Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

Re-Decode: A General-Purpose CPU architecture model for extensive FU usage

---

TESIS que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **JORGE ALBERTO PADILLA GUTIÉRREZ**

Director **CARLOS HUMBERTO VÁZQUEZ TELLO**

Tlaquepaque, Jalisco. 19 de septiembre de 2025.

The idea, development, and completion of this work would not be possible without the constant help of my parents, whose love and affection were present every day without hesitation, and also, whose support has always been present throughout my career. This thesis is a reflection of everything my parents have taught me and their efforts in guiding me to accomplish my dreams. Thanks for being my main inspiration and for believing in me, even when I doubt myself. This work is theirs as much as mine.

I would also like to include the great direction and help from Carlos Vazquez, the advisor of this work, for his constant support and patience through this process. Whose impact in academia and the industry makes a high impact, and whose guidance was more than enough for this project to be completed. His wisdom and involvement with my academic development were fundamental to achieving this objective. Thanks for taking the time to bring me your wise tips and for guiding me on every aspect of my academic career. Without his trust and orientation, this achievement would not be possible.

Gratitude is in debt to all my fellow coworkers and classmates in my master's, as well as professors and directors of the Instituto Tecnológico y de Estudios Superiores de Occidente, who brought the tools and opportunities required for my academic development. This program not only inspired me, but it also inspired many engineers in the country to be involved in the Pre-Silicon and CPU design and architecture world.

And lastly, I would like to thank my partner and love Itzel Vera, for being my constant support during this trip; thanks for your patience, your understanding, and for being by my side on every step of the way, even in the toughest moments. This work is not only the culmination of my efforts, but also of your unconditional support and the strength you give me to keep moving forward and never give up. I dedicate every single one of my achievements to you, because without you, nothing would have been possible.

Abstract

In the nature of current CPU architectures, the full utilization of the Functional Units (FUs) is not always accomplished, generating a negative impact on Instructions per Cycle (IPC). In order to get the best performance out of the FUs, the Re-Decode algorithm is implemented as an architectural model in the Gem5 architectural simulator; this model redirects the instructions in queue into similar instructions, mainly scalar or vector instructions, that will take advantage of unused FUs slots without changing the final result of the program. This architectural model takes advantage of the instructions affinity, defined as the capability of some instructions to obtain the same result value; the instructions with high affinity are pairs of General-Purpose operations and their respective Scalar operation for vector registers, as well as the Scalar operations with their respective Vector operations, meaning that regardless of being a General-Purpose Register (GPR) or a vector register with a Scalar value, the result of their execution will always be the same for an independent set of input values. This aims to distribute the instructions more efficiently between every FU, concluding with a higher IPC. The Re-Decode architectural model was able to increase up to 9% the IPC on a threaded workflow and reduce the number of instructions waiting for an FU by a factor of 52 times by being Re-Decoded.

Index

Introduction

Nowadays General-Purpose CPUs are made of a collection of complex blocks of combinational and sequential logic which are individually set to accomplish small tasks with a specific arrangement of data, this design approach brings the possibility to encapsulate the instructions that the CPU decodes and executes in different Pipeline stages and furthermore process them independently in each stage, where also multiple other instructions are being executed in parallel, this is part of the called Instruction Level Parallelism (ILP) and It is fundamental in CPU architecture since current CPUs exploit this capability [1]. Another way to increase the number of instructions that are executed in parallel on the CPU comes from the introduction of multiple Functional Blocks (FBs) in each stage of the Pipeline, enabling multiple instructions to be on the same stage while other instructions are on different stages.

In order to fully optimize the usage of the FBs in charge of executing instructions, a new core hardware architectural model is proposed, with the objective of maximizing the utilization of these FBs and reducing the number of idle cycles that impact the Instructions Per Cycle (IPC) of the processor. While there are multiple strategies on the state-of-the-art of CPU Design that focus on IPC increments, this brand-new architectural model is defined to take advantage of these strategies and formulates an extension of the most efficient architecture features. This new architectural model not only extends the FBs present on these strategies but also implements new logic that maximizes the number of executed instructions and thus provides an IPC increment.

On state-of-the-art CPU Design, the main division of FBs is composed of two clusters: the first one consist of the logic in charge of getting the next instruction pointer with branch prediction, fetching instructions and decoding them, which form a cluster called Front End; the second one is the logic in charge of executing these instructions and bring them to the registers or back to memory, this cluster is called Back End.

The resources of both Front End and Back End are bounded by timing and area limits, causing them to be in a sweet spot that manages good performance, timing and area; but uncore factors such as memory frequency generate multiple tradeoffs that limit this sweet spot. It has been

proven in CPU architectures that in order to tolerate the gap between processor and memory speeds, the size of the critical resources must be increased dramatically to impractical sizes [2].

While Front End improvement efforts include tuning on predictions and latency, as well as hierarchy of decoding, it is nontrivial to keep the Back End Functional Units (FUs) at 100% utilization; the Back End itself contains some of the FBs that limit the utilization of the FUs, one of such being the Reservation Stations, also known as Instruction Queues (IQs). The IQs are a set of queues where instructions are allocated prior to execution, and these instructions may be either waiting for the values to be ready in the registers used or waiting for an FU to be available to execute the instruction.

On the IQs, the impact of lower IPC can be seen by having them either empty or full. While the IQ is empty, the instruction flow of the program is set to use other FUs different from the one this IQs uses, thus the FU is unused; while the IQ is full, the queue no longer accepts more instructions to be fed in until the FU has space to execute, meaning that the Front End may stall instructions, and thus, these instructions last longer to execute. It has been proven that VLIW architectures with 8 ALUs, where all 8 FUs are executing at the same time, have a negligible program execution time, and most FUs work in parallel on the benchmarks, where only 5 run at 0.74% of the time in parallel [3].

The FUs also have a relevant role in the search for their full usage. On current architectures, CPUs not only work with integer values, but also with floating point values, the registers are not only independent for single values, since there are also Vector registers and Vector FUs, but while this technology is not new, these are not always fully used during the execution time. Some studies found that Vector FUs are used approximately 30% on the execution time of common programs, and less than 40% of their usage includes the whole vector size [4]. This has led to the creation of multiple extensions to the most used Instruction Set Architectures (ISAs) that include lots of functionalities to these FUs.

On the back end of the processor, where the FUs lie, the most known strategy for IPC improvement is the Tomasulo Algorithm, this is the base component of the Out-of-Order (OOO) architecture where the instructions are not necessarily executed in program order and instead the instructions are executed when their data is available, also the order will not modify the expected result of the program; the Tomasulo Algorithm is in charge of re-naming the registers of the

instructions and setting them ready when their data is available, and it keeps their real order stored, at the end of execution the data is updated but is only committed back on program order [5].

With all things considered, the Re-Decode architectural model is proposed. The Re-Decode architectural model consists of the addition of a second decoding stage during instruction allocation in the IQs, where it is reviewed if the instruction has similarities with another instruction. These similarities are considered valid if and only if the result of the execution is the same in both instructions, then the instruction is temporally changed to this other instruction moving to a different FU to be executed, and after its execution it is brought back to the original instruction form to be correctly retired by the OOO logic.

The IPC improvement of the Re-Decode Architectural Model comes from the utilization of Vector FUs when the workload is mainly on General Purpose Registers (GPRs); since Vector FUs are generally less used and GPR FUs are mainly full [4], some instructions will then be moved to the Vector FUs, thus, more instructions will be executed at the same time.

The Re-Decode Architectural Model was implemented in Gem5, an Open-Source architectural simulator for investigation in the field of CPU Architecture and Microarchitecture. It is one of the most flexible and widely used simulators in academia and research simulators [6]. In [7], Akram and Sawalha named this simulator as the best based on flexibility, configurability and precision; it is highly precise at calculating the IPC, and it is the most flexible due to the capabilities it brings to the architect.

Gem5 is based on both C++ and Python, where Python defines the top-level classes and the simulation scripts, while C++ defines the pure internal behavior of the classes. It also uses the Linux library SCons to execute. The simulation scripts instantiate every component required in the simulation, which includes the System, Clock Domain, Memory, CPU, Bus, Cache, Controllers and Ports.

The availability of multiple CPUs in Gem5 is a key part on its flexibility, since it can simulate Alpha, ARM, SPARC, MIPS, POWER, RISC-V and x86 ISAs both In-Order and Out-of-Order; it also has a direct execution simulation mode where it does not simulate pipelines and instead simulates only executions in 1 cycle [6]. Gem5 also counts with a simple results log, where every statistic is shown for ease of usage; it includes every defined statistic in the source code and every self-implemented statistic.

In this document, first the objectives and research goals are presented, followed by the Theoretical Framework, where the state-of-the-art regarding the topic that involves the Re-Decode architectural model is defined, including FUs, IQs, Instruction and Data types. Followed by the method, where every new concept, algorithm and methodology required for the Re-Decode architectural model is proposed, as well as the modifications for Gem5. After that, the experiments are included, where the benchmarks used and the metrics evaluated are shown. The final chapters include the results, where the benchmarks selected are evaluated with our metrics, future work where the things that were not included but can be implemented as improvements are redacted, and lastly our conclusions.

1. Objectives

The main objective of this Thesis is to propose the Re-Decode Architectural Model and show how its implementation would cause an increment in the IPC of a General-Purpose CPU architecture. The Re-Decode Architectural Model modifies the program instructions into similar instructions that can be executed on a different FU without changing the result of its execution.

To show these capabilities, the Re-Decode Architectural Model must be defined, modeled, implemented and tested. This document presents and expands on each of these model elaboration steps.

The specific objectives of the definition, implementation and result measurements of the Re-Decode Architectural Model are as follows:

- Identify the scenarios in which the instructions can be manipulated without changing their execution result, and while doing so, getting executed on a different FU.
- Formulate an Algorithm that defines how instructions with the found similarities can be changed, and therefore, are changed to be executed on a different FU.
- Elaborate the proposed algorithm on Gem5, the selected architecture simulator.
- Run simulations of different flows and benchmarks in both the proposed model and a reference model with no changes.
- Compare metrics obtained in the simulations, and show the IPC increment of the proposed model.

2. Theoretical Framework

There are some elements on the state-of-the-art of CPU architecture that form a key part on the concept of Re-Decode:

1. Instruction types
2. FU types
3. Instruction Queues
4. Data Type
5. Registers

First, Re-Decode takes advantage of multiple instruction types and FU types, here the instruction types are segmented by their data type, while the FU type is segmented by the registers in which it works. While the FUs are only present on the execution stage of the instruction, other functional blocks, like the Instruction Queues, are relevant to the construction of the Re-Decode Algorithm.

Also, the implementation of the Re-Decode Architectural Model takes advantage of the flexibility of Gem5; this architecture simulator has a small ramp-up process, and it has a user-friendly metrics system. It also contains a clear C++ environment where the development of the features can be done.

Another important feature of Gem5 is the variety of ISAs it can simulate. From multiple RISC architectures such as RISC-V and ARM, to also CISC like x86, the Re-Decode Architectural Model was implemented for x86, which is the ISA present on almost every Desktop and Laptop PC in the consumer market, it was developed by Intel first in 1978 and it is still being updated to this day.

2.1. Characteristics of the Functional Blocks

State-of-the-art CPU architectures are mainly Out-of-Order, *i.e.*, the execution of the instructions is not necessary in the same order as the program. This architectural feature exists due to the independence of some of the instructions, as well as the time it takes to execute them.

2. THEORETICAL FRAMEWORK

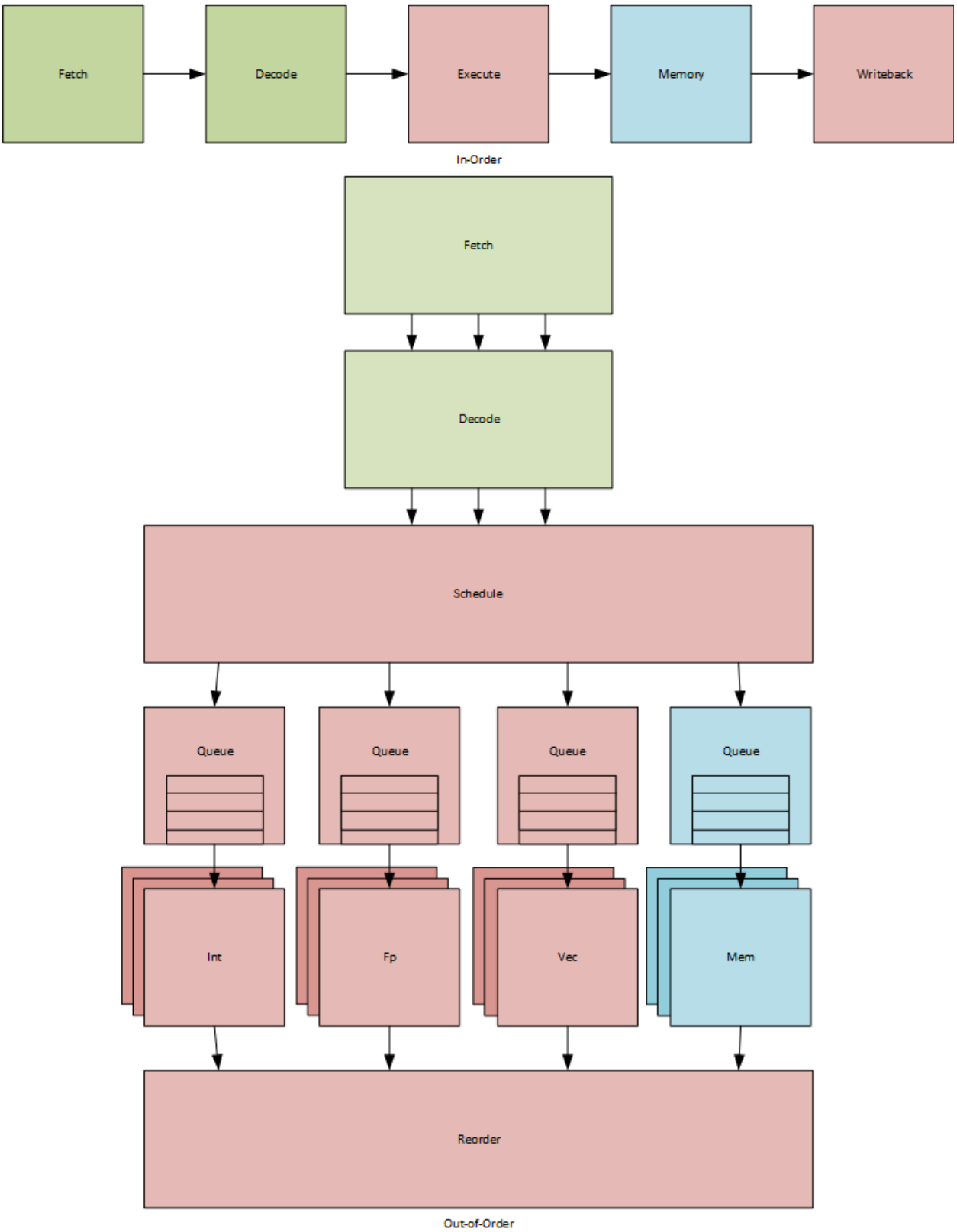

Fig. 2-1 A diagram representation of both In-Order and out-of-Order pipelines with color distinction of Front End (green) Back End (red) and memory (blue)

The Out-of-Order concept is also well known in academia, and in the book "Computer Architecture: A Quantitative Approach" it is used broadly [5]. *Fig. 2-1* shows the typical diagram representation of both In-Order and Out-of-Order architectures, this representation includes some of the main pipeline stages present in both architectures, since this may vary depending on the architecture and microarchitecture of the processor.

On these architectures multiple FBs are relevant in order to keep the correct track and flow of the instruction. Regularly the FUs are divided into different types of instructions that can be executed, and into the different latencies these instructions take to be completed. While multiple FUs are instantiated, many instructions execute at the same time.

As Dolejsi Santos *Et al.* tested in "Functional units utilization in a multiple-instruction issue architecture," [3] using a VLIW architecture, the ALUs implemented can be either instances of the same ALU that can execute the same types of instructions, completely different ALUs which each can execute independent types of instructions, or a mixture of both. Current commercial CPU architectures use both prioritizing the ALUs on which more instructions are executed.

Considering the P-Core of the Intel Core Ultra series 200, the core architecture clusters ALUs in multiple groups, and the instructions can access these groups via ports; these groups are categorized by the instruction type it will execute such as Integer, Vector and Memory access. The most common ALUs are instantiated multiple times, one for each port that can access to them, by having 6 ports that can access to a basic Integer ALU there are then 6 copies of that ALU, one for each port, while other instructions like Floating point division are only present in 2 ports of the Vector side, so there are only 2 instances of that FU [8].

Another relevant blocks present in Out-of-Order architectures are the Instruction Queues (IQs), where the instructions are stored while not being executed. Once an instruction gets to an IQ it first waits for its operands to be ready, *i.e.,* waits for their data dependencies to be completed. Then, it waits for an FU of its data type to be empty, this is done so it can now execute.

The IQs are a fundamental element of the Tomasulo algorithm, which is the base of the Out-Of-Order architecture, since it defines the flow of the instructions at the Back End in a way that they only progress in terms of their dependencies regardless of the order of the program. The IQs are the reservation stations of each ALU where these instructions are waiting for their dependencies; the instructions may only proceed to execute once no more data dependencies are pending, for simplicity, reservation stations are referred to as IQs in the remainder of this text.

While all of these happen to the instructions, a reorder logic keeps the original order of execution and retires the instructions if and only if the previous instructions are also retired. This reorder logic also helps to retrieve the state of the program if an event like an exception or an interrupt occurs.

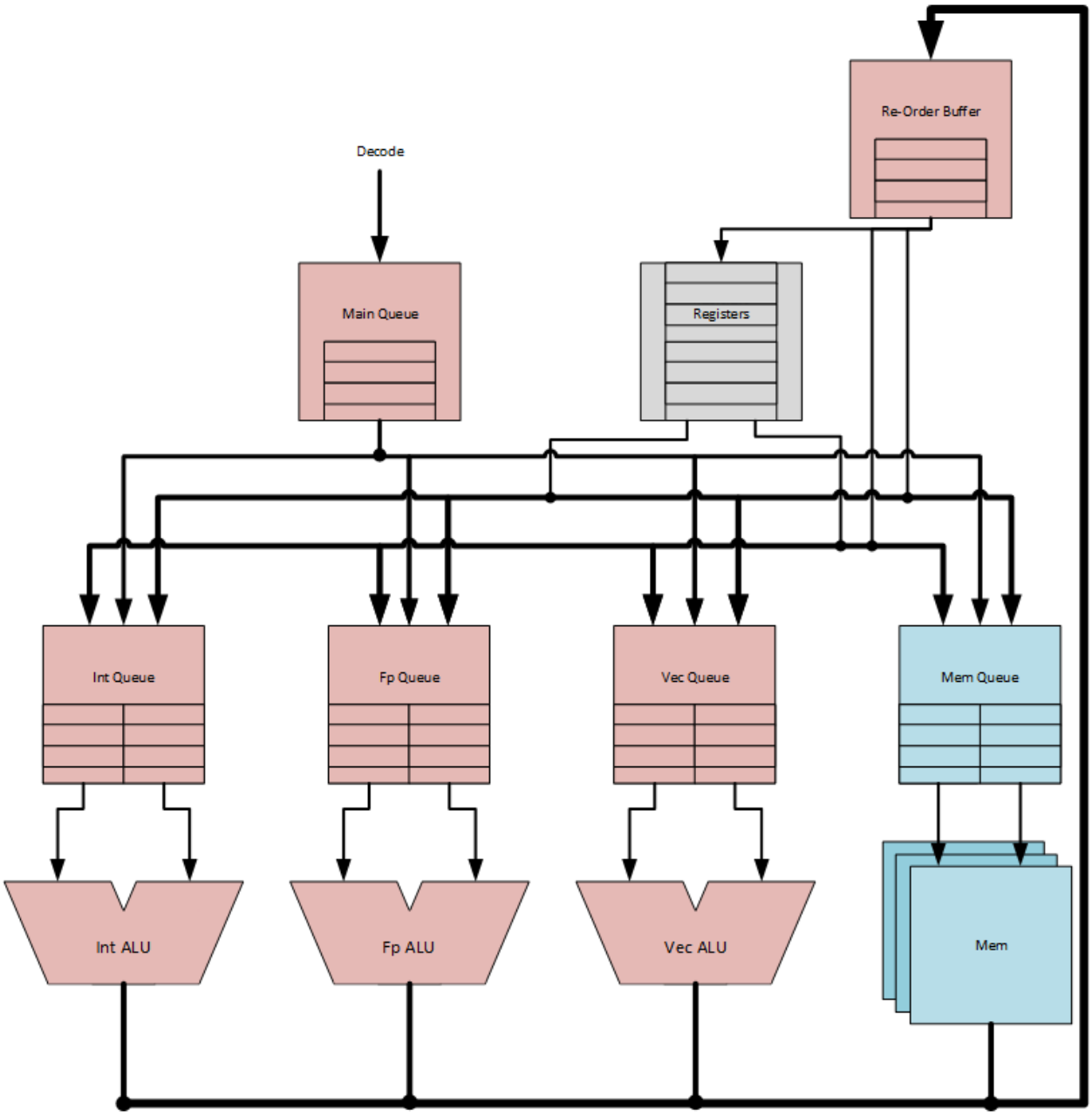

Fig. 2-2 A diagram of the Tomasulo algorithm present in Out-of-Order architectures.

The IQ's size is relevant to prevent stalling, and avoiding cycles being lost while the CPU waits for some event to happen. IQs will stall the CPU when full; this could happen in 2 different ways:

1. If there are no FUs ready to execute the instructions on the IQ
2. If every instruction on the IQ is waiting for its dependencies

A possible solution could be to make sure that the FUs are retiring enough instructions for the queues to be both filling dependencies and executing them.

Fig. 2-2 shows a representation of the Tomasulo algorithm, which is the base model used in Out-of-Order architectures. Here, the IQs are presented on the Tomasulo algorithm; there is an IQ for each ALU, which are the Int Queue, the Fp Queue, Vec Queue and Mem Queue, and they keep the instructions waiting for their dependencies.

Referring to the FUs, there are FUs capable of executing multiple data at the same time called Vector Units. These Vector Units address the SIMD concept by executing individual instructions on a defined set of data. The amount of data executed depends on both the operand format and the size of the vector register itself.

In recent years, the work on Vector Units has been focused on Vector Length Agnostic Architectures, where the concept of Agnostic Length refers to the capability of the hardware to interact with any defined vector length on the software size. This is done by changing the size of the vector from the specified on software to the one done on hardware while moving the data to one or many registers that fit the size of the physical registers [9].

Both Pohl *Et al.* in "A Performance Analysis of Vector Length Agnostic Code," [9] and Kaddar et al in "Accelerating The Vvc Decoder For Vector Length Agnostic Simd Architectures," [10] show how this technology can be an improvement over the current tendency of increasing the vector size for both software and hardware, since this tends to cause code portability issues.

The trend of extending the instruction set for more vector instructions also shows the focus of current architectures on giving more utilization to the Vector FUs. Currently, the x86 ISA is expected to be expanded on the Vector FU utilization side by adding more instructions and compatibility of bigger vectors with the Advanced Vector Extensions 10 (AVX10), published by Intel in the white paper "Intel® Advanced Vector Extensions 10 (Intel® AVX10) Architecture Specification" [11] where the Vector functionalities of x86 are expected to be incremented on every Client and Server processor.

2.2. Hardware Simulation Models with Gem5

The selected architecture simulator for the Re-Decode Architectural Model implementation is Gem5. This simulator requires two main components in order to run a simulation: the Event-Driven Source Classes and the Simulation Scripts. Once a simulation is run, a set of outputs is returned, including the list file of the Objects present during the simulation, and the statistics log, which compiles every base and user-defined statistic measurement inside the code of the Source Classes.

Gem5 also counts with a developer guide, which teaches how to use the simulator from the Linux requirements to the most advanced examples of both simulation scripts and Classes; the elaboration of an architectural model requires both Script creation and Class changes to accurately simulate the proposed architectural model.

```
import argparse
import m5
from m5.objects import *
from caches import *
'''
Simulation Parser for parameters
Using the argparse python library
'''
parser = argparse.ArgumentParser(description='Simple CPU OOO with 2-level Cache.')
parser.add_argument("binary", default="", nargs="?", type=str, help="Path to the binary to execute")
parser.add_argument("--l1i_size", help=f"L1 Instruction Cache size. Default: 16kB")
parser.add_argument("--l1d_size", help=f"L1 Data Cache size. Default: 64kB")
parser.add_argument("--l2_size", help=f"L2 Cache size. Default: 256kB")
options = parser.parse_args()
'''
System SimObject
Contains functional (not timing-level) information, such as
    memory ranges, root clock domain, root voltage domain
'''
system = System()

'''
Clock Domain definitions
SrcClockDomain() makes the clk_domain variable a Clock Domain defined by Gem5
clock will keep the frequency
voltage_domain will keep a voltage, the default is obtained by VoltageDomain()
'''
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()
'''
Memory Definition
mem_mode defines the "simulation mode" of the memory
    timing is the most used
mem_ranges defines the size, we use AddrRange() for the creation of a whole range with the defined value
'''
system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]

'''
CPU Definition
Basic example, X86TimingSimpleCPU
    timing-based, each instruction in a single clock cycle (except mem)
```

```
CPU Naming Format
    {ISA}{Type}CPU
'''
system.cpu = X86O3CPU()

'''
Memory Bus Definition
System-Wide memory bus with SystemXBar()
'''
system.membus = SystemXBar()

'''
Cache Definition
ICache and DCache are defined individualy
    If no Cache, connect to membus
Two types of Cache
    Clasic
        Regular Simple Cache
    Ruby
        Designed for Coherence
Cache Steps
    Instanciate the Caches
    Conect them to the ports
Hierarchy
    CPU to L1s
    L1s to L2 bus
    L2 bus to L2
    L2 to membus
'''
#L1s
system.cpu.icache = L1ICache(options.l1i_size)
system.cpu.dcache = L1DCache(options.l1d_size)

system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)

#L2 bus
system.l2bus = L2XBar()

system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)

#L2
system.l2cache = L2Cache(options.l2_size)
system.l2cache.connectCPUSideBus(system.l2bus)
system.l2cache.connectMemSideBus(system.membus)

#Use this instead if no caches are used, which conect directly to the membus

'''
Ports Definition
Each mem object can have two kinds of ports
Request Ports
    From Request port to Response port
Response Ports
    From Response port to Request port
Must be conected that way
    cpu.request_port = cache.response_port
We can assign an array of ports to a port as well

Required Ports
    I/O Controller conected to the memory bus
    Special Functional Only port conected to the memory bus (mem reads and writes)
x86 Ports
    PIO conected to the memory bus
    Interrupt port conected to the memory bus
        requestor conected to cpu side
```

```
        responder conected to mem side
The Interrupt Controller creation is a function of the cpu class, no need to assign to an atribute
'''
system.cpu.createInterruptController()
system.cpu.interrupts[0].pio = system.membus.mem_side_ports
system.cpu.interrupts[0].int_requestor = system.membus.cpu_side_ports
system.cpu.interrupts[0].int_responder = system.membus.mem_side_ports

system.system_port = system.membus.cpu_side_ports

'''
Memory Controller Definition
A mem controller should be conected to the membus on the mem side
We must define a dram type and its range
'''
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8()
system.mem_ctrl.dram.range = system.mem_ranges[0]
system.mem_ctrl.port = system.membus.mem_side_ports

'''
Process to Execute
Syscall Emulation emulates only CPU and Mem, it only needs the CPU to point to a compiled executable
Fullsystem Emulation emulates the entire hardware system and runs a kernel (Like a Virtual Machine)

Process is a SimObject
    The Processes command is set to the command tu run
    Like an argv
        First position is the executable
        Next positions are the arguments
    Set the CPU to use the process and its workload
    Create the functional execution context
'''
binary = 'tests/test-progs/hello/bin/x86/linux/hello'
system.workload = SEWorkload.init_compatible(binary)
process = Process()
process.cmd = [binary]
system.cpu.workload = process
system.cpu.createThreads()

'''
System Instance
Needs the creation of the Root object
The instantiaton process creates a C++ equivalent of every SimObject in Python
'''
root = Root(full_system = False, system = system)
m5.instantiate()

'''
Sinulation Execution
'''
print('Start of Simulation!')
exit_event = m5.simulate()
print('End of Simulation!')
print(f'Exit Status: {exit_event.getCause()} @ {m5.curTick()} ticks')
```

Code 2-1 Example of a Simulation Script for gem5 where every important Class is instantiated.

2.2.1 Simulation Scripts

The simulation script is written in Python. It is in charge of instantiating the classes that take part in the simulated processor; it uses the tools of M5 and the Classes defined in the source code of the simulator.

An example of a Simulation script is shown on Code 2-1, where some of the classes required for a simulation to be executed correctly can be seen. At first, the System SimObject is present, which contains functional configurations like voltage domains and clock domains, followed by the instance of these configurations and their respective variables; the next instances include the CPU, which is the main class for this implementation and the one that also gem5 provides with multiple different flavors. A deep explanation of the script is shown in Appendix A.

After the definition of the CPU, other components such as memory buses, caches, controllers and ports are specified, lastly the binary of the code that will be simulated is included. All these elements are instantiated and executed in the same Python file.

2.2.2 SimObjects

The Source Classes of Gem5, referred internally as SimObjects, are every defined class in Gem5 that will take part in the simulation. These are defined at High Level on Python and specified at functional Low Level on C++. These SimObjects can be modified to accomplish the architecture proposal, as well as integrate new SimObjects into the simulation, if so, these should match the inputs and outputs of the SimObjects which they interact with.

Gem5 works as an Event-Driven simulation instead of being determined by “only clock” cycles, which means that the SimObjects behave in terms of the events executed on certain data such as the instruction and its dependencies, these events could be defined by any number of expected cycles, as well as single cycle, which forms part of the flexibility of the simulator [6].

2.2.3 Statistics

One last crucial element of Gem5 that brings flexibility and ease of use for this implementation is its statistics. The SimObject files usually include a Statistics class connected to it, in which variables are defined that keep track of events, values, counts and other features that can be measured. While gem5 includes a large number of statistical variables by default, it is flexible for the implementation of more Statistics classes that will also be included along with every default one in the Statistics Dump at the end of the simulation.

The dump file shows every measurement done on the simulation, which goes from the basic IPC and time of execution to more specific ones like the number of instructions, or the number of times an FU was busy and tried to be accessed by an instruction, among many others. Code 2-2 shows an example of a statistics dump file with only the first metrics shown, since these files are longer; these first metrics are the most general, such as time, IPC, instructions and real time. Is in this file where the metrics will be obtained and evaluated.

Gem5's flexibility also allows that every user-defined statistic variable will be included in this dump file; since the Statistic classes are defined in conjunction with its SimObject, a hierarchy is shown where the metrics of each SimObject are grouped, this makes every default and user-defined metric present along with other statistics of the same SimObject.

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.000001                       # Number of seconds simulated (Second)
simTicks                                       954163                       # Number of ticks simulated (Tick)
finalTick                                      954163                       # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                     0.02                       # Real time elapsed on the host (Second)
hostTickRate                                48211965                       # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                     655756                       # Number of bytes of host memory used (Byte)
simInsts                                           68                       # Number of instructions simulated (Count)
simOps                                            118                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                     3413                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                       5920                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu.numCycles                              955                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                              14.044118                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                               0.071204                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                      0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                    0                       # Number of work items this cpu completed (Count)
system.cpu.instsAdded                             264                       # Number of instructions added to the IQ (excludes non-spec)
(Count)
system.cpu.instsIssued                            249                       # Number of instructions issued (Count)
system.cpu.squashedInstsExamined                  121                       # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined                85                       # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.numIssuedDist::samples                 763                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean               0.326343                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev              1.174835                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows                0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                       689     90.30%     90.30% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                        20      2.62%     92.92% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                        11      1.44%     94.36% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                        10      1.31%     95.67% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                         7      0.92%     96.59% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                        11      1.44%     98.03% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                        11      1.44%     99.48% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                         4      0.52%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                         0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                 0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                 0                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                 7                       # Number of insts issued each cycle (Count)
```

Code 2-2 Example of a statistics dump file from a simulation on gem5 with only the first metrics.

2.3. Instruction Set Architectures

The Instruction Set Architecture (ISA) of a Processor specifies the way the software and the hardware communicate; every processor has an ISA specified, and the software must speak in that ISA to run on the hardware. There are 2 main distinctions between the types of ISAs defined by the "complexity" of the instructions that the processor can execute, in conjunction with the way these instructions are stored in memory; at first, the Complex Instruction Set Computers (CISC) have the most complex instructions, and most CISC architectures will have a variable amount of bytes in memory, as an example, a CISC architecture may be able to have a single 2 bytes instruction that starts a virtualization process as well as a 5 bytes instruction that does a transformation in the Galois-Field on a Vector; in the other hand, a Reduced Instruction Set Computer (RISC) will only contain simple instructions with a fixed length, for example, it could only do additions, multiplications, jumps and memory accesses, and everything is always defined in a fixed amount of bytes.

While RISC architectures such as ARM and RISC-V are widely used, the x86 CISC architecture, which is present in almost every desktop and laptop PC, will be the one in which the Re-Decode Architectural Model is implemented. Due to the considerable number of instructions that use Vector FUs, this ISA brings a complete and easier perspective on how this architectural model can be implemented. This implementation is not forced to be x86 only, since the ideas presented can be applied to other ISAs.

The x86 ISA includes a large number of instructions in which the Re-Decode Architectural Model can be formulated, this comes from the way it manages data formats. x86 has different instructions for different data formats, and it does not require a continuous set of configuration operations that modify these on the fly. As an example, x86 can perform additions of 64-bit values followed by subtractions of 32-bit values without modifying the configuration registers.

Another characteristic of x86 is the microcode. This concept refers to the internal conversion of regular x86 instructions into RISC-like microinstructions, which are the operations that the FUs can execute. Microinstructions bring the capability of forming long flows of reduced operations into a single complex instruction which is stored in less memory.

2.4. Types of Instructions

As mentioned widely in this document, state-of-the-art processors will include Vector FUs capable of doing operations on vectors of multiple values. This brings the idea of segmenting the Instructions into different types according to the data or registers operated. For the Re-Decode Architectural Model, 4 different types of instructions are defined: Integer instructions, Floating-Point Instructions, Scalar Instructions, Vector Instructions and Non-ALU Instructions.

2.4.1 Integer Instructions

The simplest of the set are the Integer Instructions. These, as the name implies, will execute Arithmetic and Logic operations on Integer Values (Ints), which are stored on the commonly known General-Purpose Registers (GPRs). These are well-known instructions and the ones that are present in every ISA's main implementation.

Regardless of being a CISC or RISC architecture, the Integer Instructions will almost every time be RISC-like, this means that there is most likely no microcode conversion required in a CISC architecture, and these instructions are easy to understand and map on every ISA; some RISC architectures will only include these types of instructions as well as non-ALU instructions in their basic implementation. The formats in which Ints can be defined are length defined, which include 8-bits (byte), 16-bits (word), 32-bits (doubleword), and 64-bits (quadword), and could be either signed with 2's complement or unsigned.

2.4.2 Floating-Point Instructions

These instructions will also execute Arithmetic instructions in a set of values called Floating-Point Values (FPs). An FP refers to a real number, which includes natural numbers, rational numbers and irrational numbers. While the instructions are almost the same for floating-point and integer sets, the algorithms for solving these operations are more complex.

The formats for FP values are also defined by their length, some of them being used widely such as 64bits (double precision), 32-bits (single precision) 16-bits (half precision) and newer formats used mainly on GPUs such as 8-bits (fp8), 4-bits (fp4) and 128-bits (quadruple precision).

The format in which these instructions are represented changes between specific sizes, but 3 fields are shared among them:

- Sign, which indicates whether it is positive or negative.
- Exponent, which indicates the exponent part.
- Mantissa, which indicated the decimal part of the number.

Each field has a variable length depending on the format, except for the sign field which is always one bit.

2.4.3 Vector Instructions

This set of instructions includes both the Ints and the FPs formats with the addition of executing the same instruction on multiple different values. This means it executes the same Arithmetic or Logic instruction to a set of multiple Ints or multiple FPs at the same time. The reason this set includes instructions that operate on Ints and FPs is that both are stored on the same vector registers, and the instructions will determine the format of the values on those registers.

A Vector Register is bigger than an Int or an FP register since it can store multiple of these at the same time, regular vector sizes include 128-bits, 256-bits and 512-bits; these sizes are always powers of 2 and the format of theirs values will be the same as the ones on regular Int and FP instructions, so the amount of values on a vector are determined by the size of the vector and the size of the operands, for example, by having a 128-bits vector and executing an add instruction with single precision values, 4 different operands fit in the vector, and thus, executes a single instruction with 4 values at the same time, this instruction would be VADDPS on x86.

2.4.4 Scalar Instructions

The Scalar instructions are a subset of Vector Instructions which are defined separately due to their relevance for the Re-Decode Architectural Model, these are vector instructions that will only execute on 1 singular operand, that is, regardless of the size of the vector register it will only execute on the lower part of it where the singular operand is present.

There are some scenarios in which a scalar operation is required, as the name implies physics and mathematics, a scalar can be multiplied by a matrix or vector. An example of an x86 scalar instruction is VADDSD, which is a vector addition scalar in double precision values; x86 has flexibility with scalar instructions, where the 'S' in the 2nd to last letter indicates it is "scalar".

The definition of scalar instructions is fundamental for Re-Decode, since their difference against Integer and Floating-Point instructions is the register in which they execute and thus also the FU in which it executes, the Scalar Instructions execute on single values placed at Vector Registers, so they also execute in Vector FUs.

2.4.5 Non-ALU Instructions

Lastly, a set of instructions called Non-ALU Instructions is defined, these include flow control and processor configurations; this mainly includes GPRs and can be executed on the Int FUs, but for the Re-Decode Architectural Model, Integer instructions are consider mainly the ALU operations that are based on the execution of an arithmetical or logical operation with input values that results on an output value. Since the Re-Decode Architectural Model works on Arithmetic and Logic Instructions. This set of instructions will be categorized aside.

3. Method

The Re-Decode Architectural Model is defined as an algorithm to be implemented in the simulator. To achieve it, a Re-Decode hierarchy and an Affinity rule were proposed. While the hierarchy defines how the instructions will be Re-Decoded, the Affinity rule identifies if the instruction can or cannot be Re-Decoded. At the end, the Affinity concept is used between instructions of different categories, where such categories are the different instruction types (defined in Section 2.4) ordered hierarchically in such a way that the Re-Decode may have a bigger impact on the IPC.

3.1. Instruction Categories and Re-Decode Instances

The defined categories for Instruction types bring the possibility of identifying the instances of Re-Decode occurrences. Since the main benefit of the Re-Decode Architectural Model comes from increasing the usage of the FUs that are not being used, with this premise, Re-Decoding an instruction to a similar instruction that utilizes a different FU and expects the same output is the main focus of this model. With this information, it is concluded that the instances are the different transformations between the instruction types.

The possible changes between instruction types that function as Re-Decode instances are as follows:

- Integer Instructions to Scalar Instructions
- Scalar Instructions to Integer Instructions
- Floating-Point Instructions to Scalar Instructions
- Scalar Instructions to Floating-Point Instructions
- Scalar Instructions to Vector Instructions
- Vector Instructions to Scalar Instructions
- Vector Instructions to Bigger Vector Instructions
- Vector Instructions to Shorter Vector Instructions

3. METHOD

For simplicity purposes, the combination of Int instructions and FP instructions will be considered as General-Purpose Register (GPR) Instructions, while on some architectures like x86 the FP values are not present on the GPRs and instead have their own registers, the combination comes from the concept of having a singular register for a singular value. That brings us to the following instances:

- GPR Instructions to Scalar Instructions
- Scalar Instructions to GPR Instructions
- Scalar Instructions to Vector Instructions
- Vector Instructions to Scalar Instructions
- Vector Instructions to Bigger Vector Instructions
- Vector Instructions to Shorter Vector Instructions

It is important to include that transformations between GPR and Vector instructions can be achieved by doing an initial transformation to Scalar, this is done so that the transformations do not imply multiple changes individually and reduce the requirements of completing them.

3.1.1 GPR to Scalar

The first instance, expected to provide the higher IPC benefit, comes as the main focus of implementation; a GPR instruction can be converted in a Scalar instruction with the same precision, if the GPR FUs are full and the Scalar IQs are almost empty, re-decoding this will send the instruction to a different FU and thus more instructions will be executing in parallel.

Doing this kind of change also requires changing the register, since the FUs take different registers, so a pointer to the instruction should save the logical (GPR) register of the original execution plan and the new physical (vector) register, and prior the retirement of the instruction should redirect the value to a corresponding physical GPR and update the original logical register with the new value.

A basic example of this scenario is Re-Decoding an addition, by converting our integer addition instruction into a scalar addition, the only difference will be that it uses the bigger vector registers on a different FU, but the execution will result in the same value for the same pair of

inputs, extrapolating this behavior to other arithmetic instructions on GPRs the same result will be expected on Scalar registers.

3.1.2 Scalar to GPR

In contrast to the instance presented previously, the opposite move can be done in case of excessive usage of scalar instructions while the GPR FUs are empty. An example would be to transition from a scalar addition to a regular addition on GPRs, where the same concepts apply.

3.1.3 Scalar to Vector

The second fundamental case, is the instance where multiple scalar operations are on the queue and can be merged into a vector instruction. This instance of Re-Decode can be applied to instructions that have already been Re-Decoded; that is, even GPR instructions previously Re-Decoded to Scalar can also be now transformed to Vector if there are enough Scalars to merge into a Vector.

This scenario now depends on more than one instruction of the same type to be present on the queue; regardless of this Re-Decode instance merging multiple instructions instead of just converting one, the same logic that defined the possibility of converting it to a similar instruction is used independently of the number of instructions.

For example, two instances of scalar additions can be merged together into a vector addition, where the vector contains the two elements and each addition is done independently but at the same time in the same vector FU; this scenario will improve the IPC since at software level it is executing more instructions at the same time while at hardware level is merging instructions, while it's expected to have an impact on power and possible latency increase.

3.1.4 Vector to Vector

The final fundamental case, one that is proposed but its implementation is outside the scope of this project, is when the Vector Agnostic concepts are applied to now modify singular instructions into multiple similar instructions of shorter dimensions or merge multiple similar

instructions into one of bigger vector length. The Re-Decode architectural model gets benefits from vector agnostic architectures since the transformation of vectors to a length independent from the instruction itself gives the freedom to the compiler to optimize the code in the best theoretical way, and the hardware adapts it to its capabilities without changing the execution flow [9].

Increasing the size of the vector with multiple instructions is the second way in which Re-Decode can merge instructions, this is focused on the IPC since it reduces the time it takes to complete the program instructions; in the other hand, decreasing the size is only beneficial if the size of a vector agnostic architecture is small, or if there are objectives on reduction of power consumption, since these are not the focus of this project those are only mentioned as theoretical improvements for the Re-Decode Architectural Model.

Similar to the Scalar to Vector scenario, this scenario will also merge instructions. As an example, two instances of vector additions, with 2 elements each, can be merged into a bigger vector addition of now 4 elements if the implemented vector size allows it.

Since both Scalar to Vector and Vector to Vector Re-Decode instances depend on having the complete Re-Decode logic encapsulated before the retirement of the original instructions, *i.e.* the Vector instruction has to be executed and then brought back to Scalar or to the original sized Vector before retirement, and each instruction retires as their original program order. If any element of the new vector should not be retired (for example, comes from a branch mispredict) it will be executed on the Re-Decoded Vector, it will then be restored to the original instructions, and the retirement flow will follow the program order, so any instruction prior to the mispredict will correctly retire, and any instruction after will not retire. Additionally, Scalar to Vector and Vector to Vector Re-Decode can only occur if none of the instructions being merged have data dependencies on each other.

3.2. Re-Decode Hierarchy

The proposed hierarchy, that the Re-Decode Architectural Model uses, defines which types and in what order the Re-Decode can be done individually. That is, one step of the hierarchy is one individual application of Re-Decode, if the objective is to do multiple hierarchy jumps, it instead needs to go to the middle stages first.

The categories in which an instruction can be Re-Decoded are:

1. GPR (INT and FP)
2. Scalars
3. Vectors

Thus, the actual Re-Decode on an instruction implies a switch between the levels of this hierarchy:

1. GPR -> Scalar
2. Scalar -> Vector
3. Vector -> Vector

3.3. Affinity

The Affinity of an instruction is defined as the characteristic of a tuple and represents the ability of obtaining the same result from both instructions of the tuple with the same input operands, that is, the relation between 2 instructions where the result will be equal with the same input values. If an instruction has affinity with another it means that both instructions will execute towards the same solution. It is important to mention that the output of both instructions will match if the inputs of one instruction match the inputs of the other, regardless of the input value.

If an instruction has affinity with another instruction, but it is required that one of the instructions converts the format of the input to match the type of the other instruction, it is still considered that both instructions have affinity, but in a lower manner. For example, if a GPR instruction takes an integer input, converts it to floating point, and adds it to another FP value, it will have affinity with a Scalar FP addition, but it will require first a Scalar Conversion from Int to FP.

The affinity of the instructions can be obtained by analyzing the different ISA instructions and pairing the ones that accomplish the same operation but with different register sources. Vector -> Vector is the easiest to identify since the same instruction can operate on multiple instruction lengths, but the other hierarchy jumps can also be matched as well. In order to formalize these concepts, a function is proposed.

3.3.1 Affinity Function

The Affinity Function is a comparison, where the input is a tuple made of 2 instructions, and the output is a discrete value that defines the level of affinity between those 2 instructions. The affinity function does not consider the operands of the instructions, only the instructions themselves, so the level of affinity will cover every possible input on the instructions.

$$Affinity(Instruction_1, Instruction_2) = x, x \in \{high, low, null\} \quad (3\text{-}1)$$

The output of (3-1) is an element of a set of categorical elements, composed by high, low and null; high and low refer that there is affinity between both instructions, while null refers to the fact that there is no affinity between both instructions.

If both instructions have high affinity, it means that both will output the same result of execution for the same input operands.

If both instructions have low affinity, both instructions will also output the same result of execution, but some data conversion is needed to fulfill the execution. Data conversion refers to a format change (such as Int to FP) or precision change (such as single precision to double precision).

If both instructions have null affinity, it means that there is no guarantee both will result in the same output for a shared input.

3.3.2 Examples of Affinity

Considering the x86 ISA [12], which is used on this Re-Decode Architectural Model implementation, there are multiple examples of different addition types and their affinities.

Starting with GPR -> Scalar, the basic ADD instruction has high affinity with VPADD, this is a Vector Addition of Integers, by inputting one value to the vector or mask every element out but the first one, it recreates a scalar addition with an integer value; this applies for every int precision (8, 16, 32, 64) so the precision is not specified in the example, in x86 the precision is specified on GPR instructions with the register name (for example AL, AX, EAX, RAX respectively) and on vector registers with the last letter of the instruction (for example VPADDB, VPADDW, VPADDD, VPADDQ), an ‘X’ is used to imply that any of these versions work.

$$Affinity(ADD, VPADDX_{Masked}) = high \quad (3\text{-}2)$$

The FADD instruction, which is the same addition but for FP values on x87, has high affinity with the VADDS instruction since the regular FP sum and a scalar FP sum will result in the same value; an important consideration involves the default precision of the x87 which is extended precision (80bits), this is not present on vector nor scalars, but configuration registers can be programmed such that FP registers will also execute with single or double precision formats by rounding the result to the specified format, since this configurations are done at config registers level and not by adding conversion instructions, that is why is still high affinity. As mentioned in 2.4.4 the 'S' in the instruction stands for Scalar, as opposed to, if it were a vector instruction it would have been VADDP where the 'P' stands for Packed, both 'S' and 'P' are followed by the precision (32 and 64 which are S and D respectively); for x86 instructions is important to consider the position of the Packed 'P' letter, if its present before the mnemonic (ADD in these examples) it is executing on int values, and if its present after the mnemonic it is instead executing on FP values.

$$Affinity(FADD_{Round}, VADDSX) = high \quad (3\text{-}3)$$

The VPADD instruction can have a mask, which was used for (3-2) to leave only one element, and this instruction has high affinity with itself with an unmasked version (or a masked version where more than 1 value is present). This is not a GPR -> Scalar Re-Decode but instead a Scalar -> Vector instance, and since the instruction is the same, the execution of the individual operands will always be the same; thus, the VPADD masked instruction has high affinity with the regular VPADD instruction. The affinity function only implies that the VPADD masked instruction has high affinity with the regular VPADD and can be Re-Decoded, but an implementation of Re-Decode will also include a counter of how many VPADDs are, since it is a Scalar -> Vector Re-Decode, requires at least 2 instances of the same instruction to merge 2 scalar values into one vector of 2 elements.

$$Affinity(VPADDX_{Masked}, VPADDX) = high \quad (3\text{-}4)$$

The VADDS instruction has high affinity with the VADDP instruction, this uses the same argument as VPADD with mask and regular, but now on scalars and vectors of FP values, here as mentioned for (3-3) the 'P' for Packed is present, and a vector instruction is considered a 'Packed'

instruction; it also uses the same argument of considering the amount of instructions for the Re-Decode implementation while the Affinity Function will only return if it can be done regardless of the number of instructions.

$$Affinity(VADDSX, VADDPX) = high \tag{3-5}$$

Now for the Vector -> Vector scenario, it will require a transition in vector length instead of instruction, thus, the same instruction is utilized; considering the same addition type instructions, the VADDP instruction with xmm registers (of 128 bits) has high affinity with the same instruction but with ymm registers (256 bits).

$$Affinity\left(VADDPX_{xmm}, VADDPX_{ymm}\right) = high \tag{3-6}$$

The same applies for VPADD instructions, it has high affinity with itself for different vector lengths.

$$Affinity\left(VPADDX_{xmm}, VPADDX_{ymm}\right) = high \tag{3-7}$$

Going back to GPR -> Scalar, there is a scenario with low affinity for x86, and this is the case for FIADD. This is an x87 instruction that uses an int operand, converts it into FP, and then adds it to another FP value, resulting in FP. This instruction has low affinity with VADDS since the output will be the same regardless of the input if and only if a conversion is added for VADDS that converts the input from GPR to FP.

The conversion instruction depends on the precision of the integer value, and it has to be inserted before the Re-Decoded instruction. In this example, a VCVTSI2SD instruction must be added, it will do a conversion from Int to double precision FP.

$$Affinity(FIADD, VADDSX) = low \tag{3-8}$$

Lastly, considering the GPR instructions ADD and FADD, these have null affinity between themselves, since the addition is done differently for Ints and FP and thus, the result will not always be the same for matching input operands. This example does not obey the Re-Decode Hierarchy mentioned in 3.2 but it is used to show an example of null affinity.

$$Affinity(ADD, FADD) = null \tag{3-9}$$

3.3.3 Affinity Table

In terms of implementation, to keep track of which instructions which are already decoded and queued can be Re-Decoded, and also keep track of these Re-Decoded instructions up until before their retirement, so these instructions are brought back to their original instruction's registers, the implementation requires an Affinity Table and a Re-Decode Queue.

The Affinity Table is a ROM like block where every Re-Decode definition is possible, the inputs are the elements that define the instruction like its opcode, and the output is the instruction with which it has either high or low affinity, as well as the identifier of it having high affinity; if it has low, both the low affinity instruction and the conversion instruction that should be inserted prior, with the identifier of low affinity.

The Re-Decode Queue is a parallel queue from the Instruction Queue where a pointer or an identifier of the Re-Decoded instructions is placed, and it includes both the original version identifier such as the opcode and the new Re-Decoded version. This queue has the objective of keeping track of the instructions during their execution flow.

Prior to the retirement of the instruction, the Re-Decoded instruction must be returned to its original version, this will keep the execution flow defined by the program and will prevent any dependencies of Re-Decode on other external logics such as the Tomasulo algorithm. Keeping the Re-Decode logic isolated and independent by bringing back the original instruction at the end will allow the ROB logic to function as normal, and any flush or mispredict can also behave normally.

Appendix B shows the Affinity Table of x86 instructions, which includes the instructions that have either high or low affinity, their FU type, affinity type and the instruction with affinity.

3.4. Re-Decode Algorithm

In order to implement the Re-Decoding logic, an algorithm is proposed, which includes the Affinity Function and a newly defined Re-Decode Function. This algorithm works from the IQs, across the Execution pipe, and up to retirement at the same level as the Tomasulo algorithm.

These are the steps of the Re-Decode Algorithm:

1. The instructions in the IQ are inputs to the affinity table.
2. The number of instructions in the respective IQs is measured.

3. If the affinity is high or low, the number of instructions on the IQ assigned to the instruction with affinity is measured.
4. If the number of instructions on the source instructions' IQ is considerably more than the one of the affinity instructions, these instructions are Re-Decoded and sent into the affinity instructions' IQ.
5. The original instruction is stored on the Re-Decode Queue to bring it back at the end.
6. The instructions flow as the new Re-Decoded instructions to their respective new IQs and FUs.
7. After execution and before retirement, the instruction is brought back to its original form, including its format and its original registers.

These steps can be performed multiple times on the same instruction to occupy the most possible FUs in parallel, getting the instruction to do multiple jumps on the Re-Decode hierarchy. For example, first doing a GPR -> Scalar Re-Decode, followed by a Scalar -> Vector Re-Decode.

If Re-Decode is available with an instruction with low affinity, the addition of the conversion instruction should not negatively impact the IPC, since the availability of Re-Decode implies the possibility of using other FUs, and this reduces the latency of the instruction by reducing the cycles it spends on the IQs, the impact on latency of adding a conversion instruction should be smaller than the original latency on the IQ. The low affinity scenario is left as future work since it details the implementation rather than the architecture.

3.4.1 Re-Decode Function

Step 4 of the algorithm defines the inputs and outputs of the Re-Decode Function used for this algorithm; its definition is a function that receives 7 input elements:

1. Instruction type to Re-Decode.
2. The number of instructions of the same type to Re-Decode.
3. The output of the Affinity Table.
4. An indicator of the fullness of the IQ of the instruction to execute.
5. An indicator of the busyness of the FU of the instruction to execute.
6. An indicator of the fullness of the IQ of the instruction with affinity.

7. An indicator of the busyness of the FU of the instruction with affinity.

The output is a nominal indicator that determines the hierarchy jump of Re-Decode for the instruction, which could be either GPR -> Scalar, Scalar -> Vector, Vector -> Vector, or null if it cannot be Re-Decoded or will not be since it is not required.

Since the Affinity Table already gives us the instruction with Affinity, the Re-Decode Function receives this output and uses it to define its own output, the Re-Decode Function is then used as a control function that propagates the signals that indicate which instruction will be used between the original instruction and the instruction with Affinity.

$$ReDecode\left(Inst, Count, Affinity, IQFull_{Inst}, FUFull_{Inst}, IQFull_{Affinity}, FUFull_{Affinity}\right) = x \\ x \in \{GPR \rightarrow Scalar, Scalar \rightarrow Vector, Vector \rightarrow Vector, null\} \tag{3-10}$$

In the first Re-Decode hierarchy level, if the amount of GPR instructions executing and waiting on the IQ are significantly bigger than the amount of scalar instructions, and the instruction has affinity with another scalar instruction, the function outputs an identifier indicating this instruction will do a GPR -> Scalar Re-Decode; this instance does not consider the amount of equal instructions since it only converts one instruction to another without merging or splitting.

At the second Re-Decode hierarchy level, if the Scalar queue has multiple instance of the same type of instruction, since regularly this IQ is shared on scalar and vectors, and the instruction has affinity with another vector instruction, the function outputs an identifier that indicates this set of instructions will merge for a Scalar -> Vector Re-Decode; the function should consider that the amount of instructions and the precision of their operands match a vector size.

Finally, in the last Re-Decode hierarchy level, the Vector -> Vector Re-Decode also considers the number of instructions. It is expected that it will have high affinity since it will have it with itself; the change of vector size should match possible sizes according to hardware, number of instructions or power management needs.

3.5. Architectural Model Implementation

The implementation of the Re-Decode Architectural model in Gem5 consists of the addition of a class that refers to the affinity table, and the definition of the Re-Decode Function which is implemented close to the IQs.

```
class AffinityTable
{
    protected:
        // Hash table that will represent the affinity table
        // The key is the inputed OpClass
        // The value is the affinity Opclass
        std::unordered_map<OpClass,OpClass>__affinityTable;

    public:
        // Constructor
        AffinityTable();
        // Function getAffinity which returns
        // a set member of the affinity table
        OpClass getAffinity(OpClass inputOpClass)
        {
            return __affinityTable[inputOpClass];
        }
};
```

Code 3-1 Definition of the AffinityTable class in affinity_table.hh

3.5.1 The AffinityTable Class

A new class is added to Gem5 called AffinityTable, which is defined in both affinity_table.hh and affinity_table.cc files. On the header, an unordered_map is used as the stored definition of the instructions with affinity. A base constructor is defined without input arguments, and most importantly, the getAffinity() function is specified, as the return of the instruction with affinity. The header file is shown in Code 3-1.

The first consideration on the implementation is the usage of the OpClass instead of the instruction for both input and output, this is due to the simplicity of its usage to obtain the Instruction type and thus the IQ and the FU in which it will be executed; the FU types are strictly defined for their instruction types and their definition is almost identical to the types defined in this document, furthermore, the instructions can be grouped by their OpClass and the performance should be equal to the expected.

Another consideration includes the singular output of the OpClass with affinity without including the level of affinity, the reason to be implemented this way is to reuse the same argument of the OpClass usage instead of the instruction, the instructions with low affinity will still be sent

to the destination FU with affinity including the conversion instruction, that is, the conversion instruction will have the type of the affinity instruction so it executes on that same FU.

In the affinity_table.cc file the definition of the elements of this table is present, as shown in Code 3-2. As mentioned early the used keys and values of the unordered map are OpClass, which is the identifier of the instruction type, one big consideration is the definition of SimdShadow OpClasses, this OpClasses were added so that Scalar -> Vector Re-Decode is easier to define and identify on the simulation, the argument on their implementation comes from the fact that converting a scalar instruction into a vector instruction will use the same registers with the same FU, with the only difference on the amount of bits in the register that are considered.

```
AffinityTable::AffinityTable()
{
    //GPR -> SCALAR
    __affinityTable[IntAluOp] = SimdAluOp;
    __affinityTable[IntMultOp] = SimdMultOp;
    __affinityTable[IntDivOp] = SimdDivOp;
    __affinityTable[FloatAddOp] = SimdFloatAddOp;
    __affinityTable[FloatCmpOp] = SimdFloatCmpOp;
    __affinityTable[FloatCvtOp] = SimdFloatCvtOp;
    __affinityTable[FloatMultOp] = SimdFloatMultOp;
    __affinityTable[FloatMultAccOp] = SimdFloatMultAccOp;
    __affinityTable[FloatDivOp] = SimdFloatDivOp;
    __affinityTable[FloatMiscOp] = SimdFloatMiscOp;
    __affinityTable[FloatSqrtOp] = SimdFloatSqrtOp;
    // SCALAR -> VECTOR
    __affinityTable[SimdAddOp] = SimdShadowAddOp;
    __affinityTable[SimdAddAccOp] = SimdShadowAddAccOp;
    __affinityTable[SimdAluOp] = SimdShadowAluOp;
    __affinityTable[SimdCmpOp] = SimdShadowCmpOp;
    __affinityTable[SimdCvtOp] = SimdShadowCvtOp;
    __affinityTable[SimdMiscOp] = SimdShadowMiscOp;
    __affinityTable[SimdMultOp] = SimdShadowMultOp;
    __affinityTable[SimdMultAccOp] = SimdShadowMultAccOp;
    __affinityTable[SimdMatMultAccOp] = SimdShadowMatMultAccOp;
    __affinityTable[SimdShiftOp] = SimdShadowShiftOp;
    __affinityTable[SimdShiftAccOp] = SimdShadowShiftAccOp;
    __affinityTable[SimdDivOp] = SimdShadowDivOp;
    __affinityTable[SimdSqrtOp] = SimdShadowSqrtOp;
    __affinityTable[SimdReduceAddOp] = SimdShadowReduceAddOp;
    __affinityTable[SimdReduceAluOp] = SimdShadowReduceAluOp;
    __affinityTable[SimdReduceCmpOp] = SimdShadowReduceCmpOp;
```

```
    __affinityTable[SimdFloatAddOp] = SimdShadowFloatAddOp;
    __affinityTable[SimdFloatAluOp] = SimdShadowFloatAluOp;
    __affinityTable[SimdFloatCmpOp] = SimdShadowFloatCmpOp;
    __affinityTable[SimdFloatCvtOp] = SimdShadowFloatCvtOp;
    __affinityTable[SimdFloatDivOp] = SimdShadowFloatDivOp;
    __affinityTable[SimdFloatMiscOp] = SimdShadowFloatMiscOp;
    __affinityTable[SimdFloatMultOp] = SimdShadowFloatMultOp;
    __affinityTable[SimdFloatMultAccOp] = SimdShadowFloatMultAccOp;
    __affinityTable[SimdFloatMatMultAccOp] = SimdShadowFloatMatMultAccOp;
    __affinityTable[SimdFloatSqrtOp] = SimdShadowFloatSqrtOp;
    __affinityTable[SimdFloatReduceCmpOp] = SimdShadowFloatReduceCmpOp;
    __affinityTable[SimdFloatReduceAddOp] = SimdShadowFloatReduceAddOp;
    __affinityTable[SimdAesOp] = SimdAesOp;
    __affinityTable[SimdAesMixOp] = SimdAesMixOp;
    __affinityTable[SimdSha1HashOp] = SimdSha1HashOp;
    __affinityTable[SimdSha1Hash2Op] = SimdSha1Hash2Op;
    __affinityTable[SimdSha256HashOp] = SimdSha256HashOp;
    __affinityTable[SimdSha256Hash2Op] = SimdSha256Hash2Op;
    __affinityTable[SimdShaSigma2Op] = SimdShaSigma2Op;
    __affinityTable[SimdShaSigma3Op] = SimdShaSigma3Op;
    __affinityTable[SimdPredAluOp] = SimdPredAluOp;
}
```

Code 3-2 Definition of the entries of the Affinity Table in affinity_table.cc

If a “shadow” unit is defined it will only be accessed when Re-Decode is used as a new vector instruction, this FU will not function as a regular FU and thus it will not be considered as adding FUs to the design, plus, it always executes in parallel with the regular SIMD FU from which the instruction executes, so it can be considered as an extension of that same FU that will only be used when Re-Decode call it, that is, if and only if Scalar -> Vector is done.

If the OpClass of an instruction is not present on the table it means that such instruction is an instruction that cannot be Re-Decoded, which is also an argument that benefits the considerations done on the implementation of this architectural model.

3.5.2 The Re-Decode Function on inst_queue.cc

The second implementation requirement on Gem5 is the Re-Decode algorithm, which is implemented on the inst_queue.cc file and is shown on Code 3-3. This file is where the IQs are

utilized, Gem5 counts with an implementation of 2 layers of main IQs relevant for Re-Decode; the first is instList, where for every thread the instructions are waiting for their dependencies, the second is readyInsts, where for every OpClass the instructions are ready to execute and now wait for their respective FU. Since readyInsts is a set of arrays for every OpClass, readyInsts can be considered a set of queues for each FU type.

```
        OpClass redecode_op_class = affinityTable.getAffinity(op_class);
        if ((op_class != redecode_op_class) &
            (readyInsts[op_class].size() >
            readyInsts[redecode_op_class].size()) &
            !inst->isVector()){
            DPRINTF(ReDecode,"\t\tREDECODED GPR->SCALAR ADD %s TO readyInsts "
            "op_class=%d to queue:%d size_IntAlu=%d "
            "size_affinity=%d inst=%d\n", inst->staticInst->getName(),
            op_class,redecode_op_class,readyInsts[op_class].size(),
            readyInsts[redecode_op_class].size(),inst->seqNum);
            inst->redecodeInst();
            op_class = redecode_op_class;
            iqStats.instReDecoded++;
        }
        redecode_op_class = affinityTable.getAffinity(op_class);
        if ((op_class != redecode_op_class) &
            (readyInsts[op_class].size() >
            readyInsts[redecode_op_class].size()) &
            (inst->isVector() | inst->isRedecoded())){
            DPRINTF(ReDecode,"\t\tREDECODED SCALAR->VECTOR ADD %s "
            "TO readyInsts op_class=%d to queue:%d size_IntAlu=%d "
            "size_affinity=%d inst=%d\n", inst->staticInst->getName(),
            op_class,redecode_op_class,readyInsts[op_class].size(),
            readyInsts[redecode_op_class].size(),inst->seqNum);
            inst->redecodeInst();
            op_class = redecode_op_class;
            iqStats.instReDecoded++;
        }
        if (!inst->isRedecoded()) {
            DPRINTF(ReDecode,"\t\tADD %s TO readyInsts "
            "op_class=%d size_current=%d inst=%d\n",
            inst->staticInst->getName(),
            op_class,readyInsts[op_class].size(),inst->seqNum);
        }
        readyInsts[op_class].push(inst);
```

Code 3-3 Definition of the Re-Decode algorithm implemented in inst_queue.cc

```
// Redecoded setter and getter
void redecodeInst() { redecoded = true;  }
bool isRedecoded() { return redecoded; }

// The redecode() Function sets the redecodedOpClass
// to the OpClass of the afinity instruction
void redecode(OpClass newOpClass) { redecodedOpClass = newOpClass; }
```

Code 3-4 Definition of the redecode() function and the used variables.

The algorithm first gets the affinity, and with it measures the number of instructions in the queues of each FU. The implementation counts a minimal difference of 1 instruction between both queues, but this can be changed according to the architectural needs. If the difference between the number of instructions matches the required, it then Re-Decodes the instruction and sends it to the appropriate new IQ.

The algorithm Re-Decodes the instruction in 2 independent revisions, where the first is done for GPR -> Scalar and the second is Scalar -> Vector. This allows jumping in the 2 hierarchy levels on one instruction if possible. The redecode() function and its variables are defined on dyn_inst.cc and shown in Code 3-4.

The last consideration in the implementation is the simplification of OpClasses overall instead of the complete modification of the instructions. The reason this is done, other than simplicity of implementation and explanation, is by the nature of Re-Decode being executing similar instructions. For Gem5, the scalar and vector instructions share the base operation of a GPR instruction and only change the OpClass as well as sizes, thus, doing this change on simulation as well results in the same as the way is implemented. Also the registers are not changed, since in Tomasulo the registers are renamed, and at the end brought back to their original, that same renaming can be used to visualize a new register of a different type and at the end it will also get back, so doing multiple changes to the same logical register only adds logic that results in the same destination as our implementation.

With this implementation the Re-Decode Architectural Model is completed and ready to be tested, but in order to measure these changes, some Statistics have been added as well; on the iqStats class the instReDecoded statistic is added, where each instance of Re-Decoded instructions is counted, its usage is shown in Code 3-3 and its definition in Code 3-5.

```
struct IQStats : public statistics::Group
{
    IQStats(CPU *cpu, const unsigned &total_width);
    /** Stat for number of instructions added. */
    statistics::Scalar instsAdded;
    /** Stat for number of non-speculative instructions added. */
    statistics::Scalar nonSpecInstsAdded;

    statistics::Scalar instsIssued;
    /** Stat for number of integer instructions issued. */
    statistics::Scalar intInstsIssued;
    /** Stat for number of floating point instructions issued. */
    statistics::Scalar floatInstsIssued;
    /** Stat for number of branch instructions issued. */
    statistics::Scalar branchInstsIssued;
    /** Stat for number of memory instructions issued. */
    statistics::Scalar memInstsIssued;
    /** Stat for number of miscellaneous instructions issued. */
    statistics::Scalar miscInstsIssued;
    /** Stat for number of squashed instructions that were ready to
     *  issue. */
    statistics::Scalar squashedInstsIssued;
    /** Stat for number of squashed instructions examined when
     *  squashing. */
    statistics::Scalar squashedInstsExamined;
    /** Stat for number of squashed instruction operands examined when
     * squashing.
     */
    statistics::Scalar squashedOperandsExamined;
    /** Stat for number of non-speculative instructions removed due to
     *  a squash.
     */
    statistics::Scalar squashedNonSpecRemoved;
    // Also include number of instructions rescheduled and replayed.

    /** Distribution of number of instructions in the queue.
     * @todo: Need to create struct to track the entry time for each
     * instruction. */
    // statistics::VectorDistribution queueResDist;
    /** Distribution of the number of instructions issued. */
    statistics::Distribution numIssuedDist;
    /** Distribution of the cycles it takes to issue an instruction.
     * @todo: Need to create struct to track the ready time for each
     * instruction. */
```

```
        // statistics::VectorDistribution issueDelayDist;

        /** Number of times an instruction could not be issued because a
         * FU was busy.
         */
        statistics::Vector statFuBusy;
        // statistics::Vector dist_unissued;
        /** Stat for total number issued for each instruction type. */
        statistics::Vector2d statIssuedInstType;

        /** Number of instructions issued per cycle. */
        statistics::Formula issueRate;

        /** Number of times the FU was busy. */
        statistics::Vector fuBusy;
        /** Number of times the FU was busy per instruction issued. */
        statistics::Formula fuBusyRate;

        /** Number of times an instruction has been Re-Decodificated **/
        statistics::Scalar instReDecoded;
    } iqStats;
```

Code 3-5 Addition of the instReDecoded statistic in inst_queue.hh

4. Experiments

To test the efficiency of the Re-Decode Architectural Model, assessing the IPC increase, a set of benchmarks between 2 models was used, first the reference/golden model, which includes the basic implementation of Gem5, and then the Re-Decode Model, which includes the modifications mentioned in 3.5.

4.1. Model Characteristics

The Re-Decode model was constructed on top of the reference model, inheriting the same Out-of-Order x86 architecture, which is defined as X86O3CPU; this can be done since Gem5 brings the possibility of running not only multiple ISAs but also different architectures and simulation models. On top of it, the changes to include the affinity table have been implemented, as well as the Re-Decode function as defined in 3.5 for the Re-Decode model.

The amount of FUs was modified in both models to have a balanced scenario, counting with 3 IntALUs and an IQ of 32 instructions. This model also counts with 4 threads defined by the build command for Gem5 “scons build/X86/gem5.opt -j 4”. This model also has multiple instruction queues as defined in 3.5.2 where instList is a general queue for every instruction waiting for its dependencies, and readyInsts is a set of queues, one per each OpClass, where the instructions are now waiting for an available FU for execution.

This model also includes a set of Statistic classes, where multiple variables store the number of statistics that measures multiple characteristics of the simulations, including basic metrics like IPC and Throughput, and more specific ones for every component inside the simulator such as the number of instructions executed per FU, branch mispredictions and memory misses.

Inside our simulation script, the definition of the cache is controlled by a parameter with the parser for the execution command, as shown in Code 4-1. The L1 instructions cache was defined with 64kB, the L1 data cache with 128kB, and the L2 with 1MB. The command used for these experiments is “build/X86/gem5.opt src/tutorial/simple_script.py --l2_size='1MB' --l1d_size='128kB' --l1i_size='64kB'”.

```
'''
Simulation Parser for parameters
Using the argparse python library
'''
parser = argparse.ArgumentParser(description='Simple CPU OOO with 2-level Cache.')
parser.add_argument("binary", default="", nargs="?", type=str, help="Path to the binary to execute")
parser.add_argument("--l1i_size", help=f"L1 Instruction Cache size. Default: 16kB")
parser.add_argument("--l1d_size", help=f"L1 Data Cache size. Default: 64kB")
parser.add_argument("--l2_size", help=f"L2 Cache size. Default: 256kB")
options = parser.parse_args()
```

Code 4-1 Parser system for cache arguments coming from the command line.

The rest of the characteristics defined on the script are explained in Appendix A. some of those definitions include a 1GHz frequency with a 512MB external memory, these numbers are not close to a state-of-the-art CPU, but that same configuration is used in both models, thus, the IPC increment can be visualized with the difference between the 2. The definition of the frequency and external memory is shown in Code 4-2.

```
'''
Clock Domain definitions
SrcClockDomain() makes the clk_domain variable a Clock Domain defined by Gem5
clock will keep the frequency
voltage_domain will keep a voltage, the default is obtained by VoltageDomain()
'''
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()
'''
Memory Definition
mem_mode defines the "simulation mode" of the memory
    timing is the most used
mem_ranges defines the size, we use AddrRange() for the creation of a whole range with the defined value
'''
system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]
```

Code 4-2 Definition of the frequency and external memory in the simulation script.

4.2. Benchmarks

The Benchmarks selected to evaluate the IPC differences between the reference model and the Re-Decode model are: Dhrystone, Whetstone and Gem5's threads benchmark. Dhrystone is a benchmark focused on integer performance, Whetstone is a benchmark that, in contrast, focuses on FP performance, and lastly the threads benchmark which is present on the gem5 model.

The Threads Benchmark is a simple C++ code included on the Gem5 repo as well as other workflows and C++ programs, this is the longest and most complex of these included flows since it utilizes the defined number of threads on elaboration, it is based on arithmetic operations to

pointers and their respective variables [6]. This benchmark is selected since it represents what can be considered a daily basis workflow.

Dhrystone is a synthetic benchmark developed by Reinhold P. Weicker, which seeks to represent intensive programming workflows with integer instructions [13]. This benchmark was created in 1984 but has been constantly updated through the years. Its last official version, 2.1, was released in 2010, and since then it has been publicly modified. The version used for Re-Decode was refurbished in 2023 by Nick Felker [14].

Whetstone is a synthetic benchmark designed originally by the Technical Support Unit of the Department of Trade and Industry, which was seeking to evaluate the performance of computers with FP instructions. This benchmark has also been improved during the years; the version used for Re-Decode was developed by Netlib [15] and is implemented in Gem5 in the same way Dhrystone was included.

Dhrystone includes multiple scenarios in which Re-Decode is present, this benchmark is selected since it is an example of the best performance Re-Decode could have, this comes from the fact that Dhrystone exercises an excessive amount of integer instructions with combinations of both dependencies between them and independent to others, the exhaust utilization of int instructions will cause the fullness of both the FUs and the IQs for these instructions, and this scenario is where Re-Decode is mostly used for GPR -> Scalar transitions.

In contrast, Whetstone includes fewer Re-Decode scenarios; thus, this benchmark is selected as the example of the worst case scenario for Re-Decode, since Whetstone utilizes more FP instructions without removing the need of int and flow control instructions, the balance of instructions among the FUs and the IQs is not completely demanding on one instruction type and instead are more distributed, thus the scenarios of Re-Decode are not as common as on Dhrystone.

Lastly the Threads benchmark is the one that represents a common workflow instead of an specialized workflow, thus its representation of a daily basis flow complements the other 2 benchmarks, this benchmark comes on the Gem5 repository and it is utilized frequently with this simulator, the threads scenario does not specializes od any instruction type, so the balance of instruction types is similar to a regular flow of any program. With these 3 benchmarks, Re-Decode can be tested and measured completely.

4.3. Metrics

The metrics evaluated in the simulation results include the IPC, which is our main performance indicator, the amount of times an instruction tries to access an FU that is completely busy and thus it has to wait extra cycles in order to execute, the number of accesses to the individual FUs, and also the amount of Re-Decode instances, that is, how many times an instruction was Re-Decoded.

These metrics can be mapped to a Statistic variable in Gem5, at first the IPC which is used for its own purpose of Instructions per Cycle, and FuBusy which is the number of times an instruction tried to access to its FU to be executed and this one was busy, thus causing the instruction to wait more cycles.

Other metrics can be implemented with manually defined Statistics; in order to count the number of times that any instruction was Re-Decoded, instReDecoded was defined; with this last statistic is also expected to see an increment on vecAluAccesses, since vecAlus are the FUs for both scalars and vectors. Also, instReDecoded can be merged with the addition of the AluAccesses statistic to see how the instructions are distributed on the FUs after Re-Decode.

5. Results

After running the Dhrystone, Whetstone, and Threads Benchmarks on both the reference model and the Re-Decode model, an improvement of the Re-Decode model can be seen over the reference in all tested benchmarks. Appendix C shows the stats.txt file of every benchmark run of each model.

5.1. Architecture IPC

The first metric compared between both models is the IPC, the Instructions Per Cycle metric is one of the most important metrics on CPU architecture since it counts how many instructions are executed for each clock cycle. This metric is not only based on retirements, but it also includes the negative impact of branch mispredictions. This is the most used statistic overall on CPU design thanks to its simplicity and completeness.

Starting with the threads benchmark, which has an IPC of 0.988040 on the reference model, incremented on the Re-Decode model to a value of 1.078998. The Dhrystone benchmark on the reference model had an IPC of 1.948333, whereas the Re-Decode model had an IPC of 2.063805. Lastly, the Whetstone benchmark started with an IPC of 1.930105 on the reference model, while on the Re-Decode model was a value of 1.975303.

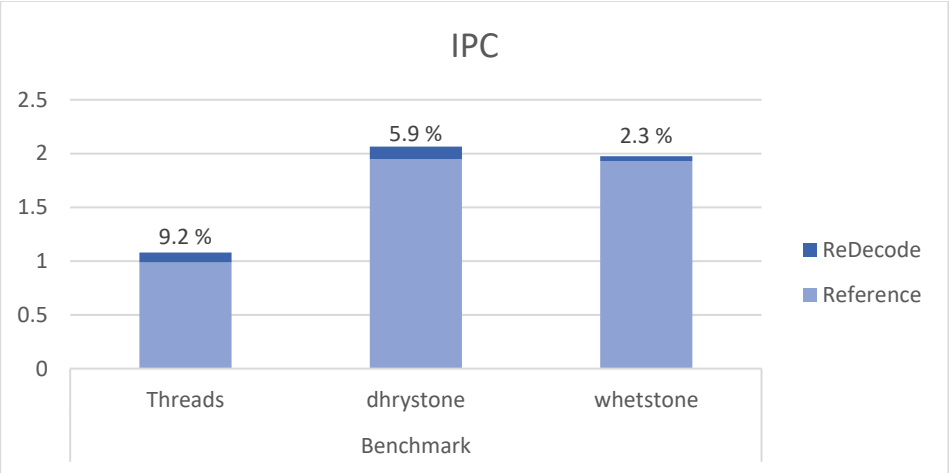

Fig. 5-1 IP Comparison between the run benchmarks, the darker color represents the increment of Re-Decode compared with the reference model.

The threads benchmark counts with the biggest IPC increment of 9.2% over the reference model, Dhrystone is close with an increment of 5.9% in the IPC and lastly Whetstone with an increment of 2.34%, the results are expected since Whetstone is the benchmark that include the least amount of Re-Decode scenarios, Dhrystone being the workflow with the most expected instances has a good increment, and the threads run which represents a daily basis comes with almost 10% of IPC increase, since it presents more common instruction distributions has a higher density GPR FUs and thus brings more Re-Decode scenarios.

The average IPC increase of the 3 benchmarks is 5.8%, which proves that the Re-Decode Architectural Model provides an IPC increment to the CPU. *Fig. 5-1* shows the benchmarks with their respective IPCs on the vertical axis; the darker color represents the increment in the Re-Decode model compared with the reference model.

5.2. FuBusy Statistic

The next metric that is evaluated is called FuBusy, which is the amount of times that an instruction tries to access to an FU but this is completely Full, thus the instruction has to wait more cycles to be executed, this metric is relevant for Re-Decode since the change of instruction type is expected to reduce the amount of accesses to busy FUs by accessing to more available FUs referred to the instruction type of the instruction with affinity.

The number of times where FUs are full and instructions in the queue cannot access to them, which is measured in Gem5 by FuBusy, was reduced considerably; this statistic is present not only as a combined counter for every FU, but it also includes the statistic for each FU, but for simplicity is considered the total FuBusy for every FU summed. The statistic of each individual FU is defined as system.cpu.statFuBusy::OpClass, and is present in Appendix C.

	Threads	Dhrystone	Whetstone
ReDecode	19559	397797	2867854
Reference	694102	3612271	150570329

Table 1 FuBusy Comparison between the 3 benchmarks, the columns represent the benchmark, and the rows represent the Model.

In *Table 1* is shown the number of times in which each benchmark attempts to access to a Busy FU and caused instructions to wait extra cycles, the Re-Decode model managed to decrease in 35 times the amount of FuBusy instances concerning the Reference model, on Dhrystone managed to reduce it by 9 times, and Whetstone managed to reduce it by 52 times, with this data is proven that Re-Decode is able to reduce the number of times the instructions are waiting for an FU by sending them to other FUs.

5.3. Instructions per FU

The last metric is the distribution of the instructions along the FUs in order to see how Vector FUs are now utilized with Re-Decode. This is expected since the most used scenario is GPR -> Scalar, with the increment of instructions executed on the Vector FU is also expected an equivalent reduction of the instructions executed in both Int and Fp units, since those are the GPR instructions that are Re-Decoded into Scalar instructions.

On Gem5, the statistics called intAluAccesses, fpAluAccesses and vecAluAccesses are referred to as the number of instructions issued on their respective FUs, so these are the metrics used to evaluate the distribution among FUs of the instructions executed on each benchmark.

On the threads benchmark, IntAlu is reduced by 43.9%, on Dhrystone by 34.7%, and Whetstone by 36.1%; similarly, FpAlu had a decrease of 19.7% in threads, 12.5% in Dhrystone, and 13.6% in Whetstone. This makes a tendency of 38% of Int and 15% of FP instructions being Re-Decoded. For this statistic, *Fig. 5-2* shows each benchmark individually, where the vertical axis is the number of instructions on each of the FUs represented on the horizontal axis, each FU has a bar for each model, the first 2 graphs representing the Threads and Dhrystone benchmarks are in logarithmic scale in order to appreciate the difference between the models due to the quantity of instructions between FUs.

5. RESULTS

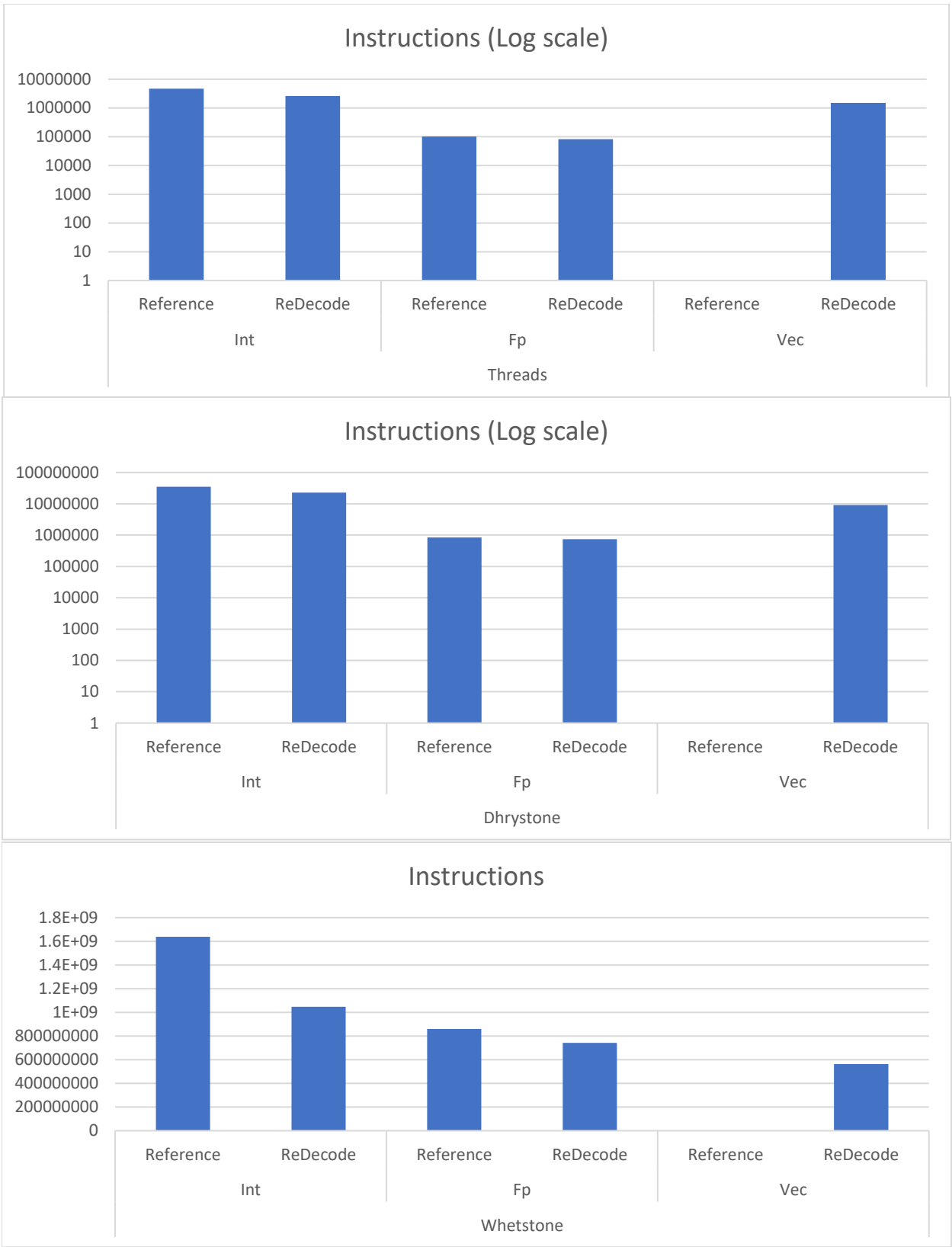

Fig. 5-2 AluAccess Comparison between the 3 benchmarks, each FU has a bar for each model, Threads and Dhrystone are on a logarithmic scale.

6. Future Work

The implementation of the Re-Decode Architectural Model presented only included 2 of the 3 theoretical instances of Re-Decode, so a plausible future work is the implementation of Vector -> Vector, this one will require the complete counter implementation without the considerations made for the implementation on this work such as the utilization of the OpClass as the main identifier of the instruction types as well as the creation of Shadow Vector Units for the increment of vector length, to simulate a more accurate version and also to include both possibilities for vector size increment and decrement. The addition of Vector -> Vector is not expected to have a major impact on the IPC as the instances implemented in this work, but its implementation together with every Re-Decode instance causes a significant impact overall.

Another possibility of improvement is the complete implementation within Gem5 of an integrated instruction modifier class to remove most design considerations done for this implementation. These may increase the precision of the results by considering every possible variable relevant to the Re-Decode logic in a methodology closer to physical implementation.

Lastly, the main follow-up for this architectural model would be the micro-architectural definition and finally implementation of the architecture itself on RTL code, such an implementation would bring simulation results that will prove the IPC increment now in a practical manner. The micro-architectural definition could be paired with the complete implementation mentioned before, and it is required to fulfill the bases for an RTL implementation.

7. Conclusions

In order to achieve an increment on the IPC of a CPU, there are multiple different domains that are not only the focus of research and development but also open for more innovative ideas. The proposed model is an innovation that brings the back end of a CPU core to the next level of utilization. The idea of constantly utilizing every FU has always been the main objective of the front end, and giving the extra layer of Re-Decode helps the CPU to achieve an increment in FU parallel usage thanks to a characteristic from the nature of the instructions themselves that is now defined and utilized as the base characteristic of Re-Decode, which is the affinity.

The concept of affinity is a fundamental part of the Re-Decode Architectural Model, since the need of instructions that will execute the same result has not been overly exploited before to increase the IPC aside from code optimization in compilers, Re-Decode can be seen as a second layer of code optimization after the compiler which lies inside the hardware itself and does not depend on the code flow but instead on the instructions themselves.

The Affinity and Re-Decode functions were defined mathematically in order to bring the availability of future improvement and development, but their concept is fully utilized for Re-Decode with the objective of having their definition easily translated into a spec file.

The utilization of vector units for scalar instructions that come from the Re-Decode of integer or floating-point instructions is the main instance of Re-Decode and thus the one that gives the main amount of IPC increment. By having this possibility as a function that depends on the number of instructions in the queue, with a trigger level open to implementation (like a reconfigurable CSR), gives this model the capability of not losing performance on code segments that are mainly vector instructions.

The decision to work with Gem5 was made thanks to its simplicity of training, understanding and usage. This simulator brings wide capabilities to hardware architects, and as an example, this implementation relied on Gem5.

Lastly, it is concluded that the implementation of this model, and the achievement of the results proving an increment of the IPC, match the objectives and goals of this project, and this model is also expected to be studied or even utilized for future CPU technologies.

8. References

[1] D. Defour and F. de Dinechin, “Software Carry-Save: A Case Study for Instruction-Level Parallelism,” in *Parallel Computing Technologies*, V. E. Malyshkin, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, doi: 10.1007/978-3-540-45145-7_18.

[2] A. Cristal, J. F. Martinez, J. Llosa, and M. Valero, “A case for resource-conscious out-of-order processors,” *IEEE Comput. Archit. Lett.*, vol. 2, no. 1, Jan. 2003, doi: 10.1109/L-CA.2003.4.

[3] A. Dolejsi Santos, A. Wolfe, and E. S. T. Fernandes, “Functional units utilization in a multiple-instruction issue architecture,” in *Proceedings 23rd Euromicro Conference New Frontiers of Information Technology - Short Contributions* -, Sep. 1997, pp. 228–233. doi: 10.1109/EMSCNT.1997.658471.

[4] L. Villa, R. Espasa, and M. Valero, “Effective usage of vector registers in decoupled vector architectures,” in *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing - PDP ’98* -, Jan. 1998, pp. 495–501. doi: 10.1109/EMPDP.1998.647238.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture, A Quantitative Approach*, 6th ed. 2017. Accessed: Jan. 24, 2025. [Online]. Available: https://shop.elsevier.com/books/computer-architecture/hennessy/978-0-12-811905-1

[6] J. Lowe-Power *et al.*, “The gem5 Simulator: Version 20.0+,” Sep. 30, 2020, *arXiv*: arXiv:2007.03152. doi: 10.48550/arXiv.2007.03152.

[7] A. Akram and L. Sawalha, “A Survey of Computer Architecture Simulation Techniques and Tools,” *IEEE Access*, vol. 7, pp. 78120–78145, 2019, doi: 10.1109/ACCESS.2019.2917698.

[8] “2024 Intel Tech Tour: Next Gen P-core-The Lion Cove Microarchitecture,” Intel. Accessed: Jan. 24, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/824430/2024-intel-tech-tour-next-gen-p-core-the-lion-cove-microarchitecture.html

[9] A. Pohl, M. Greese, B. Cosenza, and B. Juurlink, “A Performance Analysis of Vector Length Agnostic Code,” in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, Jul. 2019, pp. 159–164. doi: 10.1109/HPCS48598.2019.9188238.

[10] Y. Kaddar, A. Pohl, and B. Juurlink, “Accelerating The Vvc Decoder For Vector Length Agnostic Simd Architectures,” in *2020 IEEE International Conference on Multimedia and Expo (ICME)*, Jul. 2020, pp. 1–6. doi: 10.1109/ICME46284.2020.9102973.

[11] “Intel® Advanced Vector Extensions 10 (Intel® AVX10) Architecture Specification,” Intel. Accessed: Oct. 27, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/784267/intel-advanced-vector-extensions-10-intel-avx10-architecture-specification.html

[12] “Intel® 64 and IA-32 Architectures Software Developer Manuals.” Accessed: Feb. 24, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[13] R. P. Weicker, “Dhrystone: a synthetic systems programming benchmark,” *Commun. ACM*, vol. 27, no. 10, pp. 1013–1030, Oct. 1984, doi: 10.1145/358274.358283.

[14] N. Felker, “Building & Running Whetstone/Dhrystone in Gem5,” Medium. Accessed: Feb. 20, 2025. [Online]. Available: https://fleker.medium.com/building-running-whetstone-dhrystone-in-gem5-90d465a10bb4

[15] “The Netlib.” Accessed: Feb. 20, 2025. [Online]. Available: https://netlib.org/

Appendix

A. EXPLANATION OF THE SIMULATION SCRIPT

The Gem5 Simulation Scripts are Python files with instances of every component of the CPU being simulated. The script used for Re-Decode is shown in Code A-1. In this appendix, every component present in this script will be explained.

```
import argparse

from caches import *

import m5
from m5.objects import *

"""
Simulation Parser for parameters
Using the argparse python library
"""
parser = argparse.ArgumentParser(
    description="Simple CPU OOO with 2-level Cache."
)
parser.add_argument(
    "--l1i_size", help=f"L1 Instruction Cache size. Default: 16kB"
)
parser.add_argument("--l1d_size", help=f"L1 Data Cache size. Default: 64kB")
parser.add_argument("--l2_size", help=f"L2 Cache size. Default: 256kB")

options = parser.parse_args()

"""
System SimObject
Contains functional (not timing-level) information, such as
    memory ranges, root clock domain, root voltage domain
"""
system = System()

"""
Clock Domain definitions
SrcClockDomain() makes the clk_domain variable a Clock Domain defined by Gem5
clock will keep the frequency
voltage_domain will keep a voltage, the default is obtained by VoltageDomain()
"""
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "1GHz"
```

A. EXPLANATION OF THE SIMULATION SCRIPT

```
system.clk_domain.voltage_domain = VoltageDomain()

"""
Memory Definition
mem_mode defines the "simulation mode" of the memory
    timing is the most used
mem_ranges defines the size, we use AddrRange() for the creation of a whole range with the defined value
"""
system.mem_mode = "timing"
system.mem_ranges = [AddrRange("512MB")]

"""
CPU Definition
Basic example, X86TimingSimpleCPU
    timing-based, each instruction in a single clock cycle (except mem)
CPU Naming Format
    {ISA}{Type}CPU
"""
system.cpu = X86O3CPU()

"""
Memory Bus Definition
System-Wide memory bus with SystemXBar()
"""
system.membus = SystemXBar()

"""
Cache Definition
ICache and DCache are defined individualy
    If no Cache, conect to membus
Two types of Cache
    Clasic
        Regular Simple Cache
    Ruby
        Designed for Coherence
Cache Steps
    Instanciate the Caches
    Conect them to the ports

Hierarchy
    CPU to L1s
    L1s to L2 bus
    L2 bus to L2
    L2 to membus
```

```
"""
# L1s
system.cpu.icache = L1ICache(options.l1i_size)
system.cpu.dcache = L1DCache(options.l1d_size)

system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)

# L2 bus
system.l2bus = L2XBar()

system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)

# L2
system.l2cache = L2Cache(options.l2_size)
system.l2cache.connectCPUSideBus(system.l2bus)
system.l2cache.connectMemSideBus(system.membus)

# Use this instead if no caches are used, which conect directly to the membus
# system.cpu.icache_port = system.membus.cpu_side_ports
# system.cpu.dcache_port = system.membus.cpu_side_ports

"""
Ports Definition
Each mem object can have two kinds of ports
Request Ports
    From Request port to Response port
Response Ports
    From Response port to Request port
Must be conected that way
    cpu.request_port = cache.response_port
We can assign an array of ports to a port as well

Required Ports
    I/O Controller conected to the memory bus
    Special Functional Only port conected to the memory bus (mem reads and writes)
x86 Ports
    PIO conected to the memory bus
    Interrupt port conected to the memory bus
        requestor conected to cpu side
        responder conected to mem side
The Interrupt Controller creation is a function of the cpu class, no need to assign to an atribute
"""
```

A. EXPLANATION OF THE SIMULATION SCRIPT

```
system.cpu.createInterruptController()
system.cpu.interrupts[0].pio = system.membus.mem_side_ports
system.cpu.interrupts[0].int_requestor = system.membus.cpu_side_ports
system.cpu.interrupts[0].int_responder = system.membus.mem_side_ports

system.system_port = system.membus.cpu_side_ports

"""
Memory Controller Definition
A mem controller should be conected to the membus on the mem side
We must define a dram type and its range
"""
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8()
system.mem_ctrl.dram.range = system.mem_ranges[0]
system.mem_ctrl.port = system.membus.mem_side_ports

"""
Process to Execute
Syscall Emulation emulates only CPU and Mem, it only needs the CPU to point to a compiled executable
Fullsystem Emulation emulates the entire hardware system and runs a kernel (Like a Virtual Machine)

Process is a SimObject
    The Processes command is set to the command tu run
    Like an argv
        First position is the executable
        Next positions are the arguments
    Set the CPU to use the process and its workload
    Create the functional execution context
"""
# binary = "tests/test-progs/hello/bin/x86/linux/hello"
binary = "tests/test-progs/threads/bin/x86/linux/threads"
# binary = "tests/test-progs/benchmarks/src/dhrystone"
# binary = "tests/test-progs/benchmarks/src/whetstone"

system.workload = SEWorkload.init_compatible(binary)

process = Process()
process.cmd = [binary]
system.cpu.workload = process
system.cpu.createThreads()

"""
System Instance
```

```
Needs the creation of the Root object
The instantiaton process creates a C++ equivalent of every SimObject in Python
"""
root = Root(full_system=False, system=system)

# root.tutorial = TutorialObject(latency = '1ns', iterations = 10)
# root.tutorial.extra_object = ExtraObject(buffer_size='100B')

m5.instantiate()

"""
Sinulation Execution
"""
print("Start of Simulation!")
exit_event = m5.simulate()
print("End of Simulation!")
print(f"Exit Status: {exit_event.getCause()} @ {m5.curTick()} ticks")
```

Code A-1 Simulation script used for Re-Decode.

The first elements are the imports, where m5 and m5.objects are needed in order to run SCons, which is the tool that Gem5 utilizes to run. The extra imports are argparse and caches, the first one refers to the argparse library, which gives the possibility to the program to input variables from the command line, on the other hand, caches is a library that comes with Gem5's source code where the definition of the caches is present.

The next elements are the parser definition and inclusion in the script, first is created and instantiated with a description, then the arguments are included individually with the add_argument() function of the parser; the first argument is l1i_size, which identifies the size of the l1instruction cache, the second one being l1d_size which now refers to the l1 data cache, and lastly l2_size which now refers to the l2 cache. The parser is then saved in the options variable.

The instance of System is the top of the hierarchy, and every further instance is included inside it as a member of the system.

The first one instantiated is the clock domain, which comes from the SrcClockDomain class and is defined with a 1GHz clock; a voltage domain is also included, coming from the VoltageDomain class.

A. EXPLANATION OF THE SIMULATION SCRIPT

Next up is the Memory definition, where the timing memory mode is defined, which is the most used on Gem5, and a range of memory is defined with the AddrRange() function, a 512MB memory range is created.

The CPU definition is now included. Re-Decode is based on X86O3CPU, which is a timing-based model based on x86 with an Out-of-Order architecture. Other models included in Gem5 have the naming convention: {ISA}{Type}CPU.

Now that the Memory Bus is defined, a system-wide memory bus is defined with the SystemXBar() class.

The caches are now defined, and their arguments come from the options variable defined previously. There are 2 cache types in Gem5, which are Classic and Ruby; the main difference comes from Ruby being focused more on coherence. Classic mode was selected since only one core is used. The hierarchy of the cache is defined as follows:

1. CPU to L1s
2. L1s to L2 bus
3. L2 bus to L2
4. L2 to membus

For the first connection, the connectCPU() function was used on every L1 and the argument is the CPU, for the second connection, now connectBus() was used on these caches as well, the third connection is now done on the l2 cache with connectCPUSideBus() to the l2bus, and lastly connectMemSideBus() was used on the same L2 with the membus.

The next elements included in the script are the ports, where each memory object can have 2 kinds of ports: request ports and response ports. The request ports are connected from a request port to a response port, and the response ports are connected inversely. The implemented connection is a request port from the CPU to the response port of the cache.

The required ports are the interrupt controller, which is created with the createInterruptController() function, the PIO controller connected to the membus side ports which is present on the interrupts[0] array of the CPU object, and both the interrupt requestor and responder connected to the CPU ports and the membus ports respectively.

The Memory Controller is now defined, which is connected to the membus. The MemCtrl() class is instantiated followed by the dram, its range and ports, the dram used is the DDR3_1600_8x8 which was selected since is an example class used commonly on Gem5, the

range is defined with mem_ranges[0] which is a default value, and the port is connected to the mem side ports as well.

Those were all the classes required for the system. The next element present on the script is the Process to execute, which is the benchmarks used; the path of the binary file is stored in the binary variable. The workload is now defined in order to read a binary file, this is done with SEWorkload.init_compatible(binary)

The process is defined with the Process() class, the command is an array that includes the binary, and the Threads of the CPU are defined with createThreads() which uses the amount defined in model elaboration.

Lastly, the system is instantiated with the Root() class, full_system is false since the simulation will run only benchmarks instead of full Operating Systems. It is then executed with an instance of m5 followed by m5.simulate().

The script file includes a counter of simulation ticks obtained with curTick() and the output of the simulation with getCause(), which are printed on the command line.

B. X86 AFFINITY TABLE

The x86 affinity table is defined after analyzing the different instructions on this ISA and pairing the ones that execute the same operation but with different registers. The objective of this table is to exemplify the affinity of the instructions in this ISA and show how the hierarchy jumps are completely present.

Every instruction specified in this table that executes either on Integer and Floating Point FUs has affinity with a Scalar instruction, and every Scalar instruction has affinity with a vector instruction. The operation and operand precision that the instruction executes will be the same on both the input instruction and the instruction with affinity; this guarantees that the result will be the same for any input operand.

Input Instruction	FU type	Instruction with Affinity	Affinity FU type	High or Low Affinity	Added Instruction (Low Affinity)
ADD R/M8	Integer	VPADDB (upper zeros)	Scalar	High	
ADD R/M16	Integer	VPADDW (upper zeros)	Scalar	High	
ADD R/M32	Integer	VPADDD (upper zeros)	Scalar	High	
ADD R/M64	Integer	VPADDQ (upper zeros)	Scalar	High	
AND R/M8	Integer	VPAND (upper zeros)	Scalar	High	
AND R/M16	Integer	VPAND (upper zeros)	Scalar	High	
AND R/M32	Integer	VPANDD (upper zeros)	Scalar	High	
AND R/M64	Integer	VPANDQ (upper zeros)	Scalar	High	
ANDN R/M32	Integer	VPANDND (upper zeros)	Scalar	High	
ANDN R/M64	Integer	VPANDNQ (upper zeros)	Scalar	High	
CMP R/M8	Integer	VPCMPB (upper zeros)	Scalar	High	
CMP R/M16	Integer	VPCMPW (upper zeros)	Scalar	High	

B. X86 AFFINITY TABLE

CMP R/M32	Integer	VPCMPD (upper zeros)	Scalar	High	
CMP R/M64	Integer	VPCMPQ (upper zeros)	Scalar	High	
FADD M32	Floating Point	VADDSS	Scalar	High	
FADD M64	Floating Point	VADDSD	Scalar	High	
FIADD M16	Floating Point	VADDSS	Scalar	Low	VCVTSI2SS
FIADD M32	Floating Point	VADDSD	Scalar	Low	VCVTSI2SD
FCOM M32	Floating Point	VCMPSS	Scalar	High	
FCOM M64	Floating Point	VCMPSD	Scalar	High	
FICOM M32	Floating Point	VCMPSS	Scalar	Low	VCVTSI2SS
FICOM M64	Floating Point	VCMPSD	Scalar	Low	VCVTSI2SD
FDIV M32	Floating Point	VDIVSS	Scalar	High	
FDIV M64	Floating Point	VDIVSD	Scalar	High	
FIDIV M16	Floating Point	VDIVSS	Scalar	Low	VCVTSI2SS
FIDIV M32	Floating Point	VDIVSD	Scalar	Low	VCVTSI2SD
FILD M16	Floating Point	VCVTSI2SS	Scalar	High	
FILD M32	Floating Point	VCVTSI2SD	Scalar	High	
FIST M16	Floating Point	VCVTSS2SI	Scalar	High	
FIST M32	Floating Point	VCVTSD2SI	Scalar	High	
FISTT M16	Floating Point	VCVTTSS2SI	Scalar	High	
FISTT M32	Floating Point	VCVTTSD2SI	Scalar	High	
FLD M32	Floating Point	VMOVAPS (upper zeros)	Scalar	High	

FLD M64	Floating Point	VMOVAPD (upper zeros)	Scalar	High	
FMUL M32	Floating Point	VMULSS	Scalar	High	
FMUL M64	Floating Point	VMULSD	Scalar	High	
FIMUL M16	Floating Point	VMULSS	Scalar	Low	VCVTSI2SS
FIMUL M32	Floating Point	VMULSD	Scalar	Low	VCVTSI2SD
FST M32	Floating Point	VMOVAPS (upper zeros)	Scalar	High	
FST M64	Floating Point	VMOVAPD (upper zeros)	Scalar	High	
FSUB M32	Floating Point	VSUBSS	Scalar	High	
FSUB M64	Floating Point	VSUBSD	Scalar	High	
FISUB M16	Floating Point	VSUBSS	Scalar	Low	VCVTSI2SS
FISUB M32	Floating Point	VSUBSD	Scalar	Low	VCVTSI2SD
IMUL R/M16	Integer	VPMULLW (upper zeros)	Scalar	High	
IMUL R/M32	Integer	VPMULLD (upper zeros)	Scalar	High	
IMUL R/M64	Integer	VPMULLQ (upper zeros)	Scalar	High	
MOV R/M8	Integer	VMOVD (upper zeros)	Scalar	High	
MOV R/M16	Integer	VMOVD (upper zeros)	Scalar	High	
MOV R/M32	Integer	VMOVD	Scalar	High	
MOV R/M64	Integer	VMOVQ	Scalar	High	
MOVSX R/M16 R/M8	Integer	VPMOVSXBW (upper zeros)	Scalar	High	
MOVSX R/M32 R/M8	Integer	VPMOVSXBD (upper zeros)	Scalar	High	
MOVSX R/M64 R/M8	Integer	VPMOVSXBQ (upper zeros)	Scalar	High	
MOVSX R/M32 R/M16	Integer	VPMOVSXWD (upper zeros)	Scalar	High	

B. X86 AFFINITY TABLE

MOVSX R/M64 R/M16	Integer	VPMOVSXWQ (upper zeros)	Scalar	High	
MOVSXD R/M64 R/M32	Integer	VPMOVSXDQ (upper zeros)	Scalar	High	
MOVZX R/M16 R/M8	Integer	VPMOVZXBW (upper zeros)	Scalar	High	
MOVZX R/M32 R/M8	Integer	VPMOVZXBD (upper zeros)	Scalar	High	
MOVZX R/M64 R/M8	Integer	VPMOVZXBQ (upper zeros)	Scalar	High	
MOVZX R/M32 R/M16	Integer	VPMOVZXWD (upper zeros)	Scalar	High	
MOVZX R/M64 R/M16	Integer	VPMOVZXWQ (upper zeros)	Scalar	High	
MUL R/M32	Integer	VPMULHUW (upper zeros)	Scalar	High	
MUL R/M64	Integer	VPMULUDQ (upper zeros)	Scalar	High	
MULX R/M32	Integer	VPMULHUW (upper zeros)	Scalar	High	
MULX R/M64	Integer	VPMULUDQ (upper zeros)	Scalar	High	
NEG R/M8	Integer	VPSUBB (upper zeros)	Scalar	Low	Register of 0s
NEG R/M16	Integer	VPSUBW (upper zeros)	Scalar	Low	Register of 0s
NEG R/M32	Integer	VPSUBD (upper zeros)	Scalar	Low	Register of 0s
NEG R/M64	Integer	VPSUBQ (upper zeros)	Scalar	Low	Register of 0s
NOT R/M8	Integer	VPANDN (upper zeros)	Scalar	Low	Register of 1s
NOT R/M16	Integer	VPANDN (upper zeros)	Scalar	Low	Register of 1s
NOT R/M32	Integer	VPANDND (upper zeros)	Scalar	Low	Register of 1s
NOT R/M64	Integer	VPANDNQ (upper zeros)	Scalar	Low	Register of 1s
OR R/M8	Integer	VPOR (upper zeros)	Scalar	High	
OR R/M16	Integer	VPOR (upper zeros)	Scalar	High	

OR R/M32	Integer	VPORD (upper zeros)	Scalar	High	
OR R/M64	Integer	VPORQ (upper zeros)	Scalar	High	
ROL R/M32	Integer	VPROLD (upper zeros)	Scalar	High	
ROL R/M64	Integer	VPROLQ (upper zeros)	Scalar	High	
ROR R/M32	Integer	VPRORD (upper zeros)	Scalar	High	
ROR R/M64	Integer	VPROLQ (upper zeros)	Scalar	High	
SAL R/M16	Integer	VPSLLW (upper zeros)	Scalar	High	
SAL R/M32	Integer	VPSLLD (upper zeros)	Scalar	High	
SAL R/M64	Integer	VPSLLQ (upper zeros)	Scalar	High	
SAR R/M16	Integer	VPSRAW (upper zeros)	Scalar	High	
SAR R/M32	Integer	VPSRAD (upper zeros)	Scalar	High	
SAR R/M64	Integer	VPSRAQ (upper zeros)	Scalar	High	
SHL R/M16	Integer	VPSLLW (upper zeros)	Scalar	High	
SHL R/M32	Integer	VPSLLD (upper zeros)	Scalar	High	
SHL R/M64	Integer	VPSLLQ (upper zeros)	Scalar	High	
SHR R/M16	Integer	VPSRLW (upper zeros)	Scalar	High	
SHR R/M32	Integer	VPSRLD (upper zeros)	Scalar	High	
SHR R/M64	Integer	VPSRLQ (upper zeros)	Scalar	High	
SARX R/M32	Integer	VPSRAD (upper zeros)	Scalar	High	
SARX R/M64	Integer	VPSRAQ (upper zeros)	Scalar	High	
SHLX R/M32	Integer	VPSLLD (upper zeros)	Scalar	High	

B. X86 AFFINITY TABLE

SHLX R/M64	Integer	VPSLLQ (upper zeros)	Scalar	High	
SHRX R/M32	Integer	VPSRLD (upper zeros)	Scalar	High	
SHRX R/M64	Integer	VPSRLQ (upper zeros)	Scalar	High	
SUB R/M8	Integer	VPSUBB (upper zeros)	Scalar	High	
SUB R/M16	Integer	VPSUBW (upper zeros)	Scalar	High	
SUB R/M32	Integer	VPSUBD (upper zeros)	Scalar	High	
SUB R/M64	Integer	VPSUBQ (upper zeros)	Scalar	High	
TEST R/M8	Integer	VPTEST (upper zeros)	Scalar	High	
TEST R/M16	Integer	VPTEST (upper zeros)	Scalar	High	
TEST R/M32	Integer	VPTEST (upper zeros)	Scalar	High	
TEST R/M64	Integer	VPTEST (upper zeros)	Scalar	High	
TZCNT R/M32	Integer	VPLZCNTD (upper zeros)	Scalar	High	
TZCNT R/M64	Integer	VPLZCNTQ (upper zeros)	Scalar	High	
VADDSD R/M64	Scalar	VADDPD	Vector	High	
VADDSS R/M32	Scalar	VADDPS	Vector	High	
VCMPSD R/M64	Scalar	VCMPPD	Vector	High	
VCMPSS R/M32	Scalar	VCMPPS	Vector	High	
VCVTSD2SI R/M64	Scalar	VCVTPD2PI	Vector	High	
VCVTSI2SD R/M32	Scalar	VCVTPI2PD	Vector	High	
VCVTSI2SS R/M16	Scalar	VCVTPI2PS	Vector	High	
VCVTSS2SI R/M32	Scalar	VCVTPS2PI	Vector	High	

VCVTTSD2SI R/M64	Scalar	VCVTTPD2PI	Vector	High	
VCVTTSS2SI R/M32	Scalar	VCVTTPS2PI	Vector	High	
VDIVSD R/M64	Scalar	VDIVPD	Vector	High	
VDIVSS R/M32	Scalar	VDIVPS	Vector	High	
VMOVAPD (upper zeros)	Scalar	VMOVAPD	Vector	High	
VMOVAPS (upper zeros)	Scalar	VMOVAPS	Vector	High	
VMOVD R/M32	Scalar	VMOVDQA	Vector	High	
VMOVQ R/M64	Scalar	VMOVDQA	Vector	High	
VMULSD R/M64	Scalar	VMULPD	Vector	High	
VMULSS R/M32	Scalar	VMULPS	Vector	High	
VSUBSD R/M64	Scalar	VSUBPD	Vector	High	
VSUBSS R/M32	Scalar	VSUBPS	Vector	High	
VPADDB (upper zeros)	Scalar	VPADDB	Vector	High	
VPADDD (upper zeros)	Scalar	VPADDD	Vector	High	
VPADDQ (upper zeros)	Scalar	VPADDQ	Vector	High	
VPADDW (upper zeros)	Scalar	VPADDW	Vector	High	
VPANDB (upper zeros)	Scalar	VPANDB	Vector	High	
VPANDD (upper zeros)	Scalar	VPANDD	Vector	High	
VPANDQ (upper zeros)	Scalar	VPANDQ	Vector	High	
VPANDW (upper zeros)	Scalar	VPANDW	Vector	High	
VPANDNB (upper zeros)	Scalar	VPANDNB	Vector	High	

B. X86 AFFINITY TABLE

VPANDND (upper zeros)	Scalar	VPANDND	Vector	High	
VPANDNQ (upper zeros)	Scalar	VPANDNQ	Vector	High	
VPANDNW (upper zeros)	Scalar	VPANDNW	Vector	High	
VPCMPB (upper zeros)	Scalar	VPCMPB	Vector	High	
VPCMPD (upper zeros)	Scalar	VPCMPD	Vector	High	
VPCMPQ (upper zeros)	Scalar	VPCMPQ	Vector	High	
VPCMPW (upper zeros)	Scalar	VPCMPW	Vector	High	
VPMOVSXBD (upper zeros)	Scalar	VPMOVSXBD	Vector	High	
VPMOVSXBQ (upper zeros)	Scalar	VPMOVSXBQ	Vector	High	
VPMOVSXBW (upper zeros)	Scalar	VPMOVSXBW	Vector	High	
VPMOVSXDQ (upper zeros)	Scalar	VPMOVSXDQ	Vector	High	
VPMOVSXWD (upper zeros)	Scalar	VPMOVSXWD	Vector	High	
VPMOVSXWQ (upper zeros)	Scalar	VPMOVSXWQ	Vector	High	
VPMOVSZBD (upper zeros)	Scalar	VPMOVSZBD	Vector	High	
VPMOVSZBQ (upper zeros)	Scalar	VPMOVSZBQ	Vector	High	
VPMOVSZBW (upper zeros)	Scalar	VPMOVSZBW	Vector	High	
VPMOVSZWD (upper zeros)	Scalar	VPMOVSZWD	Vector	High	
VPMOVSZWQ (upper zeros)	Scalar	VPMOVSZWQ	Vector	High	
VPMULHUW (upper zeros)	Scalar	VPMULHUW	Vector	High	
VPMULLD (upper zeros)	Scalar	VPMULLD	Vector	High	
VPMULLQ (upper zeros)	Scalar	VPMULLQ	Vector	High	

VPMULLW (upper zeros)	Scalar	VPMULLW	Vector	High	
VPMULUDQ (upper zeros)	Scalar	VPMULUDQ	Vector	High	
VPOR (upper zeros)	Scalar	VPOR	Vector	High	
VPORD (upper zeros)	Scalar	VPORD	Vector	High	
VPORQ (upper zeros)	Scalar	VPORQ	Vector	High	
VPROLD (upper zeros)	Scalar	VPROLD	Vector	High	
VPROLQ (upper zeros)	Scalar	VPROLQ	Vector	High	
VPRORD (upper zeros)	Scalar	VPRORD	Vector	High	
VPRORQ (upper zeros)	Scalar	VPRORQ	Vector	High	
VPSLLD (upper zeros)	Scalar	VPSLLD	Vector	High	
VPSLLQ (upper zeros)	Scalar	VPSLLQ	Vector	High	
VPSLLW (upper zeros)	Scalar	VPSLLW	Vector	High	
VPSRAD (upper zeros)	Scalar	VPSRAD	Vector	High	
VPSRAQ (upper zeros)	Scalar	VPSRAQ	Vector	High	
VPSRAW (upper zeros)	Scalar	VPSRAW	Vector	High	
VPSRLD (upper zeros)	Scalar	VPSRLD	Vector	High	
VPSRLQ (upper zeros)	Scalar	VPSRLQ	Vector	High	
VPSRLW (upper zeros)	Scalar	VPSRLW	Vector	High	
VPSUBB (upper zeros)	Scalar	VPSUBB	Vector	High	
VPSUBD (upper zeros)	Scalar	VPSUBD	Vector	High	
VPSUBQ (upper zeros)	Scalar	VPSUBQ	Vector	High	

B. X86 AFFINITY TABLE

VPSUBW (upper zeros)	Scalar	VPSUBW	Vector	High	
VPTEST (upper zeros)	Scalar	VPTEST	Vector	High	
VPLZCNTD (upper zeros)	Scalar	VPLZCNTD	Vector	High	
VPLZCNTQ (upper zeros)	Scalar	VPLZCNTQ	Vector	High	
VPXORB (upper zeros)	Scalar	VPXORB	Vector	High	
VPXORD (upper zeros)	Scalar	VPXORD	Vector	High	
VPXORQ (upper zeros)	Scalar	VPXORQ	Vector	High	
VPXORW (upper zeros)	Scalar	VPXORW	Vector	High	
XOR R/M8	Integer	VPXOR (upper zeros)	Scalar	High	
XOR R/M16	Integer	VPXOR (upper zeros)	Scalar	High	
XOR R/M32	Integer	VPXORD (upper zeros)	Scalar	High	
XOR R/M64	Integer	VPXORQ (upper zeros)	Scalar	High	

C. STATS.TXT FILES

This section presents relevant simulation results, which are part of the stats.txt files generated by Gem5 during simulation. For access to the complete list of statistics, please visit the GitHub repository of this project at https://github.com/Jorge-Padilla/ReDecodification or contact Jorge Padilla at jorge.padillag@iteso.mx.

Threads Benchmark

Reference

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.002225                       # Number of seconds simulated (Second)
simTicks                                   2225251000                       # Number of ticks simulated (Tick)
finalTick                                  2225251000                       # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                     12.49                       # Real time elapsed on the host (Second)
hostTickRate                               178227851                        # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                    743716                        # Number of bytes of host memory used (Byte)
simInsts                                     2198637                        # Number of instructions simulated (Count)
simOps                                       3815141                        # Number of ops (including micro ops) simulated (Count)
hostInstRate                                  176094                        # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                    305563                        # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                         1000                        # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage           1                        # Voltage in Volts (Volt)
system.cpu.numCycles                         2225252                        # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                               1.012105                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                               0.988040                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                     0                        # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                   0                        # Number of work items this cpu completed (Count)
system.cpu.instsAdded                        4322853                        # Number of instructions added to the IQ (excludes non-spec)
system.cpu.nonSpecInstsAdded                    1416                        # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                       4116240                        # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                 16064                        # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined              509122                        # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined           1069566                        # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.squashedNonSpecRemoved                165                        # Number of squashed non-spec instructions that were removed
system.cpu.numIssuedDist::samples            2117967                        # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean               1.943486                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev              1.826941                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows               0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                   829659     39.17%     39.17% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                   112957      5.33%     44.51% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                   205750      9.71%     54.22% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                   504103     23.80%     78.02% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                   323581     15.28%     93.30% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                    84542      3.99%     97.29% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                    40846      1.93%     99.22% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                    14868      0.70%     99.92% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                     1661      0.08%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                0                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                8                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::total              2117967                       # Number of insts issued each cycle (Count)
system.cpu.statFuBusy::No_OpClass                  0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntAlu                 674715     97.21%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntMult                     0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntDiv                      0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatAdd                    0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCmp                    0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCvt                   11      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMult                   0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMultAcc                0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatDiv                    0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMisc                   0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatSqrt                   0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAdd                    16      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAddAcc                  0      0.00%     97.21% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAlu                    69      0.01%     97.22% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCmp                     0      0.00%     97.22% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCvt                    41      0.01%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMisc                   28      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMult                    0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMultAcc                 0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMatMultAcc              0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShift                  23      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShiftAcc                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdDiv                     0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSqrt                    0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAdd                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAlu                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCmp                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCvt                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatDiv                0      0.00%     97.23% # attempts to use FU when none available (Count)
```

C. STATS.TXT FILES

```
system.cpu.statFuBusy::SimdFloatMisc                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMult                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMultAcc             0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMatMultAcc            0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatSqrt                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAdd                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAlu                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceCmp                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceAdd            0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceCmp            0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAes                      0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAesMix                   0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash                 0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash2                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash               0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash2              0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma2                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma3                0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdPredAlu                  0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::Matrix                       0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixMov                    0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixOP                     0      0.00%     97.23% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemRead                   9052      1.30%     98.54% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemWrite                  9444      1.36%     99.90% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemRead               424      0.06%     99.96% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemWrite              279      0.04%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IprAccess                    0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::InstPrefetch                 0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideMaskLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideMaskStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorStridedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorStridedStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIndexedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIndexedStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorWholeRegisterLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorWholeRegisterStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerArith           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatArith             0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatConvert           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerReduce           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatReduce            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorMisc                   0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerExtension            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorConfig                 0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statIssuedInstType_0::No_OpClass        22306      0.54%      0.54% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IntAlu       3181307     77.29%     77.83% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntMult           57      0.00%     77.83% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv         40619      0.99%     78.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd         7914      0.19%     79.01% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp            0      0.00%     79.01% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt         1612      0.04%     79.05% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult            0      0.00%     79.05% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc            0      0.00%     79.05% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatDiv            0      0.00%     79.05% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMisc           0      0.00%     79.05% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatSqrt           0      0.00%     79.05% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAdd         6900      0.17%     79.22% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAddAcc            0      0.00%     79.22% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdAlu        13889      0.34%     79.55% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCmp             0      0.00%     79.55% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCvt         14308      0.35%     79.90% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMisc        13973      0.34%     80.24% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMult            0      0.00%     80.24% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMultAcc            0      0.00%     80.24% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdMatMultAcc            0      0.00%     80.24% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShift         2105      0.05%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdShiftAcc            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdDiv             0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdSqrt            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatAdd            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatAlu            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCmp            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCvt            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatDiv            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMisc            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
```

```
system.cpu.statIssuedInstType_0::SimdFloatMult            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMultAcc            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMatMultAcc            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatSqrt            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAdd            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAlu            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceCmp            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceAdd            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceCmp            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdAes            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAesMix            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash2            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash2            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma2            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma3            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdPredAlu            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::Matrix            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixMov            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixOP            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemRead       557934     13.55%     93.85% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemWrite       217085      5.27%     99.12% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMemRead        20812      0.51%     99.63% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatMemWrite        15419      0.37%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IprAccess            0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::InstPrefetch            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideMaskLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideMaskStore            0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::VectorStridedLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorStridedStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIndexedLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorIndexedStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # Number of instructions issued per FU
type, per thread (Count)
system.cpu.statIssuedInstType_0::VectorWholeRegisterLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorWholeRegisterStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIntegerArith            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorFloatArith            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorFloatConvert            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIntegerReduce            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorFloatReduce            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorMisc            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorIntegerExtension            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorConfig            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::total        4116240                       # Number of instructions issued per FU type, per thread (Count)
system.cpu.issueRate                     1.849786                       # Inst issue rate ((Count/Cycle))
system.cpu.fuBusy                          694102                       # FU busy when requested (Count)
system.cpu.fuBusyRate                    0.168625                       # FU busy rate (busy events/executed inst) ((Count/Count))
system.cpu.intInstQueueReads             10857544                       # Number of integer instruction queue reads (Count)
```

C. STATS.TXT FILES

```
system.cpu.intInstQueueWrites                4706698                       # Number of integer instruction queue writes (Count)
system.cpu.intInstQueueWakeupAccesses        3987918                       # Number of integer instruction queue wakeup accesses (Count)
system.cpu.fpInstQueueReads                   203069                       # Number of floating instruction queue reads (Count)
system.cpu.fpInstQueueWrites                  126828                       # Number of floating instruction queue writes (Count)
system.cpu.fpInstQueueWakeupAccesses           98366                       # Number of floating instruction queue wakeup accesses (Count)
system.cpu.vecInstQueueReads                       0                       # Number of vector instruction queue reads (Count)
system.cpu.vecInstQueueWrites                      0                       # Number of vector instruction queue writes (Count)
system.cpu.vecInstQueueWakeupAccesses              0                       # Number of vector instruction queue wakeup accesses (Count)
system.cpu.intAluAccesses                    4685938                       # Number of integer alu accesses (Count)
system.cpu.fpAluAccesses                      102098                       # Number of floating point alu accesses (Count)
system.cpu.vecAluAccesses                          0                       # Number of vector alu accesses (Count)
system.cpu.numSquashedInsts                    16177                       # Number of squashed instructions skipped in execute (Count)
system.cpu.numSwp                                  0                       # Number of swp insts executed (Count)
system.cpu.timesIdled                           1568                       # Number of times that the entire CPU went into an idle state and
unscheduled itself (Count)
system.cpu.idleCycles                         107285                       # Total number of cycles that the CPU has spent unscheduled due
to idling (Cycle)
system.cpu.MemDepUnit__0.insertedLoads        597785                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.insertedStores       243584                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.conflictingLoads       10864                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__0.conflictingStores       2112                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__1.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__1.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__2.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__2.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__2.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__2.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__3.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__3.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__3.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__3.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.branchPred.lookups_0::NoBranch         1214      0.27%      0.27% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::Return          13964      3.14%      3.41% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::CallDirect      17025      3.83%      7.25% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::CallIndirect     1248      0.28%      7.53% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::DirectCond      381884     85.92%     93.44% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::DirectUncond    19599      4.41%     97.85% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::IndirectCond        0      0.00%     97.85% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::IndirectUncond     9545      2.15%    100.00% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::total         444479                       # Number of BP lookups (Count)
system.cpu.branchPred.squashes_0::NoBranch        1122      1.22%      1.22% # Number of branches that got squashed (completely removed) as
an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::Return         2224      2.43%      3.65% # Number of branches that got squashed (completely removed) as an
earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::CallDirect        5904      6.44%     10.09% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::CallIndirect         624      0.68%     10.77% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::DirectCond       71828     78.32%     89.09% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::DirectUncond        5384      5.87%     94.96% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::IndirectCond           0      0.00%     94.96% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::IndirectUncond        4622      5.04%    100.00% # Number of branches that got squashed (completely
removed) as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::total        91708                       # Number of branches that got squashed (completely removed) as an
earlier branch was mispredicted. (Count)
system.cpu.branchPred.corrected_0::NoBranch          208      1.48%      1.48% # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::Return           14      0.10%      1.58% # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::CallDirect          612      4.35%      5.93% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::CallIndirect          270      1.92%      7.85% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::DirectCond        11504     81.81%     89.66% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::DirectUncond          533      3.79%     93.45% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::IndirectCond            0      0.00%     93.45% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::IndirectUncond          921      6.55%    100.00% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::total        14062                       # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.earlyResteers_0::NoBranch            0                       # Number of branches that got redirected after decode.
(Count)
```

Re-Decode

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.002038                       # Number of seconds simulated (Second)
simTicks                                   2037664000                       # Number of ticks simulated (Tick)
finalTick                                  2037664000                       # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                     22.32                       # Real time elapsed on the host (Second)
hostTickRate                                 91290789                       # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                     745896                       # Number of bytes of host memory used (Byte)
simInsts                                      2198637                       # Number of instructions simulated (Count)
simOps                                        3815141                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                    98501                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                     170921                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu.numCycles                          2037665                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                               0.926786                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                               1.078998                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                      0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                    0                       # Number of work items this cpu completed (Count)
system.cpu.instsAdded                         4285384                       # Number of instructions added to the IQ (excludes non-spec)
system.cpu.nonSpecInstsAdded                     1425                       # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                        4159143                       # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                    417                       # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined              471662                       # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined           735000                       # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.squashedNonSpecRemoved                174                       # Number of squashed non-spec instructions that were removed
system.cpu.numIssuedDist::samples             1931963                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean               2.152807                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev              2.281544                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows                0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                    765269     39.61%     39.61% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                    158979      8.23%     47.84% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                    244508     12.66%     60.50% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                    230809     11.95%     72.44% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                    185866      9.62%     82.06% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                    152126      7.87%     89.94% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                     93292      4.83%     94.77% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                     54037      2.80%     97.56% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                     47077      2.44%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                 0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                 0                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                 8                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::total               1931963                       # Number of insts issued each cycle (Count)
system.cpu.statFuBusy::No_OpClass                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntAlu                     757      3.87%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntMult                      0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntDiv                       0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatAdd                     0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCmp                     0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCvt                     0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMult                    0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMultAcc                 0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatDiv                     0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMisc                    0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatSqrt                    0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAdd                      0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAddAcc                   0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAlu                      0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCmp                      0      0.00%      3.87% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCvt                      1      0.01%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMisc                     0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMult                     0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMultAcc                  0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMatMultAcc               0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShift                    0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShiftAcc                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdDiv                      0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSqrt                     0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAdd                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAlu                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCmp                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCvt                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatDiv                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMisc                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMult                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMultAcc             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMatMultAcc          0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatSqrt                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAdd                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAlu                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceCmp                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceAdd            0      0.00%      3.88% # attempts to use FU when none available (Count)
```

C. STATS.TXT FILES

```
system.cpu.statFuBusy::SimdFloatReduceCmp            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAdd                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAddAcc              0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAlu                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowCmp                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowCvt                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMisc                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMult                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMultAcc             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMatMultAcc            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowShift               0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowShiftAcc             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowDiv                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowSqrt                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatAdd             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatAlu             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatCmp             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatCvt             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatDiv             0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMisc            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMult            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMultAcc            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMatMultAcc            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatSqrt            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceAdd            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceAlu            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceCmp            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatReduceAdd            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatReduceCmp            0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAes                       0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAesMix                    0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash                  0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash2                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash                0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash2               0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma2                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma3                 0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdPredAlu                   0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::Matrix                        0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixMov                     0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixOP                      0      0.00%      3.88% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemRead                    7403     37.85%     41.73% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemWrite                  10669     54.55%     96.27% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemRead                446      2.28%     98.55% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemWrite               283      1.45%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IprAccess                     0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::InstPrefetch                  0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideMaskLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideMaskStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdStridedLoad               0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdStridedStore              0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdIndexedLoad               0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdIndexedStore              0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdWholeRegisterLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdWholeRegisterStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideSegmentedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideSegmentedStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdExt                       0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatExt                  0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdConfig                    0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statIssuedInstType_0::No_OpClass        22511      0.54%      0.54% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IntAlu         1784589     42.91%     43.45% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntMult             55      0.00%     43.45% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv           41558      1.00%     44.45% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd          5896      0.14%     44.59% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp             0      0.00%     44.59% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt           988      0.02%     44.61% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult            0      0.00%     44.61% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc            0      0.00%     44.61% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatDiv             0      0.00%     44.61% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMisc            0      0.00%     44.61% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatSqrt            0      0.00%     44.61% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAdd           5966      0.14%     44.76% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAddAcc            0      0.00%     44.76% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdAlu        1065227     25.61%     70.37% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCmp              0      0.00%     70.37% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCvt          8240      0.20%     70.57% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMisc         13146      0.32%     70.88% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMult             0      0.00%     70.88% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMultAcc            0      0.00%     70.88% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdMatMultAcc            0      0.00%     70.88% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShift          1426      0.03%     70.92% # Number of instructions issued per FU type, per thread (Count)
```

```
system.cpu.statIssuedInstType_0::SimdShiftAcc            0      0.00%     70.92% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdDiv            0      0.00%     70.92% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdSqrt            0      0.00%     70.92% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatAdd         2007      0.05%     70.97% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatAlu            0      0.00%     70.97% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCmp            0      0.00%     70.97% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCvt          424      0.01%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatDiv            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMisc            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMult            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMultAcc            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMatMultAcc            0      0.00%     70.98% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatSqrt            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAdd            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAlu            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceCmp            0      0.00%     70.98% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceAdd            0      0.00%     70.98% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceCmp            0      0.00%     70.98% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowAdd          649      0.02%     70.99% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowAddAcc            0      0.00%     70.99% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowAlu       379550      9.13%     80.12% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowCmp            0      0.00%     80.12% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowCvt         5553      0.13%     80.25% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMisc          686      0.02%     80.27% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMult            0      0.00%     80.27% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMultAcc            0      0.00%     80.27% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMatMultAcc            0      0.00%     80.27% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowShift          612      0.01%     80.28% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowShiftAcc            0      0.00%     80.28% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowDiv            0      0.00%     80.28% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowSqrt            0      0.00%     80.28% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatAdd            0      0.00%     80.28% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatAlu            0      0.00%     80.28% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatCmp            0      0.00%     80.28% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatCvt          200      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatDiv            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMisc            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMult            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMultAcc            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMatMultAcc            0      0.00%     80.29% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatSqrt            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceAdd            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceAlu            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceCmp            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatReduceAdd            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatReduceCmp            0      0.00%     80.29% # Number of instructions issued per FU type, per
thread (Count)
```

C. STATS.TXT FILES

```
system.cpu.statIssuedInstType_0::SimdAes            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAesMix            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash2            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash2            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma2            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma3            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdPredAlu            0      0.00%     80.29% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::Matrix             0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixMov            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixOP            0      0.00%     80.29% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemRead       561783     13.51%     93.79% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemWrite       221383      5.32%     99.12% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMemRead        21258      0.51%     99.63% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatMemWrite        15436      0.37%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IprAccess            0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::InstPrefetch            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideMaskLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideMaskStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdStridedLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdStridedStore            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdIndexedLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdIndexedStore            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdWholeRegisterLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdWholeRegisterStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # Number of instructions issued per FU
type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideSegmentedLoad            0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideSegmentedStore            0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdExt            0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatExt            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdConfig            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::total        4159143                       # Number of instructions issued per FU type, per thread (Count)
system.cpu.issueRate                         2.041132                       # Inst issue rate ((Count/Cycle))
system.cpu.fuBusy                               19559                       # FU busy when requested (Count)
system.cpu.fuBusyRate                        0.004703                       # FU busy rate (busy events/executed inst) ((Count/Count))
system.cpu.instReDecoded                      1821506                       # Number of times an instruction has been Re-Decodificated
(Count)
system.cpu.intInstQueueReads                  7212166                       # Number of integer instruction queue reads (Count)
system.cpu.intInstQueueWrites                 4527830                       # Number of integer instruction queue writes (Count)
system.cpu.intInstQueueWakeupAccesses         2596555                       # Number of integer instruction queue wakeup accesses (Count)
system.cpu.fpInstQueueReads                    163276                       # Number of floating instruction queue reads (Count)
system.cpu.fpInstQueueWrites                   126750                       # Number of floating instruction queue writes (Count)
system.cpu.fpInstQueueWakeupAccesses            78090                       # Number of floating instruction queue wakeup accesses (Count)
system.cpu.vecInstQueueReads                  2894783                       # Number of vector instruction queue reads (Count)
system.cpu.vecInstQueueWrites                  104019                       # Number of vector instruction queue writes (Count)
system.cpu.vecInstQueueWakeupAccesses         1440215                       # Number of vector instruction queue wakeup accesses (Count)
system.cpu.intAluAccesses                     2627177                       # Number of integer alu accesses (Count)
system.cpu.fpAluAccesses                        81980                       # Number of floating point alu accesses (Count)
system.cpu.vecAluAccesses                     1447034                       # Number of vector alu accesses (Count)
system.cpu.numSquashedInsts                     27438                       # Number of squashed instructions skipped in execute (Count)
system.cpu.numSwp                                   0                       # Number of swp insts executed (Count)
system.cpu.timesIdled                            1583                       # Number of times that the entire CPU went into an idle state and
unscheduled itself (Count)
system.cpu.idleCycles                          105702                       # Total number of cycles that the CPU has spent unscheduled due
to idling (Cycle)
system.cpu.MemDepUnit__0.insertedLoads         595917                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.insertedStores        244095                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.conflictingLoads        10859                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__0.conflictingStores         2184                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__1.insertedLoads              0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.insertedStores             0                       # Number of stores inserted to the mem dependence unit. (Count)
```

Dhrystone

Reference

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.009490                       # Number of seconds simulated (Second)
simTicks                                   9490250000                       # Number of ticks simulated (Tick)
finalTick                                  9490250000                       # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  111.71                       # Real time elapsed on the host (Second)
hostTickRate                                84956619                       # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                    675332                       # Number of bytes of host memory used (Byte)
simInsts                                    18490168                       # Number of instructions simulated (Count)
simOps                                      32899467                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                  165524                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                    294515                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                         1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage           1                       # Voltage in Volts (Volt)
system.cpu.numCycles                          9490251                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                               0.513259                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                               1.948333                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                     0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                   0                       # Number of work items this cpu completed (Count)
system.cpu.instsAdded                       32932524                       # Number of instructions added to the IQ (excludes non-spec)
system.cpu.nonSpecInstsAdded                      82                       # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                      32920029                       # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                 1624                       # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined              33133                       # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined           59300                       # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.squashedNonSpecRemoved                46                       # Number of squashed non-spec instructions that were removed
system.cpu.numIssuedDist::samples             9466492                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean               3.477532                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev              1.477945                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows                0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                    186470      1.97%      1.97% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                    786765      8.31%     10.28% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                   1205810     12.74%     23.02% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                   2625587     27.74%     50.75% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                   2440035     25.78%     76.53% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                   1501315     15.86%     92.39% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                    510317      5.39%     97.78% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                    165278      1.75%     99.53% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                     44915      0.47%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                0                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                8                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::total               9466492                       # Number of insts issued each cycle (Count)
system.cpu.statFuBusy::No_OpClass                  0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntAlu                3267524     90.46%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntMult                     0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntDiv                      0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatAdd                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCmp                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCvt                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMult                   0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMultAcc                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatDiv                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMisc                   0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatSqrt                   0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAdd                     0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAddAcc                  0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAlu                    10      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCmp                     0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCvt                     3      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMisc                    1      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMult                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMultAcc                 0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMatMultAcc              0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShift                   0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShiftAcc                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdDiv                     0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSqrt                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAdd                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAlu                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCmp                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCvt                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatDiv                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMisc               0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMult               0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMultAcc            0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMatMultAcc          0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatSqrt               0      0.00%     90.46% # attempts to use FU when none available (Count)
```

C. STATS.TXT FILES

```
system.cpu.statFuBusy::SimdReduceAdd                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAlu                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceCmp                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceAdd            0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceCmp            0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAes                      0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAesMix                   0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash                 0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash2                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash               0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash2              0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma2                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma3                0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdPredAlu                  0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::Matrix                       0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixMov                    0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixOP                     0      0.00%     90.46% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemRead                 269829      7.47%     97.93% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemWrite                 74865      2.07%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemRead                25      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemWrite               14      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IprAccess                    0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::InstPrefetch                 0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideMaskLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideMaskStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorStridedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorStridedStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIndexedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIndexedStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideFaultOnlyFirstLoad          0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorWholeRegisterLoad           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorWholeRegisterStore          0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerArith            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatArith             0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatConvert            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerReduce           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatReduce             0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorMisc                   0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerExtension           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorConfig                 0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statIssuedInstType_0::No_OpClass       570885      1.73%      1.73% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IntAlu      20908291     63.51%     65.25% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntMult       150070      0.46%     65.70% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv        210029      0.64%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd          200      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp            0      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt            0      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult           0      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc            0      0.00%     66.34% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatDiv            0      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMisc           0      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatSqrt           0      0.00%     66.34% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAdd       120008      0.36%     66.71% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAddAcc            0      0.00%     66.71% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdAlu       120152      0.36%     67.07% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCmp             0      0.00%     67.07% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCvt       240048      0.73%     67.80% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMisc       120242      0.37%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMult            0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMultAcc           0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdMatMultAcc           0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShift           0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdShiftAcc            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdDiv             0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdSqrt            0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatAdd            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatAlu            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCmp            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCvt            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatDiv            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMisc           0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMult           0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMultAcc           0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
```

```
system.cpu.statIssuedInstType_0::SimdFloatMatMultAcc            0      0.00%     68.16% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatSqrt            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAdd            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAlu            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceCmp            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceAdd            0      0.00%     68.16% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceCmp            0      0.00%     68.16% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdAes            0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAesMix            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash2            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash2            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma2            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma3            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdPredAlu            0      0.00%     68.16% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::Matrix            0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixMov            0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixOP            0      0.00%     68.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemRead      6306497     19.16%     87.32% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemWrite      3932792     11.95%     99.27% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMemRead       240215      0.73%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatMemWrite          600      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IprAccess            0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::InstPrefetch            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideMaskLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideMaskStore            0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::VectorStridedLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorStridedStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIndexedLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorIndexedStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # Number of instructions issued per FU
type, per thread (Count)
system.cpu.statIssuedInstType_0::VectorWholeRegisterLoad            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorWholeRegisterStore            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIntegerArith            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorFloatArith            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorFloatConvert            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIntegerReduce            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorFloatReduce            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorMisc            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorIntegerExtension            0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorConfig            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::total       32920029                       # Number of instructions issued per FU type, per thread (Count)
system.cpu.issueRate                         3.468826                       # Inst issue rate ((Count/Cycle))
system.cpu.fuBusy                             3612271                       # FU busy when requested (Count)
system.cpu.fuBusyRate                        0.109729                       # FU busy rate (busy events/executed inst) ((Count/Count))
system.cpu.intInstQueueReads                 77237134                       # Number of integer instruction queue reads (Count)
system.cpu.intInstQueueWrites                32124581                       # Number of integer instruction queue writes (Count)
system.cpu.intInstQueueWakeupAccesses        32074952                       # Number of integer instruction queue wakeup accesses (Count)
system.cpu.fpInstQueueReads                   1683311                       # Number of floating instruction queue reads (Count)
system.cpu.fpInstQueueWrites                   842137                       # Number of floating instruction queue writes (Count)
```

C. STATS.TXT FILES

```
system.cpu.fpInstQueueWakeupAccesses          841569                       # Number of floating instruction queue wakeup accesses (Count)
system.cpu.vecInstQueueReads                       0                       # Number of vector instruction queue reads (Count)
system.cpu.vecInstQueueWrites                      0                       # Number of vector instruction queue writes (Count)
system.cpu.vecInstQueueWakeupAccesses              0                       # Number of vector instruction queue wakeup accesses (Count)
system.cpu.intAluAccesses                   35119736                       # Number of integer alu accesses (Count)
system.cpu.fpAluAccesses                      841679                       # Number of floating point alu accesses (Count)
system.cpu.vecAluAccesses                          0                       # Number of vector alu accesses (Count)
system.cpu.numSquashedInsts                     2142                       # Number of squashed instructions skipped in execute (Count)
system.cpu.numSwp                                  0                       # Number of swp insts executed (Count)
system.cpu.timesIdled                            341                       # Number of times that the entire CPU went into an idle state and
unscheduled itself (Count)
system.cpu.idleCycles                          23759                       # Total number of cycles that the CPU has spent unscheduled due
to idling (Cycle)
system.cpu.MemDepUnit__0.insertedLoads        6548154                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.insertedStores       3935424                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.conflictingLoads      2229480                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__0.conflictingStores       563582                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__1.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__1.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__2.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__2.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__2.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__2.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__3.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__3.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__3.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__3.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.branchPred.lookups_0::NoBranch          0      0.00%      0.00% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::Return       480956     19.73%     19.73% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::CallDirect       481108     19.73%     39.46% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::CallIndirect          90      0.00%     39.47% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::DirectCond      1084756     44.49%     83.96% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::DirectUncond       330748     13.57%     97.53% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::IndirectCond          0      0.00%     97.53% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::IndirectUncond        60275      2.47%    100.00% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::total        2437933                       # Number of BP lookups (Count)
system.cpu.branchPred.squashes_0::NoBranch          0      0.00%      0.00% # Number of branches that got squashed (completely removed) as
an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::Return          777     13.27%     13.27% # Number of branches that got squashed (completely removed) as an
earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::CallDirect          953     16.27%     29.54% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::CallIndirect           61      1.04%     30.58% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::DirectCond         3222     55.02%     85.60% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::DirectUncond          614     10.48%     96.09% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::IndirectCond          0      0.00%     96.09% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::IndirectUncond          229      3.91%    100.00% # Number of branches that got squashed (completely
removed) as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::total          5856                       # Number of branches that got squashed (completely removed) as an
earlier branch was mispredicted. (Count)
system.cpu.branchPred.corrected_0::NoBranch          0      0.00%      0.00% # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::Return          0      0.00%      0.00% # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::CallDirect          187     22.32%     22.32% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::CallIndirect          30      3.58%     25.89% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::DirectCond          438     52.27%     78.16% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::DirectUncond          120     14.32%     92.48% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::IndirectCond          0      0.00%     92.48% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::IndirectUncond          63      7.52%    100.00% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::total          838                       # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.earlyResteers_0::NoBranch          0                       # Number of branches that got redirected after decode.
(Count)
system.cpu.branchPred.earlyResteers_0::Return          0                       # Number of branches that got redirected after decode.
(Count)
system.cpu.branchPred.earlyResteers_0::CallDirect          0                       # Number of branches that got redirected after decode.
(Count)
```

Re-Decode

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.008959                       # Number of seconds simulated (Second)
simTicks                                   8959261000                       # Number of ticks simulated (Tick)
finalTick                                  8959261000                       # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  114.68                         # Real time elapsed on the host (Second)
hostTickRate                               78124701                         # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                   676604                         # Number of bytes of host memory used (Byte)
simInsts                                   18490166                         # Number of instructions simulated (Count)
simOps                                     32899465                         # Number of ops (including micro ops) simulated (Count)
hostInstRate                                 161234                         # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                   286883                         # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                        1000                         # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage          1                         # Voltage in Volts (Volt)
system.cpu.numCycles                         8959262                         # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                              0.484542                         # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                              2.063805                         # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                    0                         # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                  0                         # Number of work items this cpu completed (Count)
system.cpu.instsAdded                      32933041                         # Number of instructions added to the IQ (excludes non-spec)
system.cpu.nonSpecInstsAdded                     77                         # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                     32967928                         # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                   91                         # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined             33647                          # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined          52776                          # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.squashedNonSpecRemoved                41                         # Number of squashed non-spec instructions that were removed
system.cpu.numIssuedDist::samples             8934941                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean               3.689776                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev              1.848636                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows                0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                    175925      1.97%      1.97% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                    867269      9.71%     11.68% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                   1441339     16.13%     27.81% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                   2021887     22.63%     50.44% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                   1638110     18.33%     68.77% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                   1054986     11.81%     80.58% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                   1037270     11.61%     92.19% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                    483910      5.42%     97.60% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                    214245      2.40%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                 0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                 0                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                 8                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::total               8934941                       # Number of insts issued each cycle (Count)
system.cpu.statFuBusy::No_OpClass                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntAlu                       1      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntMult                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntDiv                       0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatAdd                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCmp                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCvt                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMult                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMultAcc                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatDiv                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMisc                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatSqrt                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAdd                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAddAcc                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAlu                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCmp                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCvt                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMisc                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMult                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMultAcc                  0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMatMultAcc               0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShift                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShiftAcc                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdDiv                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSqrt                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAdd                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAlu                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCmp                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCvt                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatDiv                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMisc                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMult                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMultAcc             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMatMultAcc           0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatSqrt                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAdd                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAlu                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceCmp                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceAdd            0      0.00%      0.00% # attempts to use FU when none available (Count)
```

C. STATS.TXT FILES

```
system.cpu.statFuBusy::SimdFloatReduceCmp            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAdd                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAddAcc              0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAlu                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowCmp                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowCvt                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMisc                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMult                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMultAcc             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMatMultAcc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowShift               0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowShiftAcc             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowDiv                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowSqrt                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatAdd             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatAlu             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatCmp             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatCvt             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatDiv             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMisc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMult            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMultAcc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMatMultAcc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatSqrt            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceAdd            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceAlu            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceCmp            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatReduceAdd            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatReduceCmp            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAes                       0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAesMix                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash                  0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash2                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash2               0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma2                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma3                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdPredAlu                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::Matrix                        0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixMov                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixOP                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemRead                  292326     73.49%     73.49% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemWrite                 105443     26.51%     99.99% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemRead                 12      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemWrite                15      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IprAccess                     0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::InstPrefetch                  0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideMaskLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideMaskStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdStridedLoad               0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdStridedStore              0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdIndexedLoad               0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdIndexedStore              0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdWholeRegisterLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdWholeRegisterStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideSegmentedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideSegmentedStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdExt                       0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatExt                  0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdConfig                    0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statIssuedInstType_0::No_OpClass       570765      1.73%      1.73% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IntAlu      11906405     36.12%     37.85% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntMult       150055      0.46%     38.30% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv        210060      0.64%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd          205      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp            0      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt            0      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult           0      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc            0      0.00%     38.94% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatDiv            0      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMisc           0      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatSqrt           0      0.00%     38.94% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAdd       112178      0.34%     39.28% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAddAcc            0      0.00%     39.28% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdAlu      6984082     21.18%     60.46% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCmp             0      0.00%     60.46% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCvt       180042      0.55%     61.01% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMisc       120223      0.36%     61.37% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMult            0      0.00%     61.37% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMultAcc            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdMatMultAcc            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShift           0      0.00%     61.37% # Number of instructions issued per FU type, per thread (Count)
```

```
system.cpu.statIssuedInstType_0::SimdShiftAcc            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdDiv            0      0.00%     61.37% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdSqrt            0      0.00%     61.37% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatAdd            3      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatAlu            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCmp            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCvt            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatDiv            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMisc            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMult            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMultAcc            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMatMultAcc            0      0.00%     61.37% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatSqrt            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAdd            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAlu            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceCmp            0      0.00%     61.37% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceAdd            0      0.00%     61.37% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceCmp            0      0.00%     61.37% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowAdd         7830      0.02%     61.40% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowAddAcc            0      0.00%     61.40% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowAlu      2140895      6.49%     67.89% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowCmp            0      0.00%     67.89% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowCvt        60005      0.18%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMisc           23      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMult            0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMultAcc            0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMatMultAcc            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowShift            0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowShiftAcc            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowDiv            0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowSqrt            0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatAdd            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatAlu            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatCmp            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatCvt            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatDiv            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMisc            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMult            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMultAcc            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMatMultAcc            0      0.00%     68.07% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatSqrt            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceAdd            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceAlu            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceCmp            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatReduceAdd            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatReduceCmp            0      0.00%     68.07% # Number of instructions issued per FU type, per
thread (Count)
```

C. STATS.TXT FILES

```
system.cpu.statIssuedInstType_0::SimdAes            0      0.00%     68.07% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAesMix          0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash        0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash2       0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash      0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash2     0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma2       0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma3       0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdPredAlu         0      0.00%     68.07% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::Matrix            0      0.00%     68.07% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixMov         0      0.00%     68.07% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixOP          0      0.00%     68.07% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemRead      6328964     19.20%     87.27% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemWrite     3933163     11.93%     99.20% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMemRead       262427      0.80%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatMemWrite          603      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IprAccess           0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::InstPrefetch        0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideLoad          0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideStore         0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideMaskLoad          0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideMaskStore         0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdStridedLoad         0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdStridedStore        0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdIndexedLoad         0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdIndexedStore        0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdWholeRegisterLoad          0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdWholeRegisterStore         0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideFaultOnlyFirstLoad          0      0.00%    100.00% # Number of instructions issued per FU
type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideSegmentedLoad          0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideSegmentedStore          0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdExt         0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatExt          0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdConfig          0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::total       32967928                       # Number of instructions issued per FU type, per thread (Count)
system.cpu.issueRate                         3.679759                       # Inst issue rate ((Count/Cycle))
system.cpu.fuBusy                              397797                       # FU busy when requested (Count)
system.cpu.fuBusyRate                        0.012066                       # FU busy rate (busy events/executed inst) ((Count/Count))
system.cpu.instReDecoded                     11213636                       # Number of times an instruction has been Re-Decodificated
(Count)
system.cpu.intInstQueueReads                 55531367                       # Number of integer instruction queue reads (Count)
system.cpu.intInstQueueWrites                32118405                       # Number of integer instruction queue writes (Count)
system.cpu.intInstQueueWakeupAccesses        23073490                       # Number of integer instruction queue wakeup accesses (Count)
system.cpu.fpInstQueueReads                   1471757                       # Number of floating instruction queue reads (Count)
system.cpu.fpInstQueueWrites                   842148                       # Number of floating instruction queue writes (Count)
system.cpu.fpInstQueueWakeupAccesses           713585                       # Number of floating instruction queue wakeup accesses (Count)
system.cpu.vecInstQueueReads                 18265561                       # Number of vector instruction queue reads (Count)
system.cpu.vecInstQueueWrites                   7126                        # Number of vector instruction queue writes (Count)
system.cpu.vecInstQueueWakeupAccesses         9132101                       # Number of vector instruction queue wakeup accesses (Count)
system.cpu.intAluAccesses                    22926310                       # Number of integer alu accesses (Count)
system.cpu.fpAluAccesses                       735888                       # Number of floating point alu accesses (Count)
system.cpu.vecAluAccesses                     9132762                       # Number of vector alu accesses (Count)
system.cpu.numSquashedInsts                      2891                       # Number of squashed instructions skipped in execute (Count)
system.cpu.numSwp                                   0                       # Number of swp insts executed (Count)
system.cpu.timesIdled                             346                       # Number of times that the entire CPU went into an idle state and
unscheduled itself (Count)
system.cpu.idleCycles                           24321                       # Total number of cycles that the CPU has spent unscheduled due
to idling (Cycle)
system.cpu.MemDepUnit__0.insertedLoads        6548267                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.insertedStores       3935558                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.conflictingLoads      2028922                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__0.conflictingStores       553043                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__1.insertedLoads              0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.insertedStores             0                       # Number of stores inserted to the mem dependence unit. (Count)
```

Whetstone

Reference

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.672981                       # Number of seconds simulated (Second)
simTicks                                   672980534000                     # Number of ticks simulated (Tick)
finalTick                                  672980534000                     # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                      # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  8070.49                       # Real time elapsed on the host (Second)
hostTickRate                                83387864                       # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                    916072                       # Number of bytes of host memory used (Byte)
simInsts                                   1298923312                       # Number of instructions simulated (Count)
simOps                                     2376845064                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                  160947                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                    294511                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                         1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage           1                       # Voltage in Volts (Volt)
system.cpu.numCycles                        672980535                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                                0.518106                      # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                                1.930105                      # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                      0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                    0                       # Number of work items this cpu completed (Count)
system.cpu.instsAdded                       2376993071                      # Number of instructions added to the IQ (excludes non-spec)
system.cpu.nonSpecInstsAdded                      122                       # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                      2376940322                      # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                 2282                         # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined             148123                         # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined          233017                         # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.squashedNonSpecRemoved                62                         # Number of squashed non-spec instructions that were removed
system.cpu.numIssuedDist::samples           672942662                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean               3.532159                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev              1.786543                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows                0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                  35933427      5.34%      5.34% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                  64871035      9.64%     14.98% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                  74817135     11.12%     26.10% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                 156832002     23.31%     49.40% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                 144906442     21.53%     70.94% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                  97372129     14.47%     85.41% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                  66574758      9.89%     95.30% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                  27081822      4.02%     99.32% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                   4553912      0.68%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                 0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                 0                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                 8                       # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::total             672942662                       # Number of insts issued each cycle (Count)
system.cpu.statFuBusy::No_OpClass                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntAlu               147800740     98.16%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntMult                      0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntDiv                       0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatAdd                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCmp                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCvt                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMult                    0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMultAcc                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatDiv                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMisc                    0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatSqrt                    0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAdd                      0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAddAcc                   0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAlu                     11      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCmp                      0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCvt                      3      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMisc                     2      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMult                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMultAcc                  0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMatMultAcc               0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShift                    0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShiftAcc                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdDiv                      0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSqrt                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAdd                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAlu                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCmp                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCvt                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatDiv                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMisc                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMult                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMultAcc             0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMatMultAcc           0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatSqrt                0      0.00%     98.16% # attempts to use FU when none available (Count)
```

C. STATS.TXT FILES

```
system.cpu.statFuBusy::SimdReduceAdd                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAlu                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceCmp                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceAdd            0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceCmp            0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAes                      0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAesMix                   0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash                 0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash2                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash               0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash2              0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma2                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma3                0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdPredAlu                  0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::Matrix                       0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixMov                    0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixOP                     0      0.00%     98.16% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemRead                2554265      1.70%     99.86% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemWrite                 43243      0.03%     99.89% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemRead            172047      0.11%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemWrite            18      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IprAccess                    0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::InstPrefetch                 0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideMaskLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideMaskStore           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorStridedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorStridedStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIndexedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIndexedStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorUnitStrideFaultOnlyFirstLoad           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorWholeRegisterLoad           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorWholeRegisterStore          0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerArith            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatArith             0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatConvert            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerReduce           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorFloatReduce             0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorMisc                   0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorIntegerExtension           0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::VectorConfig                 0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statIssuedInstType_0::No_OpClass     29131803      1.23%      1.23% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IntAlu    1228837291     51.70%     52.92% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntMult     14700270      0.62%     53.54% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv            29      0.00%     53.54% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd    172891007      7.27%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp            0      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt            0      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult           0      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc            0      0.00%     60.82% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatDiv            0      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMisc           0      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatSqrt           0      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAdd            8      0.00%     60.82% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAddAcc            0      0.00%     60.82% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdAlu     30228111      1.27%     62.09% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCmp            0      0.00%     62.09% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCvt           92      0.00%     62.09% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMisc     36907674      1.55%     63.64% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMult            0      0.00%     63.64% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMultAcc           0      0.00%     63.64% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdMatMultAcc           0      0.00%     63.64% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShift            0      0.00%     63.64% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdShiftAcc            0      0.00%     63.64% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdDiv            0      0.00%     63.64% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdSqrt            0      0.00%     63.64% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatAdd     96793729      4.07%     67.71% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatAlu            0      0.00%     67.71% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCmp            0      0.00%     67.71% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCvt     12752284      0.54%     68.25% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatDiv     11408478      0.48%     68.73% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMisc           0      0.00%     68.73% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMult     73379417      3.09%     71.82% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMultAcc           0      0.00%     71.82% # Number of instructions issued per FU type, per thread
(Count)
```

```
system.cpu.statIssuedInstType_0::SimdFloatMatMultAcc            0      0.00%     71.82% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatSqrt       930000      0.04%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAdd            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAlu            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceCmp            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceAdd           0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceCmp           0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdAes            0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAesMix            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash2            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash2            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma2            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma3            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdPredAlu            0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::Matrix             0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixMov            0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixOP            0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemRead    260239508     10.95%     82.80% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemWrite    102123861      4.30%     87.10% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMemRead    213664841      8.99%     96.09% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatMemWrite     92951919      3.91%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IprAccess            0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::InstPrefetch            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideLoad           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideStore           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideMaskLoad           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideMaskStore           0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::VectorStridedLoad           0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorStridedStore           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIndexedLoad           0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorIndexedStore           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorUnitStrideFaultOnlyFirstLoad           0      0.00%    100.00% # Number of instructions issued per FU
type, per thread (Count)
system.cpu.statIssuedInstType_0::VectorWholeRegisterLoad           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorWholeRegisterStore           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIntegerArith           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorFloatArith           0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorFloatConvert           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorIntegerReduce           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorFloatReduce           0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorMisc            0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::VectorIntegerExtension           0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::VectorConfig           0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::total     2376940322                       # Number of instructions issued per FU type, per thread (Count)
system.cpu.issueRate                         3.531960                       # Inst issue rate ((Count/Cycle))
system.cpu.fuBusy                          150570329                        # FU busy when requested (Count)
system.cpu.fuBusyRate                        0.063346                       # FU busy rate (busy events/executed inst) ((Count/Count))
system.cpu.intInstQueueReads              3866160407                        # Number of integer instruction queue reads (Count)
system.cpu.intInstQueueWrites             1525167310                        # Number of integer instruction queue writes (Count)
system.cpu.intInstQueueWakeupAccesses     1525021523                        # Number of integer instruction queue wakeup accesses (Count)
system.cpu.fpInstQueueReads               1711235510                        # Number of floating instruction queue reads (Count)
system.cpu.fpInstQueueWrites               851979496                        # Number of floating instruction queue writes (Count)
```

C. STATS.TXT FILES

```
system.cpu.fpInstQueueWakeupAccesses        851898235                       # Number of floating instruction queue wakeup accesses (Count)
system.cpu.vecInstQueueReads                       0                       # Number of vector instruction queue reads (Count)
system.cpu.vecInstQueueWrites                      0                       # Number of vector instruction queue writes (Count)
system.cpu.vecInstQueueWakeupAccesses              0                       # Number of vector instruction queue wakeup accesses (Count)
system.cpu.intAluAccesses                 1639049744                       # Number of integer alu accesses (Count)
system.cpu.fpAluAccesses                   859329104                       # Number of floating point alu accesses (Count)
system.cpu.vecAluAccesses                          0                       # Number of vector alu accesses (Count)
system.cpu.numSquashedInsts                    12710                       # Number of squashed instructions skipped in execute (Count)
system.cpu.numSwp                                  0                       # Number of swp insts executed (Count)
system.cpu.timesIdled                            540                       # Number of times that the entire CPU went into an idle state and
unscheduled itself (Count)
system.cpu.idleCycles                          37873                       # Total number of cycles that the CPU has spent unscheduled due
to idling (Cycle)
system.cpu.MemDepUnit__0.insertedLoads      473914409                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.insertedStores     195084176                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.conflictingLoads    109242058                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__0.conflictingStores     2842560                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__1.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__1.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__2.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__2.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__2.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__2.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__3.insertedLoads             0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__3.insertedStores            0                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__3.conflictingLoads          0                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__3.conflictingStores         0                        # Number of conflicting stores. (Count)
system.cpu.branchPred.lookups_0::NoBranch          0      0.00%      0.00% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::Return      22212784     18.92%     18.92% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::CallDirect     22212380     18.92%     37.83% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::CallIndirect          126      0.00%     37.83% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::DirectCond     58913148     50.17%     88.01% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::DirectUncond      9020984      7.68%     95.69% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::IndirectCond          0      0.00%     95.69% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::IndirectUncond      5060628      4.31%    100.00% # Number of BP lookups (Count)
system.cpu.branchPred.lookups_0::total      117420050                       # Number of BP lookups (Count)
system.cpu.branchPred.squashes_0::NoBranch          0      0.00%      0.00% # Number of branches that got squashed (completely removed) as
an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::Return         2556     20.19%     20.19% # Number of branches that got squashed (completely removed) as an
earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::CallDirect         2196     17.35%     37.54% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::CallIndirect           77      0.61%     38.15% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::DirectCond         6728     53.16%     91.31% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::DirectUncond          550      4.35%     95.65% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::IndirectCond          0      0.00%     95.65% # Number of branches that got squashed (completely removed)
as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::IndirectUncond          550      4.35%    100.00% # Number of branches that got squashed (completely
removed) as an earlier branch was mispredicted. (Count)
system.cpu.branchPred.squashes_0::total         12657                       # Number of branches that got squashed (completely removed) as an
earlier branch was mispredicted. (Count)
system.cpu.branchPred.corrected_0::NoBranch          0      0.00%      0.00% # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::Return            1      0.09%      0.09% # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::CallDirect          210     19.20%     19.29% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::CallIndirect           51      4.66%     23.95% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::DirectCond          563     51.46%     75.41% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::DirectUncond          161     14.72%     90.13% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::IndirectCond          0      0.00%     90.13% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::IndirectUncond          108      9.87%    100.00% # Number of branches that got corrected but not yet
commited. Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected
branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.corrected_0::total         1094                       # Number of branches that got corrected but not yet commited.
Branches get corrected by decode or after execute. Also a branch misprediction can be detected out-of-order. Therefore, a corrected branch
might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu.branchPred.earlyResteers_0::NoBranch          0                       # Number of branches that got redirected after decode.
(Count)
system.cpu.branchPred.earlyResteers_0::Return          0                       # Number of branches that got redirected after decode.
(Count)
system.cpu.branchPred.earlyResteers_0::CallDirect          0                       # Number of branches that got redirected after decode.
(Count)
```

Re-Decode

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.657582                       # Number of seconds simulated (Second)
simTicks                                   657581850000                     # Number of ticks simulated (Tick)
finalTick                                  657581850000                     # Number of ticks from beginning of simulation (restored from
checkpoints and never reset) (Tick)
simFreq                                 1000000000000                      # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  12196.90                       # Real time elapsed on the host (Second)
hostTickRate                                 53913865                       # The number of ticks simulated per host second (ticks/s)
((Tick/Second))
hostMemory                                     918368                       # Number of bytes of host memory used (Byte)
simInsts                                    1298923310                      # Number of instructions simulated (Count)
simOps                                      2376845062                      # Number of ops (including micro ops) simulated (Count)
hostInstRate                                   106496                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                     194873                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu.numCycles                         657581851                      # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                                0.506251                      # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                                1.975303                      # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                       0                      # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                     0                      # Number of work items this cpu completed (Count)
system.cpu.instsAdded                       2376993991                      # Number of instructions added to the IQ (excludes non-spec)
system.cpu.nonSpecInstsAdded                       120                      # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued                      2376947147                      # Number of instructions issued (Count)
system.cpu.squashedInstsIssued                      80                      # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined                149043                      # Number of squashed instructions iterated over during squash;
mainly for profiling (Count)
system.cpu.squashedOperandsExamined             222364                      # Number of squashed operands that are examined and possibly
removed from graph (Count)
system.cpu.squashedNonSpecRemoved                   60                      # Number of squashed non-spec instructions that were removed
system.cpu.numIssuedDist::samples            657543623                      # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::mean                3.614889                      # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::stdev               2.088687                      # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::underflows                0      0.00%      0.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::0                  44588617      6.78%      6.78% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::1                  67630862     10.29%     17.07% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::2                 102421294     15.58%     32.64% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::3                 105496306     16.04%     48.69% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::4                 121854134     18.53%     67.22% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::5                  88910321     13.52%     80.74% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::6                  56754774      8.63%     89.37% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::7                  43610526      6.63%     96.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::8                  26276789      4.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::overflows                 0      0.00%    100.00% # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::min_value                 0                      # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::max_value                 8                      # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist::total             657543623                      # Number of insts issued each cycle (Count)
system.cpu.statFuBusy::No_OpClass                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntAlu                       3      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntMult                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IntDiv                       0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatAdd                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCmp                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatCvt                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMult                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMultAcc                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatDiv                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMisc                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatSqrt                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAdd                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAddAcc                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAlu                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCmp                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdCvt                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMisc                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMult                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMultAcc                  0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdMatMultAcc               0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShift                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShiftAcc                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdDiv                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSqrt                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAdd                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatAlu                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCmp                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatCvt                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatDiv                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMisc                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMult                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMultAcc             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatMatMultAcc           0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatSqrt                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAdd                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceAlu                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdReduceCmp                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatReduceAdd            0      0.00%      0.00% # attempts to use FU when none available (Count)
```

C. STATS.TXT FILES

```
system.cpu.statFuBusy::SimdFloatReduceCmp            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAdd                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAddAcc              0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowAlu                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowCmp                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowCvt                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMisc                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMult                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMultAcc             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowMatMultAcc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowShift               0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowShiftAcc             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowDiv                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowSqrt                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatAdd             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatAlu             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatCmp             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatCvt             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatDiv             0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMisc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMult            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMultAcc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatMatMultAcc            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatSqrt            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceAdd            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceAlu            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowReduceCmp            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatReduceAdd            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShadowFloatReduceCmp            0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAes                       0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdAesMix                    0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash                  0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha1Hash2                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash                0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdSha256Hash2               0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma2                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdShaSigma3                 0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdPredAlu                   0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::Matrix                        0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixMov                     0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MatrixOP                      0      0.00%      0.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemRead                 2610150     91.01%     91.01% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::MemWrite                  39965      1.39%     92.41% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemRead             217716      7.59%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::FloatMemWrite                20      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::IprAccess                     0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::InstPrefetch                  0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideLoad             0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideMaskLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideMaskStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdStridedLoad               0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdStridedStore              0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdIndexedLoad               0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdIndexedStore              0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdWholeRegisterLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdWholeRegisterStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideFaultOnlyFirstLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideSegmentedLoad            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdUnitStrideSegmentedStore            0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdExt                       0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdFloatExt                  0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statFuBusy::SimdConfig                    0      0.00%    100.00% # attempts to use FU when none available (Count)
system.cpu.statIssuedInstType_0::No_OpClass     29261376      1.23%      1.23% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IntAlu     720515536     30.31%     31.54% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntMult     14700260      0.62%     32.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv            37      0.00%     32.16% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd    151305453      6.37%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp            0      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt            0      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult           0      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc            0      0.00%     38.53% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatDiv            0      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMisc           0      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatSqrt           0      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAdd             4      0.00%     38.53% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAddAcc            0      0.00%     38.53% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdAlu    428850960     18.04%     56.57% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCmp             0      0.00%     56.57% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdCvt            74      0.00%     56.57% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMisc     34999726      1.47%     58.04% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMult            0      0.00%     58.04% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdMultAcc            0      0.00%     58.04% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdMatMultAcc            0      0.00%     58.04% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShift           0      0.00%     58.04% # Number of instructions issued per FU type, per thread (Count)
```

```
system.cpu.statIssuedInstType_0::SimdShiftAcc            0      0.00%     58.04% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdDiv            0      0.00%     58.04% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdSqrt            0      0.00%     58.04% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatAdd    108221854      4.55%     62.60% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatAlu            0      0.00%     62.60% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCmp            0      0.00%     62.60% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatCvt     11825306      0.50%     63.09% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatDiv     11408610      0.48%     63.57% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMisc            0      0.00%     63.57% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMult     64084670      2.70%     66.27% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMultAcc            0      0.00%     66.27% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatMatMultAcc            0      0.00%     66.27% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatSqrt       930000      0.04%     66.31% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAdd            0      0.00%     66.31% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceAlu            0      0.00%     66.31% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdReduceCmp            0      0.00%     66.31% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceAdd            0      0.00%     66.31% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatReduceCmp            0      0.00%     66.31% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowAdd            4      0.00%     66.31% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowAddAcc            0      0.00%     66.31% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowAlu    109578145      4.61%     70.92% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowCmp            0      0.00%     70.92% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowCvt           20      0.00%     70.92% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMisc      1907843      0.08%     71.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMult            0      0.00%     71.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMultAcc            0      0.00%     71.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowMatMultAcc            0      0.00%     71.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowShift            0      0.00%     71.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowShiftAcc            0      0.00%     71.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowDiv            0      0.00%     71.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowSqrt            0      0.00%     71.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatAdd     10155539      0.43%     71.43% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatAlu            0      0.00%     71.43% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatCmp            0      0.00%     71.43% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatCvt       927065      0.04%     71.46% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatDiv            0      0.00%     71.46% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMisc            0      0.00%     71.46% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMult      9291840      0.39%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMultAcc            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatMatMultAcc            0      0.00%     71.86% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatSqrt            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceAdd            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceAlu            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowReduceCmp            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatReduceAdd            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdShadowFloatReduceCmp            0      0.00%     71.86% # Number of instructions issued per FU type, per
thread (Count)
```

C. STATS.TXT FILES

```
system.cpu.statIssuedInstType_0::SimdAes            0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdAesMix          0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash        0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha1Hash2       0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash      0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdSha256Hash2     0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma2       0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdShaSigma3       0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdPredAlu         0      0.00%     71.86% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::Matrix              0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixMov           0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MatrixOP            0      0.00%     71.86% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemRead    260239633     10.95%     82.80% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::MemWrite    102124058      4.30%     87.10% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMemRead    213666869      8.99%     96.09% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::FloatMemWrite     92952265      3.91%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::IprAccess           0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::InstPrefetch        0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideLoad          0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideStore         0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideMaskLoad          0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideMaskStore         0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdStridedLoad         0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdStridedStore        0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdIndexedLoad         0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdIndexedStore        0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdWholeRegisterLoad          0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdWholeRegisterStore         0      0.00%    100.00% # Number of instructions issued per FU type, per
thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideFaultOnlyFirstLoad          0      0.00%    100.00% # Number of instructions issued per FU
type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideSegmentedLoad          0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdUnitStrideSegmentedStore          0      0.00%    100.00% # Number of instructions issued per FU type,
per thread (Count)
system.cpu.statIssuedInstType_0::SimdExt         0      0.00%    100.00% # Number of instructions issued per FU type, per thread (Count)
system.cpu.statIssuedInstType_0::SimdFloatExt          0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::SimdConfig           0      0.00%    100.00% # Number of instructions issued per FU type, per thread
(Count)
system.cpu.statIssuedInstType_0::total     2376947147                       # Number of instructions issued per FU type, per thread (Count)
system.cpu.issueRate                         3.614679                       # Inst issue rate ((Count/Cycle))
system.cpu.fuBusy                            2867854                        # FU busy when requested (Count)
system.cpu.fuBusyRate                       0.001207                        # FU busy rate (busy events/executed inst) ((Count/Count))
system.cpu.instReDecoded                    662031310                       # Number of times an instruction has been Re-Decodificated
(Count)
system.cpu.intInstQueueReads               2805463193                       # Number of integer instruction queue reads (Count)
system.cpu.intInstQueueWrites              1525150262                       # Number of integer instruction queue writes (Count)
system.cpu.intInstQueueWakeupAccesses      1072624244                       # Number of integer instruction queue wakeup accesses (Count)
system.cpu.fpInstQueueReads                1483995683                       # Number of floating instruction queue reads (Count)
system.cpu.fpInstQueueWrites                851974257                       # Number of floating instruction queue writes (Count)
system.cpu.fpInstQueueWakeupAccesses        741880727                       # Number of floating instruction queue wakeup accesses (Count)
system.cpu.vecInstQueueReads               1124846975                       # Number of vector instruction queue reads (Count)
system.cpu.vecInstQueueWrites                 24028                         # Number of vector instruction queue writes (Count)
system.cpu.vecInstQueueWakeupAccesses       562421172                       # Number of vector instruction queue wakeup accesses (Count)
system.cpu.intAluAccesses                  1046153208                       # Number of integer alu accesses (Count)
system.cpu.fpAluAccesses                    742106702                       # Number of floating point alu accesses (Count)
system.cpu.vecAluAccesses                   562293715                       # Number of vector alu accesses (Count)
system.cpu.numSquashedInsts                     13280                       # Number of squashed instructions skipped in execute (Count)
system.cpu.numSwp                                   0                       # Number of swp insts executed (Count)
system.cpu.timesIdled                             548                       # Number of times that the entire CPU went into an idle state and
unscheduled itself (Count)
system.cpu.idleCycles                           38228                       # Total number of cycles that the CPU has spent unscheduled due
to idling (Cycle)
system.cpu.MemDepUnit__0.insertedLoads      473915603                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.insertedStores     195083699                       # Number of stores inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__0.conflictingLoads    108673053                        # Number of conflicting loads. (Count)
system.cpu.MemDepUnit__0.conflictingStores      4573454                         # Number of conflicting stores. (Count)
system.cpu.MemDepUnit__1.insertedLoads              0                       # Number of loads inserted to the mem dependence unit. (Count)
system.cpu.MemDepUnit__1.insertedStores             0                       # Number of stores inserted to the mem dependence unit. (Count)
```