The Germanium Core Architecture

Lilly Anderson
archaic.archea@gmail.com

1. Introduction

Germanium Core, or GeCo (pronounced the same as 'gecko'), is a RISC architecture designed to support sophisticated virtual memory systems, and stray away from safer designs such as in x86, ARM, and RISC-V which have all stuck to safer hardware managed page table systems. It is 64 bit with a framework for supporting larger 128 bit systems if the need arises. It also provides out of order execution to allow processors more options when attempting to optimize execution, this is more detailed in the EPIC and Instruction Reordering sections.

2. Features

GeCo boasts several features for supporting complex systems. To provide a framework for a complex virtual memory system we support software translation fault handlers, as well as a region and an (optional) VHPT system similar to that of Itanium's. In addition to virtual address space management, we also provide an IOMMU for controlling the IO address space, this can be programmed by the OS or disabled. For streamlining the process of communication between processors, we explicitly provide a small IPI bus to send over packets of around 128 bits, this can be used to inform the target processor of any OS or firmware specific services. Finally, we provide a multitude of options for optimizations when attempting to handle code and data running through the CPU, this comes in the form of Out of Order Execution primarily, as well as caching and processor hints.

3. Registers

GeCo provides 64 general purpose registers. The high register count acts to help improve performance, and reduce memory accesses.

4. Hints

To ensure the processor can adequately support programs that may have unusual, or more complex set of performance expectations we provide hints. These operate similarly to hints on other architectures such as x86, RISC-V, or Itanium. Supported hint types include caching hints, and execution hints. Caching hints inform the processor of how to handle a program's caching more efficiently. Execution hints tell the processor how it should handle out of order execution, if it should continue execution, or if it should change speed[1]. Hints can be represented by a 32 bit number, applicable 'hint arguments'effects can be put into the lower 14 bits these arguments are not guaranteed any explicit ordering or layout, the only requirement is that no hint have effects visible to the software.

[1]A good portion of execution hints will likely be priviledged, this is to avoid potentially letting user programs have too much control over system performance.

Value	Usage
0x00_000000 -> 0x07_FFFFFF	Reserved for standard usage
0x08_000000 -> 0xBF_FFFFFF	Reserved for platform usage
0xC0_000000 -> 0xFF_FFFFFF	Reserved for custom usage

4.1. Standardized Hints

To provide a better framework for optimizations across micro-architectures and platforms, the *GeCo* specification provides options for consistent hint encoding. This allows a user program optimized for the Champion platform, can also function, and maybe even already optimized for, a custom platform. Standardized hints take up the range `0x00->0x07` in the upper 8 bits.

4.2. Platform Specific Hints

Some platforms provide common features that may not necessarily be standardized in the *GeCo* spec. These features may be important to account for if you're targetting a platform that's optimized for a certain set of tasks such as a personal computer, or embedded platform. To help accommodate these platforms *GeCo* provides a range of hints that a platform can designate for itself, the only requirement that comes with this is that said hints are documented, and freely accessible. If you do not have the means to host your own server for distributing your specification, you may contact the Germanium project and we can attempt to accommodate you by providing file hosting[2]. The hint range provided for these processor hints falls is range `0x08->0xBF` in the upper 8 bits.

4.3. Processor Specific Hints

To help with further optimizations that some micro-architectures may need or want to engage with there is also a range of hints provided for specific processors. Like platform specific hints, processor specific hints are required to be freely available. The hint range provided for these processor hints falls is range `0xC0->0xFF` in the upper 8 bits.

5. Encodings

GeCo supports several different instruction encodings, and encoding widths, this helps to further optimize a processor for different tasks, such as low memory footprints, larger and faster memory operations, multi-threading, or larger address spaces. Although compilers must provide alignment as specified in their respective encoding sections, an implementation is not held to the same requirements and is allowed to provide a lower alignment requirement for the instruction. This means that for *GeM* instruction encoding (16-bits), the compiler must provide 2 bytes of alignment. However, a *GeCo* implementation may reduce the alignment requirement to 8 bits. It is important to note that this behavior should never be relied upon by compilers, as it is not guaranteed.

Bit range	Usage

[2]The *Germanium* project may occasionally search for platform specifications and store them in an archive server if available. If you would like to access something in this archive, request removal of content, or request a document be archived, please contact one of the authors of this paper at their provided email.

0..1	Encoding size 0
2..95	Encoding specific
96..99	Encoding size 1
100..127	Encoding specific

5.1. Germanium Micro (GeM) Encoding

The GeM Encoding is a 16 bit encoding used to help compress the size of a program, and potentially increase performance. This encoding provides access to some hints, common math, and relative memory accesses. It comes in the form of 2 primary encodings: MRAcc, Memory Relative Access; and CI, Common Instruction. Bits 16 and up of the instruction will be ignored, and the alignment of the instruction must at least be 16 bits.

5.1.1. MRAcc

MRAcc (Memory Relative Access) allows you to quickly access memory relative to a a few common registers, often known as the “High Access” registers.

6. Assistance

Many people have provided assistance in editting this document and although they are not authors, it is still important to give them credit.

- Jaiden Garcia