

Simulation of CPUID of Intel Architecture

Grigory Rechistov*

Name Surname[†]

March 16, 2014

Contents

| | | |
|----------|--|----------|
| 1 | Introduction | 1 |
| 1.1 | Contributions | 2 |
| 2 | Overview of Processor Identification | 2 |
| 2.1 | MIPS | 2 |
| 2.2 | ARM | 2 |
| 2.3 | IBM System z | 2 |
| 2.4 | PowerPC | 3 |
| 2.5 | SPARC | 3 |
| 2.6 | Intel IA-64 (Itanium) | 3 |
| 2.7 | Intel IA-32 and Intel 64 | 3 |
| 2.8 | Comparison of Processor Identification | 5 |
| 3 | What is so hard about Intel CPUID | 5 |
| 4 | Existing Approaches to CPUID Simulation | 6 |
| 4.1 | Bochs | 6 |
| 4.2 | Xen | 6 |
| 4.3 | Qemu | 6 |
| 4.4 | Simics | 6 |
| 5 | The Structured Approach | 6 |
| 6 | Evaluation | 6 |
| 6.1 | Compatibility with existing code | 6 |
| 6.2 | Extensions | 6 |
| 7 | Conclusions | 6 |
| 8 | Acknowledgements | 7 |

Abstract

In this paper we give an overview of existing microprocessor features identification facilities. Then we describe our approach to implementation of a software model of Intel IA-32 CPUID instruction. The described solution allows to define all recent CPUs' features, as well as future extensions. Our model was incorporated into the Wind River Simics* simulator framework.

Key words: cpu identification, cpu simulation, CPUID, Simics.

1 Introduction

TODO Write me

*Moscow Institute of Physics and Technology, grigory.rechistov@phystech.edu

[†]email@mail.com

1.1 Contributions

In this paper we make the following contributions.

1. Evaluate and compare existing means of processor features identification of different architectures.
2. Describe, implement and evaluate a structured solution to the simulation of CPUID instruction of Intel IA-32.

2 Overview of Processor Identification

This section outlines mechanisms for processor identification offered by several general purpose microprocessor architectures. We do not attempt to give exhaustive descriptions here, but only to reveal data that will allow us to see differences and similarities. More details can be found in documentation to respective architectures.

2.1 MIPS

This architecture processor identification with PRid register [7], which is the 15th register in Coprocessor 0. It contains 32 bits of information (Fig. 1), part of which is vendor-specific.

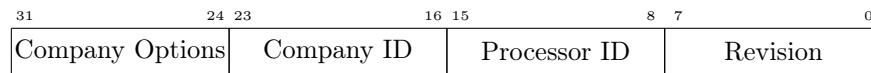


Figure 1: MIPS PRid register fields

TODO Check this
http://hwdb.mipt.cc/MIPS_PRId_register
<http://code.google.com/p/phantomuserland/source/browse/trunk/phantom/dev/mips/cpuid.c?r=1094>
<http://www.imgtec.com/mips/mips32-architecture.asp>

2.2 ARM

The popular RISC architecture provides certain means of a processor identification with CPUID base register [2]. It contains 32 bits of information (Fig. 2).

TODO Chehk this
http://infocenter.arm.com/help/topic/com.arm.doc.dai0099c/DAI0099C_core_type_rev_id.pdf
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0363e/ch01s11s01.html>
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0363e/Bcgdjadd.html>
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0388f/Babififh.html>
<http://forum.xda-developers.com/showthread.php?t=480226>

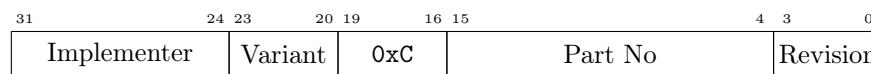


Figure 2: ARM CPUID base register fields

Examples of values [10]: Intel (XScale) PXA272 – 0x69054117, Qualcomm MSM7200A – 0x4117b362.

2.3 IBM System z

IBM System z10 [5] provides STSI and STIDP instructions. The information is rather scarce on the subject. **TODO** Expand

2.4 PowerPC

PowerPC [6] offers a 32-bit PVR register that contains just version and revision numbers.

For ISA extensions a mechanism of APU (application processor units) is employed. MSR (machine state register) is used to store information of APUs available.

TODO See also

<http://lxr.linux.no/#linux+v3.13.5/arch/powerpc/kernel/cputable.c#L2245> — identify_cpu function

http://cache.freescale.com/files/archives/doc/support_info/PPCPVR.pdf

2.5 SPARC

The SPARC standard [15] leaves quite a number of details implementation specific. To distinct between version a 64-bit register VER is defined (Fig. 3).

| | | | | | | | |
|--------------|----------------|----------|-------|-----------------|-----|------------|---|
| 63 | 48 47 | 32 31 | 24 23 | 16 15 | 8 7 | 5 4 | 0 |
| Manufacturer | Implementation | Revision | — | Max trap levels | — | Max window | |

Figure 3: SPARC VER register

2.6 Intel IA-64 (Itanium)

Intel Itanium™ [9] was designed later after the original 80x86 series and was meant to supplant it. A set of CPUID registers is used for identification purposes. The set's design somewhat resembles IA-32 CPUID instruction (discussed later) — its register numbers loosely resemble leaves of the latter. Five registers are guaranteed to be present, and bits 0–7 of CPUID [3] store the actual size of the register set (thus it is limited to 256 entries).

An example of Itanium CPUID registers content is given on Fig. 4.

2.7 Intel IA-32 and Intel 64

The common PC architecture, starting from Intel Pentium and its clones, provides CPUID [8, 1] instruction.

Since its inception it has been extended numerous number of times.

On Figure 5 an output of all defined leaves and subleaves for a modern Intel processor (microarchitecture Ivy Bridge) is demonstrated. The resulting table contains 25 tuples of 4 values of 32 bit width each, total of more than 3 kb of information. Essentially a number of values can be deduced from it.

- Processor producer brand (leaf 0) and SKU naming (leaves 0x80000002–0x80000004).
- Availability of ISA extensions such as 64 bit mode, SSEx, AVX, MOVBE etc.
- Cache configuration of all layers, both in legacy format (leaf 2) and in current format (leaf 4 with subleaves)
- Multi-processor configuration knowledge, such as availability of Intel HyperThreading, relative position inside CPU package (topology), presence of APIC (interrupt controller)
- Numerous architectural variables, such as addresses widths (leaf 0x80000008), availability of dynamic frequency scaling etc.

TODO Collect it

Figure 4: Contents CPUID registers of Intel Itanium

| Leaf | Subleaf | EAX | EBX | ECX | EDX |
|------------|---------|------------|------------|------------|------------|
| ----- | | | | | |
| 0 | 0 | 0xd | 0x756e6547 | 0x6c65746e | 0x49656e69 |
| 0x1 | 0 | 0x306a9 | 0x6100800 | 0x7f9ae3bf | 0xbfebfbff |
| 0x2 | 0 | 0x76035a01 | 0xf0b0ff | 0 | 0xca0000 |
| 0x4 | 0 | 0x1c004121 | 0x1c0003f | 0x3f | 0 |
| 0x4 | 0x1 | 0x1c004122 | 0x1c0003f | 0x3f | 0 |
| 0x4 | 0x2 | 0x1c004143 | 0x1c0003f | 0x1ff | 0 |
| 0x4 | 0x3 | 0x1c03c163 | 0x2c0003f | 0x1fff | 0x6 |
| 0x5 | 0 | 0x40 | 0x40 | 0x3 | 0x1120 |
| 0x6 | 0 | 0x77 | 0x2 | 0x9 | 0 |
| 0x7 | 0 | 0 | 0x281 | 0 | 0 |
| 0xa | 0 | 0x7300803 | 0 | 0 | 0x603 |
| 0xb | 0 | 0x1 | 0x1 | 0x100 | 0x6 |
| 0xb | 0x1 | 0x4 | 0x4 | 0x201 | 0x6 |
| 0xb | 0x2 | 0 | 0 | 0x2 | 0x6 |
| 0xb | 0x3 | 0 | 0 | 0x3 | 0x6 |
| 0xd | 0 | 0x7 | 0x340 | 0x340 | 0 |
| 0xd | 0x1 | 0x1 | 0 | 0 | 0 |
| 0xd | 0x2 | 0x100 | 0x240 | 0 | 0 |
| 0x80000000 | 0 | 0x80000008 | 0 | 0 | 0 |
| 0x80000001 | 0 | 0 | 0 | 0x1 | 0x28100800 |
| 0x80000002 | 0 | 0x20202020 | 0x20202020 | 0x65746e49 | 0x2952286c |
| 0x80000003 | 0 | 0x726f4320 | 0x4d542865 | 0x35692029 | 0x3534332d |
| 0x80000004 | 0 | 0x50432030 | 0x20402055 | 0x30312e33 | 0x7a4847 |
| 0x80000006 | 0 | 0 | 0 | 0x1006040 | 0 |
| 0x80000007 | 0 | 0 | 0 | 0 | 0x100 |
| 0x80000008 | 0 | 0x3024 | 0 | 0 | 0 |

Figure 5: Output of CPUID instruction obtained for Intel®Core™i5-3450 using `ggg-cpuid` [14]

```

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 rdtscp lm constant_tsc arch_perfmon
pebs bts nopl xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave
avx lahf_lm arat epb xsaveopt pln pts dtherm tpr_shadow vnmi flexpriority ept vpid

```

Figure 6: Part of output of `cat /proc/cpuinfo` command on GNU/Linux on a recent Intel CPU

2.8 Comparison of Processor Identification

Based on the presented data several conclusions can be made.

- CPU identification facilities differ greatly between architectures. They may be represented by instructions, registers, or groups of both.
- The most common thing that can be specified through a CPUID is a vendor identification. The next on popularity is indication of ISA extensions.
- The complexity/completeness of CPUID facilities depends on whether there are requirements to run the same binary code on hardware from multiple vendors and/or of different generations. If, in order to perform efficiently, software must “know” a list of supported instruction extensions and other types of model specific configuration, there has to be a documented way to obtain such knowledge.
- Conversely, CPUs provided by a single vendor and/or designed for specific software usually provide less means of self-identification. Most microcontrollers for embedded applications hardly provide even an idea of CPUID, compared to general purpose processors, because software is often written to be run on a specific HW; binaries are not meant to be moved to some other incarnation of the same architecture.

3 What is so hard about Intel CPUID

We now concentrate solely on the Intel IA-32 architecture and the single CPUID instruction. A complete definition of CPUID in [8] takes about 40 pages.

There is a number of complications that have resulted from long uncontrolled expansion of the CPUID.

Influence of Multiple Vendors Until recently, there were numerous CPU vendors that offered processors compatible with Intel architecture, including IBM, Cyrix, VIA, Centaur, Transmeta etc. By 2014, only two companies remain — Intel itself and AMD. Competition lead to the numerous small and subtle but essential differences in implementation. Until CPUID was introduced,

In the past, a robust identification of IA-32 processor brand and model required surprisingly intricate methods [4]. At present times, things are somewhat well documented. But then, another

Long history that lead to extremely long list of extensions. With 40 years of backwards-compatible development the IA-32 architecture collected a number of additions. Consider a list of flags which a modern GNU/Linux operating system shows for a CPUID of the laptop this paper is being written on (Fig. 6). It should be noted that only a part of information from CPUID is actually present on this list.

Now, moving to details of CPUID operation, we will show more complexity that arises from the semantics of CPUID instruction.

Elements addressing To inspect a value of a particular

- Leaves
- Subleaves
- Registers
- Bit range

Non-constant values Firmware is able to suppress certain features indicated by CPUID by manipulating bits of model specific register (MSR) IA32_MISC_ENABLE. For example: **TODO** NX, Leaf3, 1GB pages

Topology-variable elements Finally, it should be noted that, besides EAX, EBX, ECX, EDX, one more register may be affected by CPUID , namely IA32_SIGNATURE **TODO** .

4 Existing Approaches to CPUID Simulation

Every software simulator, emulator or virtual machine of a recent (Pentium or later) IA-32 system must guarantee CPUID operation of certain accuracy. If such tool is used for system firmware development, which is even more sensitive to identification information, the following requirements should be.

- Be accurate **TODO**
- Be configurable **TODO**

4.1 Bochs

Bochs [12] **TODO**

4.2 Xen

Xen [13] **TODO**

4.3 Qemu

Qemu [3] **TODO**

4.4 Simics

Wind River Simics [11] **TODO**

5 The Structured Approach

The described approach
It has the following advantages

- Uses natural unit of configuration state — a field.
- In the meantime, it allows its users to operate in terms of 32 bit leaves values which are more convenient and compact in many cases.
-

6 Evaluation

6.1 Compatibility with existing code

“legacy”

6.2 Extensions

TODO Field flags, such as “hidden”.

7 Conclusions

In this paper we described our approach to simulation of a single but complex processor instruction CPUID .

Model specific registers. Besides CPUID , processors of Intel architecture may have a quite extensive of model specific registers (MSRs), which can be read/written with RDMSR/WRMSR instructions, and also contain bits of processor identification. The simulation of MSRs is also an important aspect, but it is out of this paper focus. Other architectures may have similar facilities, e.g. special purpose registers (SPRs) in PowerPC.

8 Acknowledgements

Thanks to my parents for raising an awesome me.

References

- [1] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 1: Application Programming*, 2012.
- [2] ARM Limited. *CPUID Base Register*, 2008.
- [3] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *FREENIX Track: 2005 USENIX Annual Technical Conference*, 2005.
- [4] Robert R. Collins. CPUID algorithm wars. *Dr. Dobbs*, November 1996.
- [5] Per Fremstad, Wolfgang Fries, Marian Gasparovic, Parwez Hamid, Brian Hatfield, Dick Jorna, Fernando Nogal, and Karl-Erik Stenfors. *IBM System z10 Enterprise Class Technical Guide*. IBM, November 2009.
- [6] IBM. *PowerPC® Microprocessor Family: The Programming Environments Manual for 64-bit Microprocessors Version 3.0*, July 2005.
- [7] Imagination Technologies. *MIPS32 Architecture*, 2014.
- [8] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual. Volumes 1–3*, 2012.
- [9] Intel Corporation. *Intel® Itanium® Architecture Software Developer's Manual*, 2010.
- [10] intruders. ARM CPUID - processor model detection, 2009.
- [11] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A Full System Simulation Platform. *Computer*, 35:50–58, February 2002.
- [12] Darek Mihoka and Stanislav Shwartsman. Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure. *ISCA-35 Proceedings of the 1st Workshop on Architectural and Microarchitectural Support for Binary Translation*, 2008.
- [13] Ian Pratt. Xen and the art of virtualization, 2006.
- [14] Grigory Rechistov. ggg-cpuid, a program to print cpuid, 2014.
- [15] D.L. Weaver, T. Germond, and SPARC International. *The SPARC architecture manual: version 9*. PTR Prentice Hall, 1994.