How to detect New Instruction support in the 4th generation Intel® Core™ processor family

The 4th generation Intel® Core™ processor family (codenamed Haswell) introduces support for many new instructions that are specifically designed to provide better performance to a broad range of applications such as: media, gaming, data processing, hashing, cryptography, etc. The new instructions can be divided into the following categories:

- Intel® Advanced Vector Extensions 2 (Intel® AVX2)
- Fused Multiply Add (FMA)
- Bit Manipulation New Instructions (BMI)
- MOVBE instruction (previously supported by the Intel® Atom™ processor)
- Intel® Transactional Synchronization Extensions (Intel® TSX) (available in some models)

The details of these instructions can be found in Intel® 64 and IA-32 Architectures Software Developer Manuals and Intel® Advanced Vector Extensions Programming Reference manual.

In order to correctly use the new instructions and avoid runtime crashes, applications must properly detect hardware support for the new instructions using CPUID checks. It is important to understand that a new instruction is supported on a particular processor only if the corresponding CPUID feature flag is set. Applications must not assume support of any instruction set extension simply based on, for example, checking a CPU model or family and must instead always check for *_all_* the feature CPUID bits of the instructions being used.

Software developers can take advantage of the new instructions via writing assembly code, using intrinsic functions, or relying on compiler automatic code generation. In the latter case, it is crucial to understand what instructions the compiler(s) can generate with given switches and implement proper CPUID feature checks accordingly.

Generally, compilers and libraries generating code for 4th generation Intel Core processors are expected and allowed to use all the instructions listed above, with the exception of Intel TSX. Below is the complete list of CPUID flags that generally must be checked:

```
CPUID.(EAX=01H, ECX=0H):ECX.FMA[bit 12]==1  &&
CPUID.(EAX=07H, ECX=0H):EBX.AVX2[bit 5]==1  &&
CPUID.(EAX=07H, ECX=0H):EBX.BMI1[bit 3]==1   &&
CPUID.(EAX=07H, ECX=0H):EBX.BMI2[bit 8]==1   &&
CPUID.(EAX=80000001H):ECX.LZCNT[bit 5]==1    &&
CPUID.(EAX=01H, ECX=0H):ECX.MOVBE[bit 22]==1
```

Note: Applications using instructions from the RTM subset of Intel TSX extension need to guard the code by checking the CPUID.(EAX=07H, ECX=0H).EBX.RTM[bit 11]==1. Applications can also, but are not required to, check CPUID.(EAX=07H, ECX=0H).EBX.HLE[bit 4]==1 for HLE, because legacy processors ignore HLE hints.

For example Intel® Composer XE 2013 can automatically generate *all* the new instructions guarded by the CPUID features in the above list, using -Q**a**xCORE-AVX2 and -QxCORE-AVX2 switches on Microsoft Windows* (on Linux*: -axCORE-AVX2 and -xCORE-AVX2). The compiler switch -[Q]**a**xCORE-AVX2 generates automatic CPUID check and dispatch to the code using new instructions, while the -[Q]xCORE-AVX2 switch *assumes* the new instructions are supported and thus requires a manual implementation of the CPUID check for *all* the features in the list above. Microsoft Visual C++* 2012 compiler supports these new instructions via intrinsics as well as 32-bit inline assembler, while the GCC compiler supports both auto-generation and intrinsics with -march=core-avx2 switch starting with version 4.7, thus requiring a check of the *complete* list of CPUID features above, whenever such code is called.

Additionally, libraries such as Intel® Integrated Performance Primitives (Intel® IPP) beginning with version 7.1 may also use these new Instructions. In the case of Intel IPP, two types of interfaces are available: an automatically dispatched interface is the default, and a CPU-specific interface available via prefixes like 'h9_' (32-bit) or 'l9_' (64-bit). In the case of functions optimized for the 4th generation Intel Core processor family, applications must check for the support of *all* the features in the list above before calling these functions.

And finally, new instructions using VEX prefixes and operating on vector YMM/XMM registers continue to require checking for OS support of YMM state before using, the same check as for Intel AVX instructions.

Below is a code example you can use to detect the support of new instructions:

```
#if defined(__INTEL_COMPILER) && (__INTEL_COMPILER >= 1300)

#include <immintrin.h>

int check_4th_gen_intel_core_features()
{
    const int the_4th_gen_features =
        (_FEATURE_AVX2 | _FEATURE_FMA | _FEATURE_BMI | _FEATURE_LZCNT | _FEATURE_MOVBE);
    return _may_i_use_cpu_feature( the_4th_gen_features );
}

#else /* non-Intel compiler */

#include <stdint.h>
#if defined(_MSC_VER)
# include <intrin.h>
#endif

void run_cpuid(uint32_t eax, uint32_t ecx, uint32_t* abcd)
{
#if defined(_MSC_VER)
    __cpuidex(abcd, eax, ecx);
#else
    uint32_t ebx, edx;
# if defined( __i386__ ) && defined ( __PIC__ )
     /* in case of PIC under 32-bit EBX cannot be clobbered */
    __asm__ ( "movl %%ebx, %%edi \n\t cpuid \n\t xchgl %%ebx, %%edi" : "=D" (ebx),
# else
    __asm__ ( "cpuid" : "+b" (ebx),
```

```
# endif
              "+a" (eax), "+c" (ecx), "=d" (edx) );
    abcd[0] = eax; abcd[1] = ebx; abcd[2] = ecx; abcd[3] = edx;
#endif
}

int check_xcr0_ymm()
{
    uint32_t xcr0;
#if defined(_MSC_VER)
    xcr0 = (uint32_t)_xgetbv(0);  /* min VS2010 SP1 compiler is required */
#else
    __asm__ ("xgetbv" : "=a" (xcr0) : "c" (0) : "%edx" );
#endif
    return ((xcr0 & 6) == 6); /* checking if xmm and ymm state are enabled in XCR0 */
}


int check_4th_gen_intel_core_features()
{
    uint32_t abcd[4];
    uint32_t fma_movbe_osxsave_mask = ((1 << 12) | (1 << 22) | (1 << 27));
    uint32_t avx2_bmi12_mask = (1 << 5) | (1 << 3) | (1 << 8);

    /* CPUID.(EAX=01H, ECX=0H):ECX.FMA[bit 12]==1   &&
       CPUID.(EAX=01H, ECX=0H):ECX.MOVBE[bit 22]==1 &&
       CPUID.(EAX=01H, ECX=0H):ECX.OSXSAVE[bit 27]==1 */
    run_cpuid( 1, 0, abcd );
    if ( (abcd[2] & fma_movbe_osxsave_mask) != fma_movbe_osxsave_mask )
        return 0;

    if ( ! check_xcr0_ymm() )
        return 0;

    /*  CPUID.(EAX=07H, ECX=0H):EBX.AVX2[bit 5]==1  &&
        CPUID.(EAX=07H, ECX=0H):EBX.BMI1[bit 3]==1  &&
        CPUID.(EAX=07H, ECX=0H):EBX.BMI2[bit 8]==1  */
    run_cpuid( 7, 0, abcd );
    if ( (abcd[1] & avx2_bmi12_mask) != avx2_bmi12_mask )
        return 0;

    /* CPUID.(EAX=80000001H):ECX.LZCNT[bit 5]==1 */
    run_cpuid( 0x80000001, 0, abcd );
    if ( (abcd[2] & (1 << 5)) == 0)
        return 0;

    return 1;
}

#endif /* non-Intel compiler */


static int can_use_intel_core_4th_gen_features()
{
    static int the_4th_gen_features_available = -1;
    /* test is performed once */
    if (the_4th_gen_features_available < 0 )
        the_4th_gen_features_available = check_4th_gen_intel_core_features();
```

```
    return the_4th_gen_features_available;
}

#include <stdio.h>

int main(int argc, char** argv)
{
    if ( can_use_intel_core_4th_gen_features() )
        printf("This CPU supports ISA extensions introduced in Haswell\n");
    else
        printf("This CPU does not support all ISA extensions introduced in Haswell\n");

    return 1;
}
```

Notices