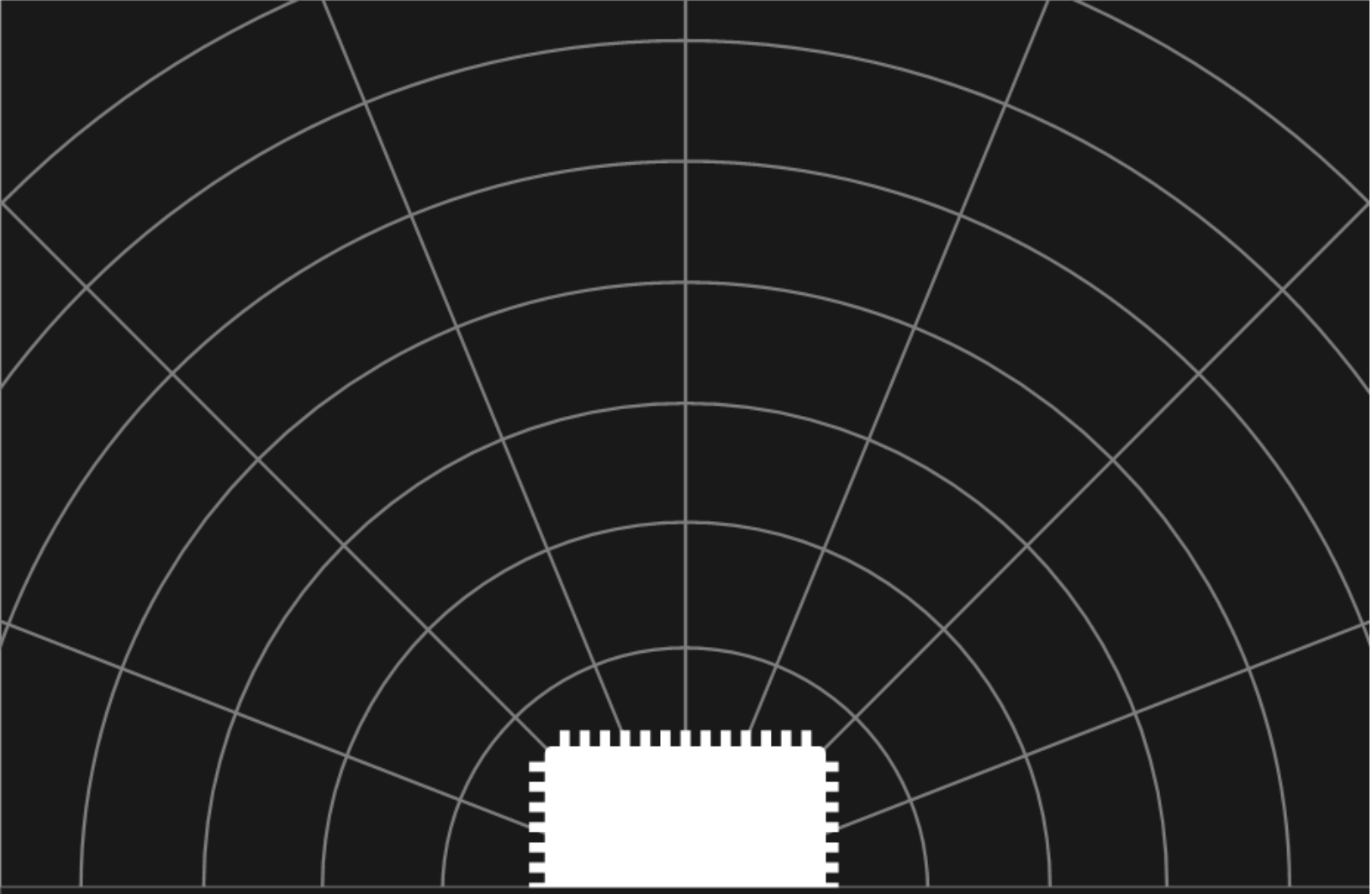

ddcpuid Technical Manual

ddcpuid Manual

Exploring your x86 processor

Version 0.21.1, October 2023

Table of Contents

Index of Tables

Table of Figures

Preface

The ddcpuid Manual was written by dd86k <dd@dax.moe> (https://github.com/dd86k) for the ddcpuid project.

This is a technical reference meant to better understand x86 features and technologies referenced in the ddcpuid software. It is also useful when combined with the Intel and AMD reference manuals, optimization guides, and future instruction references.

It's best aimed for anyone interested in their processor technologies and capabilities, the x86 instruction set architecture, and the ddcpuid project. For a better understanding of some advanced feature, it assumes that you are at minimum familiar with the x86 and AMD64 instruction set architectures.

So far, it has seen 153 written revisions so far. This revision was released October 13, 2023 and contains 23851 words. **This document is meant to be distributed freely. If you bought a copy, you got swindled!** This document is licensed under the *Creative Common Attribution-NoDerivatives 4.0 International (CC-BY-ND 4.0)* license, which permits sharing this document without alterations and allows commercial implications (such as referring to this manual for commercial purposes). See https://creativecommons.org/licenses/by-nd/4.0/ for more information.

The software project is available on Github (https://github.com/dd86k/ddcpuid) and my own repository (https://git.dd86k.space/dd86k/ddcpuid). A copy of the project license for the ddcpuid software is found at Software License.

Notations Used

Bit ranges

The format `[a:b]` is used to denote an inclusive range of bits with a starting index of zero. The value '`b`' is the starting position and the value '`a`' is the ending position.

Examples:

- [1:0] selects bits at position 0 and 1.
- [10:5] selects all bits from position 5 up to 10.

Hexadecimal numbers

Hexadecimal numbers with a notation base of 16, typically using the uppercase characters for values 10 to 15. These can come in two styles: Either must have a 'h' suffix or a '0x' prefix. These numbers may contain an underscore character for clarification for values higher than 0xffff (65535) and may be prefixed with zeroes.

Examples:

- 7_0040h
- 01h
- 0x400
- 0xff

Binary numbers

Binary numbers uses a base notation of 2. These numbers must have a mandatory 'b' suffix. These numbers may be prefixed with zeroes.

Examples:

- 10b
- 0110b

Binary sizes

This document uses the ISO binary notation (KiB, MiB, GiB, etc.) as per IEC 60027-2 A.2 and ISO/IEC 80000-13:2008 (storage capacity, storage size [M]).

While typically the ISO 'iB' suffix notation denote a base of 1024 and the ISO 'B' suffix notation denote a base of 1000, **all binary sizes in this document uses the base 1024 notation regardless of suffix**.

Examples:

- 4B
- 8KiB
- 32 MiB
- 2 TiB

CPUID bits

CPUID data are described using the format **CPUID.Xh(ECX=Yh).REG[pos]** for familiarity with the Intel syntax used in the Intel reference manuals.

- **CPUID** stands for the CPUID instruction (optional);
- **Xh** is the EAX leaf value in hexadecimal;
- **(ECX=Yh)** is the ECX sub-leaf value in hexadecimal (optional);
- **REG** is the selected register (EAX, EBX, ECX, or EDX);
- And **pos** is the bit offset index position within the register. If omitted, the entirely of the register value is used.
  Examples:
  - CPUID.1h.EAX[3:0]: Processor identifier stepping number.
  - CPUID.7h(ECX=0h).EDX[31]: Speculative Store Bypass Disable [SSDB].
  - CPUID.8000_0000h.EAX: Maximum extended leaf value.

Reference Literature

This project is based on information provided from the following documents listed below.

- Intel Corporation
  - *Intel® 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4* (June 2021)
  - *Intel® Architecture Instruction Set Extensions Programming Reference* (August 2015)
  - *Intel® Architecture Instruction Set Extensions and Future Features Programming Reference* (May 2021)
  - *Intel Speculative Execution Side Channel Mitigations Revision 3.0* (May 2018)
  - *Intel® 5-Level Paging and 5-Level EPT White Paper Revision 1.1* (May 2017)
  - *Intel SGX Product Brief: Enhanced Security for Applications and Data In-use* (2019)
  - *Intel® 64 and IA-32 Architectures Optimization Reference Manual* (May 2020)
  - *Intel® 64 Architecture Processor Topology Enumeration* (January 2018)
  - Marius Cornea, *Intel® AVX-512 Instructions and Their Use in the Implementation of Math Functions* (2015)
  - *Intel® Software Guard Extensions (Intel® SGX) SGX2* (June 2016)
- Advanced Micro Devices Incorporated
  - *AMD64 Architecture Programmer's Manual: Volumes 1-5* (March 2021)
  - *AMD64 Technology Indirect Branch Control Extension Revision 4.10.18 White Paper* (April 2018)
  - *AMD64 Technology Speculative Store Bypass Disable 5.21.18 White Paper* (May 2018)
  - *Open-Source Register Reference For AMD Family 17h Processors Models 00h-2Fh* (July 2018)
- Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard, *Malware Guard Extension: Using SGX to Conceal Cache Attacks (Extended Version)* (Feb 2017) Graz University of Technology. https://arxiv.org/pdf/1702.08719.pdf

- Vladimir Kiriansky, Carl Waldspurger, *Speculative Buffer Overflows: Attacks and Defenses* (July 2018) https://people.csail.mit.edu/vlk/spectre11.pdf
- Michael Schwarz, Samuel Weiser, Daniel Gruss, *Practical Enclave Malware with Intel SGX* (Feb 2019) Graz University of Technology. https://regmedia.co.uk/2019/02/12/sgxmalware.pdf

More sources are provided within footnotes.

Chapter 1
Introduction

The ddcpuid project is a x86 processor information tool that provides detailed processor information. By relying on the CPUID instruction, introduced in the Intel i486SL and Intel Pentium processors, the ddcpuid program is able to identify what features are available on the processor.

1.1 History

My ddcpuid utility was created out of my curiosity for system programming and assembler languages. One day, in 2016, I was tired of C#, so I decided to pick a new programming language to learn, and I stumbled on the D programming language's x86 inline assembler feature.[1]

Once I installed the Digital Mars D compiler on my system, I quickly started making examples using the CPUID instruction, and the project quickly grew since. I then decided to make it a full project by February 26, 2016.

Then, on June 26, 2016, the first git commit was pushed on Github.

On February 1st, 2017, ddcpuid got 64-bit compilation support.

On December 11, 2017, ddcpuid was rewritten to be compiled with D’s betterC feature.

On June 15, 2018, cache information was added.

Finally, on June 16, 2018, the first version of the manual was released.

1 https://dlang.org/spec/iasm.html

1.2 Processor Vendor Support

The ddcpuid software currently supports Intel and AMD processors. Other vendors are unsupported as documentation is unavailable for them.

The following table indicates possible values for the main processor vendor string.

	String	Vendor	Supported?
Current vendors	`"GenuineIntel"`	Intel Corporation	✔
	`"AuthenticAMD"`	Advanced Micro Devices Inc.	✔
	`"VIA VIA VIA "`	VIA Technologies Inc.	
	`"CentaurHauls"`	Centaur Technology	
	`"GenuineTMx86"`	Transmeta Corporation	
	`"CyrixInstead"`	Cyrix Corporation	
	`"NexGenDriven"`	NexGen	
	`"UMC UMC UMC "`	United Microelectronics Corporation	
	`"SiS SiS SiS "`	Silicon Integrated Systems	
	`"Geode by NSC"`	National Semiconductor	
	`"RiseRiseRise"`	Rise Technology	
	`"Vortex86 SoC"`	DM&P Electronics Inc.	
	`"HygonGenuine"`	Hygon Dhyana	
Obsolete vendors	`"AMDisbetter!"`	Legacy AMD string	
	`"TransmetaCPU"`	Legacy Transmeta string	
Paravirtualization vendors	`"VMwareVMware"`	VMware Inc.	
	`"bhyve bhyve "`	BHyVe ("Bee Hive"), FreeBSD	
	`"XenVMMXenVMM"`	Xen Project	
	`"Microsoft Hv"`	Hyper-V, Microsoft Corporation	✔
	`" lrpepyh  vr"`	Parallels Inc.	

	String	Vendor	Supported?
	"ACRNACRNACRN"	ACRN Project	
	"KVMKVMKVM\0\0\0"	Kernel-based Virtual Machine, Linux	✔
	"VBoxVBoxVBox"	Oracle VirtualBox	✔
	"TCGTCGTCGTCG"	Tiny Code Generator, QEMU	

Table 1-1: Vendor Strings

1.3 Inline Notation Reference

For a better readability experience, most features are printed within the same lane as their respected category. If a feature is not shown, it indicates that the feature is missing or unavailable from the processor. Such notation is explained on the table below.

Notation	Description
A_B	May denote a space, rendering as "A B".
A+B	Denotes two attached items, so A and B comes together.
+A	Denotes an extra feature to the preceding feature.
A/B	Denotes A is the vendor specific name while B is the common name, when available.
A=B	Denotes extra information on the extension or instruction. For example, `clflush=64B` indicates a flush size of 64 Bytes.
(A, B)	Denotes feature A and B are present from parent feature. For example, tsx (hle, rtm) indicates HLE and RTM enabled.

Table 1-2: Inline Notation Reference

1.4 Future Work

Despite other processors having a similar processor identification feature, the ddcpuid project is not planned to be supported on ARM (informal system registers require EL1 access or higher), PowerPC/PowerISA (the Processor Version Register requires privileged

access), MIPS (non-widespread usage), z/Architecture and System/390 (generally only available to business customers), and RISC-V (Control-Status Registers requires machine-level privilege access) instruction set architectures.

Thus, for these reasons, when compiling on other platforms other than x86, the compiler will emit "`Error: static assert: "Unsupported platform"`".

Chapter 2
Command Line Interface

The help screen can be accessed via `-h` or `--help`.

2.1 Command Line Options

`-r, --raw [level[,sublevel]]`

Show raw CPUID data in a table. Accepts a CPUID level (EAX) and a sublevel (ECX).

See CPUID Table for an output example.

`-l, --list`

Shows the full list of items. This contains a lot more information and is more suitable for parsing.

`-o`

Override maximum CPUID leaves to 0x40, 0x4000_0040, and 0x8000_0040 respectively within the scanning phase. Also useful with the CPUID table feature for diagnostic purposes.

`-b, --baseline`

Prints the Microarchitecture Feature Level with strict feature sets for x86-64 processors, otherwise its legacy baseline.

Possible values are `i386`, `i486`, `i586`, `i686`, `x86-64`, `x86-64-v2`, `x86-64-v3`, and `x86-64-v4`. A newline character is included.

`-p, --platform`

Prints the system platform. This is like the baseline, but more sensitive to the x86_64 processor bit, regardless of operating system. For example, running a virtual machine forced in a 32-bit environment (e.g., Windows XP (32-bit)) will show i686, and when in a 64-bit environment (e.g., Windows XP x64 (64-bit)) will show x86-64.

Possible values are `i386`, `i486`, `i586`, `i686`, and `x86-64`.

`--ver`

Only print the version string and exit.

`--version`

Show version screen and exit.

`-h, --help`

Show help screen and exit.

Chapter 3
Operating Modes

By default, when no options are specified, ddcpuid will proceed to its default mode of operation, Summary, which is meant for humans to be read.

Please note that results may be affected by BIOS settings, microcode revisions, paravirtualization software, emulators, and hypervisors. For the most accurate results, it is best to run the utility on the host computer, unless you seek features under a virtualized environment.

3.1 Summary Mode (Default)

In this mode, ddcpuid will proceed to fetch and display processor information in a human-readable format. This is best intended for a quick overview of processor details.

This information includes, in order:

1. Processor Vendor and Brand strings;
2. Processor display identifier numbers;
3. Physical and logical core count;
4. Maximum amount of physical and virtual memory supported;
5. Processor baseline level;
6. Processor platform category;
7. Features available, or enabled, on the processor, including SGX and SGX2;
8. Legacy and other enabled extensions;
9. All SSE (Streaming SIMD Extension) enabled extensions;
10. All AVX (Advanced Vector Extension) enabled extensions;
11. All AMX (Advanced Matrix Extension) enabled extensions;
12. Security mitigations;
13. And per level-cache information.

3.1.1 Processor Vendor String

The vendor string is vendor-issued and dictates the processor manufacturer. Possible values are discussed at Processor Vendor Support. Although paravirtualization-enabled hypervisor vendor strings are located at CPUID.4000_0000h, please be advised that virtualization software may easily change this string, and that some processors may use other vendor names for better compatibility.

The processor vendor string is typically obtained using `CPUID.01h.EBX~EDX~ECX`, and can contain up to 12 ASCII characters (3 registers × 4 bytes). This string is not null-terminated.

3.1.2 Processor Brand String

The brand string is the full model string in ASCII, such as `"Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz"`. Please be advised that virtualization software may easily change this string, or provide their own.

The processor brand string is obtained using CPUID leaves 8000_0002h to 8000_0004h by combining registers `EAX`, `EBX`, `ECX`, and `EDX` in order. The brand string can be up to 48 ASCII characters (3 CPUID calls × 4 registers × 4 bytes) and is not null-terminated. Other legacy methods, such as the brand string index, and obtaining the model name via the family identification number, are supported, but not explained in this document.

3.1.3 Processor Identifier (Family, Model, and Stepping)

Processor vendors identify processors with identification numbers named **Family**, **Model**, and **Stepping**.

By default, the effective numbers are shown, resembling the format used under Windows:

```
Family 0x6 Model 0x3a Stepping 0x9
```

The effective family and model numbers are often used in optimization guides.

Some platforms, to identify the processor, may use the identifier value (CPUID.01h.EAX) as-is.

3.1.3.1 On Intel Processors

When identifying Intel processors, Intel uses the calculated ("display") Family and Model values within their optimization manuals, notably in their latency and throughput tables

(e.g., 06_4EH refers to the Skylake microarchitecture, equivalent to `Family 0x6 Model 0x4e`).

Intel considers the release of a microcode update for a silicon revision to be the equivalent of a processor stepping and completes a full-stepping level validation for releases of microcode updates, but does not makes microcode updates exclusive to stepping revision numbers.

On Intel processors, processor identifiers are calculated according to the reference manual (Volume 2A, Figure 3-6), shown below.

```
if (BaseFamily != 15)
 Family = BaseFamily;
else
 Family = ExtendedFamily + BaseFamily;

if (BaseFamily == 6 || BaseFamily == 0)
 Model = (ExtendedModel << 4) + BaseModel;
else
 Model = BaseModel;
```

3.1.3.2 On AMD Processors

When identifying AMD processors, AMD use the calculated Family value to identify processors within the AMD optimization manuals.

On AMD processors, processor identifiers are calculated according to the reference manual (Volume 3, §E.3.2 CPUID Fn0000_0001_EAX), shown below.

```
if (BaseFamily < 15) { // 0Fh
     Family = BaseFamily;
     Model = BaseModel;
} else {
     Family = ExtendedFamily + BaseFamily;
     Model = (ExtendedModel << 4) + BaseModel;
}
```

3.1.4 Cache Information

Cache information is showed as provided from the processor per physical processor core, typically level 1 (L1) and level 2 (L2) caches. Please note that level 3 cache (L3) is usually shared across all physical processor cores and thus show the total amount of cache for the entire processor package. Processor configuration may vary depending on the microarchitecture.

The cache notation follows an "L%u-%c" format, where %u is the cache level and %c being the cache type: Instruction, Data, or Unified, where unified is both for instructions and data.

On recent processors, cache size are calculated according to the algorithm below, which gets the cache size in Bytes for the entire physical core or processor package.

```
CacheSize = Sets * Linesize * Partitions * Ways
```

On older AMD processors, when the 8000_001Dh extended leaf is not available, cache information and sizes are processed from the 8000_0006h extended leaf instead. For the level 1 (L1) cache, the size of the L1-D cache is located in the first byte of the ECX register, and the size of L1-I is located at the first byte of the EDX register (8000_0005h leaf). The L2 cache size is held in upper half of the ECX register, and when possible, the L3 cache size is located in the upper 18 bits of the EDX register using this algorithm: `((EDX >> 18) + 1) * 512`.

Legacy cache table (CPUID.02h) is supported, but not explained in this document.

3.1.5 Supported Processor Technologies

Processor technologies, mostly those publicly advertised, are shown in default mode, seen in the following sub-chapters. Technologies are not to be confused with extensions and features, such as VT-x and AMD-V which describes the virtualization extension.

The tables below show which processor technologies that will show within the Technologies section.

WARNING: Both Intel and AMD processors may report the HTT bit (CPUID.01h.EDX[28]) to design a multi-core processor, even those that do not support Hyper-Threading Technology (SMT), such as the Intel Core 2 Quad Q6600.

Intel Technology	CPUID bit
Enhanced SpeedStep (EIST)	CPUID.01h.ECX[7]
TurboBoost	CPUID.06h.EAX[1]
TurboBoost 3.0	CPUID.06h.EAX[14]
HLE (TSX)	CPUID.07h.EBX[4]
RTM (TSX)	CPUID.07h.EBX[11]
SGX	CPUID.07h(ECX=0h).EBX[2]
SGX2	CPUID.12h(ECX=0h).EAX[1]

Table 3-1: Supported Intel Technologies

AMD Technology	CPUID bit
Core Performance Boost	CPUID.8000_0007h.EDX[9]

Table 3-2: Supported AMD Technologies

3.1.6 Output Example

If no command-line switches are given, the default operating mode prints a human-readable overview of the processor the utility was running on.

```
Name:        AuthenticAMD AMD Ryzen 9 5950X 16-Core Processor
Identifier:  Family 0x19 Model 0x21 Stepping 0x0
Cores:       16 cores, 32 threads
Max. Memory: 256 TiB physical, 256 TiB virtual
Baseline:    x86-64-v3
Techs:       core-performance-boost htt
Extensions:  x87/fpu +f16c mmx extmmx amd64/x86-64 +lahf64
amd-v/vmx +svm=v1 aes-ni adx sha bmi1 bmi2
SSE:         sse sse2 sse3 ssse3 sse4.2 sse4a fma
AVX:         avx avx2
AMX:
Mitigations: ibpb ibrs ibrs_pref stibp stibp_on ssbd
Cache L1-D:   16x   32 KiB,  512 KiB total, si
Cache L1-I:   16x   32 KiB,  512 KiB total, si
Cache L2-U:   16x  512 KiB,    8 MiB total, si ci
Cache L3-U:    2x   32 MiB,   64 MiB total, si nwbv
```

3.2 List Mode

In list mode, all available processor information are displayed in a simple list, similar to the list from the `/proc/cpuinfo` device found in the Linux® operating system.

This mode is better suited for parsing.

3.2.1 Output Example

Output example when the "`--list`" command-line switch is used.

```
Vendor      : AuthenticAMD
Brand       : AMD Ryzen 9 5950X 16-Core Processor
Identifier  : 0xa20f10
Family      : 0x19
BaseFamily  : 0xf
ExtFamily   : 0xa
Model       : 0x21
BaseModel   : 0x1
ExtModel    : 0x2
Stepping    : 0x0
Cores       : 16
Threads     : 32
Extensions  : x87/fpu +f16c mmx extmmx sse sse2 sse3 ssse3 sse4.2
sse4a avx avx2 fma3 amd64/x86-64 +lahf64 amd-v/vmx +svm=v1 aes-ni
adx sha bmi1 bmi2
Extra       : monitor+mwait +min=64 +max=64 monitorx+mwaitx
pclmulqdq cmpxchg8b cmpxchg16b movbe rdrand rdseed rdmsr+wrmsr
sysenter+sysexit syscall+sysret rdtsc +tsc-invariant rdtscp rdpid
cmov fcomi+fcmov lzcnt popcnt xsave+xrstor xsetbv+xgetbv
fxsave+fxrstor skinit+stgi
Technologies: core-performance-boost htt
Cache       : clflush=64B clflushopt prefetchw
Level 1-D   : 16x     32 KiB, 8 ways, 1 parts, 64 B, 64 sets, si
Level 1-I   : 16x     32 KiB, 8 ways, 1 parts, 64 B, 64 sets, si
Level 2-U   : 16x    512 KiB, 8 ways, 1 parts, 64 B, 1024 sets, si
ci
Level 3-U   :  2x  32768 KiB, 16 ways, 1 parts, 64 B, 32768 sets,
si nwbv
```

```
System      : apic x2apic arat tm apic-id=22 max-id=32
Virtual     : vme apicv
Memory      : pae pse pse-36 page1gb amd-evp/nx pat mtrr pge smep
PhysicalBits: 48
LinearBits  : 48
Debugging   : mca mce de
Security    : ibpb ibrs ibrs_pref stibp stibp_on ssbd
Max. Leaf   : 0x10
Max. V-Leaf : 0x0
Max. E-Leaf : 0x80000023
Type        : Original
Brand Index : 0
Misc.       : fsgsbase
```

3.3 Baseline Mode

The processor baseline (optimization group), featured in the Summary Mode, can be used as stand-alone, useful in build scripts. When `-b` or `--baseline` is invoked, only the processor's baseline (optimization group) is printed.

Possible values under 32-bit hosts are `i386`, `i486` (baseline minimum), `i586` (Pentium), `i686` (Pentium Pro). Possible values under 64-bit targets are `x86-64` (minimum baseline), `x86-64-v2`, `x86-64-v3`, and `x86-64-v4`.

NOTE: While getting 32-bit results under 64-bit hosts should never happen, some 64-bit hosts may lack extensions to correspond to the GCC `x86-64` optimization group.

For more information about the feature optimization groups under x86-64 targets, see Microarchitecture Feature Levels.

3.3.1 Output Example

```
x86-64-v2
```

3.4 Raw CPUID Mode

When invoked with the “`-r`” or “`--raw`” option, a table of CPUID hexadecimal values, without the “0x” prefix, are printed. This is useful for diagnostic purposes. It can be used

to verify a set of bits not supported by ddcpuid. The table generated in this mode is compatible with the Markdown markup language.

To target a specific CPUID leaf (value in the EAX register), you may supply a numerical value to the switch. This option supports decimal and hexadecimal values.

For example, to print information from leaf CPUID.02h: "`--raw 2`". To specify a subleaf (value in the ECX register), you may supply a numerical value after a comma, additional to the leaf value. For example, to print information from leaf CPUID.04h(ECX=01h): "`--raw 0x4,0x1`".

Otherwise, the "`-o`" option overrides the requested maximum leaves (standard, virtual, and extended) to 0x40, 0x4000_0040, and 0x8000_0040 and can be used in exploring behavior past the processor's maximum supported leaf.

The default mode of operation checks the highest nibble (EAX[31:28]) with the input leaf for a maximum leaf to be used. This should automatically detect virtualization and Xeon Phi leaves.

3.4.1 Output Example

Below is an example on a processor branded as `AMD Ryzen 9 5950X 16-Core Processor` using `ddcpuid -r` running the computer host.

Leaf	Sub-leaf	EAX	EBX	ECX	EDX
0	0	10	68747541	444d4163	69746e65
1	0	a20f10	10200800	7ed8320b	178bfbff
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	40	40	3	11
6	0	4	0	1	0
7	0	0	219c97a9	40069c	10
8	0	0	0	0	0
9	0	0	0	0	0
a	0	0	0	0	0
b	0	1	2	100	10
c	0	0	0	0	0
d	0	207	988	988	0

e	0	0	0	0	0
f	0	0	ff	0	2
10	0	0	2	0	0
80000000	0	80000023	68747541	444d4163	69746e65
80000001	0	a20f10	20000000	75c237ff	2fd3fbff
80000002	0	20444d41	657a7952	2039206e	30353935
80000003	0	36312058	726f432d	72502065	7365636f
80000004	0	20726f73	20202020	20202020	202020
80000005	0	ff40ff40	ff40ff40	20080140	20080140
80000006	0	48002200	68004200	2006140	2009140
80000007	0	0	3b	0	6799
80000008	0	3030	111ef657	501f	10000
80000009	0	0	0	0	0
8000000a	0	1	8000	0	101bbcff
8000000b	0	0	0	0	0
8000000c	0	0	0	0	0
8000000d	0	0	0	0	0
8000000e	0	0	0	0	0
8000000f	0	0	0	0	0
80000010	0	0	0	0	0
80000011	0	0	0	0	0
80000012	0	0	0	0	0
80000013	0	0	0	0	0
80000014	0	0	0	0	0
80000015	0	0	0	0	0
80000016	0	0	0	0	0
80000017	0	0	0	0	0
80000018	0	0	0	0	0
80000019	0	f040f040	f0400000	0	0
8000001a	0	6	0	0	0
8000001b	0	3ff	0	0	0
8000001c	0	0	0	0	0
8000001d	0	4121	1c0003f	3f	0
8000001e	0	10	108	0	0
8000001f	0	1780f	173	1fd	1
80000020	0	0	2	0	0
80000021	0	4d	0	0	0
80000022	0	0	0	0	0
80000023	0	0	0	0	0

Running the option again with “`ddcpuid --raw 4,1`”:

```
| Leaf     | Sub-leaf | EAX      | EBX      | ECX      | EDX      |
|----------|----------|----------|----------|----------|----------|
|        4 |        1 |        0 |        0 |        0 |        0 |
```

Chapter 4
Processor Extensions

Processor extensions add a variety of new instructions to aid computation of everyday tasks at greater speeds. Some of these extensions are specific to the microarchitecture, meaning the processor vendor may choose to implement extensions in certain microarchitectures.

4.1 x87 On-Chip-FPU

Introduced	**Intel**	1989 with i486
	AMD	1993 with Am486
CPUID bit	01h.EDX[0]	
Instructions	See instructions using escape codes 0xd8 to 0xdf (ESC).	

Table 4-1: x87 FPU-On-Chip Summary

Floating-point arithmetic is a term referring to formulaic representation of real numbers, a trade-off between range and precision using approximation.

Originally, advertised as math co-processors, the Floating-Point Unit was available on a separate processor, such as the 8087, 80287, or 80387 (i387). Starting with the 80387, these floating-point coprocessors fully implemented the IEEE 754:1985 Standard for Floating-Point Arithmetic convention for mathematical computation requiring decimal precision. The 8087 and 80287 co-processors improperly compared positive and negative infinity by stating they are equal. This was fixed in the i387SX and later FPUs.

Starting with the 80486 (i486DX), the **Floating-Point Unit** now became on-board (in-die), except for the i486SX, where a separate i487SX may be available, becoming the x87 FPU.

The x87 extension adds 8 80-bit registers(R0 to R7) along with 8 stack pointers (ST(0) to ST(7)), a 16-bit Control Register (FPCSR), a 16-bit Status Register, a 16-bit Tag Register, a 48-bit Last Instruction Pointer (FCS:FIP), a 48-bit Last Data (Operand) Pointer (FDS:FDP), and a 10-bit Opcode Register.

4.2 MMX

Introduced	**Intel**	1997 with P6
	AMD	1997 with K6
CPUID bit	01h.EDX[23]	
Instructions	• MOVD • MOVQ • PACKSSWB • PACKSSDW • PACKUSWB • PUNPCKHBW • PUNPCKHWD • PUNPCKHDQ • PUNPCKLBW • PUNPCKLWD • PUNPCKLDQ • PADDB • PADDW • PADDD • PADDSB • PADDSW	• PADDUSB • PADDUSW • PSUBB • PSUBW • PSUBD • PSUBSB • PSUBSW • PSUBUSB • PSUBUSW • PMULHW • PMULLW • PMADDWD • PCMPEQB • PCMPEQW • PCMPEQD • PCMPGTB	• PCMPGTW • PCMPGTD • PAND • PANDN • POR • PXOR • PSLLW • PSLLD • PSLLQ • PSRLW • PSRLD • PSRLQ • PSRAW • PSRAD • EMMS

Table 4-2: MMX Summary

The MMX extension introduced SIMD (single instruction, multiple data) instructions with new registers: 8 MM 64-bit registers.

4.3 Extended MMX (ExtMMX)

Introduced	**Intel**	1999 with P9 (Pentium III)
	AMD	1999 with K7 (Athlon)
CPUID bit	**AMD**	8000_0001h.EDX[22]
Instructions	• PADDSIW • PAVEB • PDISTIB • PMACHRIW • PMAGW • PMULHRW • PMULHRIW • PMVZB • PMVNZB • PMVLZB • PMVGEZB • PSUBSIW	

Table 4-3: Extended MMX Summary

The Extended MMX extension, introduced in AMD Athlon processors, added SIMD instructions and are distinguished from SSE since AMD did not include some SSE instructions in their Athlon processors.

4.4 3DNow!

Introduced	**AMD**	1998 with K6-2
CPUID bit	**AMD**	8000_0001h.EDX[31]
Instructions	• PAVGUSB • PMULHRW • PI2FD • PF2ID • PFMAX • PFMIN • PFCMPEQ • PFCMPGE • PFCMPGT • PFADD • PFACC • PFSUB • PFSUBR • PFMUL • PFRCP • PFRSQRT • PFRCPIT1 • PFRCPIT2 • PFRSQIT1	

Table 4-4: 3DNow! Summary

The 3DNow! extension was added by AMD for vector processing, which is useful for video processing and three-dimensional rendering.

In 2010, AMD deprecated the 3DNow! extension except for the PREFETCH and PREFETCHW instructions. The Bulldozer, Bobcat, Zen and their derivatives architectures effectively dropped the 3DNow! extension. The last AMD processor supporting 3DNow! is the A8-3870K APU.

4.5 Extended 3DNow! (Ext3DNow!)

Introduced	**AMD**	1999 with K7 (Athlon)
CPUID bit	**AMD**	8000_0001h.EDX[30]
Instructions	• PF2IW • PI2FW • PSWAPD • PFNACC • PFPNACC	

Table 4-5: Extended 3DNow! Summary

The Extended 3DNow! extension is an extension to 3DNow! and it was added in their Athlon processors. This extension was dropped alongside 3DNow! with the Bulldozer, Bobcat, and Zen microarchitectures.

4.6 Streaming SIMD Extensions (SSE)

Introduced	**Intel**	1999 with P6 (Pentium III)
	AMD	1999 with K7 (Athlon XP)
CPUID bit	01h.EDX[25]	

Instructions	• ADDPS • ADDSS • ANDNPS • ANDPS • CMPXXPS • CMPXXSS • COMISS • CVTPI2PS • CVTPS2PI • CVTSI2SS • CVTSS2SI • CVTTPS2PI • CVTTSS2SI • DIVPS • DIVSS • EQ • FXRSTOR • FXSAVE • LDMXCSR • LE • LT • MASKMOVQ • MAXPS • MAXSS • MINPS	• MINSS • MOVAPS • MOVHLPS • MOVHPS • MOVLHPS • MOVLPS • MOVMSKPS • MOVNTPS • MOVNTQ • MOVSS • MOVUPS • MULPS • MULSS • NE • NLE • NLT • ORD • ORPS • PAVGB • PAVGW • PEXTRW • PINSRW • PMAXSW • PMAXUB • PMINSW	• PMINUB • PMOVMSKB • PMULHUW • PREFETCHNTA • PREFETCHT0 • PREFETCHT1 • PREFETCHT2 • PSADBW • PSHUFW • RCPPS • RCPSS • RSQRTPS • RSQRTSS • SFENCE • SHUFPS • SQRTPS • SQRTSS • STMXCSR • SUBPS • SUBSS • UCOMISS • UNORD • UNPCKHPS • UNPCKLPS • XORPS

Table 4-6: Streaming SIMD Extensions Summary

The Streaming SIMD Extensions were added in the Intel Pentium III and AMD AthlonXP processors. It adds 8 XMM 128-bit registers and the MXCSR status register.

4.6.1 Float16 Conversion (F16C)

Introduced	**Intel**	2012 with Ivy Bridge
	AMD	2013 with Jaguar
CPUID bit	01h.ECX[29]	
Instructions	• VCVTPH2PS • VCVTPS2PH	

Table 4-7: Float-16 Conversion Summary

If set, 16-bit float conversion is available, such as converting four packed half precision (16-bit) floating-point values or eight packed half precision (16-bit) floating-point values to a packed single-precision float-point value (VCVTPH2PS) and vice versa (VCVTPS2PH).

This extension was introduced part of the SSE extensions and was previously known as CVT16.

4.7 Streaming SIMD Extensions 2 (SSE2)

Introduced	**Intel**	2000 with NetBurst (Pentium 4)
	AMD	2003 with K8 (Athlon 64, Opteron)
CPUID bit	01h.EDX[26]	

Instructions	• ADD(PD|SD) • ANDN(PD|PS) • ANDPD • CLFLUSH • CMPPD • CMPSD • COMISD • CVTDQ2(PD|PS) • CVTPD2(DQ|PI|PS) • CVTPI2PD • CVTPS2DQ • CVTPS2PD • CVTSD2SI • CVTSD2SS • CVTSI2SD • CVTSS2SD • CVTTPD2DQ • CVTTPD2PI • CVTTPS2DQ • CVTTSD2SI • DIVPD • DIVSD • LFENCE • MASKMOVDQU • MAXPD • MAXSD • MFENCE • MINPD • MINSD • MOVAPD • MOVDQ2Q • MOVHPD • MOVLPD • MOVNTDQ	• MOVNTI • MOVNTPD • MOVQ • MOVQ2DQ • MOVSD • MOVUPD • MULPD • MULSD • ORPD • PACKSSDW • PACKSSWB • PACKUSWB • PADDB • PADDD • PADDQ • PADDSB • PADDSW • PADDUSB • PADDUSW • PADDW • PAND • PANDN • PCMPXXB • PCMPXXD • PCMPXXW • PMADDWD • PMULHW • PMULLW • PMULUDQ • POR • PSHUFD • PSHUFHW • PSHUFLW • PSLLD • PSLLDQ	• PSLLQ • PSLLW • PSRAD • PSRAW • PSRLD • PSRLDQ • PSRLQ • PSRLW • PSUBB • PSUBD • PSUBQ • PSUBSB • PSUBSW • PSUBUSB • PSUBUSW • PSUBW • PUNPCKHBW • PUNPCKHDQ • PUNPCKHQDQ • PUNPCKHWD • PUNPCKLBW • PUNPCKLDQ • PUNPCKLQDQ • PUNPCKLWD • PXOR • SQRTPD • SQRTSD • SUBPD • SUBSD • UCOMISD • UNPCKHPD • UNPCKLPD • XORPD

Table 4-8: Streaming SIMD Extensions 2 Summary

First introduced in the Intel Pentium 4 processor, the second extension to SSE features the CLFLUSH instruction, and instructions for cache control.

4.8 Streaming SIMD Extensions 3 (SSE3)

Introduced	**Intel**	2004 with NetBurst Prescott (Pentium 4)
	AMD	2005 with K8 (Athlon 64, Opeteron)
CPUID bit	01h.ECX[0]	
Instructions	• ADDSUBPD • ADDSUBPS • HADDPD • HADDPS • HSUBPD	• HSUBPS • LDDQU • MOVDDUP • MOVSHDUP • MOVSLDUP	• FISTTP • MONITOR • MWAIT

Table 4-9: Streaming SIMD Extensions 3 Summary

Introduced in the Intel Pentium 4 Prescott family, SSE3 is an extension to the SSE family of instructions.

4.9 Supplemental Streaming SIMD Extensions 3 (SSSE3)

Introduced	**Intel**	2006 with Core
	AMD	2011 with Bulldozer
CPUID bit	01h.ECX[9]	
Instructions	• PSIGND • PSIGNW • PSIGNB • PHADDD • PHADDW • PHADDSW	• PHSUBD • PHSUBW • PHSUBSW • PMADDUBSW • PABSD • PABSW	• PABSB • PMULHRSW • PSHUFB • PALIGNR

Table 4-10: Supplemental Streaming SIMD Extensions 3 Summary

Introduced in the Core architecture, the SSSE3 extension is an extension to SSE3 and was sometimes confused for SSE4 during development.

4.10 Streaming SIMD Extensions 4 (SSE4) Extensions

SSE4 is a group of extensions coming in three flavors: SSE4.1, SEE4.2, and SSE4a. All announced in 2006, but implemented in 2007, SSE4 extensions are both supported in recent Intel (Core) and AMD processors (K10).

4.10.1 Streaming SIMD Extensions 4.1 (SSE41)

Introduced	**Intel**	2008 with Nehalem
	AMD	2008 with Familly 11h (Turion X2 Ultra)
CPUID bit	01h.ECX[15]	
Instructions	• MPSADBW • PHMINPOSUW • PMULDQ • PMULLD • DPPS • DPPD • BLENDPS • BLENDPD • BLENDVPS • BLENDVPD • PBLENDVB • PBLENDW • PMINSB • PMAXSB • PMINUW • PMAXUW • PMINUD	• PMAXUD • PMINSD • PMAXSD • ROUNDPS • ROUNDSS • ROUNDPD • ROUNDSD • INSERTPS • PINSRB • PINSRD • PINSRQ • EXTRACTPS • PEXTRB • PEXTRW • PEXTRD • PEXTRQ • PMOVSXBW	• PMOVZXBW • PMOVSXBD • PMOVZXBD • PMOVSXBQ • PMOVZXBQ • PMOVXSWD • PMOVZXWD • PMOVSXWQ • PMOVZXWQ • PMOVSXDQ • PMOVZXDQ • PTEST • PCMPEQQ • PACKUSDW • MOVNTDQA

Table 4-11: Streaming SIMD Extensions 4.1 Summary

The Streaming SIMD Extensions 4.1 were introduced in Intel's Core 2 Penryn architecture.

4.10.2 Streaming SIMD Extensions 4.2 (SSE42)

Introduced	**Intel**	2008 with Nehalem
	AMD	2008 with Family 11h (Turion X2 Ultra)

CPUID bit	01h.ECX[20]		
Instructions	• CRC32 • PCMPESTRI • PCMPESTRM	• PCMPISTRI • PCMPISTRM • PCMPGTQ	• POPCNT

Table 4-12: Streaming SIMD Extensions 4.2 Summary

Introduced in Intel's Core Nehalem architecture (1st Core generation). The Streaming SIMD Extensions 4.2 was designed to speed up XML parsing[1] and introduces the CRC32 instruction.

4.10.3 Streaming SIMD Extensions 4a (SSE4a)

Introduced	**AMD**	2007 with Family 10h (Barcelona, Phenom II)	
CPUID bit	**AMD**	8000_0001h.ECX[6]	
Instructions	• LZCNT • POPCNT	• EXTRQ • INSERTQ	• MOVNTSD • MOVNTSS

Table 4-13: Streaming SIMD Extensions 4a Summary

Introduced in AMD's Barcelona (Family 10h) architecture, the SSE4a extension features the LZCNT instructions.

1 https://web.archive.org/web/20160405012724/https://software.intel.com/en-us/articles/xml-parsing-accelerator-with-intel-streaming-simd-extensions-4-intel-sse4/

4.10.4 Streaming SIMD eXtended Operation (XOP)

Introduced	**AMD**	2011 with Family 15h (Bulldozer)
CPUID bit	**AMD**	8000_0001h.ECX[1]
Instructions	• BEXTR • BLCFILL • BLCI • BLCIC • BLCMSK • BLCS • BLSFILL • BLSIC • LLWPCB • LWPINS • LWPVAL • SLWPCB • T1MSKC • TZMSK • VFRCZxx • VPCOMccx • VPHADDxx • VPMACSxxxx • VPROTx • VPSHAx	

Table 4-14: Streaming SIMD Extended Operation Summary

Introduced in the Bulldozer microarchitecture and initially intended as SSE5[1], the XOP prefix adds three more opcode maps (8, 9, and 10) to the SSE instruction and functions similarly to the 3-byte VEX prefix. It is considered to be an extension to SIMD like AVX, AVX2, and FMA4.

AMD removed XOP support from the Zen microarchitecture and onward.

4.11 AMD64/EM64T/Intel64 (x86_64)

Introduced	**AMD**	2003 with K8 (Athlon 64)
	Intel	2004 with NetBurst (Pentium 4)
CPUID bit	8000_0001h.EDX[29]	
Instructions	See REX prefix.	

Table 4-15: x86-64 Summary

Also known as the LONG operating mode, x86-64 is the common name of the 64-bit native extension to the x86 architecture. It expands operations to 64-bit wide, extends basic registers (such as RAX, RBX, RCX, RDX, RSP, RBP, RSI, and RDI), adds the 64-bit registers R8 to R15, adds additional SSE registers (XMM 8 to 16), adds 4-level paging (introducing

1 AMD, *Striking a Balance*, https://web.archive.org/web/20131104114239/http://developer.amd.com/community/blog/2009/05/06/striking-a-balance/

PML4 makes up to 48 bits in total, and grows the Directory Pointer to 9 bits), and removes multiple unused legacy features, such as the BCD instructions.

4.11.1 LAHF+SAHF in 64-bit Mode (LAHF64)

CPUID bit	8000_0001h.ECX[0]

Table 4-16: LAHF64 Summary

When this bit is set, the LAHF and SAHF instructions are available to use in the 64-bit operating mode.

4.12 Hardware Virtualization (VMX/VT-x/SVM/AMD-V/VIA-VT)

Introduced	**Intel**	2005 with Pentium 4 (Model 662 and 672)
	AMD	2006 with Athlon 64 (base, X2, and FX)
CPUID bit	**Intel**	01h.ECX[5]
	AMD	8000_0001h.ECX[2]
Instructions	**VMX** • INVEPT • INVVPID • VMCALL • VMCLEAR • VMFUNC • VMLAUNCH • VMPTRLD • VMPTRST • VMREAD • VMRESUME • VMRESUME • VMWRITE • VMXOFF • VMXON	**SVM** • INVLPGA • VMLOAD • VMMCALL • VMRUN • VMSAVE

Table 4-17: x86 Virtualization Summary

Denotes x86 hardware-assisted virtualization for running multiple instance of what can be called a virtual machine. Intel tends to name its technology VT-x or VMX and AMD tends to name it AMD-V or SVM.

AMD includes a SVM version via CPUID.8000_000Ah.EAX[7:0].

4.13 Intel TXT (SMX)

Introduced	**Intel**	2006 with Core
CPUID bit	**Intel**	01h.ECX[6]
Instructions	See the GETSEC instruction.	

Table 4-18: Intel TXT Summary

Complementing Intel's Management Engine, Intel Trusted eXecution Technology uses a Trusted Platform Module (TPM) to attest the authenticity of the platform and the operating system, assures that the operating system starts in a trusted environment in a dynamic chain of trust, and provides additional security capabilities. While the TPM v1.0 platform uses the SHA-1 hash algorithm, version 2.0 uses the SHA-256 hashing algorithm.

The Trusted Computing Group (TCG) got TPM technology standardized via ISO: See *ISO/IEC 11889-1:2009* for more details. TPM is also available for the EFI platform: See *TCG EFI Platform Specification For TPM Family 1.1 or 1.2* for more details.

4.14 Advanced Encryption Standard-New Instructions (AES-NI)

Introduced	**Intel**	2010 with Westmere
	AMD	2013 with Family 16h (Jaguar)
CPUID bit	01h.ECX[25]	
Instructions	• PCLMULQDQ • AESENC • AESDEC • AESENCLAST • AESDECLAST • AESKEYGENASSIST • AESIMC	

Table 4-19: Advanced Encryption Standard-New Instructions Summary

Includes instructions to aid calculation of AES-related encryption and decryption calculations.

AES-NI can be used to accelerate the performance of an implementation of AES by 3 to 10x over a completely software implementation.[1]

The instructions work with fixed block sizes of 128 bits of plain text and the number of rounds (10, 12, or 14) can be used depending on the desired key length (128, 192, or 256 bits respectively).

4.15 Advanced Vector Extension (AVX)

Introduced	**Intel**	2011 with Sandy Bridge
	AMD	2011 with Family 15h (Bulldozer)
CPUID bit	01h.ECX[28]	
Instructions	• VBROADCASTSS • VBROADCASTSD • VBROADCASTF128 • VINSERTF128 • VEXTRACTF128 • VMASKMOVPS • VMASKMOVPD • VPERMILPS • VPERMILPD • VPERM2F128 • VZEROALL • VZEROUPPER See other instructions with the VEX prefix.	

Table 4-20: Advanced Vector Extension Summary

Introduced in Intel's Sandy Bridge architecture, the AVX extension includes 16 YMM 256-bit registers, and 12 new instructions for 256-bit processing.

4.16 Advanced Vector Extension 2 (AVX2)

Introduced	**Intel**	2013 with Haswell
	AMD	2015 with Family 15h (Excavator)
CPUID bit	07h.EBX[5]	

1 https://software.intel.com/content/www/us/en/develop/articles/intel-advanced-encryption-standard-instructions-aes-ni.html

Instructions	• VBROADCASTSS • VBROADCASTSD • VPBROADCASTB • VPBROADCASTW • VPBROADCASTD • VPBROADCASTQ • VBROADCASTI128 • VINSERTI128 • VEXTRACTI128 • VGATHERDPD See other instructions with the VEX prefix.	• VGATHERQPD • VGATHERDPS • VGATHERQPS • VPGATHERDD • VPGATHERDQ • VPGATHERQD • VPGATHERQQ • VPMASKMOVD • VPMASKMOVQ • VPERMPS	• VPERMD • VPERMPD • VPERMQ • VPERM2I128 • VPBLENDD • VPSLLVD • VPSLLVQ • VPSRLVD • VPSRLVQ • VPSRAVD

Table 4-21: Advanced Vector Extension 2 Summary

Introduced in Intel's Haswell architecture, the AVX2 extension expends most vector integer SSE and AVX instructions to 256 bits.

4.16.1 WAITPKG

Introduced	**Intel**	Planned for late 2022 with Sapphire Rapids		
CPUID bit	**Intel**	07h.ECX[5]		
Instructions	• TPAUSE	• UMONITOR	• UMWAIT	

Table 4-22: WAITPKG Summary

Part of the AVX2 extension, the WAITPKG extension includes a few instructions to better optimize processor states such as TPAUSE, for a timed pause, UMONITOR, to setup address monitoring, and UMWAIT, to monitor a range of addresses in an implementation-dependent optimized state.

If set, the WAITPKG instructions are available.

4.17 Advanced Vector Extension 512-bit (AVX512F)

Introduced	**Intel**	2015 with Knights Landing (Xeon Phi) 2017 with Skylake-X and Cannon Lake
	AMD	2022 with Geona (EPYC 9004)
CPUID bit	**Intel**	07h.EBX[16]
	AMD	Assuming the same as Intel's
Instructions	• VALIGN(D|Q) • VBLENDM(PD|PS) • VCOMPRESSPD/PS • VCVT(T)PD2UDQ • VCVT(T)PS2UDQ • VCVTQQ2PD/PS • VCVT(T)SD2USI • VCVT(T)SS2USI • VCVTUDQ2PD/PS • VCVTUSI2USD/S • VEXPANDPD/PS • VEXTRACTF32X4/64X4 • VEXTRACTI32X4/64X4 • VFIXUPIMM(PD|PS|SD|SS) • VGETEXPPD/PS • VGETEXPSD/SS • VGETMANTPD/PS • VGETMANTSD/SS • VINSERTF32X4/64X4 • VMOVDQA32/64 • VMOVDQU32/64 • VPBLENDMD/Q • VPBROADCASTD/Q • VPCM(PD|UD|PQ|UQ) • VPCOMPRESSQ/D • VPERMI2D/Q • VPERMI2PD/PS • VPERMT2D/Q • VPERMT2PD/PS • VPEXPANDD/Q • VPMAXSQ • VPMAXUD/UQ • VPMINSQ • VPMINUD/UQ • VPMOV(S|US)(QB|QW|QD|DB|DW) • VPROLD/Q • VPROLVD/Q • VPRORD/Q • VPRORRD/Q • VPSCATTER(DD|DQ|QD|QQ) • VPSRAQ • VPSRAVQ • VPTESTNMD/Q • VPTERLOGD/Q • VPTESTMD/Q • VRCP14PD/PS • VRCP14SD/SS • VRNDSCALEPD/(PS|SS) • VRSQRT14PD/(PS|SS) • VSCALEPD/(PS|SS) • VSCATTER(DD|DQ|QD|QQ) • VSHUFF32X4/64X2 • VSHUFI32X4/64X2 See instructions with the EVEX prefix.	

Table 4-23: Advanced Vector Extension 512-bit Summary

Introduced in Intel's Landing Knights (Xeon Phi) and Skylake architectures, AVX-512 are a set of additional extensions to the AVX series supporting 512-bit vector operations. When the **AVX512F** bit is set, this feature adds 32 new ZMM 512-bit registers, and extends the total count of vector registers to 32 for YMM (up to YMM31) and XMM (up to XMM31) registers.

The following AVX-512 extensions may be additionally present (with introduced architecture):

- Exponential and Reciprocal instructions (**AVX512ER**);
- Conflict Detection instructions (**AVX512CD**);
- New Prefetch instructions (**AVX512PF**);
- BYTE and WORD instructions (**AVX512BW**);
- DWORD and QWORD instructions (**AVX512DQ**);
- Vector Length instructions (**AVX512VL**);
- Integer Fused Multiply-Add instructions, some instructions are available with 52-bit precision like VPMADD52HUQ (**AVX512_IFMA**);
- Vector Byte Manipulation Instructions (**AVX512_VBMI**, **AVX512_VBMI2**, and AVX512BW);
- Vector Neural Networking Instructions (**AVX512_VNNI**);
- 4-iterations Vector Neural Network Instructions (enhanced Word variable) (**AVX512_4VNNIW**);
- Scalar and Packed Single-Precision Floating-Point Fused Multiply-Add extension (**AVX512_4FMAPS**);
- Galois Fields New Instructions (**AVX512_GFNI**);
- AES encryption and decryption extension (**AVX512_VAES**);
- VPOPCNT instruction support for 128-bit types (**AVX512_VPOPCNTDQ**);
- Support for BFLOAT16 instructions (**AVX512_BF16**);
- Compute intersection between DWORD/QWORD to a pair of mask registers (**AVX512_VP2INTERSECT**);
- And bit algorithmic instructions (**AVX512_BITALG**).

In order to have any AVX-512 extensions, the AVX512F (Foundation) bit is required to be present from CPUID. **Unless specified, all CPUID features in this section are available when ECX is set to 0h.**

Some extensions, including some future extensions, are explained in the following sub-chapters.

4.17.1 AVX-512 Exponential and Reciprocal (AVX512ER)

Introduced	**Intel**	2015 with Xeon Phi x200 and 2017 with Skylake-SP
CPUID bit	**Intel**	07h.EBX[27]
		• VEXP2PD • VEXP2PS • VRCP28PD • VRCP28PS • VRCP28SD • VRCP28SS • VRSQRT28PD • VRSQRT28PS • VRSQRT28SD • VRSQRT28SS

Table 4-24: AVX-512 Exponential and Reciprocal Summary

Includes AVX-512 Exponential and Reciprocal instructions. If set, VEXP2PD, VEXP2PS, VRCP28xx, and VRSQRT28xx instructions are supported.

4.17.2 AVX-512 Conflict Detection (AVX512CD)

Introduced	**Intel**	2015 with Xeon Phi x200 and 2017 with Skylake-SP
CPUID bit	**Intel**	07h.EBX[28]
Instructions		• VPBROADCASTMB2Q • VPBROADCASTMW2D • VPCONFLICTTD • VPCONFLICTTQ • VPLZCNTD • VPLZCNTQ

Table 4-25: AVX-512 Conflict Detection Summary

Helps to efficiently calculate conflict-free subsets of elements in loops that otherwise would not been safely vectored.

4.17.3 AVX-512 Prefetch (AVX512PF)

Introduced	**Intel**	2015 with Xeon Phi x200 and 2017 with Skylake-SP
CPUID bit	**Intel**	07h.EBX[26]
Instructions		• VGATHERPF0DPD • VGATHERPF0DPS • VGATHERPF0QPD • VGATHERPF0QPS • VGATHERPF1DPD • VGATHERPF1DPS • VGATHERPF1QPD • VGATHERPF1QPS • VSCATTERPF0DPD • VSCATTERPF0DPS • VSCATTERPF0QPD • VSCATTERPF0QPS • VSCATTERPF1DPD • VSCATTERPF1DPS • VSCATTERPF1QPD • VSCATTERPF1QPS

Table 4-26: AVX-512 Prefetch Summary

Includes AVX-512 Prefetch instructions. If set, VGATHERPF0xxx, VGATHERPF1xxx, VSCATTERPF0xxx, and VSCATTERPF1xxx instructions are supported.

4.17.4 AVX-512 Doubleword/Quadword (AVX512DQ)

Introduced	**Intel**	2018 with Cannon Lake and 2017 with Skylake-X
CPUID bit	**Intel**	07h.EBX[17]
Instructions	• VCVTPD2QQ • VCVTPD2UQQ • VCVTPS2QQ • VCVTPS2UQQ • VCVTTPD2QQ • VCVTTPD2UQQ • VCVTTPS2QQ • VCVTTPS2UQQ • VCVTUQQ2PD	• VCVTUQQ2PS • VFPCLASSPD • VFPCLASSPS • VFPCLASSSD • VFPCLASSSS • VPMOVD2M • VPMOVM2D • VPMOVM2Q • VPMOVQ2M	• VPMULLQ • VRANGEPD • VRANGEPS • VRANGESD • VRANGESS • VREDUCEPD • VREDUCEPS • VREDUCESD • VREDUCESS

Table 4-27: AVX-512 Doubleword/Quadword Summary

Adds the doubleword and quadword versions of the instructions tagged with this feature (DQ).

4.17.5 AVX-512 Byte/Word (AVX512BW)

Introduced	**Intel**	2018 with Cannon Lake and 2017 with Skylake-X
CPUID bit	**Intel**	07h.EBX[30]
Instructions	• KUNPCKBW • KUNPCKDQ • VDBPSADBW • VPBLENDMB • VPBLENDMW • VPCMPB • VPCMPUB • VPCMPUW	• VPCMPW • VPERMI2W • VPERMT2W • VPERMW • VPMOVB2M • VPMOVM2B • VPMOVM2W • VPMOVSWB	• VPMOVUSWB • VPMOVW2M • VPMOVWB • VPTESTMB • VPTESTMW • VPTESTNMB • VPTESTNMW

Table 4-28: AVX-512 Byte/Word Summary

Adds the byte and word instruction versions tagged with this feature (BW).

4.17.6 AVX-512 Vector Length (AVX512VL)

Introduced	**Intel**	2018 with Cannon Lake and 2017 with Skylake-X
CPUID bit	**Intel**	07h.ECX[1]
Instructions	See instructions with the AVX512VL tag.	

Table 4-29: AVX-512 Vector Length Summary

Allows 128-bit (XMM) and 256-bit (YMM) operations with AVX-512 instructions.

4.17.7 AVX-512 Integer Fused Multiply-Add (AVX512_IFMA)

Introduced	**Intel**	2018 with Cannon Lake
CPUID bit	**Intel**	07h.EBX[21]
Instructions	• VPMADD52HUQ • VPMADD52LUQ	

Table 4-30: AVX-512 Integer Fused Multiply-Add Summary

AVX-512 52-bit precision, where bit 53 is set, makes the VPMADD52HUQ and VPMADD52LUQ instructions available to use.

4.17.8 AVX-512 Vector Byte Manipulation Instructions (AVX512_VBMI, AVX512_VBMI2)

Introduced	**Intel**	VBMI: 2018 with Cannon Lake VBMI2: 2019 with Ice Lake
CPUID bit	**Intel**	VBMI: 07h.EBX[31] VBMI2: 07h.ECX[6]
Instructions	**VBMI**: VPERMB, VPERMI2B, VPERMT2B, VPMULTISHIFTQB **VBMI2**: VPCOMPRESSB, VPCOMPRESSW, VPEXPANDB, VPEXPANDW, VPSHLD, VPSHLDV, VPSHRD, VPSHRDV	

Table 4-31: AVX-512 Vector Byte Manipulation Instructions Summary

Adds VPERMB, VPERMI2B, VPERMT2B, and VPMULTISHIFTQB instructions. Also adds additional capabilities not in AVX512BW.

4.17.9 AVX-512 Galois Fields New Instructions (AVX512_GFNI)

Introduced	**Intel**	2019 with Ice Lake
CPUID bit	**Intel**	07h.ECX[8]
Instructions	GF2P8AFFINEINVQB, GF2P8AFFINEQB, GF2P8MULB	

Table 4-32: AVX-512 Galois Fields New Instructions Summary

Aids calculating Galois Fields.

4.17.10 AVX-512 Vector Advanced Encryption Standard (AVX512_VAES)

Introduced	**Intel**	2019 with Ice Lake
CPUID bit	**Intel**	07h.ECX[9]
Instructions	VAESDEC, VAESDECLAST, VAESENC, VAESENCLAST	

Table 4-33: AVX-512 Vector Advanced Encryption Standard Summary

Aids encryption and decryption of AES data.

4.17.11 AVX-512 Vector Neural Networking Instructions (AVX512_VNNI)

Introduced	**Intel**	2019 with Ice Lake
CPUID bit	**Intel**	07h.ECX[11]
Instructions	VPDPBUSD, VPDPBUSDS, VPDPWSSD, VPDPWSSDS	

Table 4-34: AVX-512 Vector Neural Networking Instructions Summary

Aids in vector neural networking calculations.

4.17.12 AVX-512 Bit Algorithms (AVX512_BITALG)

Introduced	**Intel**	2019 with Ice Lake
CPUID bit	**Intel**	07h.ECX[12]
Instructions	VPOPCNTB, VPOPCNTW, VPSHUFBITQMB	

Table 4-35: AVX-512 Bit Algorithms Summary

Two bit-counting instructions and a bit shuffle instruction.

4.17.13 AVX-512 VPOPCNTDQ (AVX512_VPOPCNTDQ)

Introduced	**Intel**	2017 with Knights Mill and 2019 with Ice Lake
CPUID bit	**Intel**	07h.ECX[14]
Instructions	VPOPCNT	

Table 4-36: AVX-512 VPOPCNTDQ Summary

Allows using VPOPCNT on YMM and XMM registers with writemask k1.

4.17.14 AVX-512 Vector Neural Network Instructions Word variable precision (AVX512_4VNNIW)

Introduced	**Intel**	2017 with Knights Mill
CPUID bit	**Intel**	07h.EDX[2]
Instructions	VP4DPWSSDS and VP4DPWSSD	

Table 4-37: AVX-512 Vector Neural Network Instructions Word variable-precision Summary

Vector instructions to help for deep learning.

4.17.15 AVX-512 Fused Multiply Accumulation Packed Single-precision (AVX512_4FMAPS)

Introduced	**Intel**	2017 with Knights Mill
CPUID bit	**Intel**	07h.EDX[3]
Instructions	V4FMADDPS, V4FNMADDPS, V4FMADDSS, V4FNMADDSS	

Table 4-38: AVX-512 Fused Multiply Accumulation Packed Single-precision Summary

Enables FMA operations on packed single-precision floating-point numbers.

4.17.16 AVX-512 VP2INTERSECT (AVX512_VP2INTERSECT)

Introduced	**Intel**	2020 with Tiger Lake
CPUID bit	**Intel**	07h.EDX[8]
Instructions	VP2INTERSECTD and VP2INTERSECTQ	

Table 4-39: AVX-512 VP2INTERSECT Summary

Allows intersection vector pairs to another pair of mask registers.

4.17.17 BFLOAT16 Number Type (AVX512_BF16)

Introduced in the Ice Lake architecture, the Brain floating-point format 16-bit is a binary number format that occupies 2 bytes of data, complying to IEEE 754. It is designed to be used in hardware-accelerated machine learning algorithms. The bfloat16 format features a truncated mantissa compared to the traditional float16 format, and a matching number of bits for the exponent for easier conversion, shown below.

float32 (~1^{-38} to ~3^{38})

	Exponent (8 bits)								Mantissa (23 bits)																							
S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

float16 (~5.96^{-8} to 65,504)

	Exponent (5)					Mantissa (10 bits)										
S	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M

bfloat16 (~1^{-38} to ~3^{38})

	Exponent (8 bits)								Mantissa (7 bits)							
S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M

Table 4-40: float32, float16, and bfloat16 Comparisons

Intel describes further how to use and convert the bfloat16 number in *BFLOAT16 – Hardware Numerics Definition White Paper* (November 2018, Version 1.0).

BFLOAT16 operations are available when the CPUID.07h(ECX=1h).EAX[5] bit is set.

4.18 Multi-Precision Add-Carry Instruction Extension (ADX)

Introduced	**Intel**	2014 with Broadwell
	AMD	2017 with Family 17h (Zen)
CPUID bit	07h.EBX[19]	
Instructions	• MULX	• ADCX • ADOX

Table 4-41: ADX Summary

Introduced in the Intel Broadwell and AMD Ryzen microarchitectures, the ADX extension features a few complementary instructions that extends the initial ADC (Add with Carry) instruction.

4.19 Fused-Multiply-Add (FMA) Extensions

The Fused-Multiply-Add instructions, grouped with SSE extensions, are a series of extensions to perform a single-rounding of a multiplication and addition instruction with multiple operands: a = (a · b) + c for FMA and d = (a · b) + c for FMA4). These instructions may use the VEX prefixes (C4h, C5h).

4.19.1 Fused-Multiply-Add 3-operand (FMA3)

Introduced	**AMD**	2012 with Family 15h (Piledriver)
	Intel	2013 with Haswell
CPUID bit	01h.ECX[12]	
Instructions	• VFMADD132PDy • VFMADD132PSy • VFMADD132PDx • VFMADD132PSx • VFMADD132SD • VFMADD132SS	• VFMADD213PDy • VFMADD213PSy • VFMADD213PDx • VFMADD213PSx • VFMADD213SD • VFMADD213SS

Table 4-42: Fused-Multiply-Add 3-operand Summary

Introduced in AMD's Piledriver and Intel's Haswell microarchitectures, mostly known as FMA, the FMA3 extension permits multiplication and addition with three operands. The Zen and Zen+ microarchitectures lack FMA3, but it was added back into the Zen 2 microarchitecture.

4.19.2 Fused-Multiply-Add 4-operand (FMA4)

Introduced	**AMD**	2011 with Family 15h (Bulldozer)
CPUID bit	**AMD**	8000_0001h.ECX[16]
Instructions	• VFMADDPDx • VFMADDPDy	• VFMADDPSx • VFMADDPSy

Table 4-43: Fused-Multiply-Add 4-operand Summary

Introduced in the Bulldozer microarchitecture, FMA4 extends on FMA3 by adding an additional operand per instruction. Zen, Zen+, and Zen 2 microarchitectures lack FMA4.

4.20 Trailing Bit Manipulation (TBM)

Introduced	**AMD**	2012 with Family 15h (Piledriver)
CPUID bit	**AMD**	8000_0001h.ECX[21]
Instructions	• BEXTR • BLCFILL • BLCI • BLCIC • BLCMSK • BLCS • BLSFILL • BLSIC • T1MSKC • TZMSK	

Table 4-44: Trailing Bit Manipulation Summary

Introduced in the Piledriver microarchitecture, the TBM extension features complementary instructions to BMI1. TBM support was removed in Jaguar and Zen processors.

4.21 Bit Manipulation Groups (BMI1, BMI2)

Introduced	**Intel**	BMI1 and BMI2: 2013 with Haswell
	AMD	BMI1: 2012with Family 15h (Piledriver) BMI2: 2015 with Family 15h (Excavator)
CPUID bits	BMI1: 07h.EBX[3] BMI2: 07h.EBX[8]	
Instructions	**BMI1** • ANDxx • ANDNxx • BEXTR • BLSI • BLSMSK • BLSR • TZCNT	**BMI2** • BZHI • MULX • PDEP • PEXT • RORX • SARX • SHRX • SHLX

Table 4-45: Bit Manipulation Groups Summary

Defines groups of bit manipulation instructions as BMI1 and BMI2 for advanced bit manipulation.

Please note that Advanced Bit Manipulation extension (ABM) from AMD features the LZCNT instruction, and the LZCNT instruction has its own CPUID bit (CPUID.8000_0001h.ECX[5]).

4.22 Intel SHA (SHA)

Introduced	**Intel**	2016 with Goldmont
CPUID bit	**Intel**	07h(ECX=00h).EBX[29]
Instructions	**Intel**	• SHA1RNDS4 • SHA1NEXTE • SHA1MSG1 • SHA1MSG2 • SHA256RNDS2 • SHA256MSG1 • SHA256MSG2

Table 4-46: Intel SHA Summary

An Intel extension to aid the process of the Secure Hash Algorithm (SHA) on the Goldmont microarchitecture (Intel Atom), notably the SHA-1 and SHA-256 variants, using the XMM registers.

4.23 Intel AMX

Introduced	**Intel**	Planned for 2022 with Sapphire Rapids
CPUID bit	**Intel**	07h.EDX[24]
Instructions	**Intel**	• LDTILECFG • STTILECFG • TDPBF16PS • TDPBSSD • TDPBSUD • TDPBUSD • TDPBUUD • TILELOADD • TILELOADDT1 • TILERELEASE • TILESTORED • TILEZERO

Table 4-47: AMX Summary

The Advanced Matrix Extensions, known as Intel® AMX, is an extension that brings new instructions to compute 2-dimensional arrays working on the principle of tiles and palettes. This adds 8 TMM 1 KiB registers and a Tile Matrix Multiply Unit (TMUL) within the first iteration of the extension.

This extension is only available in 64-bit (LONG) mode and depends on the XSAVE functionality to be enabled.

The extension also defines additional CPUID bits:

- **AMX-BF16** (07h.EDX[22]): If set, Bfloat16 is supported.
- **AMX-INT8** (07h.EDX[25]): If set, computational operations on 8-bit integers are supported.
- **XTILECFG** (0Dh.EAX[17]): If set, the 64-byte TILECFG register is available within XSAVE. CPUID.0Dh(ECX=17h) enumerates more information about XTILECFG.
- **XTILEDATA** (0Dh.EAX[18]): If set, 8192 bytes of tile data is available within XSAVE (so all TMM registers). CPUID.0Dh(ECX=18h) enumerates more information about XTILECFG.
- **XFD** (0Dh(ECX=01h).EAX[18]): If set, MSRs IA32_XFD (MSR address 1C4h) and IA32_XFD_ERR (1C5h) are available. Extended Feature Disable is an extension to XSAVE.

This extension is planned to appear in the Sapphire Rapids Xeon processor.

Chapter 5
Extra Instructions

This chapter describes instructions not part of any processor extensions as "extras". These instructions were added separately or alongside an extension with unique CPUID bits.

5.1 MONITOR and MWAIT

CPUID bit	01h.ECX[3]

Table 5-1: MONITOR + MWAIT Summary

MONITOR can be used to monitor a linear address range and MWAIT can be used to provide hints to the processor to enter an implementation dependent optimized state for address-range monitors and power management (basically, loops).

If available, the minimum and maximum MWAIT monitor-line sizes are shown in bytes as +MIN:N (CPUID.05h.AX) and +MAX:N (CPUID.05h.BX).

If set, MONITOR and MWAIT are available.

MONITORX and MWAITX: (AMD) If CPUID.8000_0001h.ECX[29] is set, then MONITORX and MWAITX are available.

5.2 PCLMULQDQ

CPUID bit	01h.ECX[1]

Table 5-2: PCLMULQDQ Summary

Carry-Less Multiplication Quadword. If set, this instruction is available.

5.3 CMPXCHG8B

CPUID bit	01h.EDX[8]

Table 5-3: CMPXCHG8B Summary

Compare and Exchange 8 Bytes. If set, this instruction is available.

5.4 CMPXCHG16B

CPUID bit	01h.ECX[13]

Table 5-4: CMPXCHG16B Summary

Compare and Exchange 16 Bytes. If set, this instruction is available.

5.5 MOVBE

CPUID bit	01h.ECX[22]

Table 5-5: MOVBE Summary

Move Data After Swapping Bytes. MOVBE is mainly available in Intel Atom processors and a few AMD processors.

If set, this instruction is available.

5.6 RDRAND

CPUID bit	01h.ECX[30]

Table 5-6: RDRAND Summary

Read Random Number. Conforms to NIST SP 800-90A, FIPS 140-2, and ANSI X9.82.

If set, this instruction is available.

5.7 RDSEED

CPUID bit	07h.EBX[18]

Table 5-7: RDSEED Summary

Similar to RDRAND, but generates a number from an enhanced non-deterministic random bit generator, having lower-level access to the entropy-generating hardware. Conforms to NIST SP 800-90B and NIST SP 800-90C in the XOR construction mode.

If set, this instruction is available.

5.8 RDMSR and WRMSR

CPUID bit	01h.EDX[5]

Table 5-8: RDMSR+WRMSR Summary

Read from Model-Specific Register and Write to MSR.

If set, RDMSR and WRMSR instructions are available.

5.9 SYSENTER and SYSEXIT

CPUID bit	01h.EDX[11]

Table 5-9: SYSENTER+SYSEXIT Summary

Fast System Call. Transfers control to a fixed entry point in an operating system. Used in legacy mode only (32-bit, IA-32e).

If set, SYSENTER and SYSEXIT instructions are available.

5.10 SYSCALL and SYSRET (SCE)

CPUID bit	8000_0001h.EDX[11]

Table 5-10: SYSCALL+SYSRET Summary

Fast System Call. Transfers control to a fixed entry point in an operating system. SYSCALL has reduced number of checks compared to SYSENTER.

If set, SYSCALL and SYSRET instructions are available.

5.11 RDTSC

CPUID bit	8000_0001h.EDX[27]

Table 5-11: RDTSC Summary

Read Timestamp Counter from the 64-bit IA32_TIME_STAMP_COUNTER MSR into EDX:EAX. Useful for Ring 3 (CPL=3) user programs to count cycles in-between instructions.

Please note that the counter can be disable via CR4[2] (TSD).

If set, this instruction is available.

5.11.1 TSC-Deadline

CPUID bit	**Intel**	01h.ECX[24]

Table 5-12: TSC-Deadline Summary

If set, the processor’s local APIC timer supports one-shot operation using a TSC deadline value at the IA32_TSC_DEADLINE MSR.

5.11.2 TSC-Invariant

CPUID bit	8000_0007h.EDX[8]

Table 5-13: TSC-Invariant Summary

If set, the timestamp counter is not affected in any C, P, and T processor states while in the S0 state.

5.12 RDTSCP

CPUID bit	8000_0001h.EDX[27]

Table 5-14: RDTSCP

Exactly like RDTSC, but also reads the Processor ID from the IA32_TSC_AUX MSR (C0000103h) into ECX.

If set, this instruction is available.

5.13 RDPID

CPUID bit	07h.ECX[22]

Table 5-15: RDPID

Read processor ID. Reads the value from the IA32_TSC_AUX MSR (C0000103h).

If set, this instruction is available.

5.14 CMOV, including FCOMI and FCMOV

CPUID bit	01h.EDX[15]

Table 5-16: CMOV Summary

Conditional Move, usually shown as CMOVcc.

If set, this instruction is available.

If set, and the processor is equipped with an on-chip FPU, **FCOMI** and **FCMOV** are also available.

5.15 LZCNT

CPUID bit	8000_0001h.ECX[5]

Table 5-17: LZCNT Summary

Count the number of leading zero bits, added alongside SSE4a.

If set, this instruction is available.

5.16 POPCNT

CPUID bit	01h.ECX[23]

Table 5-18: POPCNT Summary

Return the count of number of bits set (population count), added alongside SSE4.2.

If set, this instruction is available.

5.17 XSAVE and XRSTOR (XSAVE)

CPUID bit	01h.ECX[26]

Table 5-19: XSAVE Summary

Save or restore the processor extended states to or from EDX:EAX. Like FXRSTOR and FXSAVE, the memory format used for the x87 state depends on the REX.W prefix.

If set, XSAVE and XRSTOR are available.

5.18 XSETBV and XGETBV (OSXSAVE)

CPUID bit	01h.ECX[27]

Table 5-20: XSAVE Summary

Set or get extended control register from or EDX:EAX into or from the 64-bit extended control register (XCR0) specified in ECX.

If set, XSETBV and XGETBV are available.

5.19 FXSAVE and FXSTOR (FXSR)

CPUID bit	01h.EDX[24]

Table 5-21: FXSAVE + FXRSTOR

Save or restore x87 FPU, MMX, and SSE states. If FXSAVE is not available, see FSAVE. The REX.W prefix changes the memory format.

If set, FXSAVE and FXSTOR are available.

5.20 PCONFIG

CPUID bit	**Intel**	07h(ECX=0h).EDX[18]

Table 5-22: PCONFIG Summary

PCONFIG (Platform Configuration) is an instruction that allows software to configure certain platform features.

PCONFIG is planned to appear in the Intel Ice Lake architecture (10nm) or later.

5.21 CLDEMOTE

CPUID bit	**Intel**	07h(ECX=0h).ECX[25]

Table 5-23: CLDEMOTE Summary

CLDEMOTE (Cache Line Demote) is an instruction that hints the processor to demote a line cache to a superior level.

CLDEMOTE is planned to appear in the Intel Future Tremont architecture or later.

5.22 MOVDIRI and MOVDIR64B

CPUID bit	**Intel**	MOVDIRI: 07h.ECX[27] MOVDIR64B: 07h.ECX[28]
Instructions	• MOVDIRI • MOVDIR64B	

Table 5-24: MOVDIRI/MOVDIR64B Summary

The MOVDIRI and MOVDIR64B instructions were introduced in the Tremont microarchitecture.

The MOVDIRI (Move Doubleword as Direct Store) instruction moves a DWORD value from a register to memory using a direct-store operation.

The MOVDIR64B (Move 64 Bytes as Direct Store) instruction enables moving 64 Bytes between two memory locations using a direct-store operation atomically.

The definition of direct-store as written from the Intel® Architecture Instruction Set Extensions and Future Features Programming Reference manual (MOVDIRI) seen below.

> *"The direct-store is implemented by using write combining (WC) memory type protocol for writing data. Using this protocol, the processor does not write the data into the cache hierarchy, nor does it fetch the corresponding cache line from memory into the cache hierarchy. If the destination address is cached, the line is written-back (if modified) and invalidated from the cache, before the direct-store. Unlike stores with non-temporal hint that allow uncached (UC) and write-protected (WP) memory-type for the destination to override the non-temporal hint, direct-stores always follow WC memory type protocol irrespective of the destination address memory type (including UC and WP types)."*

5.23 ENQCMD

CPUID bit	**Intel**	07h.ECX[29]

Table 5-25: ENQCMD Summary

En-queue commands to en-queue registers using memory-mapped I/O (MMIO).

ENQCMD is planned to appear in the Sapphire Rapids architecture or later.

5.24 SKINIT and STGI

CPUID bit	**AMD**	8000_0001h.ECX[12]

Table 5-26: SKINIT and STGI Summary

The SKINIT (Secure Init and Jump with Attestation) and STGI (Set Global Interrupt Flag) instructions are complementary AMD-V instructions.

5.25 SERIALIZE

CPUID bit	**Intel**	07h.EDX[14]

Table 5-27: SERIALIZE Summary

SERIALIZE does not modify registers, arithmetic flags, or memory, it simplies *wait* until flags, registers, and memory operations are completed before fetching and executing the next instruction.

Other serializing instructions include (when CPL>0) CPUID, IRET, RSM, (when CPL=0) INVD, INVEPT, INVLPG, INVVPID, LGDT, LIDT, LLDT, MOV (to Control Registers, except CR8, and to Debuger Registers), WBINVD, and WRMSR.

The purpose of SERIALIZE is thus fully for this particular, optimized usage.

Chapter 6
Processor Features

Processor features aid the processor to accelerate the computing of various tasks. Such technologies change the processor microarchitecture and may have little to none new instructions.

This chapter attempts to explain some of the technologies in a comprehensible way.

6.1 Intel® TurboBoost and AMD Turbo Core

All processors operate at a stock frequency, which determines the number of cycles a processor is capable of operating within a second, in Hertz.

Depending on the workload, processors may temporarily overclock the base operating frequency (e.g., the Intel Core i7-3770 varying its frequency between its core frequency of 3.4 GHz and its turbo frequency of 3.9 GHz) to provide a temporary performance boost.

Intel TurboBoost 2.0 availability depends on these factors[1]:

- Type of workload;
- Number of active cores;
- Estimated current consumption;
- And Processor temperature.

Intel TurboBoost 3.0 availability depends on these factors[2]:

- Type of workload;
- Number of active cores;
- Estimated current consumption;
- Processor temperature;
- And Drive support.[3]

1 https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html
2 https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-max-technology.html
3 No further information is provided on Drive support.

AMD Turbo Core (Core-Performance-Boost) is activated upon the operating system's request[1] with these factors[2]:

- Power draw

AMD Turbo Core was introduced in the Bulldozer microarchitecture and a few A-series processors.

6.2 Intel® SpeedStep (EIST), AMD PowerNow!, and AMD Cool'n'Quiet

While the processor is idling, software, such as the operating system, may hint the processor to reduce its clock rate, reducing the power draw (voltage), and allowing lower heat generation.

Enhanced Intel SpeedStep may be abbreviated as EIST.

AMD PowerNow! was aimed for laptop computers and AMD Cool'n'Quiet was aimed for desktop and processors. While similar, both operated differently.

6.3 Intel® Hyper-Threading Technology

Introduced in 2002 on the Intel Xeon and Intel Pentium 4 processors, and sometimes known as "HTT", Hyper-Threading Technology is an Intel SMT technology that improves computation parallelization, which allows a physical processor core to share two logical cores ("thread" or "logical processor"), acting as virtual processor cores.

These logical cores contains architectural states (register state including the Instruction Pointer and sometimes the cache state) for a particular task. The execution engine, contained in the physical core, is capable of processing two tasks concurrently – switching between two tasks rapidly – making it useful when a task is, for example, waiting on an I/O operation (e.g. memory fetching).

1 https://www.amd.com/en/technologies/turbo-core
2 https://web.archive.org/web/20130311221620/http://blogs.amd.com/work/2011/01/31/bulldozer-goes-to-11/

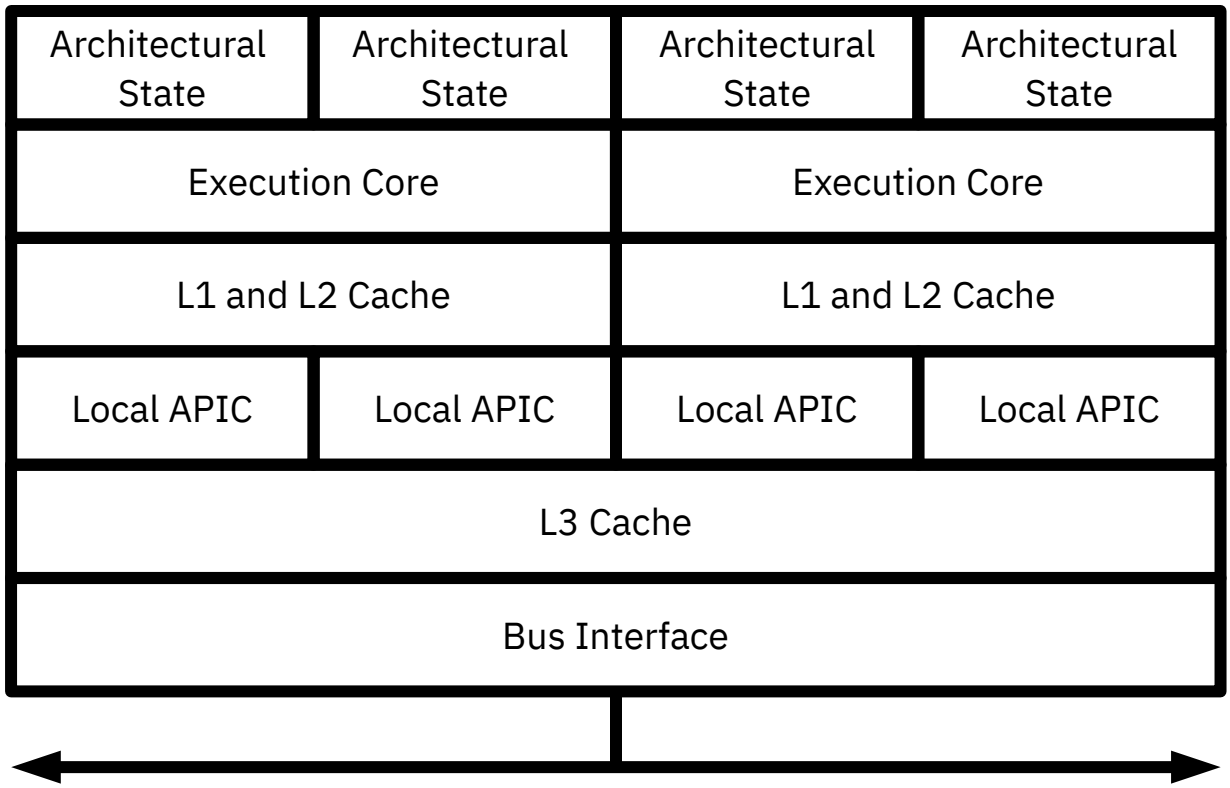

Figure 6-1: Dual-Core Processor Design Example with Hyper-Threading

Some models are known to have L1 cache state per architectural states, one execution engine per architectural with shared L2 cache, or even one local APIC per execution engine.

AMD may refer to Hyper-Threading Technology as HTT (CPUID.01h.EDX[28]), describing the CPUID flag as "Indicates either that there is more than one thread per core or more than one core per processor" (*AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions*). Starting with the Zen microarchitecture, AMD processors started to support the SMT architecture.

While most processors implement 2-way Hyper-Threading, recent Xeon Phi coprocessors implement 4-way Hyper-Threading (4 architectural states per execution core).

6.3.1 Security Concerns

Intel recommends disabling Hyper-Threading if the user is unaware of the technology. Since 2018, the OpenBSD operating system disables Hyper-Threading by default for security reasons, following Spectre-style concerns.

6.4 Intel® Software Guard Extensions (SGX)

Introduced in the Skylake microarchitecture in 2015, the Intel Security Guard Extensions (SGX) is a hardware-assisted security mechanism and application layer that helps prevent unauthorized exterior access of secret data from other applications, the operating system, and other hardware components.

These closed-off areas are called enclaves, and they help protect information from external access such as the memory system, the memory bus, and the processor package. They can be used in many scenarios such as public cloud environments, electronic medical records, and blockchain supply chains.

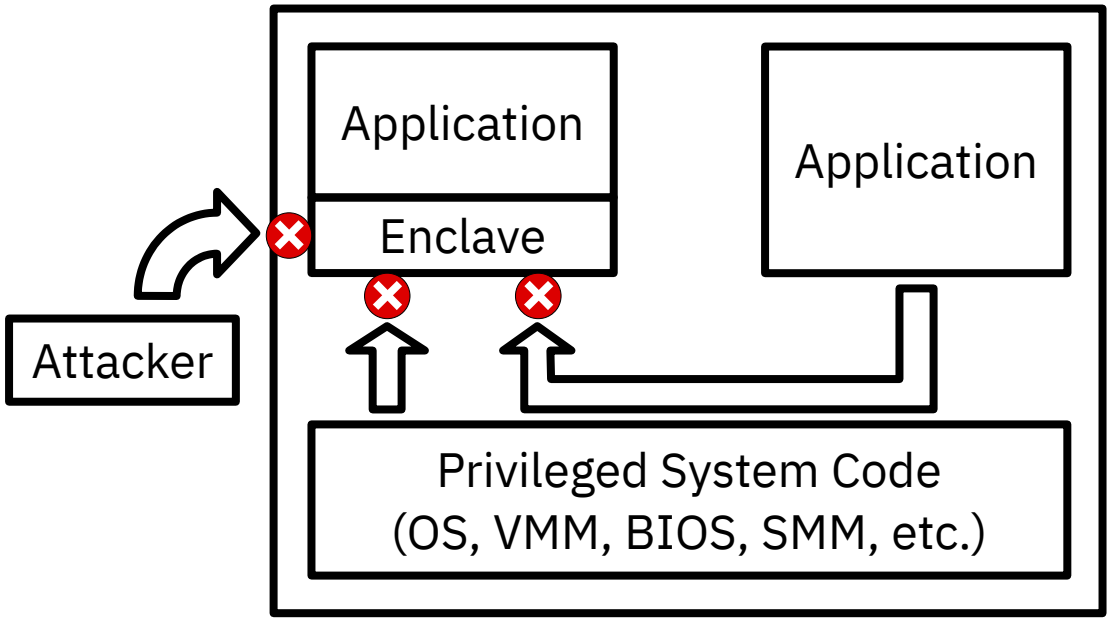

Figure 6-2: Intel SGX Overview

Enclaves may contain protected code and data. Code with its own data in enclaves may only be accessed through a Call Gate. Enclaves have their own entry table, heap, stack, and code.

Windows® supports the creation of SGX enclaves with a ENCLAVE_CREATE_INFO_SGX structure.[1] Linux® supports SGX via various frameworks such as Intel's linux-sgx package. By Linux 5.13, there is a kernel integration for KVM guests using SGX.[2]

1 https://docs.microsoft.com/en-us/windows/win32/api/enclaveapi/nf-enclaveapi-createenclave
2 http://lkml.iu.edu/hypermail/linux/kernel/2104.3/01565.html

When SGX support is enabled in the system's BIOS/EFI, the system advertises this feature via CPUID.7h.EBX[2]. The size of the enclave can also be changed in BIOS, but is fixed upon system startup.

WARNING: Intel has deprecated SGX under 11th and 12th Gen Core processors (11000 and 12000 series), breaking some applications, such as a Windows driver that enabled Blu-ray video playback for Ultra HD discs (in 4K).[1]

6.4.1 SGX Operation Overview

To use SGX, the BIOS/EFI firmware, operating system, user software, and processor must have the SGX feature compiled and enabled. The operation workflow is discussed below.

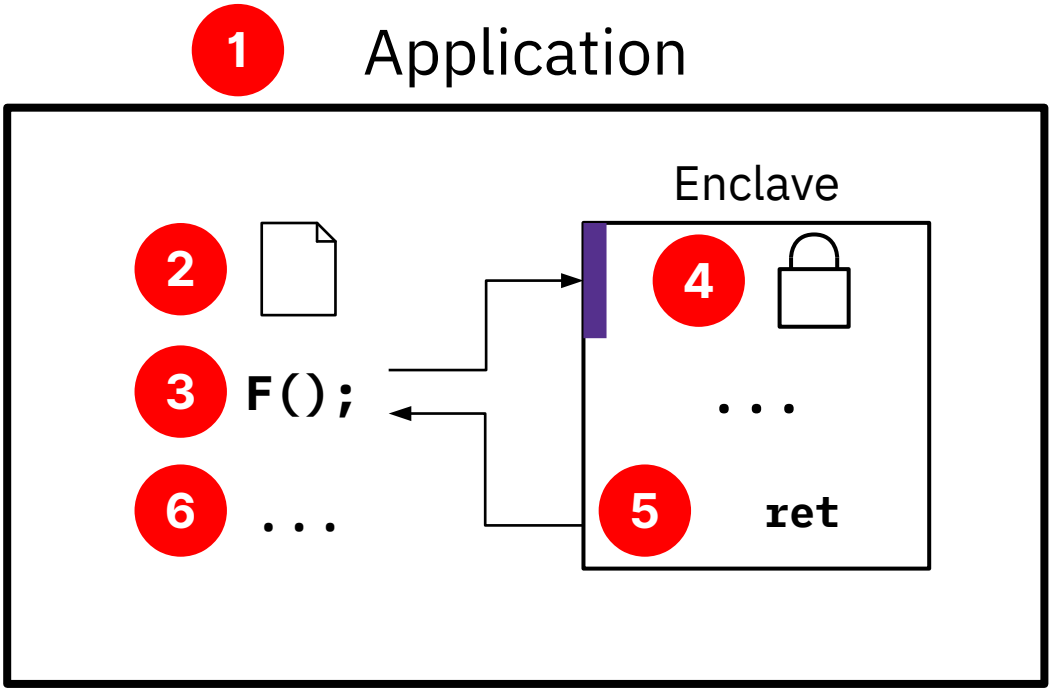

Figure 6-3: Intel SGX Runtime Example

Figure 6-3 shows an example running Intel SGX technology with the following actions.

1. The application contains enclave data and code. Intel SGX allows protected sections to be distributed in the clear. Before the enclave is created, the code and data are free to be inspected and analyzed.

1 https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/

2. The application creates an enclave. Memory access controls are applied in place to restrict external software access. External code is not allowed to read enclave data and code.
3. The application calls a trusted function, transferring the execution flow to the enclave through the call gate.
4. Enclave now sees all process data in clear within the trusted memory zone.
5. Once finished, the flow of execution is returned to the trusted function. The enclave remains in memory in the trusted memory zone, and may be closed if needed.
6. The application resumes non-trusted code and may call the trusted function at a later time.

6.4.2 Security Considerations

On March 2017, a Prime+Probe attack was discovered on the Intel SGX platform, making it possible to leak cryptography keys from enclaves.[1]

On January 2018, a Spectre-like vulnerability was discovered. With an adapted Spectre attack, it is possible to attack secure enclaves.[2] On August 2018, the Foreshadow vulnerability combines speculative execution and buffer overflow to bypass SGX technology.[3]

On February 2019, an enclave attack was found possible to perform within the enclave. This allows malware to be executed within the enclave, rendering anti-virus software unable to pick up the malware.[4]

On January 2020, the L1D Eviction Sampling[5] (L1DES, CacheOut), and the Vector Register Sampling[6] (VRS) exploits were published. The L1DES speculative execution attack is a variant of RIDL and MDS, and still allows unprivileged reading of SGX enclaves. Using the MDS or TAA methods, the VRS exploits lets an attack obtain previous read values from previously read vector registers.

1 Malware Guard Extension: Using SGX to Conceal Cache Attacks (Extended Version) https://arxiv.org/abs/1702.08719
2 https://github.com/lsds/spectre-attack-sgx
3 Speculative Buffer Overflows: Attacks and Defenses https://arxiv.org/abs/1807.03757
4 Practical Enclave Malware with Intel SGX https://arxiv.org/abs/1902.03256
5 https://software.intel.com/security-software-guidance/software-guidance/l1d-eviction-sampling
6 https://software.intel.com/security-software-guidance/software-guidance/vector-register-sampling

More information can be found on the MDS Attacks Website at https://mdsattacks.com/.

On June 2020, the CrossTalk attack make it possible to leak data regardless of the physical processor core used.[3] The attack focuses on SGX and the CPUID and RDRAND instructions, targeting the staging buffer.

On June 2020, SGAxe proves how SGX fails in practice using an updated CacheOut attack.[4] The paper concludes that, targeting the Line Fill Buffer, a transient execution attack can be used to recover SGX attestation keys from a fully updated Intel machine, trusted by Intel's attestation server.

For more information about SGAxe, visit https://sgaxe.com/.

On October 2021, the paper on SmashEx is publicly released.[5] This method abuses the exception handling mechanism on the operating system for a re-entry vulnerability. By crafting a certain set of registers and adjusting the permissions outside of the enclave, it is possible to trick the exception handler to enter the enclave using ERESUME (Enclave Resume).[6]

6.4.3 SGX2 (EDMM)

Introduced in the Gemini Lake microarchitecture in 2017 and the Ice Lake microarchitecture in 2019, SGX2, also known as Enclave Dynamic Memory Management (EDMM), is an extension to SGX that enhances enclave memory management by reissuing permissions to memory pages, removing memory pages, and expanding enclaves to allow more threads.

Because memory in SGX2 is directly managed by the processor, and not by the operating system unlike SGX1 with fixed memory enclave sizes, SGX2 enclaves are created on top of SGX1 enclaves.

3 https://www.vusec.net/projects/crosstalk/
4 https://sgaxe.com/, van Schaik, Stephan and Kwong, Andrew and Genkin, Daniel and Yarom, Yuval, *How SGX Fails in Practice*
5 https://dl.acm.org/doi/10.1145/3460120.3484821
6 https://jasonyu1996.github.io/SmashEx/

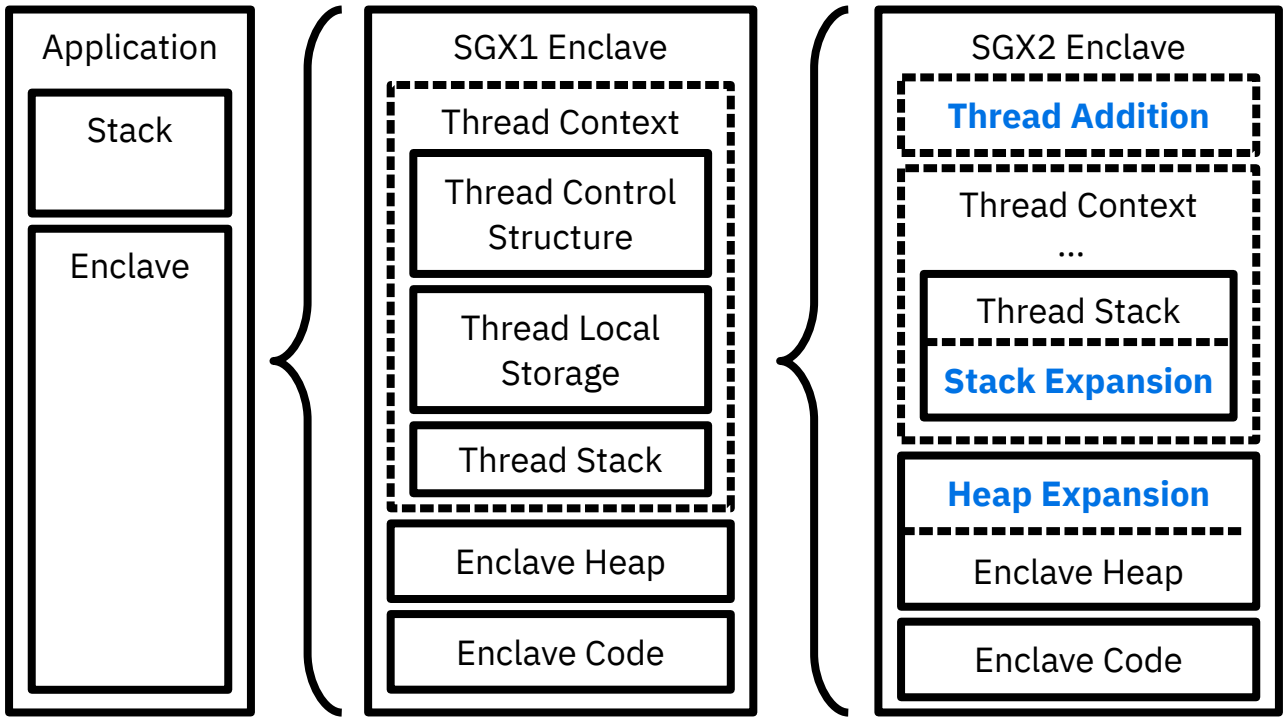

Figure 6-4: SGX2 Overview

When user-controlled SGX support is enabled in the system, the system advertises support for SGX1 (fixed system memory size) via CPUID.12h.EAX[0] and SGX2 (user-managed memory) via CPUID.12h.EAX[1]. There is also the maximum enclave size advertised in CPUID.12h.EDX[7:0] for non-64-bit environments and CPUID.12h.EDX[15:8] for 64-bit environments in size of 2^n bytes.

6.4.4 SGX2 Security Considerations

At the moment of writing, no vulnerabilities have been found specifically for SGX2.

6.5 Intel® Transactional Synchronization Extensions (TSX)

Introduced in the Haswell microarchitecture, the Intel Transactional Synchronization Extensions (TSX), being similar to AMD's Advanced Synchronization Facility (ASF)[1], adds transactional memory support, accelerating execution of multi-threaded applications through lock elision, bringing safe access to shared memory among computation threads.

While Hardware Lock Elision (HLE, CPUID.07H.EBX[4]) provides the legacy method to perform locks via the XACQUIRE and XRELEASE prefixes, the more recent Restricted Transactional Memory (RTM, CPUID.07H.EBX[11]) instruction set add XBEGIN and XEND instructions.

The XACQUIRE and XRELEASE prefixes reuse the REPNE and REPE operation codes (F2h and F3h), enabling support for processors that support Intel TSX while keeping backward compatibility.

6.5.1 Security Concerns

On July 2016, Intel TSX (HLE) was used to break kernel Address Space Layout Randomization (ASLR) with 100% accuracy on Linux platforms and around 100% accuracy on Windows platforms.[2]

On June 2019, a leap of faith with Intel TSX (HLE) can be used to escape a transactional jail and retrieve private data via previously traced data. According to the author, "[...], it is unlikely that this issue poses a risk to real-world applications.".[3]

1 As of June 2019, ASF is still at the proposal stage and has yet to be implemented in any AMD processors.
2 Breaking Kernel Address Space Layout Randomization with Intel TSX https://www.blackhat.com/docs/us-16/materials/us-16-Jang-Breaking-Kernel-Address-Space-Layout-Randomization-KASLR-With-Intel-TSX-wp.pdf
3 In Transactional Memory, No One Can Hear You Scream. Attacking Intel's Transactional Synchronization Extensions http://blog.ret2.io/2019/06/26/attacking-intel-tsx/

Chapter 7
Advanced Processor Features

This section attempts to explain some of the more advanced processor features, typically not widely shown in product specifications.

7.1 Cache Features

7.1.1 Per-Level Cache Features

This section describes per-level cache flags. For more information about cache, see Cache Information. Intel cache topology is located at CPUID.04h. AMD cache topology is located at CPUID.8000_0005h~8000_0006h (legacy) and at CPUID.8000_001Dh.

7.1.1.1 Self Initializing (SI)

If set, the cache is initialized by hardware. Otherwise software, such as an operating system, must initialize this cache level.

7.1.1.2 Fully Associative (FA)

If set, denotes the cache line is fully associative. In typical scenarios, there can be directly mapped cache (one memory address entry fits one block), 2-way cache (sets can fit two blocks each), 4-way cache (sets can fit four blocks each), and fully associative cache (block is allowed anywhere in the cache). Fully associative permits flexibility in block placement, but every tag must be compared when finding a block.

7.1.1.3 No Write-Back Invalidation (NWBI)

If set, the INVD and WBINVD instructions are not guaranteed to act on lower cache levels for non-originating threads sharing this cache.

7.1.1.4 Cache Inclusiveness (CI)

If set, the cache is included with lower levels.

7.1.1.5 Complex Cache Indexing (CCI)

If set, a complex function is used to index the cache, possibly using all address bits.

7.1.2 CLFLUSH

CPUID bit	01h.EDX[19]

Table 7-1: CLFLUSH Summary

Flush Cache Line. Invalidates all cache levels associated with the linear address contained in the memory operand.

If set, this instruction is available. The line size in bytes is supplied from bits CPUID.01h.EBX[15:8].

7.1.3 CLFLUSHOPT

CPUID bit	07h.EBX[23]

Table 7-2: CLFLUSHOPT Summary

Optimized Flush Cache Line. Flushes the cache line specified by the linear-address. The instruction checks all levels of the cache hierarchy-internal caches and external caches-and invalidates the cache line in every cache in which it is found.

If set, CLFLUSHOPT is available.

7.1.4 L1 Context ID (CNXT-ID)

CPUID bit	**Intel**	01h.ECX[10]

Table 7-3: L1 Context ID Summary

The L1 data cache may be adapted to two strategies: adaptive and shared. The L1 context ID serves as setting the L1 data context strategy.

If set, the L1 Context ID feature is available.

7.1.5 Self Snoop (SS)

CPUID bit	**Intel**	01h.EDX[27]

Table 7-4: Self Snoop Summary

Management of conflicting memory types may be done by performing a snoop of its own cache structure for transactions issued to the bus, thus self snooping.

If set, memory self snooping is available.

7.1.6 PREFETCHW

CPUID bit	8000_0001h.ECX[8]

Table 7-5: PREFETCHW Summary

Fetch data of cache line that contains the specified byte with the source operand to a location and invalidate other cache instances (L1 or L2) for the cache line, and hints a cache write.

If set, this instruction is available.

7.1.7 INVPCID

CPUID bit	**Intel**	07h.EBX[10]

Table 7-6: INVPCID Summary

Invalidate process-context identifier for a PID, such as mappings in the translation lookaside buffers (TLBs), specified in register and memory.

If set, the INVPCID instruction is available.

7.1.8 WBNOINVD

CPUID bit	**Intel**	8000_0008h.EBX[9]

Table 7-7: WBNOINVD Summary

Write back modified cache lines in the processor’s internal cache to main memory and avoid invalidating cache.

If set, the WBNOINVD instruction is available.

7.2 System Features

7.2.1 Advanced Configuration and Power Interface (ACPI)

CPUID bit	01h.EDX[22]

Table 7-8: Advanced Configuration and Power Interface Summary

The Advanced Configuration and Power Interface was introduced to help power and thermal management through the operating system.

If set, the processor currently uses ACPI.

7.2.2 Advanced Programmable Interrupt Controller (APIC)

CPUID bit	01h.EDX[9]

Table 7-9: Advanced Programmable Interrupt Controller Summary

The Advanced Programmable Interrupt Controller is an updated standard from Intel to replace the older 8259-based programmable interrupt controllers (PIC) in multi-processor systems. It is used to redirect interrupts more effectively and also provides the high-resolution local APIC timer on the order of one microsecond or better.

APIC was introduced in the Intel Pentium processors with an external 82489DX APIC.

xAPIC was an extension to APIC introduced in the Intel Pentium 4 processors.

If set, an APIC is present.

Initial ID (CPUID.01h.EBX[31:24]) refers to the initial APIC ID. This typically is the core that the program is being run on. On modern operating systems, this will likely be scheduled depending on the operating system scheduler strategy.

Max ID (CPUID.01h.EBX[23:16]) refers to the maximum APIC ID. This typically is the maximum amount of cores the program can run under. Note that Intel processor may have this value doubled to the logical core count when Hyper-Threading technology is enabled.

x2APIC (Intel: CPUID.01h.ECX[21], AMD: CPUID.8000_0001h.ECX[3]) describes an extension for xAPIC. For more information, see Intel64® Architecture x2APIC Specification

(Reference Number: 318148-004, March 2010), or the AMD BKDG (offset 400h from the "APIC Base Address Register").

7.2.3 Always-Running-APIC-Timer (ARAT)

CPUID bit	06h.EAX[2]

Table 7-10: Always-Running-APIC-Timer Summary

If set, indicates that the timebase for the local APIC timer is not affected by any processor P-states.

7.2.4 Thermal Monitor (TM)

CPUID bit	**Intel**	01h.EDX[29]
	AMD	8000_0007h.EDX[4]

Table 7-11: Thermal Monitor Summary

The first thermal monitor thermally-initiates (on-die) modulations for the stop-clock duty cycle for reduced power consumption.

If set, a first thermal monitor is available.

Requires ACPI feature.

7.2.5 Thermal Monitor 2 (TM2)

CPUID bit	**Intel**	01h.ECX[8]

Table 7-12: Thermal Monitor 2 Summary

The second thermal monitor performs frequency transitions for reduced power consumption.

If set, a second thermal monitor is available.

Requires ACPI feature.

7.3 Virtualization Features

In computing, virtualization consists of recreating a version of computer hardware as virtual. This, for example, allows the host system to create many virtual processors, which in turn can run different operating systems on the same host machine at the same time.

Intel and AMD processors often support hardware-assisted virtualization through technologies such as VT-x and AMD-V. These technologies facilitates the creation of virtual processors by emulating such with a Virtual Machine Monitor (VMM).

7.3.1 Virtual 8086 Mode Enhancements (VME)

CPUID bit	01h.EDX[1]

Table 7-13: Virtual 8086 Mode Enhancements Summary

A number of enhancements were added within the Pentium architecture to the virtual 8086 mode, including virtual interrupts.

If set, the virtual 8086 mode enhancements are available.

7.3.2 APICv

CPUID bit	**AMD**	8000_000Ah.EDX[13]

Table 7-14: APICv Summary

In a virtualization context, a hardware virtual Advanced Programmable Interrupt Controller (APICv) enables hardware interrupt handling in a virtual environment, including guest interrupts. This improves performance by avoiding virtual machine (VM) exits to the Virtual Machine Monitor (VMM) and thus reducing I/O overhead.

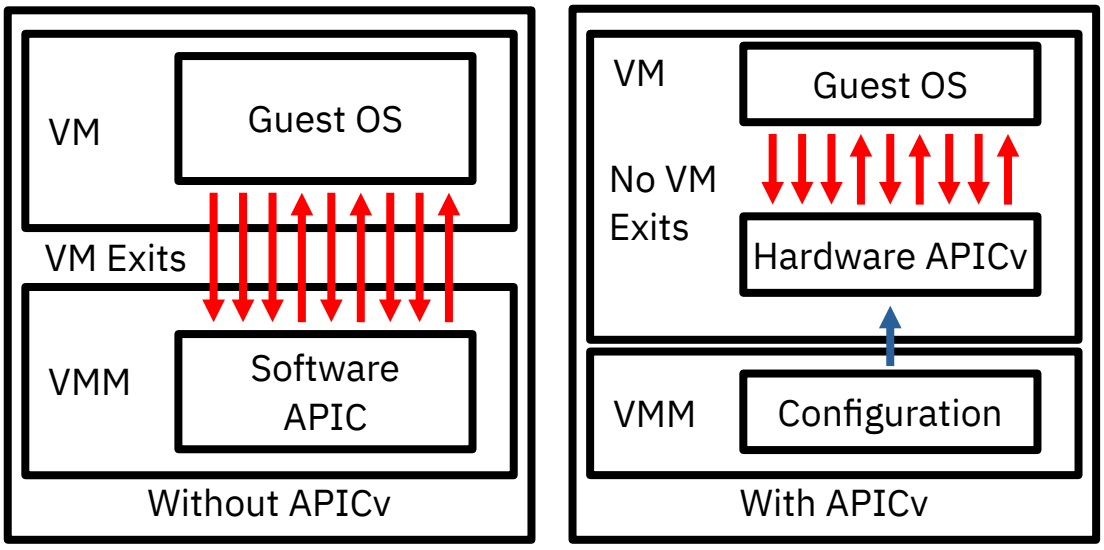

Figure 7-1: VM-VMM Interaction Without and With an APICv

Intel does not advertise this technology through a CPUID bit, but a Model Specific Register (MSR), because this feature is only available outside the processor in the motherboard controller. Not to be confused with VT-d (hardware accelerated IOMMU) and VT-i (Itanium Secure Virtual Machine).

7.3.3 Paravirtualization Features

To emulate a complete computer, hypervisors are required to emulate hardware. In a traditional environment, this means that the hypervisor must trap hardware requests, which may introduce inefficiency.

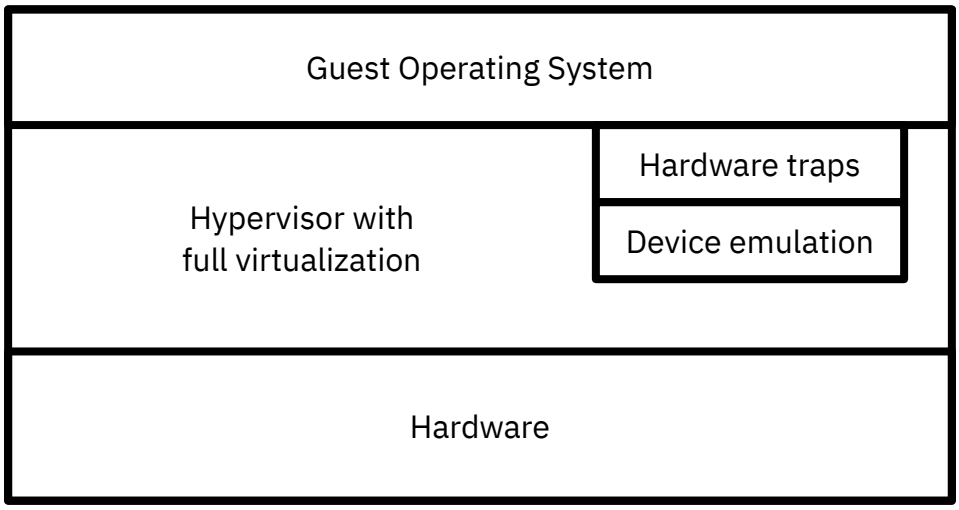

Figure 7-2: Hypervisor with full virtualization

To solve this, paravirtualization was introduced as an abstraction layer residing between the guest operating system and hypervisor. This, with the help of guest additions, makes the OS aware of the hypervisor features and can result in more efficient virtual machine transitions and communications.

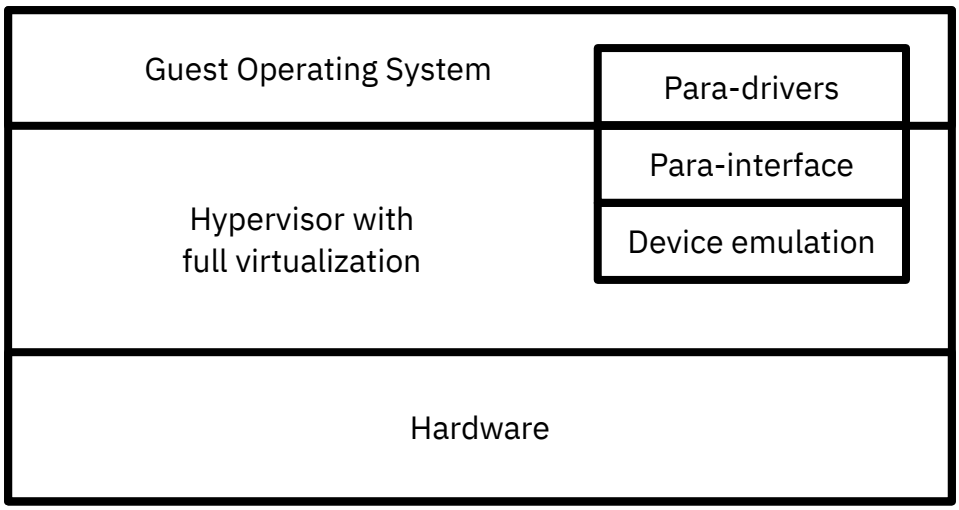

Figure 7-3: Hypervisor with full virtualization and paravirtualization

For para-drivers to detect and use these features, paravirtualization interfaces often emit CPUID information at level 4000_0000h, which details what is available on these para-interfaces.

This chapter is dedicated to present these various interfaces and their CPUID bits emitted.

7.3.3.1 Virtual Vendor Name (HOST)

This field indicates that a supported hypervisor or virtualization software has a vendor name. Every field after this lists vendor-specific features.

The following list shows possible para-interface vendor names.

Vendor String	Vendor	Method	Note
"KVMKVMKVM\0\0\0"	Linux	EBX~ECX~EDX	Linux's built-in Kernel-based Virtual Machine interface.[1]
"Microsoft Hv" "VBoxVBoxVBox"[2]	Microsoft VirtualBox	EBX~ECX~EDX	Hyper-V interface.
(empty)	VirtualBox	N/A	VirtualBox Minimal interface (CPUID.4000_0010h).

1 https://www.kernel.org/doc/html/latest/virt/kvm/cpuid.html

Table 7-15: Supported Virtualization Vendor Strings

7.3.3.2 Linux Kernel Virtual Machine (KVM) Paravirtualization Features

7.3.3.2.1 KVM_FEATURE_CLOCKSOURCE

CPUID bit	4000_0001h.EAX[0]

Table 7-16: KVM_FEATURE_CLOCKSOURCE Summary

If set, the kvmclock interface is available at MSRs 0x11 and 0x12.

7.3.3.2.2 KVM_FEATURE_NOP_IO_DELAY

CPUID bit	4000_0001h.EAX[1]

Table 7-17: KVM_FEATURE_NOP_IO_DELAY Summary

If set, it is not necessary to perform delays on Programmed I/O (PIO) operations (kvm_stat/kvm_pio).

7.3.3.2.3 KVM_FEATURE_MMU_OP

CPUID bit	4000_0001h.EAX[2]

Table 7-18: KVM_FEATURE_MMU_OP Summary

This feature is deprecated, and no further documentation has been provided.

7.3.3.2.4 KVM_FEATURE_CLOCKSOURCE2

CPUID bit	4000_0001h.EAX[3]

Table 7-19: KVM_FEATURE_CLOCKSOURCE2 Summary

If set, the kvmclock interface (remapped) is available at MSRs 0x4b564d00 and 0x4b564d01.

7.3.3.2.5 KVM_FEATURE_ASYNC_PF

CPUID bit	4000_0001h.EAX[4]

Table 7-20: KVM_FEATURE_ASYNC_PF Summary

If set, the Asynchronous Page Fault feature can be enabled via MSR 0x4b564d02.

2 VirtualBox's interface does not report as Hyper-V due to a bug, so its internal ID is translated within the library.

7.3.3.2.6 KVM_FEATURE_STEAL_TIME

CPUID bit	4000_0001h.EAX[5]

Table 7-21: KVM_FEATURE_STEAL_TIME Summary

If set, the steal time feature can be enabled via MSR 0x4b564d03.

7.3.3.2.7 KVM_FEATURE_PV_EOI

CPUID bit	4000_0001h.EAX[6]

Table 7-22: KVM_FEATURE_PV_EOI Summary

If set, the Paravirtualized End Of the Interrupt handler feature can be enabled via MSR 0x4b564d04.

7.3.3.2.8 KVM_FEATURE_PV_UNHAULT

CPUID bit	4000_0001h.EAX[7]

Table 7-23: KVM_FEATURE_PV_UNHAULT Summary

Guest operating systems should test this bit before enabling the paravirtualized spinlock feature.

7.3.3.2.9 KVM_FEATURE_PV_TLB_FLUSH

CPUID bit	4000_0001h.EAX[9]

Table 7-24: KVM_FEATURE_PV_TLB_FLUSH Summary

Guest operating systems should test this bit before enabling the paravirtualized TLB flush feature.

7.3.3.2.10 KVM_FEATURE_ASYNC_PF_VMEXIT

CPUID bit	4000_0001h.EAX[10]

Table 7-25: KVM_FEATURE_ASYNC_PF_VMEXIT Summary

If set, the Asynchronous Page Fault at VM exit feature can be enabled via MSR 0x4b564d02 by setting bit 2 (MSR[2]).

7.3.3.2.11 KVM_FEATURE_PV_SEND_IPI

CPUID bit	4000_0001h.EAX[11]

Table 7-26: KVM_FEATURE_PV_SEND_IPI Summary

Guest operating systems should test this bit before enabling the paravirtualized SEBD inter-processor-interrupt (IPI) feature.

7.3.3.2.12 KVM_FEATURE_PV_POLL_CONTROL

CPUID bit	4000_0001h.EAX[12]

Table 7-27: KVM_FEATURE_PV_POLL_CONTROL Summary

If set, host-side polling on HLT (instruction) can be disabled by writing to MSR 0x4b564d05.

7.3.3.2.13 KVM_FEATURE_PV_SCHED_YIELD

CPUID bit	4000_0001h.EAX[13]

Table 7-28: KVM_FEATURE_PV_SCHED_YIELD Summary

Guest operating systems should test this bit before using the paravirtualized scheduler yield feature.

7.3.3.2.14 KVM_FEATURE_CLOCSOURCE_STABLE_BIT

CPUID bit	4000_0001h.EAX[13]

Table 7-29: KVM_FEATURE_CLOCSOURCE_STABLE_BIT Summary

If set, the host will warn guest if not guest-side per-cpu warps are expected from the kvmclock interface.

7.3.3.2.15 KVM_HINTS_REALTIME

CPUID bit	4000_0001h.EDX[0]

Table 7-30: KVM_HINTS_REALTIME

If set, the guest operating system checks this bit to determine that vCPUs are never preempted for an unlimited amount of time, allowing optimizations.

7.3.3.3 VirtualBox Paravirtualization Features

7.3.3.3.1 TSC_FREQ_KHZ

CPUID bit	4000_0010h.EAX[31:0]

Table 7-31: TSC_FREQ_KHZ Summary

This represents the Timestamp Counter frequency in kilohertz. This applies for the VirtualBox Minimal paravirtualization interface. This typically is the frequency of the processor.

7.3.3.3.2 APIC_FREQ_KHZ

CPUID bit	4000_0010h.EBX[31:0]

Table 7-32: APIC_FREQ_KHZ Summary

This represents the Advanced Programmable Interrupt Controller (APIC) frequency in kilohertz. This applies for the VirtualBox Minimal paravirtualization interface.

7.3.3.4 Hyper-V Paravirtualization Features

7.3.3.4.1 Guest OS ID Structure (MSR_GIM_HV_GUEST_OS_ID)

CPUID bit	4000_0002h.EDX:EAX

Table 7-33: Guest OS ID Structure (MSR_GIM_HV_GUEST_OS_ID) Summary

The MSR_GIM_HV_GUEST_OS_ID MSR, if supported, allows the guest operating system to hint the host operating system about various versions and build IDs. The format (usually residing in EDX:EAX, being 64-bit) is explained below. Based on information from the VirtualBox source located at `src/VMM/include/GIMHvInternal.h`.

```
EDX=B1000001h
    ||||||++-- Major version (HV_GUEST_OS_ID_MAJOR_VERSION)
    ||||++---- OS Variant (HV_GUEST_OS_ID_OS_VARIANT)
    |+++------ Vendor ID (HV_GUEST_OS_ID_VENDOR)
    +-- 1011b
        |+++-- Unused
        +----- Open-source (HV_GUEST_OS_ID_IS_OPENSOURCE)

EAX=00001DB1h
    ||||++++-- Build ID: 7601 (HV_GUEST_OS_ID_BUILD)
    ||++------ Service version (HV_GUEST_OS_ID_SERVICE_VERSION)
    ++-------- Minor version (HV_GUEST_OS_ID_MINOR_VERSION)
```

Figure 7-4: MSR_GIM_HV_GUEST_OS_ID Hyper-V Paravirtualization Interface MSR Example and Description on a Windows 7 Guest using VirtualBox

7.3.3.4.2 HV_BASE_FEAT_VP_RUNTIME_MSR

CPUID bit	4000_0003h.EAX[0]

Table 7-34: HV_BASE_FEAT_VP_RUNTIME_MSR Summary

If set, the virtual processor runtime MSR is available.

7.3.3.4.3 HV_BASE_FEAT_PART_TIME_REF_COUNT_MSR

CPUID bit	4000_0003h.EAX[1]

Table 7-35: HV_BASE_FEAT_PART_TIME_REF_COUNT_MSR Summary

If set, the partition reference counter MSR is available.

7.3.3.4.4 HV_BASE_FEAT_BASIC_SYNIC_MSRS

CPUID bit	4000_0003h.EAX[2]

Table 7-36: HV_BASE_FEAT_BASIC_SYNIC_MSRS Summary

If set, the Basic Synthetic Interrupt Controller MSRs are available.

7.3.3.4.5 HV_BASE_FEAT_STIMER_MSRS

CPUID bit	4000_0003h.EAX[3]

Table 7-37: HV_BASE_FEAT_STIMER_MSRS Summary

If set, the Synthetic Timer MSRs are available.

7.3.3.4.6 HV_BASE_FEAT_APIC_ACCESS_MSRS

CPUID bit	4000_0003h.EAX[4]

Table 7-38: HV_BASE_FEAT_APIC_ACCESS_MSRS Summary

If set, APIC access MSRs (EOI, ICR, TPR) are available.

7.3.3.4.7 HV_BASE_FEAT_HYPERCALL_MSRS

CPUID bit	4000_0003h.EAX[5]

Table 7-39: HV_BASE_FEAT_HYPERCALL_MSRS Summary

If set, the Hypercalls API MSRs are available for use.

For more examples, see the VirtualBox source at `src/VMM/include/GIMHvInternal.h`.

7.3.3.4.8 HV_BASE_FEAT_VP_ID_MSR

CPUID bit	4000_0003h.EAX[6]

Table 7-40: HV_BASE_FEAT_VP_ID_MSR Summary

If set, access to the vCPU index MSR is available.

7.3.3.4.9 HV_BASE_FEAT_VIRT_SYS_RESET_MSR

CPUID bit	4000_0003h.EAX[7]

Table 7-41: HV_BASE_FEAT_VIRT_SYS_RESET_MSR Summary

If set, the virtual system reset MSR is available.

7.3.3.4.10 HV_BASE_FEAT_STAT_PAGES_MSR

CPUID bit	4000_0003h.EAX[8]

Table 7-42: HV_BASE_FEAT_STAT_PAGES_MSR Summary

If set, the statistic pages MSRs are available.

7.3.3.4.11 HV_BASE_FEAT_PART_REF_TSC_MSR

CPUID bit	4000_0003h.EAX[9]

Table 7-43: HV_BASE_FEAT_PART_REF_TSC_MSR Summary

If set, the partition reference timestamp counter MSR is available.

7.3.3.4.12 HV_BASE_FEAT_GUEST_IDLE_STATE_MSR

CPUID bit	4000_0003h.EAX[10]

Table 7-44: HV_BASE_FEAT_GUEST_IDLE_STATE_MSR Summary

If set, the virtual guest idle state MSR is available.

7.3.3.4.13 HV_BASE_FEAT_TIMER_FREQ_MSRS

CPUID bit	4000_0003h.EAX[11]

Table 7-45: HV_BASE_FEAT_TIMER_FREQ_MSRS Summary

If set, the timer frequency MSRs (TSC and APIC) are available.

7.3.3.4.14 HV_BASE_FEAT_DEBUG_MSRS

CPUID bit	4000_0003h.EAX[12]

Table 7-46: HV_BASE_FEAT_DEBUG_MSRS Summary

If set, the debug MSRs are available.

7.3.3.4.15 HV_PART_FLAGS_CREATE_PART

CPUID bit	4000_0003h.EBX[0]

Table 7-47: HV_PART_FLAGS_CREATE_PART Summary

If set, partitions can be created.

7.3.3.4.16 HV_PART_FLAGS_ACCESS_PART_ID

CPUID bit	4000_0003h.EBX[1]

Table 7-48: HV_PART_FLAGS_ACCESS_PART_ID Summary

If set, partitions IDs can be accessed.

7.3.3.4.17 HV_PART_FLAGS_ACCESS_MEMORY_POOL

CPUID bit	4000_0003h.EBX[2]

Table 7-49: HV_PART_FLAGS_ACCESS_MEMORY_POOL Summary

If set, the memory pool can be accessed.

7.3.3.4.18 HV_PART_FLAGS_ADJUST_MSG_BUFFERS

CPUID bit	4000_0003h.EBX[3]

Table 7-50: HV_PART_FLAGS_ADJUST_MSG_BUFFERS Summary

If set, it is possible to adjust message buffers.

7.3.3.4.19 HV_PART_FLAGS_POST_MSGS

CPUID bit	4000_0003h.EBX[4]

Table 7-51: HV_PART_FLAGS_POST_MSGS Summary

If set, it is possible to send messages.

7.3.3.4.20 HV_PART_FLAGS_SIGNAL_EVENTS

CPUID bit	4000_0003h.EBX[5]

Table 7-52: HV_PART_FLAGS_SIGNAL_EVENTS Summary

If set, it is possible to signal events.

7.3.3.4.21 HV_PART_FLAGS_CREATE_PORT

CPUID bit	4000_0003h.EBX[6]

Table 7-53: HV_PART_FLAGS_CREATE_PORT Summary

If set, it is possible to create ports.

7.3.3.4.22 HV_PART_FLAGS_CONNECT_PORT

CPUID bit	4000_0003h.EBX[7]

Table 7-54: HV_PART_FLAGS_CONNECT_PORT Summary

If set, it is possible to connect to ports.

7.3.3.4.23 HV_PART_FLAGS_ACCESS_STATS

CPUID bit	4000_0003h.EBX[8]

Table 7-55: HV_PART_FLAGS_ACCESS_STATS Summary

If set, statistics are available to access.

7.3.3.4.24 HV_PART_FLAGS_DEBUGGING

CPUID bit	4000_0003h.EBX[11]

Table 7-56: HV_PART_FLAGS_DEBUGGING Summary

If set, debugging features are available.

7.3.3.4.25 HV_PART_FLAGS_CPU_MGMT

CPUID bit	4000_0003h.EBX[12]

Table 7-57: HV_PART_FLAGS_CPU_MGMT Summary

If set, processor management features are available.

7.3.3.4.26 HV_PART_FLAGS_CPU_PROFILER

CPUID bit	4000_0003h.EBX[13]

Table 7-58: HV_PART_FLAGS_CPU_PROFILER Summary

If set, processor profiling features are available.

7.3.3.4.27 HV_PART_FLAGS_EXPANDED_STACK_WALK

CPUID bit	4000_0003h.EBX[14]

Table 7-59: HV_PART_FLAGS_EXPANDED_STACK_WALK Summary

If set, extended stack walking features are enabled.

7.3.3.4.28 HV_PART_FLAGS_ACCESS_VSM

CPUID bit	4000_0003h.EBX[16]

Table 7-60: HV_PART_FLAGS_ACCESS_VSM Summary

If set, the virtual system monitor (VSM) is accessible.

7.3.3.4.29 HV_PART_FLAGS_ACCESS_VP_REGS

CPUID bit	4000_0003h.EBX[17]

Table 7-61: HV_PART_FLAGS_ACCESS_VP_REGS Summary

If set, the virtual private registers are accessible.

7.3.3.4.30 HV_PART_FLAGS_EXTENDED_HYPERCALLS

CPUID bit	4000_0003h.EBX[20]

Table 7-62: HV_PART_FLAGS_EXTENDED_HYPERCALLS Summary

If set, the extended hypercalls API are available (>=8000h).

7.3.3.4.31 HV_PART_FLAGS_START_VP

CPUID bit	4000_0003h.EBX[20]

Table 7-63: HV_PART_FLAGS_START_VP Summary

If set, the virtual processor has started.

7.3.3.4.32 HV_PM_MAX_CPU_POWER_STATE_C0

CPUID bit	4000_0003h.ECX[0]

Table 7-64: HV_PM_MAX_CPU_POWER_STATE_C0 Summary

If set, the processor's C0 state is the maximum state.

7.3.3.4.33 HV_PM_MAX_CPU_POWER_STATE_C1

CPUID bit	4000_0003h.ECX[1]

Table 7-65: HV_PM_MAX_CPU_POWER_STATE_C1 Summary

If set, the processor's C1 state is the maximum state.

7.3.3.4.34 HV_PM_MAX_CPU_POWER_STATE_C2

CPUID bit	4000_0003h.ECX[2]

Table 7-66: HV_PM_MAX_CPU_POWER_STATE_C2 Summary

If set, the processor's C2 state is the maximum state.

7.3.3.4.35 HV_PM_MAX_CPU_POWER_STATE_C3

CPUID bit	4000_0003h.ECX[3]

Table 7-67: HV_PM_MAX_CPU_POWER_STATE_C3 Summary

If set, the processor's C3 state is the maximum state.

7.3.3.4.36 HV_PM_HPET_REQD_FOR_C3

CPUID bit	4000_0003h.ECX[4]

Table 7-68: HV_PM_HPET_REQD_FOR_C3 Summary

If set, a high precision event timer (HPET) is required to enter the C3 state.

7.3.3.4.37 HV_MISC_FEAT_MWAIT

CPUID bit	4000_0003h.EDX[0]

Table 7-69: HV_MISC_FEAT_MWAIT Summary

If set, the MWAIT instruction is available to the guest operating system.

7.3.3.4.38 HV_MISC_FEAT_GUEST_DEBUGGING

CPUID bit	4000_0003h.EDX[1]

Table 7-70: HV_MISC_FEAT_GUEST_DEBUGGING Summary

If set, the guest operating system supports debugging.

7.3.3.4.39 HV_MISC_FEAT_PERF_MON

CPUID bit	4000_0003h.EDX[2]

Table 7-71: HV_MISC_FEAT_PERF_MON Summary

If set, the performance monitor support is available.

7.3.3.4.40 HV_MISC_FEAT_PCPU_DYN_PART_EVENT

CPUID bit	4000_0003h.EDX[3]

Table 7-72: HV_MISC_FEAT_PCPU_DYN_PART_EVENT Summary

If set, support for physical CPU dynamic partitioning events is available.

7.3.3.4.41 HV_MISC_FEAT_XMM_HYPERCALL_INPUT

CPUID bit	4000_0003h.EDX[4]

Table 7-73: HV_MISC_FEAT_XMM_HYPERCALL_INPUT Summary

If set, passing hypercall input parameter block via XMM registers is supported.

7.3.3.4.42 HV_MISC_FEAT_GUEST_IDLE_STATE

CPUID bit	4000_0003h.EDX[5]

Table 7-74: HV_MISC_FEAT_GUEST_IDLE_STATE Summary

If set, virtual guest supports idle state.

7.3.3.4.43 HV_MISC_FEAT_HYPERVISOR_SLEEP_STATE

CPUID bit	4000_0003h.EDX[6]

Table 7-75: HV_MISC_FEAT_HYPERVISOR_SLEEP_STATE Summary

If set, hypervisor supports sleep state.

7.3.3.4.44 HV_MISC_FEAT_QUERY_NUMA_DISTANCE

CPUID bit	4000_0003h.EDX[7]

Table 7-76: HV_MISC_FEAT_QUERY_NUMA_DISTANCE Summary

If set, querying NUMA distances is supported.

7.3.3.4.45 HV_MISC_FEAT_TIMER_FREQ

CPUID bit	4000_0003h.EDX[8]

Table 7-77: HV_MISC_FEAT_TIMER_FREQ Summary

If set, determining timer frequencies is supported.

7.3.3.4.46 HV_MISC_FEAT_INJECT_SYNMC_XCPT

CPUID bit	4000_0003h.EDX[9]

Table 7-78: HV_MISC_FEAT_INJECT_SYNMC_XCPT Summary

Support for injecting synthetic machine checks.

7.3.3.4.47 HV_MISC_FEAT_GUEST_CRASH_MSRS

CPUID bit	4000_0003h.EDX[10]

Table 7-79: HV_MISC_FEAT_GUEST_CRASH_MSRS Summary

If set, guest crash MSRs are supported.

7.3.3.4.48 HV_MISC_FEAT_DEBUG_MSRS

CPUID bit	4000_0003h.EDX[11]

Table 7-80: HV_MISC_FEAT_DEBUG_MSRS Summary

If set, the debug MSRs are supported.

7.3.3.4.49 HV_MISC_FEAT_NPIEP1

CPUID bit	4000_0003h.EDX[12]

Table 7-81: HV_MISC_FEAT_NPIEP1 Summary

No documentation is available for this field.

7.3.3.4.50 HV_MISC_FEAT_DISABLE_HYPERVISOR

CPUID bit	4000_0003h.EDX[13]

Table 7-82: HV_MISC_FEAT_DISABLE_HYPERVISOR Summary

If set, the hypervisor can be disabled.

7.3.3.4.51 HV_MISC_FEAT_EXT_GVA_RANGE_FOR_FLUSH_VA_LIST

CPUID bit	4000_0003h.EDX[14]

Table 7-83: HV_MISC_FEAT_EXT_GVA_RANGE_FOR_FLUSH_VA_LIST Summary

If set, the extended guest virtual address (GVA) ranges for FlushVirtualAddressList are available.

7.3.3.4.52 HV_MISC_FEAT_HYPERCALL_OUTPUT_XMM

CPUID bit	4000_0003h.EDX[15]

Table 7-84: HV_MISC_FEAT_HYPERCALL_OUTPUT_XMM Summary

If set, returning hypercall output via XMM registers is supported.

7.3.3.4.53 HV_MISC_FEAT_SINT_POLLING_MODE

CPUID bit	4000_0003h.EDX[17]

Table 7-85: HV_MISC_FEAT_SINT_POLLING_MODE Summary

If set, the synthetic interrupt source polling mode is available.

7.3.3.4.54 HV_MISC_FEAT_HYPERCALL_MSR_LOCK

CPUID bit	4000_0003h.EDX[18]

Table 7-86: HV_MISC_FEAT_HYPERCALL_MSR_LOCK Summary

If set, the Hypercall MSR lock feature is available.

7.3.3.4.55 HV_MISC_FEAT_USE_DIRECT_SYNTH_MSRS

CPUID bit	4000_0003h.EDX[19]

Table 7-87: HV_MISC_FEAT_USE_DIRECT_SYNTH_MSRS Summary

If set, it is possible to directly use the synthetic MSRs.

7.3.3.4.56 HV_HINT_HYPERCALL_FOR_PROCESS_SWITCH

CPUID bit	4000_0004h.EAX[0]

Table 7-88: HV_HINT_HYPERCALL_FOR_PROCESS_SWITCH Summary

If set, the hint implies that the guest should use the Hypercall API for address space switches rather than MOV CR3.

7.3.3.4.57 HV_HINT_HYPERCALL_FOR_TLB_FLUSH

CPUID bit	4000_0004h.EAX[1]

Table 7-89: HV_HINT_HYPERCALL_FOR_TLB_FLUSH Summary

If set, the hint implies that the guest should use the Hypercall API for local TLB flushes rather than INVLPG/MOV CR3.

7.3.3.4.58 HV_HINT_HYPERCALL_FOR_TLB_SHOOTDOWN

CPUID bit	4000_0004h.EAX[2]

Table 7-90: HV_HINT_HYPERCALL_FOR_TLB_SHOOTDOWN Summary

If set, the hint implies that the guest should use the Hypercall API for inter-CPU TLB flushes rather than inter-processor-interrupts (IPIs).

7.3.3.4.59 HV_HINT_MSR_FOR_APIC_ACCESS

CPUID bit	4000_0004h.EAX[3]

Table 7-91: HV_HINT_MSR_FOR_APIC_ACCESS Summary

If set, the hint implies that the guest should use the MSRs for APIC access (EOI, ICR, TPR) rather than memory-mapped input/output (MMIO).

7.3.3.4.60 HV_HINT_MSR_FOR_SYS_RESET

CPUID bit	4000_0004h.EAX[4]

Table 7-92: HV_HINT_MSR_FOR_SYS_RESET Summary

If set, the hint implies that the guest should use the hypervisor-provided MSR for a system reset instead of traditional methods.

7.3.3.4.61 HV_HINT_RELAX_TIME_CHECKS

CPUID bit	4000_0004h.EAX[5]

Table 7-93: HV_HINT_RELAX_TIME_CHECKS Summary

If set, the hint implies that the guest should relax timer-related checks (watchdogs/deadman timeouts) that rely on timely deliver of external interrupts.

7.3.3.4.62 HV_HINT_DMA_REMAPPING

CPUID bit	4000_0004h.EAX[6]

Table 7-94: HV_HINT_DMA_REMAPPING Summary

If set, the hint implies that the guest should use the direct memory access (DMA) remapping.

7.3.3.4.63 HV_HINT_INTERRUPT_REMAPPING

CPUID bit	4000_0004h.EAX[7]

Table 7-95: HV_HINT_INTERRUPT_REMAPPING Summary

If set, the hint implies that the guest should use the interrupt remapping.

7.3.3.4.64 HV_HINT_X2APIC_MSRS

CPUID bit	4000_0004h.EAX[8]

Table 7-96: HV_HINT_X2APIC_MSRS Summary

If set, the hint implies that the guest should use the X2APIC MSRs rather than memory mapped input/output (MMIO).

7.3.3.4.65 HV_HINT_DEPRECATE_AUTO_EOI

CPUID bit	4000_0004h.EAX[9]

Table 7-97: HV_HINT_DEPRECATE_AUTO_EOI Summary

If set, the hint implies that the guest should deprecate Auto EOI (End Of Interrupt) features.

7.3.3.4.66 HV_HINT_SYNTH_CLUSTER_IPI_HYPERCALL

CPUID bit	4000_0004h.EAX[10]

Table 7-98: HV_HINT_SYNTH_CLUSTER_IPI_HYPERCALL Summary

If set, the hint implies that the guest should use the SyntheticClusterIpi Hypercall.

7.3.3.4.67 HV_HINT_EX_PROC_MASKS_INTERFACE

CPUID bit	4000_0004h.EAX[11]

Table 7-99: HV_HINT_EX_PROC_MASKS_INTERFACE Summary

If set, the hint implies that the guest should use the newer ExProcessMasks interface over ProcessMasks.

7.3.3.4.68 HV_HINT_NESTED_HYPERV

CPUID bit	4000_0004h.EAX[12]

Table 7-100: HV_HINT_NESTED_HYPERV Summary

If set, the hint implies that a Hyper-V instance is nested within a Hyper-V partition.

7.3.3.4.69 HV_HINT_INT_FOR_MBEC_SYSCALLS

CPUID bit	4000_0004h.EAX[13]

Table 7-101: HV_HINT_INT_FOR_MBEC_SYSCALLS Summary

If set, the hint implies that the guest should use the INT instruction for Mode Based Execution Control (MBEC) system calls.

7.3.3.4.70 HV_HINT_NESTED_ENLIGHTENED_VMCS_INTERFACE

CPUID bit	4000_0004h.EAX[14]

Table 7-102: HV_HINT_NESTED_ENLIGHTENED_VMCS_INTERFACE Summary

If set, the hint implies that the guest should use enlightened Virtual Machine Control Structure (VMCS) interfaces and nested enlightenment.

7.3.3.4.71 HV_HOST_FEAT_AVIC

CPUID bit	4000_0006h.EAX[0]

Table 7-103: HV_HOST_FEAT_AVIC Summary

If set, the hypervisor is using the Advanced Virtual Interrupt Controller (AVIC) overlay.

7.3.3.4.72 HV_HOST_FEAT_MSR_BITMAP

CPUID bit	4000_0006h.EAX[1]

Table 7-104: HV_HOST_FEAT_MSR_BITMAP Summary

If set, the hypervisor is using MSR bitmaps.

7.3.3.4.73 HV_HOST_FEAT_PERF_COUNTER

CPUID bit	4000_0006h.EAX[2]

Table 7-105: HV_HOST_FEAT_PERF_COUNTER Summary

If set, the hypervisor supports the architectural performance counter.

7.3.3.4.74 HV_HOST_FEAT_NESTED_PAGING

CPUID bit	4000_0006h.EAX[3]

Table 7-106: HV_HOST_FEAT_NESTED_PAGING Summary

If set, the hypervisor is using nested paging.

7.3.3.4.75 HV_HOST_FEAT_DMA_REMAPPING

CPUID bit	4000_0006h.EAX[4]

Table 7-107: HV_HOST_FEAT_DMA_REMAPPING Summary

If set, the hypervisor is using direct memory access (DMA) remapping.

7.3.3.4.76 HV_HOST_FEAT_INTERRUPT_REMAPPING

CPUID bit	4000_0006h.EAX[5]

Table 7-108: HV_HOST_FEAT_INTERRUPT_REMAPPING Summary

If set, the hypervisor is using interrupt remapping.

7.3.3.4.77 HV_HOST_FEAT_MEM_PATROL_SCRUBBER

CPUID bit	4000_0006h.EAX[6]

Table 7-109: HV_HOST_FEAT_MEM_PATROL_SCRUBBER Summary

If set, the hypervisor's memory patrol scrubber is present. A memory scrubber is in charge of reading memory, correct any bit errors (if any) with error-correcting code (ECC), and write the corrected data back to the same location.

7.3.3.4.78 HV_HOST_FEAT_DMA_PROT_IN_USE

CPUID bit	4000_0006h.EAX[7]

Table 7-110: HV_HOST_FEAT_DMA_PROT_IN_USE Summary

If set, the hypervisor is using direct memory access (DMA) protection.

7.3.3.4.79 HV_HOST_FEAT_HPET_REQUESTED

CPUID bit	4000_0006h.EAX[8]

Table 7-111: HV_HOST_FEAT_HPET_REQUESTED Summary

If set, the hypervisor requires a High Precision Event Timer (HPET).

7.3.3.4.80 HV_HOST_FEAT_STIMER_VOLATILE

CPUID bit	4000_0006h.EAX[9]

Table 7-112: HV_HOST_FEAT_STIMER_VOLATILE Summary

If set, the hypervisor's synthetic timers are volatile.

7.4 Memory features

7.4.1 Physical and Linear Address Size (PhysicalBits/LinearBits)

CPUID bit	PhysicalBits: 8000_0008h.EAX[7:0] LinearBits: 8000_0008h.EAX[15:8]

Table 7-113: Physical and Linear Address Size Summary

PhysicalBits: The physical memory address size width, represented in bits, is the maximum physical address size that the processor supports. For example, a value of 36 bits means that the processor supports up to 2^{36} bytes (64 GiB) of physical memory.

LinearBits: The linear memory address size width, represented in bits, is the maximum virtual address size that the processor supports. For example, a value of 48 bits means that the processor supports up to 2^{48} bytes (256 TiB) of virtual memory.

7.4.2 Physical Address Extension (PAE)

CPUID bit	01h.EDX[6]

Table 7-114: Physical Address Extension Summary

If set, this extension introduces 3-level paging by squeezing the highest 2 bits for the four Page Directory Pointer Table Entry (PDPTE) registers. This feature is useful for 32-bit operations systems that need to translate, and reach, memory addresses above 4 GiB.

This also adds support for 2 MiB pages by disabling the Page Table field, leaving 21 bits for the memory offset alone. To enable this, the Page Directory Entry (PDE) must have the PS bit set.

7.4.3 1 GiB Pages (Page1GB)

CPUID bit	8000_0001h.EDX[26]

Table 7-115: 1 GiB Paging Summary

If set, the processor supports 1 GiB pages by removing the Page Directory and Page Table fields, allowing memory offsets of 30 bits. To enable this, the Page Directory Pointer Table Entry (PDPTE) PS bit must be set.

This feature was introduced with x86-64.

7.4.4 Page Size Extension (PSE)

CPUID bit	01h.EDX[3]

Table 7-116: Page Size Extension Summary

If set, the processor supports 4 MiB pages by removing the Page Table field. This feature was introduced in the Intel Pentium processor.

If PAE is used, the page size is reduced to 2 MiB, and PSE is always enabled regardless of the previous value in CR4.PSE.

7.4.5 36-bit Page Size Extension (PSE-36)

CPUID bit	01h.EDX[17]

Table 7-117: 36-bit Page Size Extension Summary

If set, the processor supports 36-bit linear addresses up to 64 GiB by adding 4 additional bits in the Page Directory field. When x86-64 was introduced, this was further extended to 40 bits (1 TiB) when operating in legacy mode. This feature may also be known as ESMA (Intel Extended Server Memory Architecture).

Most operating systems do not support this extension.

7.4.6 Intel XD and AMD EVP (NX)

CPUID bit	8000_0001h.EDX[20]

Table 7-118: NX bit Summary

The No Execute bit serves to mark a page of memory as not to be executed by the processor.

This feature was introduced in x86-64 (AMD64).

If set, this memory feature is available.

7.4.7 Memory Type Range Registers (MTRR)

CPUID bit	01h.EDX[12]

Table 7-119: Memory Type Range Registers Summary

A set of additional control registers to fine-tune memory regions that should be cached by the processor.

If set, these specific control registers are available.

7.4.8 Page Attribute Table (PAT)

CPUID bit	01h.EDX[16]

Table 7-120: Page Attribute Table Summary

The page attribute table allows users to control attributes on a per-page basis. For example, marking memory as should-be cached.

If set, the processor supports the page attribute table feature.

7.4.9 Page Global Bit (PGE)

CPUID bit	01h.EDX[13]

Table 7-121: Page Global Bit Summary

The global bit in a page table is used to prevent the TLB from updating the address in cache if CR3 is reset.

If set, indicates support for the global bit in the page table.

7.4.10 Direct Cache Access (DCA)

CPUID bit	**Intel**	01h.ECX[18]

Table 7-122: Direct Cache Access Summary

Direct cache access is an I/O technology that permits other devices to directly place data into the processor's cache.

If set, direct cache access is available for other devices to use.

7.4.11 Hardware Lock Elision (HLE)

CPUID bit	**Intel**	07h.EBX[4]
Instructions	• XAQUIRE • XRELEASE • XTEST See other instructions with the LOCK prefix	

Table 7-123: Hardware Lock Elision Summary

Part of the Intel® TSX extension specifications proposed in 2012 and introduced in the Haswell family of Intel Core processors in 2013, the Hard Lock Elision prefix bytes (F2h (REP) and F3h (REPE)) are used to better hint thread transactions.

The AMD equivalent, Advanced Synchronization Facility (ASF), has yet to be implemented in any AMD processor to this date.

More information can be found at Intel TSX.

7.4.12 Restricted Transactional Memory (RTM)

CPUID bit	**Intel**	07h.EBX[11]
Instructions	• XBEGIN • XABORT • XEND • XTEST	

Table 7-124: Restricted Transactional Memory Summary

Part of the Intel® TSX extension, Restricted Transactional Memory features new instructions for safer shared multi-threaded access management.

More information can be found at Intel TSX.

7.4.13 TSX Suspend Load Address Tracking (TSXLDTRK)

CPUID bit	**Intel**	07h.EDX[16]
Instructions	• XSUSLDTRK • XRESLDTRK	

Table 7-125: TSX Suspend Load Address Tracking Summary

This extension permits which memory accesses do not need to be tracked in the TSX read table.

Intel specifies the use of the instructions as follow.

> *The XSUSLDTRK instruction specifies the start of a suspend region (addresses of subsequent loads will not be added to the transaction read set), and the XRESLDTRK instruction specifies the end of a suspend region (addresses of subsequent loads will be added to the transaction read set).*

If set, XSUSLDTRK and XRESLDTRK are available instructions.

7.4.14 Supervisor Mode Execution Protection (SMEP)

CPUID bit	07h.EBX[7]

Table 7-126: Supervisor Mode Execution Protection Summary

Instruction execution from user-mode addresses may be restricted in privileged addresses. SMEP has been filed as a patent by Intel Corporation. (International Publication Number: 2013/101059 A1, International Application Number: PCT/US2011/067838)

If set, supervisor mode execution protection is available (CR4.SMEP).

7.4.15 Supervisor Mode Access Protection (SMAP)

CPUID bit	**Intel**	07h.EBX[20]

Table 7-127: Supervisor Mode Access Protection Summary

Instruction fetches from user-mode addresses may be restricted in privileged addresses.

If set, supervisor mode execution protection is available (CR4.SMAP).

7.4.16 Protection Key Units (PKU)

CPUID bit	**Intel**	07h.ECX[3]

Table 7-128: Protection Key Units Summary

Protection keys provide an additional mechanism to control access to user-mode addresses when using 4-level pages. CR4.PKE controls this features.

If set, protection keys are available to use with the PKRU register.

7.4.17 5-Level Paging (5PL, LA57)

Introduced	**Intel**	2019 with Ice Lake
	AMD	2022 with Genoa (EPYC 9004)
CPUID bit	**Intel** **AMD**	07h.ECX[16]

Table 7-129: 5-Level Paging Summary

The AMD64 extension introduced 4-level paging with a maximum limit of 48 bits (256 TiB) of linear memory. 5-level paging extends this limit to 57 bits (128 PiB) of linear memory and 4 PiB of physical memory by introducing an additional level paging level called PML5 (5-level EPT).

This feature is referenced as LA57 in Intel manuals. Intel published its *5-Level Paging and 5-Level EPT Whitepaper Revision 1.1* about this topic in 2017.

It is important to note that the Linux kernel, since version 4.14, implemented 5-level paging, and emulates page levels if missing.

If set, 5-level paging and the 5th level Extended Paging Table are available.

7.4.18 Fast Short REP MOVSB (FSRM)

CPUID bit	**Intel**	07h.ECX[4]
Instructions	• (REP) MOVSB • (REP) STOSB	

Table 7-130: Fast Short REP MOVSB/STOSB Summary

This feature enhances string copy performance of lengths between 1 and 128 bytes long by directly copying memory data.

FSRM implies Enhanced REP MOVSB/STOSB (ERMS), which improved byte transfer rates.

The Fast Short REP MOVSB feature was introduced in the Ice Lake architecture (2019).

For more information, please consult the Intel® 64 and IA-32 Architectures Optimization Reference Manual, §3.7.6.1.

7.4.19 Linear Address Masking (LAM)

CPUID bit	01h.EDX[14]

Table 7-131: Linear Address Masking Summary

Linear Address Masking is an extension to the 5-level paging feature that allows the use of untranslated address bits for metadata. It is only supported in 64-bit mode.

The upper address bits are reserved through the concept of *canonicality*. Such as the upper bits of a 48-bit address (63:49) can be reserved for metadata. The LAM extension allows users to use these addresses with metadata without having to manually mask the linear addresses.

For example, LAM48 (4-level paging) can be enabled for user pointers if bit 63 and 47 in the linear address are unset. This allows bits 62:48 to be used even with user metadata.

The CR4[61] (LAM_U48), CR3[62] (LAM_U57), and CR4[28] (LAM_SUP) bits can be set to control the LAM behavior.

If set, this feature is available.

7.5 Debugging Features

7.5.1 Machine Check Architecture (MCA)

CPUID bit	01h.EDX[14]

Table 7-132: Machine Check Architecture Summary

Mechanism that enables hardware error reporting to the operating system.

If set, machine check architecture is available.

7.5.2 Machine Check Exception (MCE)

CPUID bit	01h.EDX[7]

Table 7-133: Machine Check Exception Summary

On hardware errors, the processor may throw a machine-check exception, which may include errors about system buses errors, ECC errors, parity errors, etc.

This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns.

If set, the processor supports machine-check exceptions.

7.5.3 Debugging Extensions (DE)

CPUID bit	01h.EDX[2]

Table 7-134: Debugging Extensions Summary

Defines support for I/O breakpoints.

If set, the processor supports I/O breakpoints.

7.5.4 Debug Store (DS)

CPUID bit	**Intel**	01h.EDX[21]

Table 7-135: Debug Store Summary

A debug store is a processor feature where the processor is able to write debugging information to a memory buffer.

If set, the processor may use the debug store.

7.5.5 Debug Store CPL (DS-CPL)

CPUD bit	**Intel**	01h.ECX[4]

Table 7-136: Debug Store CPL Summary

The processor may supported the extensions to the debug store feature allowing for branch messages storage qualified by the CPL (Current Privilege Level).

If set, indicates support for debug store CPL message branching.

7.5.6 64-bit DS Area (DTES64)

CPUID bit	**Intel**	01h.ECX[2]

Table 7-137: 64-bit DS Area Summary

Allows the use of 64-bit addresses in the debug store area.

If set, the debug store area can be used using 64-bit addresses.

7.5.7 Perfmon And Debug Capability (PDCM)

CPUID bit	**Intel**	01h.ECX[15]

Table 7-138: Perfmon And Debug Capability Summary

Indicates performance and debug feature indication in the IA32_PERF_CAPABILITIES MSR.

If set, IA32_PERF_CAPABILITIES MSR is available for use.

7.5.8 Silicon Debug (SDBG)

CPUID bit	**Intel**	01h.ECX[11]

Table 7-139: Silicon Debug Summary

Firmware software may use the IA32_DEBUG_INTERFACE MSR for silicon debugging.

If set, silicon debugging is available via the IA32_DEBUG_INTERFACE MSR.

7.5.9 Pending Break Enable (PBE)

CPUID bit	**Intel**	01h.EDX[31]

Table 7-140: Pending Break Enable Summary

In interrupt handling, the processor may use FERR# and PBE# pins in a stop-clock state (STPCLK#) that an interrupt in pending and that the processor should return to normal.

If set, FERR# and PBE# pins are supposed in a stop-clock state.

7.6 Security Features

Security and mitigation related features are listed below.

7.6.1 IA32_ARCH_CAPABILITIES

CPUID bit	**Intel**	07h(ECX=0h).EDX[29]

Tableau 7-141: IA32_ARCH_CAPABILITIES Summary

The IA32_ARCH_CAPABILITIES MSR (address 10Ah) contains additional information on available security mitigations.

If set, this Model Specific Register is available.

7.6.2 Indirect Branch Predictor Barrier (IBPB)

Introduced	2018	
CPUID bit	**Intel**	07h(ECX=0h).EDX[26]
	AMD	8000_0008h.EBX[12]

Table 7-142: Indirect Branch Predictor Barrier Summary

Related to the Spectre vulnerability, Indirect Branch Predictor Barrier is an indirect branch control mechanism, establishing a barrier to prevent software executing before the barrier to control the predicted targets of indirect branches executed after the barrier on the same logical processor.

This feature can be toggled in the IA32_SPEC_CTRL MSR.

7.6.3 Indirect Branch Restricted Speculation (IBRS)

Introduced	2018	
CPUID bit	**Intel**	07h(ECX=0h).EDX[26]
	AMD	8000_0008h.EBX[14]

Table 7-143: Indirect Branch Restricted Speculation Summary

Related to the Spectre vulnerability, Indirect Branch Restricted Speculation is a mechanism that restricts speculation of indirect branches. This feature can be toggled in IA32_SPEC_CTRL.

(Intel) IBRS_ON indicates that IBRS is always enabled (Enhanced IBRS). This feature can be detected via the IA32_ARCH_CAPABILITIES[1] MSR.

(AMD) IBRS_ON (CPUID.8000_0008.EBX[16]) indicates that IBRS is always enabled.

(AMD) IBRS_PREF (CPUID.8000_0008.EBX[18]) indicates that the processor will prefer IBRS over software mitigation, such as the reptoline software mitigation.

7.6.4 Single Thread Indirect Branch Predictors (STIBP)

Introduced	2018	
CPUID bit	**Intel**	07h(ECX=0h).EDX[27]
	AMD	8000_0008h.EBX[15]

Table 7-144: Single Thread Indirect Branch Predictors Summary

Related to the Spectre vulnerability, Single Thread Indirect Branch Predictors is an indirect branch control mechanism that restricts sharing branch predictions between logical processors on a physical core.

This feature can be toggled in IA32_SPEC_CTRL (MSR).

(AMD) STIBP_ON (CPUID.8000_0008.EBX[17]) indicates that STIBP is always enabled.

7.6.5 Speculative Store Bypass Disable (SSDB)

Introduced	2018	
CPUID bit	**Intel**	07h(ECX=0h).EDX[31]
	AMD	8000_0008h.EBX[24]

Table 7-145: Speculative Store Bypass Disable Summary

Relating to the Spectre vulnerability, Speculative Store Bypass Disable delays speculative execution of a load address until the addresses for all older stores are known.

Developers are encouraged to look at the IA32_SPEC_CTRL MSR (Intel) and VIRT_SPEC_CTRL MSR (AMD) for more information about additional security features.

7.6.6 L1D_FLUSH

Introduced	**Intel**	2018
CPUID bit	**Intel**	07h(ECX=0h).EDX[28]

Table 7-146: L1D_FLUSH Summary

Relating to the Foreshadow vulnerability (L1 Terminal Fault, L1TF), L1D_FLUSH indicates that the processor flushes and invalidates data in the L1 data cache when required.

Developers are encouraged to peek in IA32_ARCH_CAPABILITIES (MSR) (CPUID.07h(ECX=0h).EDX[29]) for more information about additional security features.

AMD stated "we believe our processors are not susceptible to these new speculative execution attack variants" (https://www.amd.com/en/corporate/security-updates).

7.6.7 MD_CLEAR

Introduced	**Intel**	2019
CPUID bit	**Intel**	07h(ECX=0h).EDX[10]

Table 7-147: MD_CLEAR Summary

Relating to the Microarchitectural Data Samping (MDS) side-channel vulnerability, indicating if the processor will overwrite buffer values affected by MDS if this correction is applied.

AMD processors seem not to be affected by MDS.

7.6.8 Indirect Branch Tracking (CET_IBT)

Introduced	**Intel**	2018
CPUID bit	**Intel**	07h(ECX=0h).EDX[20]

Table 7-148: CET_IBT Summary

Part of Intel's Control-flow Enforcement Technology (CET), the Indirect Branch Tracking (IBT) is a mechanism that tracks past branch taken to fend off return-oriented programming (ROP) and call-jump-oriented programming (COP/JOP) attacks[1].

1 https://phoronix.com/scan.php?page=news_item&px=Intel-CET-IBT-SHSTK-glibc

If set, the processor supports CET indirect branch tracking. Processors define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs.

7.6.9 Shadow Stack (CET_SS)

Introduced	**Intel**	2020 with Tiger Lake
CPUID bit	**Intel**	07h(ECX=0h).ECX[7]

Table 7-149: CET_SS Summary

Intel's Control-flow Enforcement Technology (CET) Shadow Stack (SS) extension is a mechanism capable of defending user programs against return-oriented programming (ROP) malware attacks.[1]

If set, this feature is enabled. To enable it, define bits 1:0 of the IA32_U_CET and IA32_S_CET Model-Specific Registers.

7.7 Miscellaneous Features

7.7.1 Processor Type

CPUID bits for the processor type is a deprecated feature in Intel processors and may show within the following values:

- "`Original`" (Original OEM Processor)
- "`OverDrive`" (Intel OverDrive Processor)
- "`Dual`" (Dual Processor)
- "`Reserved`" (Reserved)

All recent Intel processors show "Original". AMD does not support this feature and shows "Original".

Intel OverDrive processors, such as the Intel i486SX, i486SX2, i486DX2, and i486DX4, were marked as upgrade processors. Usually fitted in an Intel OverDrive socket or as a co-processor on a compatible socket. These processors features different pinout layouts, or voltage handling compared to the standard processors (e.g. i486DX).

1 https://docs.microsoft.com/en-us/cpp/build/reference/cetcompat?view=msvc-160

7.7.2 Brand Index

CPUID bits	01h.EBX[7:0]

Table 7-150: Brand Index Summary

Before the processor brand string, an index in a list of processor strings was used instead. For example, a brand index of 04h indicates "Intel(R) Pentium(R) III processor".

If cleared, the processor does not support the brand index identification, and the Processor Brand String should be used instead.

Any values higher than 18h and higher are reserved (Intel). AMD recommends seeing the appropriate processor revision guide to program the processor name string, as it never supported this brand index.

Below is the Intel brand string table.

Index (hex)	Brand String
00H	Not supported
01H	Intel(R) Celeron(R) processor
02H	Intel(R) Pentium(R) III processor
03H	Intel(R) Pentium(R) III Xeon(R) processor; If processor signature = 000006B1h, then Intel(R) Celeron(R) processor
04H	Intel(R) Pentium(R) III processor
06H	Mobile Intel(R) Pentium(R) III processor-M
07H	Mobile Intel(R) Celeron(R) processor
08H	Intel(R) Pentium(R) 4 processor
09H	Intel(R) Pentium(R) 4 processor
0AH	Intel(R) Celeron(R) processor
0BH	Intel(R) Xeon(R) processor; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor MP
0CH	Intel(R) Xeon(R) processor MP
0EH	Mobile Intel(R) Pentium(R) 4 processor-M; If processor signature = 0xf13h, then Intel(R) Xeon(R) processor

0FH	Mobile Intel(R) Celeron(R) processor
11H	Mobile Genuine Intel(R) processor
12H	Intel(R) Celeron(R) M processor
13H	Mobile Intel(R) Celeron(R) processor
14H	Intel(R) Celeron(R) processor
15H	Mobile Genuine Intel(R) processor
16H	Intel(R) Pentium(R) M processor
17H	Mobile Intel(R) Celeron(R) processor
18H-FFH	Reserved

Table 7-151: Intel Processor Brand Strings

7.7.3 xTPR Update Control (xTPR)

CPUID bit	**Intel**	01h.ECX[14]

Table 7-152: xTPR Update Control Summary

Intel has yet to disclose information about xTPR Update Control.

If set, the processor supports changing IA32_MISC_ENABLES[23].

7.7.4 Process-Context Architecture (PCID)

CPUID bit	**Intel**	01h.ECX[17]
Instructions	• INVPCID (CPUID.07h.EBX[10])	

Table 7-153: Process-Context Architecture Summary

Enables up to 4,096 processes to be created a managed with a unique ID (PCID) via the processor.

If set, the PCID feature is available.

7.7.5 Processor Serial Number (PSN)

CPUID bit	**Intel**	01h.EDX[18]

Table 7-154: Processor Serial Number Summary

Introduced only in the Pentium III, the processor serial number is supposed to be a unique serial number per processor packages.

If set, the processor has a serial number.

7.7.6 FSGSBASE

CPUID bit	07h.EBX[0]

Table 7-155: FSGSBASE Summary

Enables FS and GS registers write and read support. Under Windows® and Linux®, the GS register is used for Thread Local Storage.

If set, RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE are available instructions to use.

7.7.7 User Interrupts (UINTR)

CPUID bit	**Intel**	07h(ECX=00h).EBX[0]
Instructions	• UINTR (User-Interrupt) • UIRET (User-Interrupt Return)	

Table 7-156: User Interrupts (UINTR) Summary

This feature permits user-defined interrupts when running in 64-bit mode with the CPL (Current Privilege Level) set to 3 (user code), as new events in the architecture.

If set, the UINTR and UIRET instructions are available.

For more information, consult the Intel® Architecture Instruction Set Extensions and Future Features Programming Reference (December 2020) manual chapter 11.

Appendix A
Alphabetical Index

Appendix B
Microarchitecture Release Dates

The following tables represents a list of microarchitectures released over time from Intel and AMD.

These tables are vastly simplified. For further information, feel free to visit Wikipedia (List of Intel CPU microarchitectures, List of AMD CPU microarchitectures), and WikiChip (List of Intel processor families, List of AMD microarchitectures).

B.1. Intel microarchitectures

Year Released	Microarchitecture	Lithography (nm)	Notes
1979	8086	3000	8088 used in the IBM 5150 PC (1981).
1982	80186	3000	i186.
	80286	1500	i286, introduced Protected Mode.
1985	80386	1500	i386, introduced 32-bit mode.
1989	80486	1000	i486.
1993	P5	800, 600, 350	80586, Pentium.
1995	P6	500, 350, 250	80686, Pentium Pro, Pentium II.
1999	P6	250, 180, 130	(Copper Mine) Pentium III.
2000	NetBurst	180	(Willamette) Pentium 4.
2002	NetBurst	130	(Northwood, Gallatin) Pentium 4.
2003	NetBurst	130, 90, 65	Pentium M.
2004	NetBurst	90	(Prescott) Pentium 4, introduced EM64T.
2006	Core	65	Core 2 Duo, EM64T → Intel64.

Year Released	Microarchitecture	Lithography (nm)	Notes
2007	Penryn	45	Core 2 Quad.
2008	Nehalem	45	Core ix-7xx/8xx/9xx.
	Bonnell	45	Atom.
2010	Westmere	32	Core ix-5xx/6xx/9xx. Intel HD Graphics.
2011	Saltwell	32	Atom.
	Sandy Bridge	32	Core ix-2xxx.
2012	Ivy Bridge	22	Core ix-3xxx.
2013	Silvermont	22	Atom.
	Haswell	22	Core ix-4xxx.
2014	Broadwell	14	Core ix-5xxx.
2015	Airmont	14	Atom.
	Skylake	14	Core ix-6xxx.
2016	Goldmont	14	Atom.
	Kaby Lake	14	Core ix-7xxx.
2017	Goldmont Plus	14	Atom.
	Coffee Lake	14	Core ix-8xxx, Core ix-9xxx.
2018	Cannon Lake	10	AVX-512.
	Whiskey Lake	14	
	Amber Lake	14	
2019	Cascade Lake	14	Core i9-10xxxX, hardware Meltdown and Spectre mitigations.
	Comet Lake	14	Core ix-10xxx.
	Ice Lake	10	Core ix-10xxG7, Core ix-10xxxU.
2020	Cooper Lake	14	Xeon, adds BFLOAT16.

Year Released	Microarchitecture	Lithography (nm)	Notes
	Lakefield	10-14[1]	Hybrid processor. Consists of 1 "big" Sunny Cove core and 4 "little" Tremont cores.
	Willow Cove (Tiger Lake)	10	Core ix-11xxx.
2021	Tremont (Jasper Lake)	10	Atom.
	Gracemont	7	SoC and Atom.
	Golden Cove	7	Alder Lake (client), Core 12th Gen. Introduced hybrid processors.
2022	Golden Cove	7	Sapphire Rapids (server), Intel AMX.
	Raptor Lake	7	Core 13th Gen.
(2023)	Emerald Rapids	7	Xeon
	Meteor Lake	4	

Table B-1: Intel microarchitectures

B.2. AMD microarchitectures

Year Released	Microarchitecture	Lithography (nm)	Notes
1984	80286	1500	Am286
1991	80386	800	Am386
1993	80486	700, 500, 350	Am486
1995	80486	350	Am5x86, Am486Plus
1996	K5	500, 350	Based on Intel's P6 (Pentium Pro), this was the first original microarchitecture from AMD
1997	K6	350	Pentium II-compatible

1 The compute die is 10nm while the base die is 14nm. Source: Wikipedia

Year Released	Microarchitecture	Lithography (nm)	Notes
1998	K6-2	250	3DNow!
1999	K6-III Sharptooth	250, 180	
1999	K7 Athlon	250, 130	Athlon, Sempron
2001	K7 Athlon	180, 130	Athlon XP/MP
2003	K8 Hammer	130, 65	AMD64, Athlon 64, Turion 64 Mobile, Opteron
2007	K10 Barcelona	65	Family 10h, Phenom
2008	Turion X2 Ultra	45	Family 11h, based on K8 and K10
2009	Athlon II	45, 32	Based on K10 and Phenom II
2011	Fusion	32	Family 12h, based on K10
	Bobcat	40	Family 14h, Z-Series, C-Series
	Bulldozer	32	Family 15h, FX-Series
2012	Piledriver	32	A10-Series
2013	Jaguar	28	Family 16h
2014	Steamroller	28	
	Puma	28	
2015	Excavator	28	
2017	Zen	14	Family 17h, Ryzen 1xxx
2018	Zen+	12	Ryzen 2xxx
2019	Zen 2	7	Ryzen 3xxx
2020	Zen 3	7	Family 19h, Ryzen 4xxx, Ryzen 5xxx
2022	Zen 4	5	Ryzen 7xxx
(2024)	Zen 5	3	

Table B-2: AMD microarchitectures

Family 18h was given to Chengdu Haiguang IC Design Co., Ltd (vendor id: "HygonGenuine").[1]

1 http://lkml.iu.edu/hypermail/linux/kernel/1806.1/00730.html?anz=print

Appendix C
Microarchitecture Feature Levels

In 2020, the GNU Compiler Collection group (GCC) introduced the idea of feature levels, also known as optimization groups. This concept serves as a new baseline to determine the guaranteed processor features to use when optimizing programs for x86-64 targets.[1]

The following table lists every feature level with their initial amount of features available and the minimum year that the last feature was introduced for the according microarchitecture. Values were taken from x86-64-psABI-2021-05-20.pdf, Table 3.1.[2]

Level	Minimum Microarchitecture	Features
x86-64	**Intel**: Pentium 4 (2000) **AMD**: Athlon 64, Opteron (2003, K8)	FPU, CMOV, CMPXCHG8B, FXSR, MMX, SSE, SSE2, SYSCALL
x86-64-v2	**Intel**: Nehalem (2008) **AMD**: Turion X2 Ultra (2008, Family 11h)	CMPXCHG16B, LAHF-SAHF, POPCNT, SSE3, SSSE3, SSE4.1, SSE4.2
x86-64-v3	**Intel**: Haswell (2013) **AMD**: Excavator (2015, Family 15h)	AVX, AVX2, BMI1, BMI2, F16C, FMA3, LZCNT, MOVBE, OSXSAVE
x86-64-v4	**Intel**: Skylake X (2017), Cannon Lake (2020)	AVX512F, AVX512BW, AVX512CD, AVX512DQ, AVX512VL

Table C-1: Microarchitecture Feature Levels

Since GCC 11 and LLVM Clang 12, it is possible to specify the feature level to use with the `-march` switch.[3]

Please note that, on ddcpuid, the OSFXSR (OSXSAVE) and SCE (SYSCALL) features need to be enabled in the OS via the CR4 control register and the IA32_EFER MSR for the CPUID bits to be active. These requirements may be lifted in future releases. For best results, it is recommended to run the utility on the host processor.

1 https://www.phoronix.com/scan.php?page=news_item&px=GCC-11-x86-64-Feature-Levels
2 https://gitlab.com/x86-psABIs/x86-64-ABI/-/wikis/x86-64-psABI
3 https://clang.llvm.org/docs/UsersManual.html#x86

Appendix D
Processor Revisions Numbers

The processor revision number, also known as microcode version, CPU Revision, or the microcode patch level, denotes the microcode program version used. While in cases for Intel processors, the microcode revision number may be included in stepping version numbers, **this revision number is not to be confused with the processor stepping number.**

The BIOS signature MSR (IA32_BIOS_SIGN_ID) is available through the use of the `WRMSR` and `RDMSR` instructions. However, these instructions are only available in a privileged context (Ring 0, so CPL=0, or real-mode). Model Specific Registers are not available from the `CPUID` instruction.

The Intel Reference Manual Volume 3 describes the MSR being available per logical processor core (see chapter *Microcode Update Resources*):

> *Each logical processor has its own BIOS signature MSR (IA32_BIOS_SIGN_ID at MSR address 8BH).*

The following assembler snippet shows how to retrieve the revision in the EDX register (see chapter *Assembly Code to Retrieve the Update Revision*):

```
MOV ECX,08BH  ; Load IA32_BIOS_SIGN_ID to ECX
XOR EAX,EAX   ; Clear EAX
XOR EDX,EDX   ; Clear EDX
WRMSR         ; Write to Model Specific Register
MOV EAX,1     ; Set leaf to 01H
CPUID         ; Non-privileged serializing instruction
MOV ECX,08BH  ; Load IA32_BIOS_SIGN_ID to ECX
RDMSR         ; Read MSR, EDX populated with revision number
```

For more information, read the Intel Reference Manual, Volume 3, chapter 9.11: Microcode Update Facilities. AMD also uses the MSR address 0x8B for the microcode revision, noted as the *microcode patch level*.

Operating systems may include an update facility to upgrade the processor microcode when starting up and each may provide their own version of the microcode. More information is available down below.

D.1. Revision Numbers on Windows®

Under the Windows® operating system, this information can be retrieved using the Windows Registry. Typically, the `HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\0` (numbered per logical processor, but typically the odd numbers have the most up to date microcode update) key contains a field named `Update Revision`. It is a little-endian (on x86 platforms) binary field containing values of the EAX:EDX pair register values following instruction output of RDMSR. For example, a value of 00 00 00 00 1f 00 00 00 indicates EAX=0x00000000 and EDX=0x0000001F. Thus meaning the current microcode is at revision 0x1f.

Beware of the revision number provided by some tools like CPUID™ CPU-Z™: The revision code, for example: `E1/L1`, is an extension to the stepping number, resulting to Mobile Core i7-3000 (Ivy Bridge E1/L1). This information is calculated using a string matching algorithm against the sample specification names (S-Spec). For example, an Intel® Core™ i7-3770 (SR0PK) matches i7-3000 (stepping E1) and i3-3000 (L1), thus giving E1/L1.

D.2. Revision Numbers on macOS®

Under the macOS® operating system, this information can be retrieved with the `sysctl machdep.cpu.microcode_version` command. For example, an Intel® Core™ i7-3770 may show a value of `machdep.cpu.microcode_version: 25`, which indicates a revision of 0x19. To remove the field from the output, use the `-n` option.

D.3. Revision Numbers on Linux®

Under the Linux® operating system, this information can be retrieved under the `/proc/cpuinfo` proc file under the `microcode` field. For example, an Intel® Core™ i7-3770 may show a value of 0x1f. Using standard GNU tools, this can be done with `cat /proc/cpuinfo | grep microcode`.

D.4. Revision Numbers on FreeBSD® and DragonFlyBSD™

Under FreeBSD® operating systems, there are currently two options. All presented modules can be installed via pkg(1).

D.4.1 Via devcpu-data

The first option is the microcode update module (`sysutils/devcpu-data`). As of writing, there are two methods to load the module.[1]

The first recommended method is enabling the update at boot-time. To enable the module, add the following to `/boot/loader.conf`:

```
cpu_microcode_load="YES"
cpu_microcode_name="/boot/firmware/intel-ucode.bin"
```

This enables updating the microcode before the operating system starts its processor feature detection, but only supported on FreeBSD 12.0 and later.

The second method is enabling the update via a rc script. To enable such feature, you can append the following in `/etc/rc.conf`:

```
microcode_update_enable="YES"
```

Then either reboot the system or run `service microcode_update start` as a privileged user to run the service.

Finally, to retrieve the information, running `cpucontrol -m 0x8b /dev/cpuctl0` as a privileged user will output the following information:

```
MSR 0x8b: 0x00000022 0x00000000
```

The first half denotes the value of the EDX, thus representing the microcode revision with the value 0x22.

1 https://www.freebsd.org/doc/en_US.ISO8859-1/books/faq/compatibility-processors.html#microcode

D.4.2 Via cpupdate

The second option is the cpu update module (`sysutils/cpupdate`). Currently, it only supports Intel processors. After installing, performing `cpupdate -i` presents with the following information below.

```
Found CPU(s) from Intel
Core 0 to 1: CPUID 306a9 Fam 06 Mod 3a Step 09 Flag 02 uCode
00000022
```

The `306a9` portion denotes the calculated family, model, and stepping numbers in hexadecimal. The revision number is provided at `uCode` with the value 0x22.

D.5. Revision Numbers on NetBSD®

Under the NetBSD® operating system, it is possible to retrieve the revision number with the `cpuctl identify 0` command as a privileged user. For example, an Intel® Core™ i7-3770 will show, at the end of the list, `cpu0: microcode version 0x19, platform ID 1`.

D.6. Revision Numbers on OpenBSD®

Under the OpenBSD® operating system, a list of installed *firmware names* can be obtained with `fw_update -i`. However, this outputs a list of *filenames*, such as `Installed: intel-firmware-20210216v0`.

Firmwares can be downloaded from Github: https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files. The revision number should be located at byte 4 of the file with a length of 4 bytes (little-endian).

Appendix E
Compiling ddcpuid

At the moment of writing, ddcpuid is only supported on x86 and x86-64 platforms. Releases currently available on Windows® through version XP[1] to 10 and any supported versions of Linux®. It's also possible to build ddcpuid wherever the DMD, GDC, or the LDC compilers are available, such as macOS, FreeBSD, OpenBSD, NetBSD, and HaikuOS[2].

E.1. Current Compiler Issues

E.1.1 GDC

GDC versions 10 and earlier will not compile using the switch `-fno-druntime` (an equivalent of `-betterC`) due to linking issues: `undefined reference to` `` `__gdc_personality_v0' ``.

E.1.2 LDC

On Windows, LDC versions 1.13 and 1.14 do not include `legacy_stdio_definitions.lib` when linking, making it impossible to compile the project using `-betterC`.

E.2. Compiling with DUB

Using dub(1) is rather straightforward.

Once installed, navigate to the root directory of the project and to perform a debug build, simply do: `dub build`

For a release build: `dub build -b release-nobounds`

1 Windows XP support is only available via the DMD compiler using the -m32 flag (OMF format). Not guaranteed to work.

2 Building on HaikuOS requires building the DMD compiler.

To select a different compiler: `dub build --compiler=ldc2`

For more information, visit the DUB usage page: https://dub.pm/commandline.html

E.3. Compiling with make(1)

The make(1) ease typing in compilation commands. The Makefile supports DMD, GDC, and LDC on UNIX platforms only. Currently, the Makefile was only tested on GNU Make (gmake), but should work on other make platforms.

There are two variables, DC (D Compiler) and PREFIX (installation path prefix), which can be changed when invoking the make command with the VARIABLE=VALUE syntax.

Here are the available actions:

- debug (default) – Compile debug build
- release – Compile release build
- clean – Remove ddcpuid.o and ddcpuid
- install – Install ddcpuid in PREFIX/bin and ddcpuid.1 in PREFIX/share/man/man1
- uninstall – Remove ddcpuid and ddcpuid.1 from installation paths

Examples:

- Compile ddcpuid with LDC and install it locally alongside the manpage.
  - `make release DC=ldc; make install`

E.4. Compiling manually

Since ddcpuid only consists of two source files, both being in the `src` folder, it's still pretty simple to perform a compilation by hand.

Here's an example that works on all three compilers:

```
dmd src/ddcpuid.d src/main.d -ofddcpuid
ldc2 src/ddcpuid.d src/main.d -ofddcpuid
gdc src/ddcpuid.d src/main.d -oddcpuid
```

If you want an optimized release build:

```
dmd -betterC -release -O -boundscheck=off src/ddcpuid.d src/main.d
-oddcpuid
```

```
ldc2 -betterC -release -O -boundscheck=off src/ddcpuid.d src/main.d
-oddcpuid
gdc -fno-druntime -release -O -fbounds-check=off src/ddcpuid.d
src/main.d -oddcpuid
```

You get the idea.

Appendix F
Software License