

Standardizing CPUID Data for the Open-source x86 Ecosystem

Linux Plumbers Conference
November 13-15, 2023

Refereed Track Session

Ahmed S. Darwish <darwi@linutronix.de>

Standardizing CPUID Data for the Open-source x86 Ecosystem

Preliminaries — The CPUID instruction

The x86 CUID instruction

Primary method on x86 CPUs for runtime identification of CPU features.

- **Input:** a “leaf” number in the *EAX* register.
- **Optional Input:** a “subleaf” number in the *ECX* register.
- **Output:** Information regarding *CPU identification* and *supported features*.
- **Output Format:** Bitfields the *EAX*, *EBX*, *ECX*, and *EDX* registers (even on x86-64.)

The x86 CUID instruction (2)

Availability: All modern x86 CPUs

- Due to its consistent availability since i586, most x86 software unconditionally assume CUID insn. support.
- *Technically*, an EFLAGS ID bit signals its availability.

Permissions: Can be executed at any privilege level

- It can be, and frequently is, invoked from user-space.

Other: A serializing instruction.

CPUID instruction: GCC/LLVM intrinsics

- Both the GCC and LLVM compilers provide a `__cpuid_count(leaf, subleaf, eax, ebx, ecx, edx)` macro at “cpuid.h”
- Both implement the `__builtin_cpu_supports("feature")` intrinsic
 - `__builtin_cpu_supports("sse3")`
 - `__builtin_cpu_supports("sse4.1")`
 - `__builtin_cpu_supports("avx512f")`
 - ...

CPUID instruction: demo

```
static void noinline cpuid(uint32_t leaf,  
                           uint32_t output[])  
{  
    asm volatile ("cpuid"  
        : "=a" (output[0]), "=b" (output[1]),  
          "=c" (output[2]), "=d" (output[3])  
        : "a" (leaf), "c" (0)  
        : "cc", "memory");  
}
```

000000000000001159 <cpuid>:

...

1165: 0f a2 cpuid

CPUID instruction: demo (2)

```
for (uint32_t leaf = 0; leaf <= 2; leaf++) {  
    uint32_t output[4];  
  
    /* CPUID input: leaf number */  
    printf("Leaf %08x: ", leaf);  
  
    /* Invoke CPUID instruction */  
    cpuid(leaf, output);  
  
    /* CPUID output: EAX => EDX registers */  
    printf("eax = %08x, ebx = %08x\n"  
        "ecx = %08x, edx = %08x\n",  
        output[0], output[1], output[2], output[3]);  
}
```

CPUID instruction: demo (3)

Leaf 0x00000000: eax=0x00000016 ebx=0x756e6547
ecx=0x6c65746e edx=0x49656e69

Max
Standard
Leaf =
0x16

Leaf 0x00000001: eax=0x000806eb ebx=0x05100800
ecx=0x7ffaebbfbf edx=0xbfebfbff

Vendor ID
String 1 =
“Genu”

Leaf 0x00000002: eax=0x76036301 ebx=0x00f0b5ff
ecx=0x00000000 edx=0x00c30000

Vendor ID
String 2 =
“ineI”

Vendor ID
String 3 =
“ntel”

Standardizing CPUID Data for the Open-source x86 Ecosystem

Preliminaries, Pt. 2 — Sample CPUID Data

Sample CPUID data (1)

```
$ cpuid --leaf=1
```

```
feature information (1/edx):
```

```
  x87 FPU on chip                      = true
```

```
  VME: virtual-8086 mode enhancement  = true
```

```
  DE: debugging extensions             = true
```

```
  PSE: page size extensions            = true
```

```
  TSC: time stamp counter              = true
```

```
  RDMSR and WRMSR support              = true
```

```
  PAE: physical address extensions     = true
```

```
  MCE: machine check exception         = true
```

```
  CMPXCHG8B inst.                     = true
```

```
  APIC on chip                         = true
```

```
  ...
```

Sample CUID data (2)

...

feature information (1/ecx):

PNI/SSE3: Prescott New Instructions	= true
PCLMULDQ instruction	= true
DTS64: 64-bit debug store	= true
MONITOR/MWAIT	= true
CPL-qualified debug store	= true
VMX: virtual machine extensions	= true
SMX: safer mode extensions	= false
Enhanced Intel SpeedStep Technology	= true
TM2: thermal monitor 2	= true
SSSE3 extensions	= true

...

Sample CUID data (3)

```
$ cpuid --one-cpu --leaf=0xb --subleaf=0
```

```
--- level 0 ---
```

```
level type = thread (1)
```

```
bit width of level = 1
```

```
number of logical processors at level = 2
```

```
$ cpuid --one-cpu --leaf=0xb --subleaf=1
```

```
--- level 1 ---
```

```
level type = core (2)
```

```
bit width of level = 6
```

```
number of logical processors at level = 56
```

Note: Total calculated CPU number above might differ from “lscpu” or /proc/cpuinfo due to kernel CPU hotplug. It might differ from “nproc”/ sched_getaffinity(2) due to CPU hotplug or cpuset(7) cgroups.

Sample CUID data: Hierarchical HW structures

Subleaves often numerate hierarchical hardware structures:

- **Leaf 0x4:** *Deterministic cache parameters:* L1/L2/... cache levels
- **Leaf 0x10:** *Intel RDT / AMD PQoS*
 - ResID 1 (L1 CAT) ➔ ResID 2 (L2 CAT) ➔ ResID 3 (MBA)
- **Leaf 0x12:** *Intel SGX:* EPC (Enclave Page Cache) descriptors
- **Leaf 0x18:** Descriptors for i/dTLB1, TLB2, ...
- **Leaf 0x1b:** *PCONFIG* Targets enumeration (MKTME, ...)
- **Leaf 0x1f:** *Intel CPU topology v2* enumeration
 - Thread ➔ Core ➔ Module ➔ Tile ➔ Die ➔ Die group
- **Leaf 0x80000026:** *AMD CPU topology v2*

Standardizing CUID Data for the Open-source x86 Ecosystem

*52 Standard + Extended CUID leaves
830+ Publicly-known bitfields*

Vendor-specific bitfields interpretation

Data scattered around 15+ Intel/AMD manuals

Standardizing CPUID Data for the Open-source x86 Ecosystem

Origin Story — How things came to be

Pressures for an in-Kernel CUID data model:

(1) Possible future hardware

- **Heterogeneous cores** on a single die might be a reality
 - ➔ A *per-CPU* in-kernel CUID data representation will be needed in that case.
- For now, all cores on a die group, *feature-wise*, are the same
 - ➔ Performance (P) cores vs. Energy (E) cores

Note: Progression towards heterogeneous architectures is merely a prevalent trajectory within the industry. The speculation regarding heterogeneous cores is not derived from, nor does it reflect, any proprietary insights or confidential information regarding future Intel or AMD product plans.

Pressures for an in-Kernel CPUID data model:

(2) CPU vulnerabilities

- **Live micro-code updates** might change some of the CPUID bitfields *post* Operating System initialization.
- **Operating System masking** of CPUID bits, due to known security issues, might be needed.

Note: AMD CPUID Leaf **0x80000021**, EAX bit 17, indicates HWCR[CpuidUserDis] support: “Trigger a #GP(0) when the CPUID instruction is executed by non-privileged software (CPL > 0) outside SMM” → User-space CPUID invocations can now be trapped.

Pressures for an in-Kernel CPUID data model:

(3) Existing code

- Drivers are manually invoking cpuid everywhere:
`cpuid_count()`, `cpuid_e[abcd]x()`,
 - ➔ *No central authority* for CPUID information in the kernel
 - ➔ *Cannot* support heterogeneous cores, live microcode, etc.
- **On-the-spot parsing** of cpuid bitfields, through *ugly* bitwise operations, is common; e.g. `(cpuid_ebx(1) >> 8) & 0xff`
- Incomplete **X86_FEATURE_*** bits at `cpufeatures.h`
- Hack-ish **/proc/cpuinfo** flags at `cpufeatures.h`

In-kernel CPUID data model: Genesis (1)

- **2018:** **Borislav Petkov**, x86 tree co-maintainer, started a basic clean-up at a personal local branch: cpuid bitfield descriptions using clean C structures with C99 bitfields. ¹
- **2023 Q3:** **Thomas Gleixner**, x86-tree co-maintainer, asked me to look at Boris's branch and use it as a base for an in-kernel CPUID data model.
 - **Data model should cover:** Leaf and subleaf bitfields, x86 CPU vendor filtering, X86_FEATURE_* encodings, and /proc/cpuinfo flags.

¹ <https://git.kernel.org/pub/scm/linux/kernel/git/bp/bp.git/log/?h=cpuid>

In-kernel CPUID data model: Genesis (2)

- 2023 Q3: I prototype some models...

```
/**
 * @cpufv_t: CPU feature bit is exposed at /proc/cpuinfo
 * @cpufh_t: CPU feature bit is not exposed at /proc/cpuinfo
 */
typedef u32 cpufv_t, cpufh_t;

struct cpuid_level {
    u32 level,      sublevel;
};

static struct cpuid_level cpuid_levels[] = {
    /* @level,      @sublevel */
    { 0,            },
    { 1,            },
    { 0x800000001,  },
};
```

In-kernel CPUID data model: Genesis (3)

```
struct cpuid_bits {  
    // Level 0  
    u32      max_lvl, vendor0, vendor2, vendor1;  
  
    // Level 1  
    cpufv_t  xmm3      : 1,    /* "pni" SSE-3 */  
             pclmulqdq : 1,    /* PCLMULQDQ instruction */  
             dtes64    : 1,    /* 64-bit Debug Store */  
             ...;  
  
    cpufv_t  fpu        : 1,    /* Onboard FPU */  
             vme        : 1,    /* Virtual Mode Extensions */  
             de         : 1,    /* Debugging Extensions */  
             ...;  
  
    // Level 8000_0001  
    u32      stepping   : 4, base_model: 4, base_family: 4, ...;  
};
```

In-kernel CPUID data model: Genesis (4)

```
struct cpuid_bits {  
    // Level 0  
    u32      max_lvl, vendor0, vendor2, vendor1;  
  
    // Level 1  
    cpufv_t  xmm3      : 1,    /* "pni" SSE-3 */  
             pclmulqdq : 1,    /* PCLMULQDQ instruction */  
             dtes64    : 1,    /* 64-bit Debug Store */  
             ...;  
  
    cpufv_t  fpu        : 1,    /* Onboard FPU */  
             vme        : 1,    /* Virtual Mode Extensions */  
             de         : 1,    /* Debugging Extensions */  
             ...;  
  
    // Level 8000_0001  
    u32      stepping   : 4, base_model: 4, base_family: 4, ...;  
};
```

In-kernel CPUID model: Genesis (4)

- **2023 Q3:** **We** all realize that the raw bitfield descriptions along with its attached information like `/proc/cpuinfo` visibility, `/proc/cpuinfo` cpu flag names, x86 CPU vendor filtering, `X86_FEATURE` flags etc. *is not workable in C*.
 - ➔ *First*, A machine-readable database is needed.
 - ➔ *Later*, we can worry about the in-kernel C data model.
- **2023 Q3:** **Thomas Gleixner** and **Andrew Cooper** (Xen maintainer) prototype an initial data format.

A CPUID Database

- **2023 Q3:** I evaluate different languages for the machine-readable syntax: JSON, YAML, TOML, S-expressions, and XML.
 - Native support for an *extensible annotated tree* was desired for extensibility without excessive vertical lines.
- To the best of our knowledge, only XML and S-expressions were capable of supporting an annotated extensible tree.
 - XML was ultimately picked, due to its mature ecosystem, especially regarding schema (XSD), querying (XPath), and transformation capabilities (XSLT).

Standardizing CPUID Data for the Open-source x86 Ecosystem

A CPUID Database

```

<leaf id="1">
  <desc>CPU FMS (Family/Model/Stepping) + standard feature flags</desc>
  <subleaf id="0">
    <eax>
      <bit0 len="4" id="stepping" desc="Stepping ID">
        <vendors>
          <vendor>Intel</vendor>
          <vendor>AMD</vendor>
        </vendors>
      </bit0>
      <bit4 len="4" id="base_model" desc="Base CPU model ID">
        <vendors>
          <vendor>Intel</vendor>
          <vendor>AMD</vendor>
        </vendors>
      </bit4>
    </eax>
  </subleaf>
</leaf>

```

```

U: @- - - leaf_01.xml 1% (8,39) Git (nXML Valid AR
Quit

```

```

<bit6 len="1" id="smx" desc="Safer Mode Extensions">
  <vendors>
    <vendor>Intel</vendor>
  </vendors>
  <linux feature="true" proc="true" />
  <xen feature="true" attrs="" />
</bit6>
<bit7 len="1" id="est" desc="Enhanced Intel SpeedStep Technology">
  <vendors>
    <vendor>Intel</vendor>
  </vendors>
  <linux feature="true" proc="true" />
  <xen feature="true" attrs="" altid="eist" />
</bit7>
<bit8 len="1" id="tm2" desc="Thermal Monitor 2">
  <vendors>

```

```

U: @--- leaf_01.xml 20% (118,0) Git (nXML Valid AR
Quit

```

```

<bit31 len="1" id="guest_status" desc="System is run
  <text>
    Note: for a guest running within an Intel Trusted-Domain run-
    this bit is also set. If cpuid leaf 0 @max_std_leaf ≥ 0x21, t
    be used by such a guest to further enumerate the trusted-dom
  </text>
  <vendors>
    <vendor>Intel</vendor>
    <vendor>AMD</vendor>
  </vendors>
  <hypervisors>
    <hypervisor>Xen</hypervisor>
    <hypervisor>KVM</hypervisor>
  </hypervisors>
  <xen feature="true" attrs="!A" altid="hypervisor" />
</bit31>

```

```

<ecx>
  <bit0 len="8" id="domain_level" desc="This domain level (su
  <bit8 len="8" id="domain_type" desc="This domain type">
    <values>
      <value id="1" desc="Logical CPU; smallest
      <value id="2" desc="Core" />
      <value id="3" desc="Module" />
      <value id="4" desc="Tile" />
      <value id="5" desc="Die" />
      <value id="6" desc="Die group" />
    </values>
  </bit8>
</ecx>
<edx>
  <bit0 len="32" id="x2apic_id" desc="x2APIC ID of current
    <text>

```

```
<xsl:variable name="leafBiggestNameLen"
               select="max(//*[starts-with(local-name(), 'bit')]/string-length(@id))" />
```

```
<!-- Round-up a number to the nearest multiple -->
<xsl:function name="lx:round-up"          as="xs:decimal">
  <xsl:param name="number"                as="xs:decimal"/>
  <xsl:param name="multiple"              as="xs:decimal"/>
  <xsl:sequence select="ceiling($number div $multiple) *
$multiple"/>
</xsl:function>
```

```
<!-- Repeat a string n times -->
<xsl:function name="lx:repeat"            as="xs:string">
  <xsl:param name="string"                as="xs:string"/>
```

```
>
-: @--- cstruct.xslt Top (3,0) Git (nXML Invalid
Quit
```

```
$ cpuidgen --help
usage: cpuidgen.py [-h] [--kcpuid] [--leaf LEAF]
```

CPUID leaves bitfield generator

options:

-h, --help	show this help message and exit
--kcpuid, -k	Generate linux-kernel kcpuid compatible CSV file
--leaf LEAF, -l LEAF	Generate a C structure for given CPUID hex-formatted LEAF

Generate CPUID data structures in different output formats

Leaf 8000001DH

AMD deterministic cache parameters

0x8000001d,	31:0,	eax,	4:0,	cache_type	, Cache type fi
0x8000001d,	31:0,	eax,	7:5,	cache_level	, Cache level (
0x8000001d,	31:0,	eax,	8,	cache_self_init	, Self-initiali
0x8000001d,	31:0,	eax,	9,	fully_associative	, Fully-associa
0x8000001d,	31:0,	eax,	25:14,	num_threads_sharing	, Number of log
0x8000001d,	31:0,	ebx,	11:0,	cache_linesize	, System cohere
0x8000001d,	31:0,	ebx,	21:12,	cache_npartitions	, Physical line
0x8000001d,	31:0,	ebx,	31:22,	cache_nways	, Ways of assoc
0x8000001d,	31:0,	ecx,	30:0,	cache_nsets	, Cache number
0x8000001d,	31:0,	edx,	0,	wbinvd_rll_no_guarantee,	WBINVD/INVD n
0x8000001d,	31:0,	edx,	1,	ll_inclusive	, Cache is incl

Leaf 8000001EH

AMD CPU topology enumeration

0x8000001e,	0,	eax,	31:0,	ext_apic_id	, Extended APIC
0x8000001e,	0,	ebx,	7:0,	core_id	, Unique per-so

938,0-1 89%

Leaf AH

Intel PMU (Performance Monitoring Unit) enumeration

0xa,	0,	eax,	7:0,	pmu_version	, Performance m
0xa,	0,	eax,	15:8,	pmu_n_gcounters	, Number of gen
0xa,	0,	eax,	23:16,	pmu_gcounters_nbits	, Bitwidth of P
0xa,	0,	eax,	31:24,	pmu_cpuid_ebx_bits	, Length of cpu
0xa,	0,	ebx,	0,	no_core_cycle_evt	, Core cycle ev
0xa,	0,	ebx,	1,	no_insn_retired_evt	, Instruction r
0xa,	0,	ebx,	2,	no_refcycle_evt	, Reference cyc
0xa,	0,	ebx,	3,	no_llc_ref_evt	, LLC-reference
0xa,	0,	ebx,	4,	no_llc_miss_evt	, LLC-misses ev
0xa,	0,	ebx,	5,	no_br_insn_ret_evt	, Branch instru
0xa,	0,	ebx,	6,	no_br_mispredict_evt	, Branch mispre
0xa,	0,	ebx,	7,	no_td_slots_evt	, Topdown slots
0xa,	0,	ecx,	31:0,	pmu_fcounters_bitmap	, Fixed-functio
0xa,	0,	edx,	4:0,	pmu_n_fcounters	, Number of fix
0xa,	0,	edx,	12:5,	pmu_fcounters_nbits	, Bitwidth of P
0xa,	0,	edx,	15,	anythread_depr	, AnyThread dep

311,1

29%

```
Activities Terminal Nov 14 11:34 •
x + darwi@lx-t490: ~/pictures/Screenshots/pics
struct {
    // eax
    u32    cache_type          : 5, // Cache type field
          cache_level         : 3, // Cache level (1-based)
          cache_self_init     : 1, // Self-initialializing cache level
          fully_associative   : 1, // Fully-associative cache
                                   : 4, // Reserved
          num_threads_sharing : 12, // Number logical CPUs sharing this cache
          num_cores_on_die    : 6; // Number of cores in the physical package

    // ebx
    u32    cache_linesize      : 12, // System coherency line size (0-based)
          cache_npartitions    : 10, // Physical line partitions (0-based)
          cache_nways          : 10; // Ways of associativity (0-based)

    // ecx
    u32    cache_nsets         : 31, // Cache number of sets (0-based)
                                   : 1; // Reserved

    // edx
    u32    wbinvd_rll_no_guarantee : 1, // WBINVD/INVD not guaranteed for Remote Lower-Le
          ll_inclusive          : 1, // Cache is inclusive of Lower-Level caches
          complex_indexing      : 1, // Not a direct-mapped cache (complex function)
                                   : 29; // Reserved
} leaf4_sl0;
```

129,1-8

8%

```
struct {  
    // eax  
    u32    sgx1          : 1, // SGX1 leaf functions supported  
          sgx2          : 1, // SGX2 leaf functions supported  
          : 3, // Reserved  
          enclv_leaves  : 1, // ENCLV leaves (E{INC,DEC}VIRTCHILD, ESETCONTEXT  
          encls_leaves  : 1, // ENCLS leaves (ENCLS ETRACKC, ERDINFO, ELDBC, E  
          enclu_everifyreport2 : 1, // ENCLU leaf EVERIFYREPORT2 supported  
          : 2, // Reserved  
          encls_eupdatesvn : 1, // ENCLS leaf EUPDATESVN supported  
          enclu_edeccssa  : 1, // ENCLU leaf EDECCSSA supported  
          : 20; // Reserved  
    // ebx  
    u32    miscselect_exinfo : 1, // SSA.MISC frame: reporting #PF and #GP exceptio  
          miscselect_cpinfo  : 1, // SSA.MISC frame: reporting #CP exceptions insid  
          : 30; // Reserved  
    // edx  
    u32    max_enclave_sz_not64 : 8, // Maximum enclave size in non-64-bit mode (log2)  
          max_enclave_sz_64    : 8, // Maximum enclave size in 64-bit mode (log2)  
          : 16; // Reserved  
} leaf12_sl0;  
  
struct {
```

558,1-8

38%

Questions / Comments

[ANNOUNCE] x86-cpuid.org: A machine-readable CPUID data repository

<http://x86-cpuid.org/>

Thank you for your attention.

darwi@linutronix.de
info@linutronix.de