|  |
| --- |
| . |
| Theia.v3 architecture specification |
| Version 0.1  Last update Sunday, April 12, 2015 |

|  |
| --- |
|  |

Revision & Review

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description | Author(s) | Date <yyyy-mm-dd> |
| 0.1 | Initial document | Diego Valverde Garro | 2015-12-19 |
|  |  |  |  |
|  |  |  |  |

Table of contents

[Revision & Review 1](#_Toc347658231)

[Table of contents 2](#_Toc347658232)

[Table of tables 7](#_Toc347658233)

[1. Introduction 11](#_Toc347658234)

[1.1. Vector processing 13](#_Toc347658235)

[1.2. Combining Vector processing and out-of-order execution. 14](#_Toc347658236)

[2. System Overview 15](#_Toc347658237)

[2.1. Control processor Overview 17](#_Toc347658238)

[1.1. Vector processors (VP) 18](#_Toc347658239)

[2. Control FSM 20](#_Toc347658240)

[3. Vector Processor CORE (VP CORE) 21](#_Toc347658241)

[3.1. Introduction 21](#_Toc347658242)

[3.1.1. Single Thread execution example 22](#_Toc347658243)

[3.2. VP Architecture 25](#_Toc347658244)

[3.3. Word size and Endianness 28](#_Toc347658245)

[3.4. Fixed point arithmetic. 28](#_Toc347658246)

[3.5. Instruction overview 29](#_Toc347658247)

[3.5.1. Instruction operation codes 30](#_Toc347658248)

[3.5.2. Instruction destination block selector 32](#_Toc347658249)

[3.5.3. Instruction source modifiers 33](#_Toc347658250)

[3.5.4. Data dependencies and source modifiers 36](#_Toc347658251)

[3.5.4.1. VP Flags 40](#_Toc347658252)

[3.5.5. Execution units and reservation stations 41](#_Toc347658253)

[3.5.6. VP Stall conditions 44](#_Toc347658254)

[3.6. Instruction addressing modes 45](#_Toc347658255)

[3.7. Instruction word fields 49](#_Toc347658256)

[3.8. Addressing mode encoding 50](#_Toc347658257)

[3.9. Selecting the Arithmetic operation 54](#_Toc347658258)

[3.10. Fixed point Square Root unit 55](#_Toc347658259)

[3.11. Bitwise logic operations 56](#_Toc347658260)

[3.12. Destination write channel control 57](#_Toc347658261)

[3.13. Operand Scale control 58](#_Toc347658262)

[3.14. Operand Sign control 58](#_Toc347658263)

[3.15. Operand swizzle control 59](#_Toc347658264)

[3.16. Branching operations 61](#_Toc347658265)

[3.16.1. Unconditional branches 62](#_Toc347658266)

[3.16.2. Conditional Branches 63](#_Toc347658267)

[4. VP Data path 64](#_Toc347658268)

[4.1.1. Instruction issue unit (IIU) 68](#_Toc347658269)

[Dependency Table 70](#_Toc347658270)

[1.1.1. Source Modification unit (SMU) 72](#_Toc347658271)

[1.1.1.1. Issue Bus (IBUS) 74](#_Toc347658272)

[1.1.1.2. Commit Bus (CBUS) 75](#_Toc347658273)

[2. VP SMT (simultaneous multithreading) 76](#_Toc347658274)

[3. VP IO 77](#_Toc347658275)

[3.1. IO instructions 77](#_Toc347658276)

[3.2. Output memory OMEM 78](#_Toc347658277)

[3.3. Texture memory TMEM 82](#_Toc347658278)

[4. VP Register specification 84](#_Toc347658279)

[4.1. General purpose registers (GPRs) 84](#_Toc347658280)

[4.1.1. Zero register – R0. 86](#_Toc347658281)

[4.1.2. Return address register – R2.x 87](#_Toc347658282)

[4.1.3. Offset registers – R3.x, R2.y 87](#_Toc347658283)

[4.1.4. Function parameters– R4 - R9 88](#_Toc347658284)

[4.2. Shadowed GPRs 89](#_Toc347658285)

[4.3. Special purpose registers (SPRs) 90](#_Toc347658286)

[5. Control Processor architecture (CP) 93](#_Toc347658287)

[5.1.1. Data block copy operations 93](#_Toc347658288)

[5.1.2. Control processor messages 95](#_Toc347658289)

[5.1.3. Mail-boxing 96](#_Toc347658290)

[5.2. CP Instruction set 98](#_Toc347658291)

[5.3. CP COPYBLOCK command 101](#_Toc347658292)

[5.4. CP Special general purpose registers 102](#_Toc347658293)

[5.5. CP Special purpose registers (SPRs) 102](#_Toc347658294)

[5.6. CP Branching 104](#_Toc347658295)

[6. Internal Memory Controller (MCU) Architecture 106](#_Toc347658296)

[7. Appendix A: VP Issue unit encoding table 106](#_Toc347658297)

[8. Appendix B: VP addressing mode examples 0](#_Toc347658298)

[Works Cited 9](#_Toc347658299)

[Figure 1 THEIA environment overview 12](#_Toc347658300)

[Figure 2 Data pipeline in an execution unit 13](#_Toc347658301)

[Figure 3 X, Y and X data vector data lanes 14](#_Toc347658302)

[Figure 4 Convoy chaining 15](#_Toc347658303)

[Figure 5 The GPU simplified block diagram 16](#_Toc347658304)

[Figure 6 Control Processor within the system 18](#_Toc347658305)

[Figure 7 The main blocks of a CORE 19](#_Toc347658306)

[Figure 8 The CONTROL FSM, the CP and the VP Core 20](#_Toc347658307)

[Figure 9 Control FSM 21](#_Toc347658308)

[Figure 10 A sample code (single thread) 23](#_Toc347658309)

[Figure 11 Behaviour of the execution units over time for the example from Figure 13 25](#_Toc347658310)

[Figure 12 VP architecture 26](#_Toc347658311)

[Figure 13 Vector word layout 28](file:///C:\Users\diego\Dropbox\outofoderALU\doc\Theia%20architecture%20specification8.docx#_Toc347658312)

[Figure 14 Storing Fixed point numbers in a 96 bit word 28](#_Toc347658313)

[Figure 15 Instruction Layout 29](file:///C:\Users\diego\Dropbox\outofoderALU\doc\Theia%20architecture%20specification8.docx#_Toc347658314)

[Figure 16 Immediate bit and the way the instruction is interpreted by the IIU 30](file:///C:\Users\diego\Dropbox\outofoderALU\doc\Theia%20architecture%20specification8.docx#_Toc347658315)

[Figure 17 Modifying the individual signs of the instruction sources 33](#_Toc347658316)

[Figure 18 Modifying the scale of the instruction sources 33](#_Toc347658317)

[Figure 19 Swizzling instruction sources 34](#_Toc347658318)

[Figure 20 Combining several source modifiers in a single instruction 35](#_Toc347658319)

[Figure 21 Example of data dependencies when using source modifiers 36](#_Toc347658320)

[Figure 22 IIU issues a division 37](#_Toc347658321)

[Figure 23 The IIU issues an addition operation 38](#_Toc347658322)

[Figure 24 The DIV UE commits the results to the CBUS and the SMU. The SMU presents the first data dependency to the reservation stations. 39](#_Toc347658323)

[Figure 25 The SMU presents the second data dependency to the reservation stations. The ADD EU commits the result to the RF. 40](#_Toc347658324)

[Figure 26 An example code written in T-Language. 43](#_Toc347658325)

[Figure 27 The code from Figure 29 translated into assembly language 44](#_Toc347658326)

[Figure 28 Direct addressing mode 45](#_Toc347658327)

[Figure 29 Direct Addressing with displacement 46](#_Toc347658328)

[Figure 30 direct addressing with displacement 46](#_Toc347658329)

[Figure 31 Displacement and Index 47](#_Toc347658330)

[Figure 32 Indirect addressing mode 47](#_Toc347658331)

[Figure 33 Indirect addressing with displacement 48](#_Toc347658332)

[Figure 34 indirect addressing example 48](#_Toc347658333)

[Figure 35 T-Language code Snippet to calculate 1 Newton iteration of the inverse square root 56](#_Toc347658334)

[Figure 36 Operand swizzle logic 61](file:///C:\Users\diego\Dropbox\outofoderALU\doc\Theia%20architecture%20specification8.docx#_Toc347658335)

[Figure 37 VP data path Walk Through 66](#_Toc347658336)

[Figure 38 The decoded instruction presented by the IIU to the SMU 67](#_Toc347658337)

[Figure 39 The packet presented by the SMU to the reservation stations (RS). 67](#_Toc347658338)

[Figure 40 Block diagram of the IIU 69](#_Toc347658339)

[Figure 41 SMU simplified diagram 72](#_Toc347658340)

[Figure 42 multithreading 76](#_Toc347658341)

[Figure 43 A typical pixel color stored as a 32 bit value in VP’s the OMEM 79](#_Toc347658342)

[Figure 44 The OMI inside the IO unit 79](#_Toc347658343)

[Figure 45 EXE and OMI signals 80](#_Toc347658344)

[Figure 46 - VP writing data to an OMEM. 81](#_Toc347658345)

[Figure 47 - Cross bar bus example 82](file:///C:\Users\diego\Dropbox\outofoderALU\doc\Theia%20architecture%20specification8.docx#_Toc347658346)

[Figure 48 - CORE reading data from TMEM. 83](#_Toc347658347)

[Figure 49 Using the R0 register 86](#_Toc347658348)

[Figure 50 Using the R2 register 87](#_Toc347658349)

[Figure 51 Example of using the offset register R30 to allocate memory for automatic variables. 88](#_Toc347658350)

[Figure 52 General function registers and function arguments 89](#_Toc347658351)

[Figure 53 Example of an SPR shadowing R30 90](#_Toc347658352)

[Figure 54 Control processor CP 93](#_Toc347658353)

[Figure 55 CP “data block copy command” format 94](#_Toc347658354)

[Figure 56 CP control processor message. 95](#_Toc347658355)

[Figure 57 Mailboxing registers 96](#_Toc347658356)

[Figure 58 Mail box write data path 97](#_Toc347658357)

[Figure 59 Mail box write read path 97](#_Toc347658358)

[Figure 60 CP Instruction format 98](#_Toc347658359)

[Figure 61 SR0 special fields for copy block operations 98](#_Toc347658360)

[Figure 62 102](#_Toc347658361)

[Figure 63 CP block transfer high level syntax 104](#_Toc347658362)

[Figure 64 CP copy data block high level syntax 104](#_Toc347658363)

[Figure 65 104](#_Toc347658364)

Table of tables

[Table 1 Acronyms and Abbreviations 7](#_Toc416641335)

[Table 2 Reference Documents 7](#_Toc416641336)

[Table 3 Ray data structure fields 13](#_Toc416641337)

[Table 52 RGU Instruction set 15](#_Toc416641338)

Table 1 Acronyms and Abbreviations

|  |  |
| --- | --- |
| Acronym | Description |
| ALU | Arithmetic and logical unit |
| RGU | Ray generation Unit |
| RF | Register File |
| EA | Effective Address |
| RTL | Register transfer level |

Table 2 Reference Documents

|  |  |
| --- | --- |
| References | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

THIS PAGE IS INTENTIONALLY LEFT BLANK

# Introduction

TheiaV3 is the third iteration of the THEIA ray casting multicore GPU. The Theia project is an Open Source platform to play and experiment with programmable 3D graphic hardware.

The main goal of the THEIA project is to provide an open source environment including functional RTL, test bench environment and an open source high level programming language/compiler called T-Language.

The present document is dedicated to describe and specify the hardware architecture of the THEIA GPU system and related hardware subsystems.

The THEIA hardware is described using RTL (register transfer level), written in Verilog 2001 HDL. In order to perform a full RTL simulation, the HDL model needs a series of input files which represent the various input parameters and the binary representation of the user code (written in T-Language or in THEIA-Assembly language).

Scene geometry

THEIA Compiler

Binary code

User code



THEIA SIMULATION ENVIRONMENT

Simulation parameters

CAD software

Figure 1 THEIA environment overview

# System Overview

THEIA is a multicore, vector graphic processing unit (GPU). The THEIA GPU is comprised of different hardware blocks that interact with other in order to render an image frame.

Main Memory

Scheduler

**GPU**

Ray Generation Unit (RGU)

GT0

GT1

GT3

GT2

MMU

Figure 5 The GPU simplified block diagram

Figure 5 presents the GPU main functional blocks and also an external memory called “Main Memory” that is outside of the GPU. The Main memory is a large RAM that is used as a repository where the textures, code, geometry, etc. can be stored.

# Geometry Traversal Units (GT)

WIP

# Ray Generation Unit (RGU)

The RGU is a programmable engine in charge of generating RayDataStructures that are feed into the scheduler unit. The RGU has a very limited set of arithmetic operations and the instruction set is more focused towards generation of ray, thus lacking most control flow instructions.

{32’b1, RayDataStruct }

RGU

Scheduler

# The Ray Data structure

The output from the RPU is a ray data structure. The ray data structure is made of the following fields:

Pos.x

32 bits

Pos.y

32 bits

Pos.z

32 bits

1/Dir.x

32 bits

1/Dir.y

32 bits

1/Dir.z

32 bits

Table 3 Ray data structure fields

|  |  |
| --- | --- |
| Field | Description |
| Pos.x | The x Component of the Ray Origen |
| Pos.y | The y Component of the Ray Origen |
| Pos.z | The z Component of the Ray Origen |
| 1/Dir.x | Inverse of the X component of the normalized Ray Direction |
| 1/Dir.y | Inverse of the Y component of the normalized Ray Direction |
| 1/Dir.z | Inverse of the Z component of the normalized Ray Direction |

# RGU Instruction set

The RGU features a very simple instruction set targeted towards ray regeneration. Each instruction is RGU\_INSN\_SZ bit wide; the following figure illustrates a RGU instruction.

STOP

21 bits

20

RSVR

BKP

19

18

OP

17:15

DST

14:10

SRC\_A

9:5

SRC\_B

4:0

Figure 63 RGU Instruction format

|  |  |  |
| --- | --- | --- |
| Field | Size | Description |
| STOP | 1 | Stop Bit. Upon reaching instruction with this bit set. The RGU shall stop execution and send the oOutput StackCommit signal to the scheduler. |
| BKP | 1 | Breakpoint bit. The RGU shall halt execution upon encountering this bit. |
| RSVR | 1 | Reserverd for future use |
| OP | 3 | Operation field: Specifies ALU operation to execute |
| RESERVED | 4 |  |
| DST | 5 | Destination field: Specifies destination register of current operation. |
| SRC\_A | 5 | Register source A of current operation |
| SRC\_B | 5 | Register source B of current operation |

Figure 64 SR0 special fields for copy block operations

Table 52 RGU Instruction set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP | Value [[1]](#footnote-1) | DST | SRC\_A | SCR\_B | Description |
| NOP | 0 | n/a | n/a | n/a | No operation |
| MUL | 1 | Destination of operation | First operand | Second operand | Signed Fixed point multiplication.  RF[ DST ] = SRC1 \* SCR0 |
| ADD | 2 | Destination of operation | First operand | Second operand | Addition.  RF[ DST ] = SRC1 + SCR0 |
| SUB | 3 | Destination of operation | First operand | Second operand | Subtraction (2 complement).  RF[ DST ] = SRC1 – SCR0 |
| RESERVED | 4 |  |  |  |  |
| INV\_SQRT | 5 | Destination of operation | operand | n/a | Inverse Square Root  RF[ DST ] = 1 / SQRT( SRC1 ) |
| PUSH | 6 | n/a | operand | n/a | Push value to the output stack  PUSH[ SRC1 ] |

# RGU General Purpose registers

The RGU has 32x32 bit general purpose registers which can be used as source or destinations of the operations from the previous section.

General purpose commands registers are the register from R[0] to R[31]

# RGU Special purpose registers (SPRs)

TBD

# Works Cited

|  |  |
| --- | --- |
| [1] | D. P. John Hennessy, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 5 edition (September 30, 2011). |

1. In decimal [↑](#footnote-ref-1)