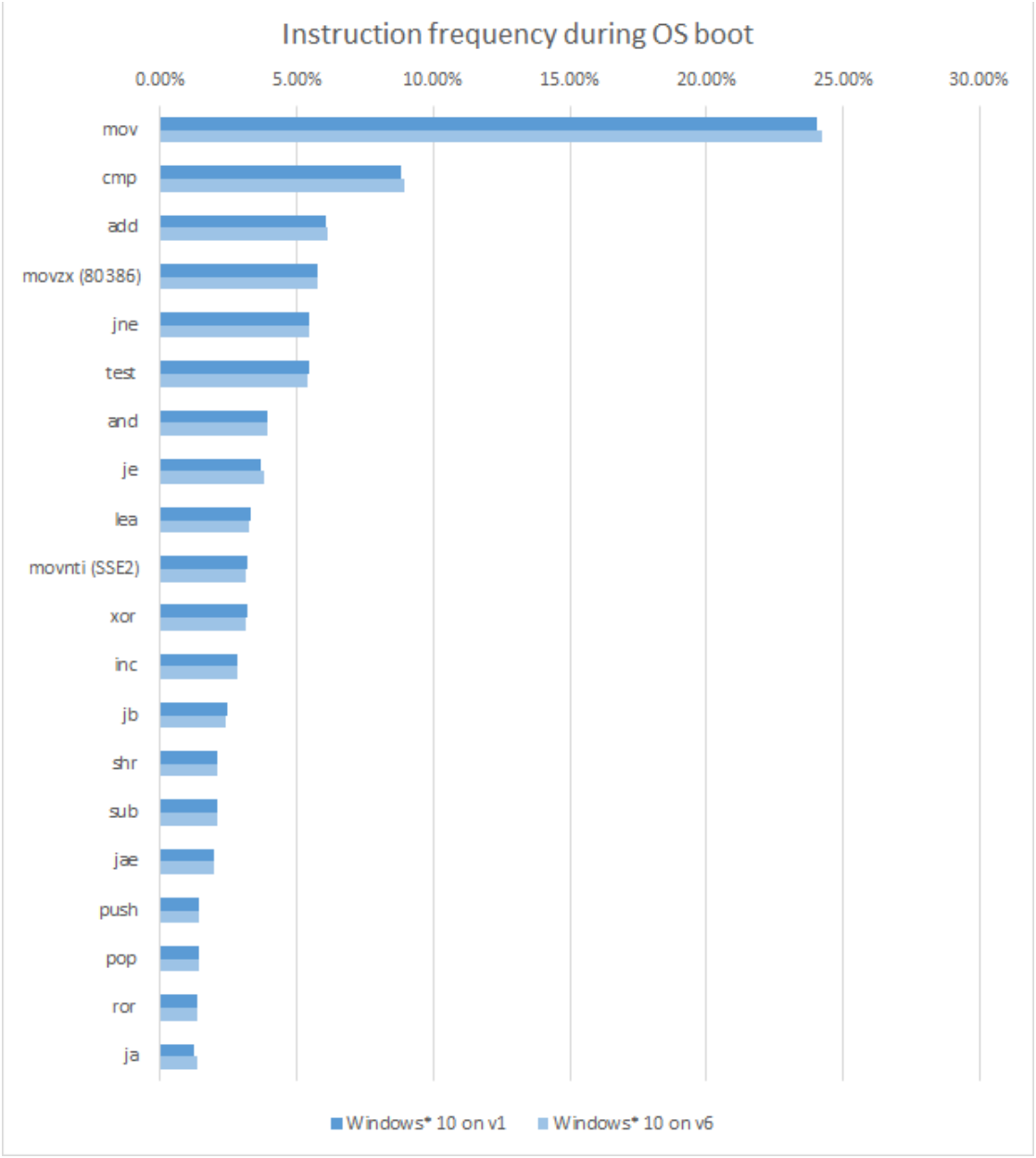
以下は、統計を収集するためいくつかの Windows* 10 ターゲットを起動しているラップトップのスクリーンショットです。



起動中、Simics* ツール環境で使用されている命令の種類について統計を収集しました。前回と同様に、ニーモニックごとに命令の使用回数をカウントしました。さらに、以下に示すように、異なる条件に基づいて命令ストリームを検証するため、数回実行しました。

複数の世代にわたる命令

最初に、起動中に動的命令の 1 % 以上を占めるすべての命令について調べました。以下のグラフに結果を示します。



Windows* のデータと以前のブログ (英語) の Ubuntu* 16 のデータは非常によく似ています。プロセッサーの世代によりわずかな違いがありますが、ほとんどの場合、プロセッサーに関係なく同じコードが実行されます。

これは、Ubuntu* や Windows* などの一般的な汎用オペレーティング・システムでは想定外のことではありません。以前のブログでは、Yocto* Linux* ビルドにおいてプロセッサーの世代間の違いが最も顕著でした。Yocto* では、Linux* を自身でビルドすることができます。通常、広範なユーザーベースをサポートする必要がないため、新しい命令セットを使用するコードを含めてより積極的に使用することができます。幅広いユーザーベースと非常に多くのユーザー向けに安定した動作を提供するという共通の目標を持つ Ubuntu* や Windows* 10 では、ハードウェアの世代間の違いが多すぎると、テストと品質管理が難しくなります。自分で Linux* をビルドする場合とは異なり、単一のシステム向けに積極的に最適化しないことは理にかなっています。

グラフから、最もよく使用されてる命令は、移動、比較、ジャンプ、基本的な算術演算であることが分かります。これは、Linux* のテスト結果と非常によく似ていますが、使用された命令はやや異なります。

Linux* との比較

オペレーティング・システムの変更により、コードのビルドに使用するコンパイラー (Linux* の gcc から Windows* の Microsoft* コンパイラーへ) と、関数やオペレーティング・システムの呼び出し規則が変わります。これらはすべてコンパイラーの命令選択プロセスに影響し、その結果、実際に使用される命令がワークロード間で異なる可能性があります。実際、いくつかの命令が、1 つのワークロードでのみ独特な使われ方をしているのを見つけました。これに関して、いくつかの例があります。

Windows* 10 は、LEAVE または ENTER 命令を使用しません。以前のブログ (英語) で述べたとおり、古い Linux* 2.6 Busybox* では LEAVE が広く使用されていましたが、より新しい Linux* ディストリビューションでは使用されていません。Windows* 10 は新しいソフトウェア・スタックであるため、LEAVE 命令を使用していないことは当然と言えるでしょう。

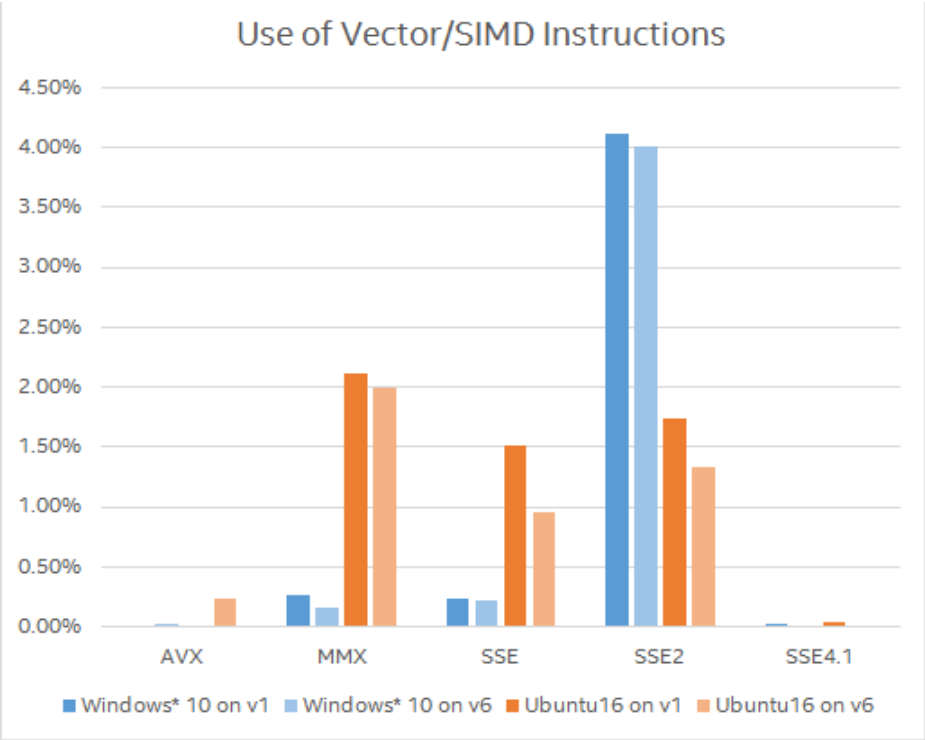
Windows* 10 では使用され、Linux* ではどの環境でも使用されないいくつかの命令があります。その最も重要なものが、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2) の MOVNTI 命令です。Windows* 上の命令の 3%。さらに、上記の一般的な命令のほかに、Windows* 10 は Linux* では使用されないいくつかのユニークなベクトル命令を使用します: PADDW、PSRLW、PMOVZXBW、PUNPCKHQDQ、PSUBW、PMADDWD。豊富なベクトル命令セットがあることを考えれば、これは驚くべきことではありません。

8086 命令セットの CMC (補数キャリーフラグ) 命令も、Windows* では使用されますが、Linux* では使用されません。80386 命令セットの BSR (逆方向のビットスキャン) も同様です。これらの命令は一般的ではありませんが (0.01% 未満)、Linux* の起動で全く使用されていないのは興味深いです。

これは、オペレーティング・システムの起動プロセスに関してのみ言えることです。アプリケーション・ソフトウェアでは結果が異なるでしょう。実際、Linux* でいくつかのほかのテストを行ったところ、以下に示すように全く異なる結果になりました。

ベクトルと SIMD

ベクトル命令は、起動中にあまり使用されません。第 1 世代と第 6 世代のプロセッサーの違いは、Windows* ではそれほどではなく、Linux* のほうが顕著です。しかし、Windows* 10 と Ubuntu* 16 を比較すると興味深い結果になります。



全体として、Windows* と Ubuntu* で起動時に使用されるベクトル命令の割合はほぼ同じ (約 5%) です。しかし、ベクトル命令の分布は異なります。Windows* ではインテル® SSE2 命令がよく使用され、Ubuntu* ではインテル® MMX® 命令が使用されています。また、Windows* では、世代間で使用される命令にほとんど違いがありません。第 6 世代のプロセッサーでインテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) がわずかに使用されているくらいです。

その他の例

この命令ニーモニックの調査は、仮想プラットフォームでインストルメンテーションを使用してソフトウェアを実行することで観察できるものを示す単純な例です。有益な情報が得られるだけでなく、さまざまなものを観察し、カウントすることができます。Simics* ユーザーは、仮想プラットフォームの一部であれば任意のものを収集するツールをプログラムできます。

例えば、以下は第 6 世代のプロセッサーで Windows* 10 を起動中の命令サイズの分布です。

Size	Count	Percentage
1	1628069423	2.43%
2	14095489620	21.04%
3	20701199506	30.91%
4	12645794342	18.88%
5	7513969415	11.22%
6	5684886517	8.49%
7	2741747372	4.09%
8	1438923660	2.15%
9	241637983	0.36%
10	266236763	0.40%
11	18008375	0.03%
12	2035468	0.00%
13	6	0.00%
14	19463	0.00%

平均サイズは、実行された命令あたり約 3.73 バイトになります。これは、コードのサイズとは関係がありません。キャッシュシステムとプロセッサーのデコーダーの負荷を示すものです。インテル® アーキテクチャー (IA) は、典型的な可変長命令セットを採用しています。上記の表でも、1 バイトから 14 バイトまでさまざまなサイズの命令が確認できます。非常に大きなサイズの命令はまれです。

命令を考察する別の方法として、オペランド型とオペコードで分類できます。これは、上記のニーモニックよりも細かく命令を分類します。例えば、MOV 命令には、最も一般的なバリエーションとして次の 20 命令があります。

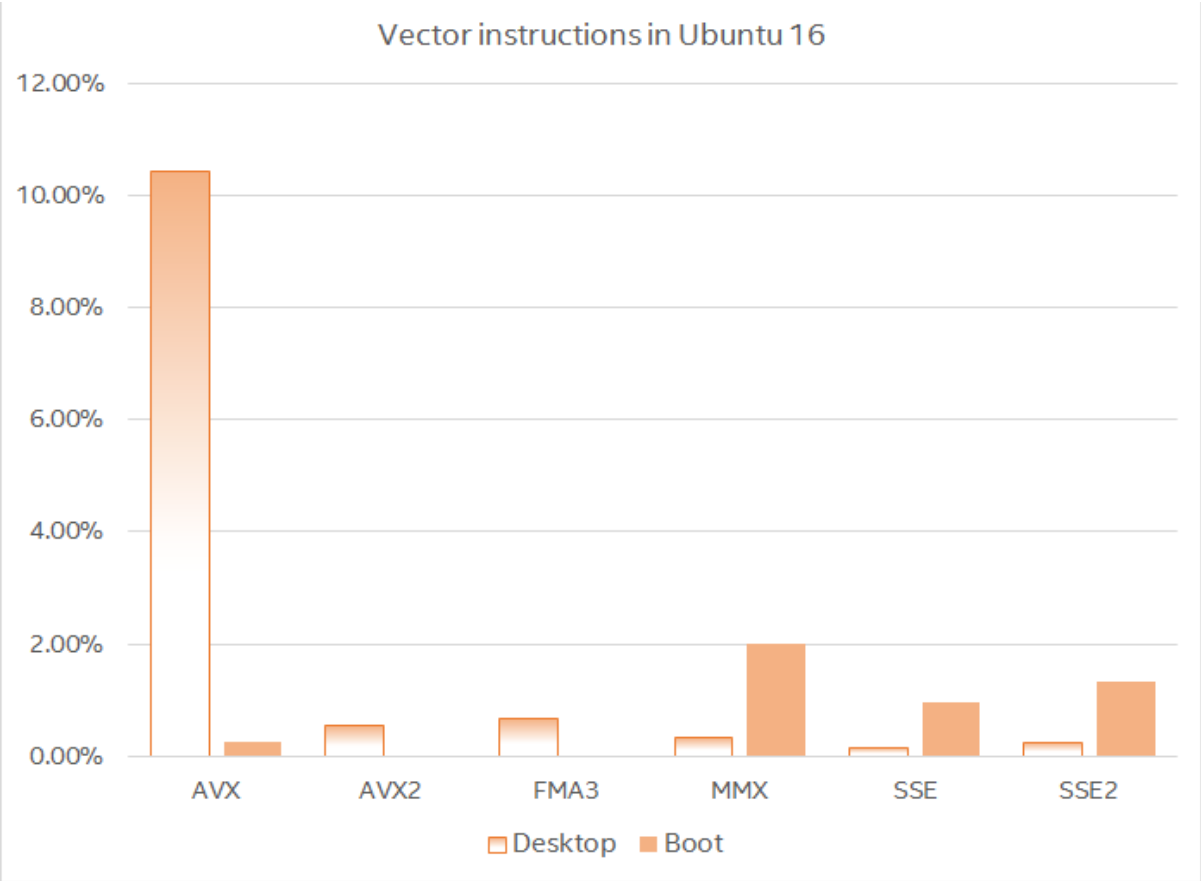
Variant	Percentage
mov r32,r32	5.25
mov r64,qword ptr [r64+imm]	3.81
mov r64,r64	2.78
mov qword ptr [r64+imm],r64	2.43
mov r32,imm	1.44
mov r32,dword ptr [r64+imm]	1.34
mov dword ptr [r64+imm],r32	0.95
mov r64,qword ptr [r64]	0.89
mov qword ptr [r64],r64	0.37
mov r64,imm	0.35
mov r32,dword ptr [r64+r64*imm]	0.33
mov r32,dword ptr [r64]	0.33
mov r64,qword ptr [r64-imm]	0.26
mov qword ptr [r64-imm],r64	0.25
mov byte ptr [r64+imm],r8	0.23
mov word ptr [r64+imm],r16	0.18
mov byte ptr [r64],r8	0.17
mov r64,qword ptr [r64+r64*imm]	0.17
mov dword ptr [r64],r32	0.16
mov r64,qword ptr [rip+imm]	0.16

このほかにも、特定のアドレスモードを使用するものが多数あります。これは典型的なロングテール型の分布です。最も一般的なモードがすべての MOV 操作の大半を占めていますが、より複雑なモードもある程度は使用されています。

ここでは、すべてのサイズの MOV 操作をリストしています。64 ビットの Windows* オペレーティング・システムを 64 ビットのプロセッサーで実行しているからと言って、すべての操作のサイズが実際に 64 ビットであるとは限りません。バイト (8 ビット)、ワード (16 ビット)、ダブルワード (32 ビット) 操作も使用されます。32 ビットの操作は、64 ビットの操作と同じくらい一般的です。

Linux* デスクトップでのベクトルと SIMD

これらの測定結果について同僚と話し合ったところ、一般的なベクトル命令とインテル® AVX 命令、そしてそれらが使用されるワークロードにどのように依存するかという疑問が浮かびました。オペレーティング・システムの起動では、ごくわずかな暗号化操作や高度に最適化されたメモリーコピー操作ぐらいにしか、それらの命令は使用されないでしょう。しかし、同僚はシステムをインタラクティブに使用しているときに、いくつかのほかの動作に遭遇しました。このため、もう 1 つテストを行うことになりました。第 6 世代のプロセッサーで Ubuntu* を起動後に、ターミナルを開いて新しい Firefox* プロセスを開始しました。



上記のグラフから、デスクトップ・アクティビティーは、新しいインテル® AVX2 命令と FMA3 (英語) 命令を含むインテル® AVX 命令を広く使用しています。実際、ベクトル命令は、実行されたすべての命令 (ユーザーレベル・コードやグラフィックス・サブシステムだけでなく、マシン全体のすべての命令) の 12% を超えています。

まとめ

ここでは、異なるプロセッサーとワークロードでさまざまな種類の命令を実行して、その結果をグラフや数値を用いて示しました。私のようなコンピューター・アーキテクチャー・オタク向けのデータと言えます。ここで最も興味深い点は、Simics* とそのインストルメンテーション機能を使用した数値の収集方法です。Simics* は、ほとんどのシステムをシミュレーションでき、非干渉的に調査とデバッグが可能です。ここで紹介したような命令統計の収集は、プロセッサー設計者、ソフトウェア・エンジニア、研究者、学生に役立つ情報を提供します。

Simics* は、学術機関に無料で提供されており、コンピューター・アーキテクチャー、オペレーティング・システム、ネットワーキング、組込みシステム、シミュレーション、低水準プログラミングを含む非常に広範な分野向けのツールです。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。