**Goblin-Core Architecture Specification**

Version 1.0.0

Legal Information

**Change Log**

|  |  |
| --- | --- |
| Date | Description |
| 06/26/13 | v.1.0.0 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Table of Contents**

1. Introduction 6

1.1. Revision 6

1.2. Authors 6

1.3. Terms and Abbreviations 6

1.4. References 6

1.5. Document Conventions 6

2. Architecture Overview 6

2.1. GC64 Architecture Components 7

2.1.1. Task Group 10

2.1.2. Task Processor 10

2.1.3. Task Unit 10

2.1.4. External Bus Architecture 11

2.1.5. MMU Bus Architecture 11

2.1.6. Local Memory Interface 11

2.1.7. Global Memory Interface 12

3. Addressing Modes & Operand Conventions 12

3.1. Addressing Modes 12

3.1.1. Local Register Indirect with Displacement 12

3.1.2. Global Register Indirect with Displacement 12

3.1.3. Instruction Pointer [IP] Relative 13

3.2. Operand Conventions 13

4. Execution Modes 14

4.1. Scalar Mode 14

4.2. Vector Mode 14

4.3. Task Switch Algorithm 14

5. Exception Model 14

6. Memory Management Units 14

6.1. Local MMU 14

6.2. Global MMU 14

7. Hazard Model 14

8. Interrupt Model 14

9. Instruction Set 14

10. Debug Model 14

11. Application Binary Interface 15

11.1. Function Call Conventions 15

11.1.1. Call Semantics 15

11.1.2. Parameter Lists 15

11.1.3. Return Codes 15

11.2. System Calls 15

12. Register Organization 15

12.1. Scalar Register Indexing 15

12.2. Vector Register Indexing 17

12.3. GCONST Register 18

12.4. Performance Counter Access 19

13. Instruction Set Architecture 19

13.1. Instruction Set Format 19

13.2. Instruction Descriptions 22

14. Appendix A. Instruction Set Table 22

15. Appendix B. Performance Counters 22

16. Appendix NNNN. 22

**Table of Figures**

# Introduction

# Revision

**Version 1.0.0**: Initial Release of Goblin-Core 64 Architecture Specification

# Authors

**Architecture Lead:** John D. Leidel, Texas Tech University

# Terms and Abbreviations

**ABI** – Application Binary Interface specification.

**AMO** – Atomic Memory Operation

**GC64** – Abbreviation for the Goblin-Core 64-bit processor architecture.

**ILP** – Instruction Level Parallelism

**MIMD** – Multiple Instruction Multiple Data

**MMU** – Memory Management Unit

**PGAS** – Partitioned Global Address Space [See, UPC, SHMEM]

**Rn** – General purpose register [where ‘n’ is 0-31]

**SHMEM** – Shared Memory Programming Model [Does not refer to SysV IPC]

**UPC** – Unified Parallel C Programming Model

**Vn** – Vector register [where ‘n’ is 0-7]

# References

# Document Conventions

# Architecture Overview

The Goblin-Core architecture [herein referred to as *GC64*] is a part of a new class of core processors that provide direct architectural support for higher level programming models. The goals of the GC64 architecture are as follows:

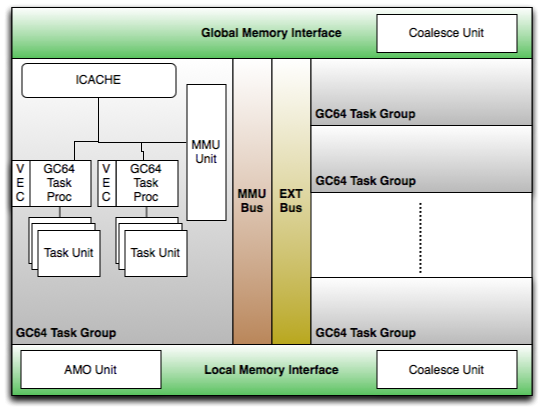
* Provide simple architectural features that dramatically increase the efficiency of MIMD-style programming models [threading, tasking, et,al]
* Provide a powerful platform well suited to solve non-deterministic or data-intensive algorithms [graph theory, combinatorics, et.al.]
* Provide native Partitioned Global Address [PGAS] instruction set support and memory addressing
* Provide a very compact and compressed ISA format that supports both scalar operation and SIMD [ILP] operation
* Provide high bandwidth, intelligent coalescing memory interfaces
* Provide a flexible platform for future expansion of computational density as process techniques continue to improve

In the future, the GC64 architecture may also become a novel deployment infrastructure for technologies such as the Micron Hybrid Memory Cube [HMC] and various silicon photonics technologies.

# GC64 Architecture Components

The GC64 architecture consists of a set of modular components. These components and all the software simulation tools associated with the GC64 project are designed to interact with one another in a modular fashion such that the size and shape of individual components is flexible based upon the specific research and development needs. The following section defines the individual component modules and any respective requirements as to their function and existence within the overall infrastructure.

At a high level, the GC64 utilizes a RISC core architecture that maintains in order execution. Parallelism is obtained via the use of tasks, which are mapped to threads or processes by the operating system and/or programming model. Parallelism is also obtained by the use of short SIMD instructions within a given task. The memory model is weakly ordered with many outstanding requests such that applications may cover the latency to memory. There are no explicit or implicit data caches in the memory pipeline. There are, however, memory coalescing units on both the local and global memory pipelines in order to increase the theoretical bandwidth. The local memory unit also contains an atomic memory operation [AMO] unit.



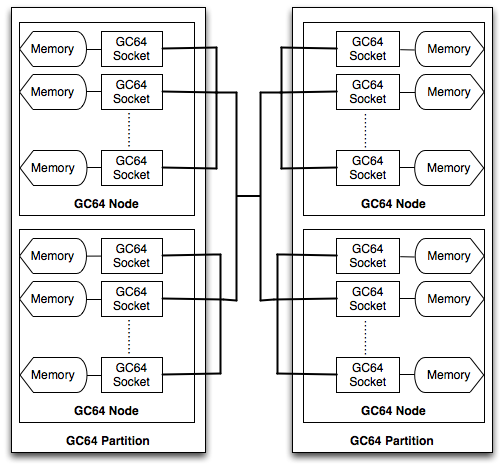
**Figure N. GC64 Architecture**

The distribution of modules within the GC64 architecture is symmetric. All GC64 sockets contain the same number of Task Groups. All Task Groups contain the same number of Task Processors and all Task Processors contain the same number Task Units.

|  |  |  |  |
| --- | --- | --- | --- |
| Unit | Minimum Architectural Constraint | Maximum Architectural Constraint | Notes |
| GC64 System Partitions | 1 | 65536 |  |
| GC64 Nodes Per Partition | 1 | 65536 |  |
| GC64 Sockets per Node | 1 | 256 |  |
| Task Groups per Socket | 1 | 256 |  |
| Task Processors per Group | 1 | 256 |  |
| Task Units per Task Processor | 1 | 256 |  |

**Figure N. GC64 Architecture Dimensions**

The GC64 architecture also contains a set of unique PGAS architectural and memory addressing characteristics. The GC64 system architecture has the notion of a *partition* as well as a *node*. Each partition is made up one or more nodes. Each node is made up of one or more sockets. This interconnection schema is extended into a global addressing format that directly maps virtual address blocks with their respective locality information within the total system. As such, memory requests can be executed to load, modify or store blocks of data at addresses that combine a base virtual address and the GC64 locality data. These *global addresses* [GA] provide the foundation for PGAS-style parallelism in the GC64 architecture.

****

**Figure N. GC64 PGAS System Architecture**

**[does not reflect actual system interconnect topology]**

# Task Group

The GC64 architecture contains the notion of a *group* of processing units. This *Task Group* is an important modular hardware unit in the GC64 architecture. The Task Group module contains one or more Task Processors and they’re associated Task Units. The minimum number of Task Groups per GC64 socket is one [1]. The maximum number of Task Groups per GC64 socket is two hundred fifty six [256].

Within each Task Group, there exists a shared instruction cache [ICACHE] unit. This ICACHE unit is shared amongst all the Task Processors within the respective Task Group. Currently, the ICACHE unit is an 8-way set associated instruction cache with a variable number of sets. The size of the ICACHE is dependent upon the number of Task Processors and Task Units. The pressure on the instruction cache is a function of the number of Task Processors and Task Units.

The Task Group also serves as each Task Processor’s interface to the Memory Management Bus. The Task Group memory management unit [MMU] interface provides the necessary transaction ID’s for the respective Task Processor’s memory requests. It also provides the ability to define any prospective ordering guarantees for the respective Task Processors without perturbing the weakly ordered nature of the entire memory bus architecture.

# Task Processor

Each Task Group contains one or more *Task Processors*. The Task Processor is analogous to a *core* in traditional microprocessor architectures. The minimum number of Task Processors in each Task Group is one [1]. The maximum number of Task Processors in each Task Group is two hundred fifty six [256].

Each Task Processor represents a single in-order, RISC core with both a scalar and a short SIMD [vector] unit. Each Task Processor has some number of Task Units associated with it. Task Processors within a Task Group operate independently from one another. Each Task Processor shares it access to the execution pipelines [both scalar and SIMD] amongst the Task Units associated with it. Section 9 describes the instruction set architecture implemented by each Task Processor.

# Task Unit

The smallest unit of divisible MIMD parallelism in the GC64 architecture is the Task Unit. Each Task Unit is designed to represent a task, thread, process or other divisible unit of software parallelism within the scope of the respective operating system or programming model. There exists a minimum of one [1] Task Unit associated with each Task Processor. There exists a maximum of two hundred fifty six [256] associated with each Task Processor.

Each Task Unit operates autonomously from the other Task Units associated with the respective Task Processor. Each Task Unit has a unique and private register file and hazard block. Time in the respective Task Processor pipeline is shared amongst its associated Task Units. Task Units have no ability to utilize execution resources for Task Processors outside of what it is directly associated with. The time sharing algorithms implemented in the GC64 Task Processor that controls the access to the execution resources is described in Section 4.3.

# External Bus Architecture

The GC64 External Bus [External System Bus] provides access to interrupt controller resources, PCIe resources and other off-chip, non-memory resources.

# MMU Bus Architecture

The GC64 Memory Bus Architecture serves three main purposes. First, it provides access from each Task Group to the Local Memory Interface unit. All local memory requests from all Task Groups travel via the MMU Bus. Second, it provides access from each Task Group to the Global Memory Interface unit. All global memory requests from all Task Groups travel via the MMU Bus. Finally, memory requests incoming from adjacent GC64 sockets, nodes or partitions arrive via the Global Memory Interface. Once arrived, the Global Memory Unit forwards the request to the Local Memory Unit via the Memory Bus.

# Local Memory Interface

The GC64 Local Memory Interface provides access to the directly connected main memory storage units associated with the respective socket. The Local Memory Interface provides three main sub functions. First, it provides a memory coalescing unit whose logic shall recognize aligned memory references and coalesce these references into single requests. In doing so, the Local Memory Unit makes more efficient use of the respective main memory storage buses.

Second, the Local Memory Unit provides an atomic memory operation unit that interoperates with the coalescing unit. The AMO unit acts not only as a read-modify-write engine, but it also acts as an AMO cache. Values that are frequently accessed via AMO transactions are held in an AMO cache such that multiple AMO operations can be performed with a single read operation. Intermediate values are flushed in order to their respective physical locations in order to maintain state.

Finally, the Local Memory Interface contains the necessary Translation Lookaside Buffer [TLB] that performs the required page table management and virtual to physical translation.

# Global Memory Interface

The GC64 Global Memory Interfaces provides access from the local socket to global memory regions and from remote sockets to local [remotely addressed] memory regions. In this manner, the Global Memory Interface has the ability to encode requests for remote sockets and decode requests from remote sockets. Any incoming requests from remote sockets are verified and subsequently forwarded to the Local Memory Interface via the MMU bus.

The Global Memory Interface also contains a memory coalescing unit. In similar fashion as to the Local Memory Interface, the Global coalescing unit attempts to block multiple aligned memory transactions into a single request in order to optimize the use of global bandwidth.

# Addressing Modes & Operand Conventions

The following section describes the local and global address mode conventions for the Goblin-Core architecture family. This includes local and global address modes as well as the associated operand conventions as they appear in the instruction set mnemonics.

# Addressing Modes

# Local Register Indirect with Displacement

# Global Register Indirect with Displacement

# Instruction Pointer [IP] Relative

# Operand Conventions

The architecture defines 8-bit byte word, 16-bit word, 32-bit double word, and 64-bit quad word. The architecture defines an IEEE-754 32-bit single precision floating point representation as well as an IEEE-754 64-bit double precision floating point representation. The architecture also defines a 128-bit global addressing format. The most significant quad word [64-bits] contains the address portion of the global address operand. The least significant quad word [64-bits] contains the interconnect locality portion of the address operand. However, 128-bit global address operands map to 64-bit quad word data payloads.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Mnemonic | Size [bytes] | Notes |
| Unsigned Byte | UB | 1 |  |
| Unsigned Word | UW | 2 |  |
| Unsigned Double | UD | 4 |  |
| Unsigned Quad | UQ | 8 |  |
| Signed Byte | SB | 1 |  |
| Signed Word | SW | 2 |  |
| Signed Double | SD | 4 |  |
| Float Single | FS | 4 | IEEE-754 32-bit single precision floating point |
| Float Double | FD | 8 | IEEE-754 64-bit double precision floating point |
| Global Address | GA | 16 | Global Addressing operand mode. Most significant quad contains the address portion of the operand. Least significant quad contains the interconnect locality portion. |

**Figure N. Operand Conventions**

# Execution Modes

# Scalar Mode

# Vector Mode

# Task Switch Algorithm

# Exception Model

# Memory Management Units

# Local MMU

# Global MMU

# Hazard Model

# Interrupt Model

# Instruction Set

# Debug Model

# Application Binary Interface

# Function Call Conventions

# Call Semantics

# Parameter Lists

# Return Codes

# System Calls

# Register Organization

The GC64 architecture defines a set of user-accessible user registers. These user registers are split into two banks, user writable registers and architecture constants. Registers are accessed via their direct index values as register renaming is not supported in the GC64 architecture. As such, the maximum number of user registers per the instruction set format is 64 [index 0x3F]. All read+write registers are indexed 0x00 – 0x34. All read-only registers are indexed 0x35 - 0x3F.

# Scalar Register Indexing

The GC64 base scalar mode contains thirty two [32] general-purpose scalar user registers. These registers are referred to as ‘R’ registers in the assembly mnemonics [R0-R31]. The remainder of the read-write registers have a pre-defined purpose, respectively. The following table defines the set of scalar-accessible registers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Name | Size [bits] | Mode | Description |
| 0x00 | R0 | 64 | Read-Write | General register |
| 0x01 | R1 | 64 | Read-Write | General register |
| 0x02 | R2 | 64 | Read-Write | General register |
| 0x03 | R3 | 64 | Read-Write | General register |
| 0x04 | R4 | 64 | Read-Write | General register |
| 0x05 | R5 | 64 | Read-Write | General register |
| 0x06 | R6 | 64 | Read-Write | General register |
| 0x07 | R7 | 64 | Read-Write | General register |
| 0x08 | R8 | 64 | Read-Write | General register |
| 0x09 | R9 | 64 | Read-Write | General register |
| 0x0A | R10 | 64 | Read-Write | General register |
| 0x0B | R11 | 64 | Read-Write | General register |
| 0x0C | R12 | 64 | Read-Write | General register |
| 0x0D | R13 | 64 | Read-Write | General register |
| 0x0E | R14 | 64 | Read-Write | General register |
| 0x0F | R15 | 64 | Read-Write | General register |
| 0x10 | R16 | 64 | Read-Write | General register |
| 0x11 | R17 | 64 | Read-Write | General register |
| 0x12 | R18 | 64 | Read-Write | General register |
| 0x13 | R19 | 64 | Read-Write | General register |
| 0x14 | R20 | 64 | Read-Write | General register |
| 0x15 | R21 | 64 | Read-Write | General register |
| 0x16 | R22 | 64 | Read-Write | General register |
| 0x17 | R23 | 64 | Read-Write | General register |
| 0x18 | R24 | 64 | Read-Write | General register |
| 0x19 | R25 | 64 | Read-Write | General register |
| 0x1A | R26 | 64 | Read-Write | General register |
| 0x1B | R27 | 64 | Read-Write | General register |
| 0x1C | R28 | 64 | Read-Write | General register |
| 0x1D | R29 | 64 | Read-Write | General register |
| 0x1E | R30 | 64 | Read-Write | General register |
| 0x1F | R31 | 64 | Read-Write | General register |
| 0x20 | SP | 64 | Read-Write | Stack Pointer |
| 0x21 | FP | 64 | Read-Write | Frame Pointer |
| 0x22 | PIC | 64 | Read-Write | Processor Independent Code Pointer |
| 0x23 | UP | 64 | Read-Write | Uplevel Frame Pointer |
| 0x24 | RP | 64 | Read-Write | Return Pointer |
| 0x25 | CC0 | 64 | Read-Write | Compare Register 0 |
| 0x26 | CC1 | 64 | Read-Write | Compare Register 1 |
| 0x27 | CC2 | 64 | Read-Write | Compare Register 2 |
| 0x28 | CC3 | 64 | Read-Write | Compare Register 3 |
| 0x29 | TQ | 64 | Read-Write | Task Queue |
| 0x2A | VL | 64 | Read-Write | Vector Length |
| 0x2B | VS | 64 | Read-Write | Vector Stride |
| 0x2C | GMODE | 64 | Read-Write | GC64 Mode |
| 0x2D | PMASK | 64 | Read-Write | Performance Counter Mask |
| 0x2E | PREAD | 64 | Read-Write | Performance Counter Read Index |
| 0x2F | TID | 64 | Read-Write | Thread ID |
| 0x30 | (--) | (--) | (--) | (--) |
| 0x31 | (--) | (--) | (--) | (--) |
| 0x32 | (--) | (--) | (--) | (--) |
| 0x33 | (--) | (--) | (--) | (--) |
| 0x34 | (--) | (--) | (--) | (--) |
| 0x35 | ZERO | 64 | Read-Only | Zero: 0x00LL |
| 0x36 | IMM32 | 64 | Read-Only | 4-Byte Immediate |
| 0x37 | IMM64 | 64 | Read-Only | 8-Byte Immediate |
| 0x38 | GCONST | 64 | Read-Only | GC64 Architecture Constant |
| 0x39 | EQ | 64 | Read-Only | Compare Val: Equal |
| 0x3A | GT | 64 | Read-Only | Compare Val: Greater Than |
| 0x3B | GTE | 64 | Read-Only | Compare Val: Greater Than + Equal |
| 0x3C | LT | 64 | Read-Only | Compare Val: Less Than |
| 0x3D | LTE | 64 | Read-Only | Compare Val: Less Than + Equal |
| 0x3E | NE | 64 | Read-Only | Compare Val: Not Equal |
| 0x3F | PVAL | 48 | Read-Only | Performance Counter Value |

**Figure N. Register Index Table**

# Vector Register Indexing

The GC64 architecture supports an aliased vector indexing mode into the aforementioned register file. Vector indexes map to multiple, strided scalar *%Rn* registers. Vector registers are denoted with the mnemonic *%Vn* by the assembler. Changing index modes from scalar [*%Rn*] to vector [*%Vn*] and vice versa does not modify the contents of the register. The only registers available via the vector aliased mode are the general purpose registers. All control and constant registers are always considered scalars.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vector Register | Vector Element | Scalar Register | Register Index | Size [bits] |
| V0 | Elem 0 | R0 | 0x00 | 64 |
| V0 | Elem 1 | R1 | 0x01 | 64 |
| V0 | Elem 2 | R2 | 0x02 | 64 |
| V0 | Elem 3 | R3 | 0x03 | 64 |
| V1 | Elem 0 | R4 | 0x04 | 64 |
| V1 | Elem 1 | R5 | 0x05 | 64 |
| V1 | Elem 2 | R6 | 0x06 | 64 |
| V1 | Elem 3 | R7 | 0x07 | 64 |
| V2 | Elem 0 | R8 | 0x08 | 64 |
| V2 | Elem 1 | R9 | 0x09 | 64 |
| V2 | Elem 2 | R10 | 0x0A | 64 |
| V2 | Elem 3 | R11 | 0x0B | 64 |
| V3 | Elem 0 | R12 | 0x0C | 64 |
| V3 | Elem 1 | R13 | 0x0D | 64 |
| V3 | Elem 2 | R14 | 0x0E | 64 |
| V3 | Elem 3 | R15 | 0x0F | 64 |
| V4 | Elem 0 | R16 | 0x10 | 64 |
| V4 | Elem 1 | R17 | 0x11 | 64 |
| V4 | Elem 2 | R18 | 0x12 | 64 |
| V4 | Elem 3 | R19 | 0x13 | 64 |
| V5 | Elem 0 | R20 | 0x14 | 64 |
| V5 | Elem 1 | R21 | 0x15 | 64 |
| V5 | Elem 2 | R22 | 0x16 | 64 |
| V5 | Elem 3 | R23 | 0x17 | 64 |
| V6 | Elem 0 | R24 | 0x18 | 64 |
| V6 | Elem 1 | R25 | 0x19 | 64 |
| V6 | Elem 2 | R26 | 0x1A | 64 |
| V6 | Elem 3 | R27 | 0x1B | 64 |
| V7 | Elem 0 | R28 | 0x1C | 64 |
| V7 | Elem 1 | R29 | 0x1D | 64 |
| V7 | Elem 2 | R30 | 0x1E | 64 |
| V7 | Elem 3 | R31 | 0x1F | 64 |

**Figure N. Vector Register Aliasing Table**

# GCONST Register

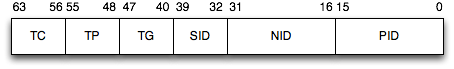
The GCONST register contains hardware locality information and hardware spatial information. This register information maps directly to the global address [\*.GA] locality information. The least significant bit field [PID] signifies the PartitionID of the respective GC64 unit. The second field [NID] signifies the NodeID of the respective GC64 unit within the target partition. The third filed [SID] signifies the SocketID of the respective GC64 unit within the target node.

The fourth field [TG] signifies the number of Task Groups in the respective GC64 socket. The fifth field [TP] signifies the number of Task Processors within each Task Group. The final field [TC] signifies the number of hardware tasks associated with each Task Processor.

The following table describes each field:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Bits | Size [bits] | Description |
| PID | [15,0] | 16 | Partition ID |
| NID | [31,16] | 16 | Node ID |
| SID | [39,32] | 8 | Socket ID |
| TG | [47,40] | 8 | Number of Task Groups |
| TP | [55,48] | 8 | Number of Task Processors per Task Group |
| TC | [63,56] | 8 | Number of Tasks per Task Processor |

**Figure N. GCONST Register Fields**



**Figure N. GCONST Register**

# Performance Counter Access

# Instruction Set Architecture

# Instruction Set Format

The GC64 instruction format is designed explicitly to provide dense instruction packing and a high degree of flexibility within a fixed target area. As such, the GC64 architecture utilizes a single, 64-bit instruction format. Two individual 32-bit instruction payloads are packed into the 64-bit format. Optionally, GC64 can utilize a second, 64-bit payload containing two 4-byte or one 8-byte immediate value. The instruction payloads must be aligned on 8-byte boundaries. The immediate payload must fall at *Instruction Pointer+0x08*.

The instruction payloads are each split into five [5] fields. The lower three fields contain six [6] bit register operands, respectively. The register indexes are noted in Section 12.1. The fourth field represents the eight [8] bit opcode field. The fifth field represents the instruction control field.

The instruction control field contains several sub-fields that control the execution and register arguments for the respective instruction payload. The least significant control bit [26,58] is known as the “vector” bit. This enables the respective instruction payload to perform a vector/SIMD operation rather than a simple scalar operation. Only certain instructions are permitted to operate in vector mode. See Appendix A for more details on candidate vector instructions.

The next two control fields, known as V0 [27,59] and V1 [28,60], enable register arguments R0 and R1 to operate in vector mode [vector register indexing, See Section 12.2]. When operating in vector mode, the target register, R2, is always considered to be a vector register [when enabled].

The fourth control field, BRK [29,61], instructs the pipeline to take a breakpoint and perform a context save operation.

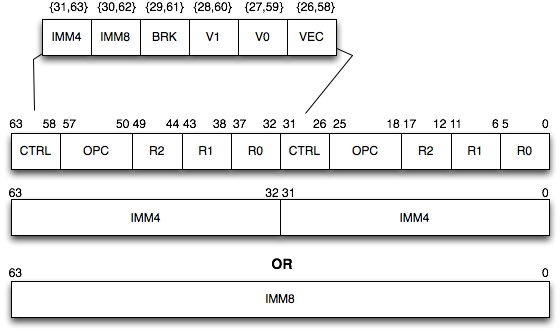
The fifth and sixth fields indicate the presence of a single 8-byte immediate value, IMM8 [30,62], or two 4-byte immediate values, IMM4 [31,63]. It is not required that both 4-byte instruction payloads utilize the immediate value in the immediate payload.

The following table describes each field.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Bits | Size [bits] | Description |
| Lower Instruction Payload | | | |
| R0 | [5,0] | 6 | Lower instruction, register argument 0 |
| R1 | [11,6] | 6 | Lower instruction, register argument 1 |
| R2 | [17,12] | 6 | Lower instruction, register argument 2 |
| OPC | [25,18] | 8 | Lower instruction, opcode field |
| CTRL | [31,26] | 6 | Lower instruction, control field |
| Upper Instruction Payload | | | |
| R0 | [37,32] | 6 | Upper instruction, register argument 0 |
| R1 | [43,38] | 6 | Upper instruction, register argument 1 |
| R2 | [49,44] | 6 | Upper instruction, register argument 2 |
| OPC | [57,50] | 8 | Upper instruction, opcode field |
| CTRL | [63,58] | 6 | Upper instruction, control field |
| Control Fields | | | |
| VEC | {58,26} | 1 | Vector instruction |
| V0 | {59,27} | 1 | Register operand 0 is a vector register |
| V1 | {60,28} | 1 | Register operand 1 is a vector register |
| BRK | {61,29} | 1 | Signals a breakpoint on this instruction payload |
| IMM8 | {62,30} | 1 | Signals the presence of 8-byte immediate payload |
| IMM4 | {63,31} | 1 | Signals the presence of 4-byte immediate payload |

**Figure N. Instruction Field Descriptions**

***NOTE:*** An instruction payload quad word and an immediate quad word CANNOT span page boundaries. We set this restriction in order to refrain from taking page faults during an ICACHE miss.



**Figure N. GC64 Instruction Format**

# Instruction Descriptions

# Appendix A. Instruction Set Table

# Appendix B. Performance Counters

# Appendix NNNN.