UNIX & LINUX

What do the flags in /proc/cpuinfo mean?

Ask Question

205

126

How can I tell whether my processor has a particular feature? (64-bit instruction set, hardware-assisted virtualization, cryptographic accelerators, etc.) I know that the file `/proc/cpuinfo` contains this information, in the `flags` line, but what do all these cryptic abbreviations mean?

For example, given the following extract from `/proc/cpuinfo`, do I have a 64-bit CPU? Do I have hardware virtualization?

```
model name      : Intel(R) Core(TM)2 Duo CPU     E8400  @ 3.00GHz
…
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm
constant_tsc arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl
vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 lahf_lm tpr_shadow vnmi flexpriority
```

linux cpu arm x86

asked Jul 19 '12 at 22:52

Gilles
545k 129 1107
1623

267

x86

(32-bit a.k.a. i386–i686 and 64-bit a.k.a. amd64. In other words, your workstation, laptop or server.)

FAQ: Do I have…

- 64-bit (x86_64/AMD64/Intel64)? `lm`
- Hardware virtualization (VMX/AMD-V)? `vmx` (Intel), `svm` (AMD)
- Accelerated AES (AES-NI)? `aes`
- TXT (TPM)? `smx`
- a hypervisor (announced as such)? `hypervisor`

Most of the other features are only of interest to compiler or kernel authors.

All the flags

The full listing is in the kernel source, in the file `arch/x86/include/asm/cpufeatures.h` .

Intel-defined CPU features, CPUID level 0x00000001 (edx)

See also Wikipedia and table 2-27 in Intel Advanced Vector Extensions Programming Reference

- `fpu` : Onboard FPU (floating point support)
- `vme` : Virtual 8086 mode enhancements
- `de` : Debugging Extensions (CR4.DE)
- `pse` : Page Size Extensions (4MB memory pages)
- `tsc` : Time Stamp Counter (RDTSC)
- `msr` : Model-Specific Registers (RDMSR, WRMSR)
- `pae` : Physical Address Extensions (support for more than 4GB of RAM)
- `mce` : Machine Check Exception
- `cx8` : CMPXCHG8 instruction (64-bit compare-and-swap)
- `apic` : Onboard APIC

- `pge` : Page Global Enable (global bit in PDEs and PTEs)
- `mca` : Machine Check Architecture
- `cmov` : CMOV instructions (conditional move) (also FCMOV)
- `pat` : Page Attribute Table
- `pse36` : 36-bit PSEs (huge pages)
- `pn` : Processor serial number
- `clflush` : Cache Line Flush instruction
- `dts` : Debug Store (buffer for debugging and profiling instructions)
- `acpi` : ACPI via MSR (temperature monitoring and clock speed modulation)
- `mmx` : Multimedia Extensions
- `fxsr` : FXSAVE/FXRSTOR, CR4.OSFXSR
- `sse` : Intel SSE vector instructions
- `sse2` : SSE2
- `ss` : CPU self snoop
- `ht` : Hyper-Threading
- `tm` : Automatic clock control (Thermal Monitor)
- `ia64` : Intel Itanium Architecture 64-bit (not to be confused with Intel's 64-bit x86 architecture with flag `x86-64` or "AMD64" bit indicated by flag `lm`)
- `pbe` : Pending Break Enable (PBE# pin) wakeup support

AMD-defined CPU features, CPUID level 0x80000001

See also Wikipedia and table 2-23 in Intel Advanced Vector Extensions Programming Reference

- `syscall` : SYSCALL (Fast System Call) and SYSRET (Return From Fast System Call)
- `mp` : Multiprocessing Capable.
- `nx` : Execute Disable
- `mmxext` : AMD MMX extensions
- `fxsr_opt` : FXSAVE/FXRSTOR

- `rdtscp` : Read Time-Stamp Counter and Processor ID
- `lm` : Long Mode (x86-64: amd64, also known as Intel 64, i.e. 64-bit capable)
- `3dnowext` : AMD 3DNow! extensions
- `3dnow` : 3DNow! (AMD vector instructions, competing with Intel's SSE1)

Transmeta-defined CPU features, CPUID level 0x80860001

- `recovery` : CPU in recovery mode
- `longrun` : Longrun power control
- `lrti` : LongRun table interface

Other features, Linux-defined mapping

- `cxmmx` : Cyrix MMX extensions
- `k6_mtrr` : AMD K6 nonstandard MTRRs
- `cyrix_arr` : Cyrix ARRs (= MTRRs)
- `centaur_mcr` : Centaur MCRs (= MTRRs)
- `constant_tsc` : TSC ticks at a constant rate
- `up` : SMP kernel running on UP
- `art` : Always-Running Timer
- `arch_perfmon` : Intel Architectural PerfMon
- `pebs` : Precise-Event Based Sampling
- `bts` : Branch Trace Store
- `rep_good` : rep microcode works well
- `acc_power` : AMD accumulated power mechanism
- `nopl` : The NOPL (0F 1F) instructions
- `xtopology` : cpu topology enum extensions
- `tsc_reliable` : TSC is known to be reliable
- `nonstop_tsc` : TSC does not stop in C states

- `extd_apicid` : has extended APICID (8 bits)
- `amd_dcm` : multi-node processor
- `aperfmperf` : APERFMPERF
- `eagerfpu` : Non lazy FPU restore
- `nonstop_tsc_s3` : TSC doesn't stop in S3 state
- `tsc_known_freq` : TSC has known frequency
- `mce_recovery` : CPU has recoverable machine checks

Intel-defined CPU features, CPUID level 0x00000001 (ecx)

See also Wikipedia and table 2-26 in Intel Advanced Vector Extensions Programming Reference

- `pni` : SSE-3 ("Prescott New Instructions")
- `pclmulqdq` : Perform a Carry-Less Multiplication of Quadword instruction — accelerator for GCM)
- `dtes64` : 64-bit Debug Store
- `monitor` : Monitor/Mwait support (Intel SSE3 supplements)
- `ds_cpl` : CPL Qual. Debug Store
- `vmx` : Hardware virtualization: Intel VMX
- `smx` : Safer mode: TXT (TPM support)
- `est` : Enhanced SpeedStep
- `tm2` : Thermal Monitor 2
- `ssse3` : Supplemental SSE-3
- `cid` : Context ID
- `sdbg` : silicon debug
- `fma` : Fused multiply-add
- `cx16` : CMPXCHG16B
- `xtpr` : Send Task Priority Messages
- `pdcm` : Performance Capabilities
- `pcid` : Process Context Identifiers
- `dca` : Direct Cache Access
- `sse4_1` : SSE-4.1
- `sse4_2` : SSE-4.2
- `x2apic` : x2APIC

- `popcnt` : Return the Count of Number of Bits Set to 1 instruction (Hamming weight, i.e. bit count)
- `tsc_deadline_timer` : Tsc deadline timer
- `aes` / `aes-ni` : Advanced Encryption Standard (New Instructions)
- `xsave` : Save Processor Extended States: also provides XGETBY,XRSTOR,XSETBY
- `avx` : Advanced Vector Extensions
- `f16c` : 16-bit fp conversions (CVT16)
- `rdrand` : Read Random Number from hardware random number generator instruction
- `hypervisor` : Running on a hypervisor

VIA/Cyrix/Centaur-defined CPU features, CPUID level 0xC0000001

- `rng` : Random Number Generator present (xstore)
- `rng_en` : Random Number Generator enabled
- `ace` : on-CPU crypto (xcrypt)
- `ace_en` : on-CPU crypto enabled
- `ace2` : Advanced Cryptography Engine v2
- `ace2_en` : ACE v2 enabled
- `phe` : PadLock Hash Engine
- `phe_en` : PHE enabled
- `pmm` : PadLock Montgomery Multiplier
- `pmm_en` : PMM enabled

More extended AMD flags: CPUID level 0x80000001, ecx

- `lahf_lm` : Load AH from Flags (LAHF) and Store AH into Flags (SAHF) in long mode
- `cmp_legacy` : If yes HyperThreading not valid
- `svm` : “Secure virtual machine”: AMD-V
- `extapic` : Extended APIC space

- `sse4a` : SSE-4A
- `misalignsse` : indicates if a general-protection exception (#GP) is generated when some legacy SSE instructions operate on unaligned data. Also depends on CR0 and Alignment Checking bit
- `3dnowprefetch` : 3DNow prefetch instructions
- `osvw` : indicates OS Visible Workaround, which allows the OS to work around processor errata.
- `ibs` : Instruction Based Sampling
- `xop` : extended AVX instructions
- `skinit` : SKINIT/STGI instructions
- `wdt` : Watchdog timer
- `lwp` : Light Weight Profiling
- `fma4` : 4 operands MAC instructions
- `tce` : translation cache extension
- `nodeid_msr` : NodeId MSR
- `tbm` : Trailing Bit Manipulation
- `topoext` : Topology Extensions CPUID leafs
- `perfctr_core` : Core Performance Counter Extensions
- `perfctr_nb` : NB Performance Counter Extensions
- `bpext` : data breakpoint extension
- `ptsc` : performance time-stamp counter
- `perfctr_l2` : L2 Performance Counter Extensions
- `mwaitx` : `MWAIT` extension (`MONITORX` / `MWAITX`)

Auxiliary flags: Linux defined - For features scattered in various CPUID levels

- `ring3mwait` : Ring 3 MONITOR/MWAIT
- `cpuid_fault` : Intel CPUID faulting
- `cpb` : AMD Core Performance

support

- `cat_l3` : Cache Allocation Technology L3
- `cat_l2` : Cache Allocation Technology L2
- `cdp_l3` : Code and Data Prioritization L3
- `invpcid_single` : effectively `invpcid` and `CR4.PCIDE=1`
- `hw_pstate` : AMD HW-PState
- `proc_feedback` : AMD ProcFeedbackInterface
- `sme` : AMD Secure Memory Encryption
- `pti` : Kernel Page Table Isolation (Kaiser)
- `retpoline` : Retpoline mitigation for Spectre variant 2 (indirect branches)
- `retpoline_amd` : AMD Retpoline mitigation
- `intel_ppin` : Intel Processor Inventory Number
- `avx512_4vnniw` : AVX-512 Neural Network Instructions
- `avx512_4fmaps` : AVX-512 Multiply Accumulation Single precision
- `mba` : Memory Bandwidth Allocation
- `rsb_ctxsw` : Fill RSB on context switches

Virtualization flags: Linux defined

- `tpr_shadow` : Intel TPR Shadow
- `vnmi` : Intel Virtual NMI
- `flexpriority` : Intel FlexPriority
- `ept` : Intel Extended Page Table
- `vpid` : Intel Virtual Processor ID
- `vmmcall` : prefer `VMMCALL` to `VMCALL`

Intel-defined CPU features, CPUID level 0x00000007:0 (ebx)

- `fsgsbase` : {RD/WR} {FS/GS}BASE instructions
- `tsc_adjust` : TSC adjustment

- `hle` : Hardware Lock Elision
- `avx2` : AVX2 instructions
- `smep` : Supervisor Mode Execution Protection
- `bmi2` : 2nd group bit manipulation extensions
- `erms` : Enhanced REP MOVSB/STOSB
- `invpcid` : Invalidate Processor Context ID
- `rtm` : Restricted Transactional Memory
- `cqm` : Cache QoS Monitoring
- `mpx` : Memory Protection Extension
- `rdt_a` : Resource Director Technology Allocation
- `avx512f` : AVX-512 foundation
- `avx512dq` : AVX-512 Double/Quad instructions
- `rdseed` : The RDSEED instruction
- `adx` : The ADCX and ADOX instructions
- `smap` : Supervisor Mode Access Prevention
- `clflushopt` : `CLFLUSHOPT` instruction
- `clwb` : `CLWB` instruction
- `intel_pt` : Intel Processor Tracing
- `avx512pf` : AVX-512 Prefetch
- `avx512er` : AVX-512 Exponential and Reciprocal
- `avx512cd` : AVX-512 Conflict Detection
- `sha_ni` : SHA1/SHA256 Instruction Extensions
- `avx512bw` : AVX-512 Byte/Word instructions
- `avx512vl` : AVX-512 128/256 Vector Length extensions

Extended state features, CPUID level 0x0000000d:1 (eax)

- `xsaveopt` : Optimized `XSAVE`
- `xsavec` : `XSAVEC`

Intel-defined CPU QoS sub-leaf, CPUID level 0x0000000F:0 (edx)

- `cqm_llc` : LLC QoS

Intel-defined CPU QoS sub-leaf, CPUID level 0x0000000F:1 (edx)

- `cqm_occup_llc` : LLC occupancy monitoring
- `cqm_mbm_total` : LLC total MBM monitoring
- `cqm_mbm_local` : LLC local MBM monitoring

AMD-defined CPU features, CPUID level 0x80000008 (ebx)

- `clzero` : `CLZERO` instruction
- `irperf` : instructions retired performance counter
- `xsaveerptr` : Always save/restore FP error pointers

Thermal and Power Management leaf, CPUID level 0x00000006 (eax)

- `dtherm` (formerly `dts`): digital thermal sensor
- `ida` : Intel Dynamic Acceleration
- `arat` : Always Running APIC Timer
- `pln` : Intel Power Limit Notification
- `pts` : Intel Package Thermal Status
- `hwp` : Intel Hardware P-states
- `hwp_notify` : HWP notification
- `hwp_act_window` : HWP Activity Window
- `hwp_epp` : HWP Energy Performance Preference
- `hwp_pkg_req` : HWP package-level request

AMD SVM Feature Identification, CPUID level 0x8000000a (edx)

- `npt` : AMD Nested Page Table support

- `svm_lock` : AMD SVM locking MSR
- `nrip_save` : AMD SVM next_rip save
- `tsc_scale` : AMD TSC scaling support
- `vmcb_clean` : AMD VMCB clean bits support
- `flushbyasid` : AMD flush-by-ASID support
- `decodeassists` : AMD Decode Assists support
- `pausefilter` : AMD filtered pause intercept
- `pfthreshold` : AMD pause filter threshold
- `avic` : Virtual Interrupt Controller
- `vmsave_vmload` : Virtual VMSAVE VMLOAD
- `vgif` : Virtual GIF

Intel-defined CPU features, CPUID level 0x00000007:0 (ecx)

- `avx512vbmi` : AVX512 Vector Bit Manipulation instructions
- `umip` : User Mode Instruction Protection
- `pku` : Protection Keys for Userspace
- `ospke` : OS Protection Keys Enable
- `avx512_vbmi2` : Additional AVX512 Vector Bit Manipulation instructions
- `gfni` : Galois Field New Instructions
- `vaes` : Vector AES
- `vpclmulqdq` : Carry-Less Multiplication Double Quadword
- `avx512_vnni` : Vector Neural Network Instructions
- `avx512_bitalg` : VPOPCNT[B,W] and VPSHUF-BITQMB instructions
- `avx512_vpopcntdq` : POPCNT for vectors of DW/QW
- `la57` : 5-level page tables
- `rdpid` : RDPID instruction

- `overflow_recov` : MCA overflow recovery support
- `succor` : uncorrectable error containment and recovery
- `smca` : Scalable MCA

Detected CPU bugs (Linux-defined)

- `f00f` : Intel F00F
- `fdiv` : CPU FDIV
- `coma` : Cyrix 6x86 coma
- `amd_tlb_mmatch` : `tlb_mmatch` AMD Erratum 383
- `amd_apic_c1e` : `apic_c1e` AMD Erratum 400
- `11ap` : Bad local APIC aka 11AP
- `fxsave_leak` : FXSAVE leaks FOP/FIP/FOP
- `clflush_monitor` : AAI65, CLFLUSH required before MONITOR
- `sysret_ss_attrs` : SYSRET doesn't fix up SS attrs
- `espfix` : "" IRET to 16-bit SS corrupts ESP/RSP high bits
- `null_seg` : Nulling a selector preserves the base
- `swapgs_fence` : SWAPGS without input dep on GS
- `monitor` : IPI required to wake up remote CPU
- `amd_e400` : CPU is among the affected by Erratum 400
- `cpu_meltdown` : CPU is affected by meltdown attack and needs kernel page table isolation
- `spectre_v1` : CPU is affected by Spectre variant 1 attack with conditional branches
- `spectre_v2` : CPU is affected by Spectre variant 2 attack with indirect branches
- `spec_store_bypass` : CPU is affected by the Speculative Store Bypass vulnerability (Spectre variant 4).

P.S. This listing was derived from `arch/x86/include/asm/cpufeatures.h` in the kernel source. The flags are listed in the same order as the source code. Please

unexpressive names, and by updating the list for new kernel versions. The current list is from Linux 4.15 plus some later additions.

edited Feb 14 at 16:24

community wiki
21 revs, 11 users 76%
Gilles

1 Thanks @Gilles and editors for an informative question & its summarized and detailed answer. Now, to check whatever CPU capabilities, I use the following taken from NixCraft, eg for Intel CPUs: `$ egrep -wo '^flags|vmx|ept|vpid|npt|tpr_shadow|flexpriority|vnmi|lm|aes' /proc/cpuinfo --color | sort -u` . And there's also the CLI/GUI excellent i-nex. – tuk0z Sep 22 '15 at 14:55

Outstanding collection of explanations & links; thanks to everyone who contributed. – Paul Gear Mar 8 '17 at 4:11

Where did the bugs data come from? It doesn't appear to be listed in the cpufeatures.h file. – Drazisil Jun 13 '18 at 2:12

@Drazisil As far as I remember all the entries are from the indicated version of `cpufeatures.h` . The descriptions are edited to make them more comprehensible and more informative where someone took the effort to do that. – Gilles Jun 13 '18 at 6:22

@Gilles That seems to be the case for all but the bugs. Setting aside that those aren't `features` I don't see them in that file. – Drazisil Jun 13 '18 at 19:40

69

ARM

On ARM processors, a few features are mentioned in the `features:` line. Only features directly related to the ARM architecture are mentioned there, not features specific to a silicon manufacturer or system-on-chip.

The features are obtained from looking up the CPU id with `read cpuid()` and looking it up in the processor type definitions known at compile time where the features are

In the list below, ARMv6 introduced SIMD instructions and datatypes. ARMv7 provided Advanced SIMD instructions and datatypes. On 32-bit ARM machines, `neon` signals Advanced SIMD; while `asimd` signals Advanced SIMD on 64-bit arm machines.

- `swp` : `SWP` instruction (atomic read-modify-write)
- `half` : Half-word loads and stores
- `thumb` : Thumb (16-bit instruction set)
- `26bit` : "26 Bit" Model (Processor status register folded into program counter)
- `fastmult` : 32×32→64-bit multiplication
- `fpa` : Floating point accelerator
- `vfp` : VFP (early SIMD vector floating point instructions)
- `edsp` : DSP extensions (the 'e' variant of the ARM9 CPUs, and all others above)
- `java` : Jazelle (Java bytecode accelerator)
- `iwmmxt` : SIMD instructions similar to Intel MMX
- `crunch` : MaverickCrunch coprocessor (if kernel support enabled)
- `thumbee` : ThumbEE
- `neon` : Advanced SIMD/NEON (`asimd` on AArch64 older kernels)
- `vfpv3` : VFP version 3
- `vfpv3d16` : VFP version 3 with 16 D-registers
- `tls` : TLS register
- `vfpv4` : VFP version 4 with fast context switching
- `idiva` : `SDIV` and `UDIV` hardware division in ARM mode
- `idivt` : `SDIV` and `UDIV` hardware division in Thumb mode
- `vfpd32` : VFP with 32 D-registers
- `lpae` : Large Physical Address Extension (>4GB physical memory on 32-bit architecture)

- `aes` : hardware-accelerated AES (secret-key cryptography)
- `pmull{2}` : 64×64→128-bit F_{2^m} multiplication — acceleration for the GCM mode of authenticated encryption
- `sha1` : hardware-accelerated SHA-1
- `sha2` : hardware-accelerated SHA-256
- `crc32` : hardware-accelerated CRC-32

Beyond that, the `Hardware:` line indicates the processor model. Depending on the model, there may be other information in other files under `/proc` or `/sys`, or in boot-time kernel log messages. Unfortunately each ARM CPU manufacturer has its own method for reporting processor features, if any.

edited Nov 14 '17 at 16:32

community wiki
12 revs, 5 users 78%
Gilles

10

x86

Find it yourself in 4.1.3 x86 and the Intel manual

`arch/x86/include/asm/cpufeature.h` contains the full list.

The define values are of type:

```
X*32 + Y
```

E.g.:

```
#define X86_FEATURE_FPU     ( 0*32
```

The features flags, extracted from CPUID, are stored inside the:

- `__u32 x86_capability[NCAPINTS + NBUGINTS];` field
- of `struct cpuinfo_x86 boot_cpu_data`
- defined at `x86/kernel/setup.c`

Where each `x86_capability` array element comes from:

index	eax	ecx	output
0	1	0	edx
1	80000001		edx
2	80860001		edx
3			
4	1	0	ecx
5	C0000001		edx
6	80000001		ecx
7			
8			
9	7	0	ebx
10	D	1	eax
11	F	0	edx
12	F	1	edx

Notes:

- empty entries mean: "from various places" or "not available"
- `index` : is the index of `x86_capability` , e.g. `x86_capability[0]`
- `eax` and `exc` : are the input values for CPUID in hex. Inputs that use `exc` , which are fewer, call it the *subleaf* (of a 2 level tree with `eax` at the root).
- `output` : is the register from which CPUID output is taken
- `file` : is the file where those fields are defined. Paths are relative to `arch/x86/kernel/cpu/` .
- `transmeta` : was the name of a CPU vendor https://en.wikipedia.org/wiki/Transmeta that was acquired by Novafora https://www.crunchbase.com/organization/novafora
- `centaur` : was the name of a CPU vendor https://en.wikipedia.org/wiki/Centaur_Technology that was acquired by VIA https://en.wikipedia.org/wiki/VIA_Technologies. Cyrix is another one.

Conclusions:

- most entries come directly from CPUID output registers and are set in `common.c` by something like:

- the others are scattered throughout the source, and are set bit by bit with `set_cpu_cap`.

  To find them, use `git grep X86_FEATURE_XXX` inside `arch/x86`.

  You can usually deduce what CPUID bit they correspond to from the surrounding code.

Other fun facts

- The flags are actually printed at `arch/x86/kernel/cpu/proc.c` with the code:

```
seq_puts(m, "flags\t\t:");
for (i = 0; i < 32*NCAPINTS; i
    if (cpu_has(c, i) && x86_c
        seq_printf(m, " %s", x
```

  Where:

  - `cpu_has` does the main check for the feature.
  - `x86_cap_flags[i]` contains strings that correspond to each flags.

  This gets passed as a callback to the `proc` system setup. The entry point is at `fs/proc/cpuinfo.c`.

- `x86_cap_flags` strings are generated by `arch/x86/kernel/cpu/mkcapflags.h` directly from `arch/x86/include/asm/cpufeature.h` by "parsing" it with `sed` ...

  The output goes to `arch/x86/kernel/cpu/capflags.c` of the build directory, and resulting array looks like:

```
const char * const x86_cap_fla
    [X86_FEATURE_FPU]       =
    [X86_FEATURE_VME]       =
```

  so for example `X86_FEATURE_FPU` corresponds to the string `"fpu"` and so on.

- `cpu_has` breaks down into two cases with code:

```
#define cpu_has(c, bit)
    (__builtin_constant_p(bit)
    test_cpu_cap(c, bit))
```

- `__builtin_constant_p(bit) && REQUIRED_MASK_BIT_SET(bit)` : the flag is required for the kernel to run.

  This is determined by data inside `required-features.h` , which comments:

  ```
  Define minimum CPUID featu
  really early to actually c
  kernel dies.  Make sure tc
  ```

  Since those are known at compile time (kernel requirements), have already been checked at startup, the check can be resolved at compile time if `bit` is known at compile time.

  Thus the `__builtin_constant_p(bit)` which checks if `bit` is a compile time constant.

- `test_cpu_cap` : this uses up `CPUID` data from the `struct cpuinfo_x86 boot_cpu_data` global

edited Aug 1 '15 at 21:05

Gilles
545k 129 1107 1623

answered Aug 1 '15 at 20:**57**

Ciro Santilli 新疆改造中心 六四事件 法轮功
5,380 2 46 44

2 You've explained how to go from the abbreviation to a longer name, but often that longer name isn't much more comprehensible, and `cpuid` does it in a more convenient way. I asked that question to have a place where the names are documented. – Gilles Aug 1 '15 at 21:04

@Gilles this is mostly for those who want to make the tables / cannot find their feature in the table, like me :-) But still, for most cases, once you look at the right point of the source, the map to CPUID is immediate. – Ciro Santilli 新疆改造中心 六四事件 法轮 Aug 1 '15 at 21:07

9 repository. It dumps every possible info about your CPU with some explanations, so you don't get those obscure flags.

answered May 22 '14 at 13:19

hurufu
261 3 2

`cpuid` expands the abbreviations. I wouldn't really call its output *explanations*. Knowing that `ht` means “Hyper Threading” explains it to some extent, but knowing that `mmx` means “MMX instruction set”, not so much, and that `mca` means “Machine Check Architecture”, hardly. – Gilles May 22 '14 at 18:10

6 @Gilles ...and yet, "Machine Check Architecture" is certainly better Google query than "mca" ;) – Alois Mahdal Jun 6 '14 at 10:07