

Gem5 RVV

1 GEM5 安装运行

1.1 git

```
1 //官网
2 git clone https://github.com/gem5/gem5.git
3 //RVV Support
4 git clone https://github.com/rivosinc/gem5
5 commit version
6 //other Support
7 git clone https://github.com/nthu-pllab/RVV_gem5/tree/develop/src/cpu/vector_en
```

1.2 安装

https://www.gem5.org/documentation/learning_gem5/introduction/

https://www.gem5.org/documentation/general_docs/building

Dependencies

- **git**: gem5 uses git for version control.
- **gcc**: gcc is used to compiled gem5. **Version >=7 must be used**. We support up to gcc Version 12.
- **Clang**: Clang can also be used. At present, we support Clang 6 to Clang 14 (inclusive).
- **SCons**: gem5 uses SCons as its build environment. SCons 3.0 or greater must be used.
- **Python 3.6+**: gem5 relies on Python development libraries. gem5 can be compiled and run in environments using Python 3.6+.
- **protobuf 2.1+** (Optional): The protobuf library is used for trace generation and playback.
- **Boost** (Optional): The Boost library is a set of general purpose C++ libraries. It is a necessary dependency if you wish to use the SystemC implementation.

```
1 If compiling gem5 on Ubuntu 22.04, or related Linux distributions,
2 you may install all these dependencies using APT:
3 sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
4     libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perf-tools-dev \
```

1.3 编译

```
1 python3 `which scons` ./build/RISCV/gem5.opt -j50
```

gem5是一个由许多不同的模块组成的复杂软件系统，它需要在构建过程中将这些模块组合在一起，并且需要根据用户的需求来决定构建哪些模块。因此，需要使用SCons来进行构建，它能够根据SConstruct文件中的指示来进行编译、链接和安装，最终得到一个完整的gem5系统。

在构建完成后，可以使用build/RISCV/gem5.opt来执行模拟脚本。gem5.opt是gem5的优化版本，它能够提供更高的性能，更快地进行仿真。因此，构建完整的gem5系统是为了能够使用gem5.opt来进行仿真，而不是直接使用gem5.opt来构建gem5系统。

该命令的运行过程可以分为以下几个步骤：

- (1) 解析命令行参数
- (2) 加载构建环境
- (3) 加载构建规则
- (4) 构建gem5

编译命令解析

(1) scons

scons是一款开源的构建工具，它可以在后台自动化整个构建过程，提高构建效率。在gem5中，scons工具用于构建gem5的可执行文件，其中，build/RISCV/gem5.opt指定了构建的目标文件的位置和名称。gem5.opt是gem5的可执行文件，它包含了gem5的所有模块和功能。通过运行gem5.opt，可以启动gem5的模拟环境，并进行模拟实验。

scon是gem5中的一个工具，它用于解析和执行gem5的脚本文件。gem5的脚本文件是用python语言编写的，包含了用于构建和配置模拟系统的指令。scon工具可以读取脚本文件，并执行其中的指令，从而完成模拟系统的构建和配置。

scon构建环境的过程如下：

- (1) 使用scons工具读取gem5的脚本文件，并解析其中的指令。
- (2) 执行脚本文件中的指令，完成模拟系统的构建和配置。
- (3) 根据脚本文件中的指令，生成gem5的可执行文件。

(4) 运行gem5的可执行文件

该命令会**检查构建环境，并加载构建配置和构建脚本**。但是，该命令不会真正执行构建操作，只会准备好构建环境。如果要执行构建操作，需要在命令行中指定构建目标

(2) ISA

gem5支持常见的各种指令集：X86、ARM、RISCV、SPARC、POWER、MIPS。不同的指令集包含了每条指令执行的完整过程。当前gem5中可用于全系统模拟的ISA主要有ARM、X86和RISCV；

Ø RISCV

目前支持的拓展包括：

Ø ARM

模拟了Cortex-A9，支持Thumb，Thumb-2，VFPv3和NEON指令集。

Ø X86

模拟了64位x86 CPU，能够在SMP的配置模式下启动原始Linux kernel。

Ø SPARC

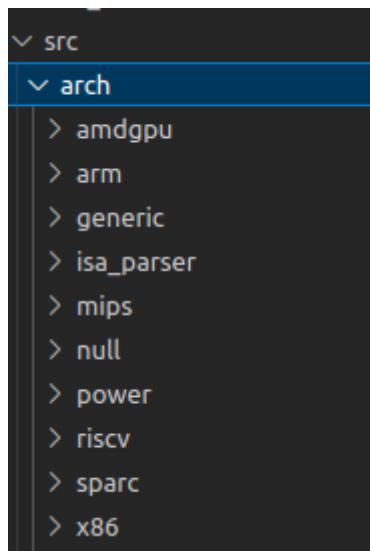
模拟了UltraSPARCT1处理器，能够启动Solaris操作系统。

Ø PowerPC

模拟了基于POWER ISAv2.06 B的32位处理器。

Ø MIPS

模拟了32位处理器核。



(3) gem5.*

关于gem5.debug, gem5.opt, gem5.fast, gem5.prof and gem5.perf的特点：

gem5.debug 关闭了优化。保证变量不会被优化掉，功能不会被意外的内联（inlined），以及控制流行为正常。该版本与gdb类的工具合作良好，然而关闭优化会造成该版本明显慢于其它版本。当

使用gdb或valgrind等工具并且不希望任何细节被模糊掉时应该选择该版本，否则建议选择其它版本。包含多种debug标志，在开发以及测试阶段使用，模拟速度慢。

gem5.opt 打开程序优化的同时保留了部分debug标志（例如，断言和DPRINTFs）。可以在试验阶段通过debug标志监控系统组件状态，模拟速度比debug类型得二进制要快。即该版本良好地平衡了模拟速度与调试观察，是所有环境中最优的版本。

gem5.fast 打开优化并关闭调试部分。编译优化程度最高，没有debug标志，最优的模拟速度，代价是不能进行运行时错误检查与调试输出。一般完成功能验证后，使用该二进制文件进行模拟实验。如果确信所有功能可以正确运行并想要获得峰值性能，建议使用该版本。

gem5.prof 类似于gem5.fast，但仍然保留了一些功能（instrumentation）可以用于gprof分析工具。该版本不常用，但可以用于找出gem5中应当被注意的部分以提升性能。不常用。

gem5.perf 同gem5.prof，但是instrumentation使用google perftools，允许被google-pprof分析。该分析版本是gem5.prof的补充，可能可以在所有基于Linux的系统中替换gem5.prof。不常用。

(4) -j* 选项启用多线程构建

目前遇到的问题：

(1) Python3 版本

已经安装的python 版本和需求的版本不一致

解决方法：

A. 重新卸载安装

B. 重新进行软链接

(2) 编译过程中，collection2: ld terminated with signal 9 错误

原因：交换空间不够

解决方法：参照以下链接

<https://www.jianshu.com/p/a59a46694dab>

Linux权限详解(chmod、600、644、700、711、755、777、4755、6755、7755)_chmod644_林20
的博客-CSDN博客

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUIDs as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
#   <type> <options> <dump> <pass>
#
# / was on /dev/sda1 during installation
UUID=0410845-afe0-0b40-8776-117ad740bc88 / ext4 errors=remount-ro 1
# /boot/efi vfat unask=0977 0 1
# /swapfile none swap sw 0 0
# /swapfile swap swap default 0 0
```

```
-rw----- 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo mkswap /swapfile^C
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 611 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ ls -l /swapfile
-rw---x--x 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 644 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ ls -l /swapfile
-rw-r--r-- 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo mkswap /swapfile
mkswap: /swapfile: 不安全的权限 0644, 建议使用 0600。
正在设置交换空间版本 1, 大小 = 30 GiB (32212250624 个字节)
无标签, UUID=25fd8c29-84fd-496f-a9e5-39dbf8ead00d
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 600 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo mkswap /swapfile
mkswap: /swapfile: 警告, 将擦除旧的 swap 签名。
正在设置交换空间版本 1, 大小 = 30 GiB (32212250624 个字节)
无标签, UUID=47d2c753-b162-448a-971b-b73153049345
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo swapon /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ free -m
总计 已用 空闲 共享 缓冲/缓存 可用
内存: 7879 1807 4449 340 1622 5459
交换: 30719 0 30719
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo vim /etc/fstab
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ 
```

3、停掉挂载的交换空间的文件

```
sudo swapoff /swapfile
```

4、删除交换空间文件

```
sudo rm /swapfile
```

5、创建新的挂载文件以及设置大小

```
sudo fallocate -l 20G /swapfile
```

可以ls -l /swapfile查看下创建的swapfile有没有相关权限

如果没有可以使用sudo chmod 600 /swapfile给文件赋予权限

6、将文件挂载到交换空间

```
sudo mkswap /swapfile
```

7、启动交换空间

```
sudo swapon /swapfile
```

可以free -m查看下，是否设置成功。完整的操作如下截图

8、最后使用sudo vim /etc/fstab将默认的/swapfile那行注释，在最后添加如下一行设置。

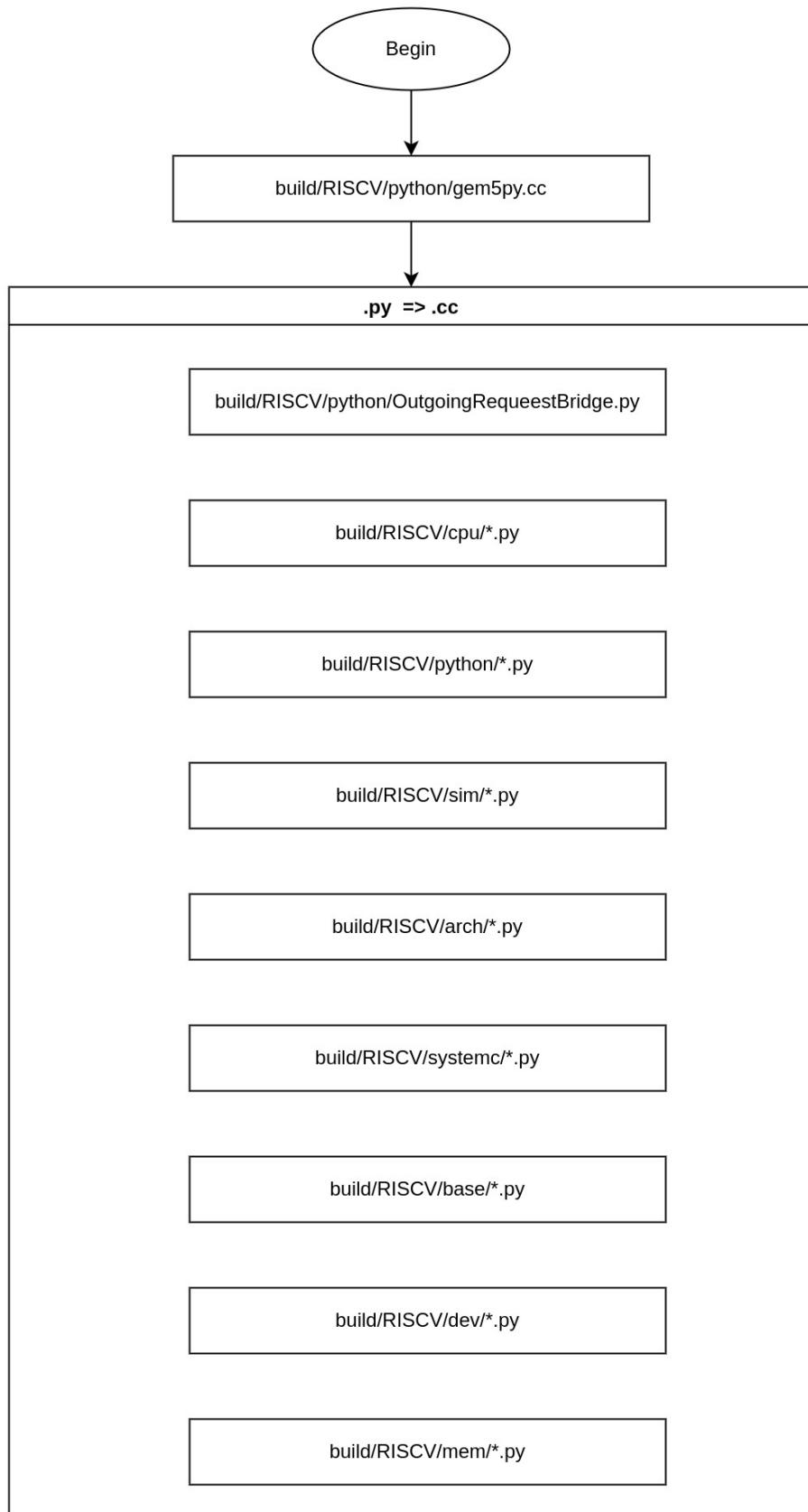
```
对不起, 请重试。
[sudo] yuanmiaomiao 的密码:
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ ls -l /swapfile
-rw-r--r-- 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 600 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ ls -l /swapfile
-rw----- 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo mkswap /swapfile^C
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 611 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ ls -l /swapfile
-rw---x--x 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 644 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ ls -l /swapfile
-rw-r--r-- 1 root root 32212254720 3月 24 09:06 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo mkswap /swapfile
mkswap: /swapfile: 不安全的权限 0644, 建议使用 0600。
正在设置交换空间版本 1, 大小 = 30 GiB (32212250624 个字节)
无标签, UUID=25fd8c29-84fd-496f-a9e5-39dbf8ead00d
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo chmod 600 /swapfile
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ sudo mkswap /swapfile
mkswap: /swapfile: 警告, 将擦除旧的 swap 签名。
正在设置交换空间版本 1, 大小 = 30 GiB (32212250624 个字节)
无标签, UUID=47d2c753-b162-448a-971b-b73153049345
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~$ 
```

```
root@yuanmiaomiao-ThinkCentre-M860t-D065:~/swap# 
```

```
root@yuanmiaomiao-ThinkCentre-M860t-D065:/swap# free -m
总计 已用 空闲 共享 缓冲/缓存 可用
内存: 7878 1843 1817 324 4218 5409
交换: 2047 940 1107
root@yuanmiaomiao-ThinkCentre-M860t-D065:/swap# ls
root@yuanmiaomiao-ThinkCentre-M860t-D065:/swap# sudo dd if=/dev/zero of=swapfile bs=16M count=1k
记录了 1024+0 的读入
记录了 1024+0 的写出
17179869184 字节 (17 GB, 16 GiB) 已复制, 104.128 s, 165 MB/s
root@yuanmiaomiao-ThinkCentre-M860t-D065:/swap# 
```

编译流程

```
Checking for compiler -fno-tree-nonheap-object support... yes
scons: done reading SConscript files.
scons: Building targets ...
[    CXX] RISCV/python/gem5py.cc -> .pyo
[    LINK] -> RISCV/gem5py
[EMBED PY] RISCV/sst/OutgoingRequestBridge.py -> .cc
[    CXX] RISCV/sst/OutgoingRequestBridge.py.cc -> .o
[    CXX] RISCV/sst/OutgoingRequestBridge.py.cc -> .pyo
[EMBED PY] RISCV/cpu/FuncUnit.py -> .cc
[    CXX] RISCV/cpu/FuncUnit.py.cc -> .pyo
[EMBED PY] RISCV/cpu/StaticInstFlags.py -> .cc
[    CXX] RISCV/cpu/StaticInstFlags.py.cc -> .pyo
[EMBED PY] RISCV/cpu/InstPBTrace.py -> .cc
[    CXX] RISCV/cpu/InstPBTrace.py.cc -> .pyo
[EMBED PY] RISCV/cpu/CheckerCPU.py -> .cc
[    CXX] RISCV/cpu/CheckerCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/BaseCPU.py -> .cc
[    CXX] RISCV/cpu/BaseCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/CPUTracers.py -> .cc
[    CXX] RISCV/cpu/CPUTracers.py.cc -> .pyo
[EMBED PY] RISCV/cpu/TimingExpr.py -> .cc
[    CXX] RISCV/cpu/TimingExpr.py.cc -> .pyo
[EMBED PY] RISCV/cpu/DummyChecker.py -> .cc
[    CXX] RISCV/cpu/DummyChecker.py.cc -> .pyo
[EMBED PY] RISCV/cpu/pred/BranchPredictor.py -> .cc
[    CXX] RISCV/cpu/pred/BranchPredictor.py.cc -> .pyo
[EMBED PY] RISCV/cpu/kvm/KvmVM.py -> .cc
[    CXX] RISCV/cpu/kvm/KvmVM.py.cc -> .pyo
[EMBED PY] RISCV/cpu/kvm/BaseKvmCPU.py -> .cc
[    CXX] RISCV/cpu/kvm/BaseKvmCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/minor/BaseMinorCPU.py -> .cc
[    CXX] RISCV/cpu/minor/BaseMinorCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/minor/MinorCPU.py -> .cc
[    CXX] RISCV/cpu/minor/MinorCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/BaseAtomicSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/BaseAtomicSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/BaseNonCachingSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/BaseNonCachingSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/BaseTimingSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/BaseTimingSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/BaseSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/BaseSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/AtomicSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/AtomicSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/NonCachingSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/NonCachingSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/TimingSimpleCPU.py -> .cc
[    CXX] RISCV/cpu/simple/TimingSimpleCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/simple/probes/SimPoint.py -> .cc
[    CXX] RISCV/cpu/simple/probes/SimPoint.py.cc -> .pyo
[EMBED PY] RISCV/cpu/trace/TraceCPU.py -> .cc
[    CXX] RISCV/cpu/trace/TraceCPU.py.cc -> .pyo
[EMBED PY] RISCV/cpu/testers/memtest/MemTest.py -> .cc
[    CXX] RISCV/cpu/testers/memtest/MemTest.py.cc -> .pyo
```



1.4 运行

运行命令

```
1 build/RISCV/gem5.opt configs/learning_gem5/part1/two_level.py
```

该命令的运行过程可以分为以下几个步骤：

- (1) 解析命令行参数
- (2) 加载配置脚本
- (3) 创建模拟对象
- (4) 创建模拟系统
- (5) 配置模拟参数
- (6) 启动模拟

配置参数时遇到的问题

```
1 --l1d_size=2kB  
2 //NOTE: k 是小写
```

```
root@17649972b2b1:/home/ymm/gem5/configs/learning_gem5/part1# ../../../../build/RISCV/gem5.opt two_level.py --l1d_size=2KB  
gem5 Simulator System. https://www.gem5.org  
gem5 is copyrighted software; use the --copyright option for details.  
  
gem5 version 22.1.0.0  
gem5 compiled Jan 13 2023 21:06:10  
gem5 started Jan 15 2023 02:03:09  
gem5 executing on 17649972b2b1, pid 44  
command line: ../../../../build/RISCV/gem5.opt two_level.py --l1d_size=2KB  
  
ValueError: cannot convert '2KB' to memory size  
Error setting param L1DCache.size to 2KB  
  
At:  
  build/RISCV/python/m5/util/convert.py(144): convert  
  build/RISCV/python/m5/util/convert.py(166): toNum  
  build/RISCV/python/m5/util/convert.py(182): toInteger  
  build/RISCV/python/m5/util/convert.py(190): toBinaryInteger  
  build/RISCV/python/m5/util/convert.py(263): toMemorySize  
  build/RISCV/python/m5/params.py(802): __init__  
  build/RISCV/python/m5/params.py(222): convert  
  build/RISCV/python/m5/SimObject.py(884): __setattr__  
  /home/ymm/gem5/configs/learning_gem5/part1/caches.py(106): __init__  
  two_level.py(91): <module>  
  build/RISCV/python/m5/main.py(597): main
```

(4) AttributeError

```
root@17649972b2b1:/home/ymm/gem5/configs/learning_gem5/part1# ../../../../build/RISCV/gem5.opt two_level.py --l1d_size=2kB  
gem5 Simulator System. https://www.gem5.org  
gem5 is copyrighted software; use the --copyright option for details.  
  
gem5 version 22.1.0.0  
gem5 compiled Jan 13 2023 21:06:10  
gem5 started Jan 15 2023 02:04:07  
gem5 executing on 17649972b2b1, pid 45  
command line: ../../../../build/RISCV/gem5.opt two_level.py --l1d_size=2kB  
  
AttributeError: Class RiscvInterrupts has no parameter pio  
  
At:  
  build/RISCV/python/m5/SimObject.py(912): __setattr__  
  two_level.py(116): <module>  
  build/RISCV/python/m5/main.py(597): main
```

```
1 Build ISA != "x86", Remove PIO + Interrupt  
2 PIO + Interrupt Special Requirement  
3 //NOTE: 只有X86 需要PIO 中断
```

```
__pycache__ caches.py libutil simple-arm.py simple-rtscv.py simple.py two_level.py
root@17649972b2b1:/home/ymm/gem5/configs/learning_gem5/part1# ../../../../build/RISCV/gem5.opt two_level.py
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.1.0.0
gem5 compiled Jan 13 2023 21:06:10
gem5 started Jan 15 2023 06:05:45
gem5 executing on 17649972b2b1, pid 70
command line: ../../../../build/RISCV/gem5.opt two_level.py

AttributeError: Class RiscvInterrupts has no parameter pio

At:
  build/RISCV/python/m5/SimObject.py(912): __setattr__
    two_level.py(116): <module>
  build/RISCV/python/m5/main.py(597): main
root@17649972b2b1:/home/ymm/gem5/configs/learning_gem5/part1# vi two_level.py
```

```
# create the interrupt controller for the CPU
system.cpu.createInterruptController()
## x86 special requirement: PIO + Interrupt Port connect to memory bus
#if m5.defines.buildEnv[`TARGET_ISA`] == "x86":
#    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
```

```
root@17649972b2b1:/home/ymm/gem5/configs/learning_gem5/part1# ../../build/RISCV/gem5.opt two_level.py --l1d_size=4kB
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.1.0.0
gem5 compiled Jan 13 2023 21:06:10
gem5 started Jan 15 2023 17:57:41
gem5 executing on 17649972b2b1, pid 87
command line: ../../build/RISCV/gem5.opt two_level.py --l1d_size=4kB

Global frequency set at 100000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
build/RISCV/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
)
0: system.remote_gdb: listening for remote gdb on port 7000
Beginning simulation!

Two Level Cache

build/RISCV/sim/simulate.cc:192: info: Entering event queue @ 0. Starting simulation...
build/RISCV/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
    Returning '/home/ymm/gem5/tests/test-progs/hello/bin/riscv/linux/hello'
build/RISCV/sim/mem_state.cc:443: info: Increasing stack size by one page.
Hello, world!
```

1.5 vscode 配置

为了方便debug, 可以用vscode 配置gem5

Vscode 配置gem5 方法如下：

```
1 Vscode debug environments with gem5
2 step1 which scons => /usr/bin/scons
3 step2 cp /usr/bin/scons to_gem5_path
4 step3 edit launch.json
5 {
6     "version": "0.2.0",
7     "configurations": [
8         {
9             "name": "Python: 当前文件"
10        }
11    ]
12}
```

```

10         "type": "python",
11         "request": "launch",
12         "program": "${file}",
13         "console": "integratedTerminal",
14         "justMyCode": false,
15         "args": ["./build/RISCV/gem5.opt"]
16     }
17 ]
18 }
19 step4 in current top file debug => add breakpoint; first_step

```

2 GEM5 基础知识

2.1 Gem5 代码结构

[gem5学习5——源代码文件结构作用介绍_ivy_reny的博客-CSDN博客](#)

注 常用的模块已经标红

build-opt:主要是构建不同构建配置的默认设置文件。例如：X86， RISCV

SConstruct:构建系统的一部分。作为build-opt目录

configs:用python编写的模拟配置脚本。目录中的文件提供一些基本的预打包功能来帮助简化编写配置，并提供了一些示例，可以直接使用，也可在自己的脚本中使用。

ext:gem5的依赖，并不是gem5的一部分。构建gem5所需的不太常见的外部软件包。

src:gem5的源代码

arch:ISA实现

generic:在其他ISA中使用的通用文件。

isa_parser.py:解析ISA描述的解析器。

ISA directories:与给定ISA关联的文件

OS directories:通常在SE模式下用于支持ISA/OS组合的代码。

isa:ISA描述文件。

base:可能对其他项目有用的常规数据结构。

loader:用于加载二进制文件和读取符号表的代码。

stats:用于保存统计信息并将数据写入文件或数据库的代码。

vnc:VNC支持

cpu:CPU模型

dev:设备模型

ISA directories:特定于给定ISA的设备模型

doxygen:Doxygen模板和输出

kern:特定于操作系统但与体系结构无关的代码（例如，数据结构的类型）。

OS directories:特定于给定模拟操作系统的代码。

mem:内存系统模型和基础架构

cache:在经典内存系统中实现缓存模型的代码。

ruby:实现ruby内存模型的代码。

protocol:Ruby协议定义。

slicc:slicc编译器。

python:用于配置和更高级别功能的Python代码。

sim:实现基本的基本模拟器功能的代码。

system:用于模拟系统的固件或引导程序之类的低级软件

alpha:Alpha控制台和palcode。

arm:一个简单的ARM引导程序。

tests:与gem5的回归测试有关的文件

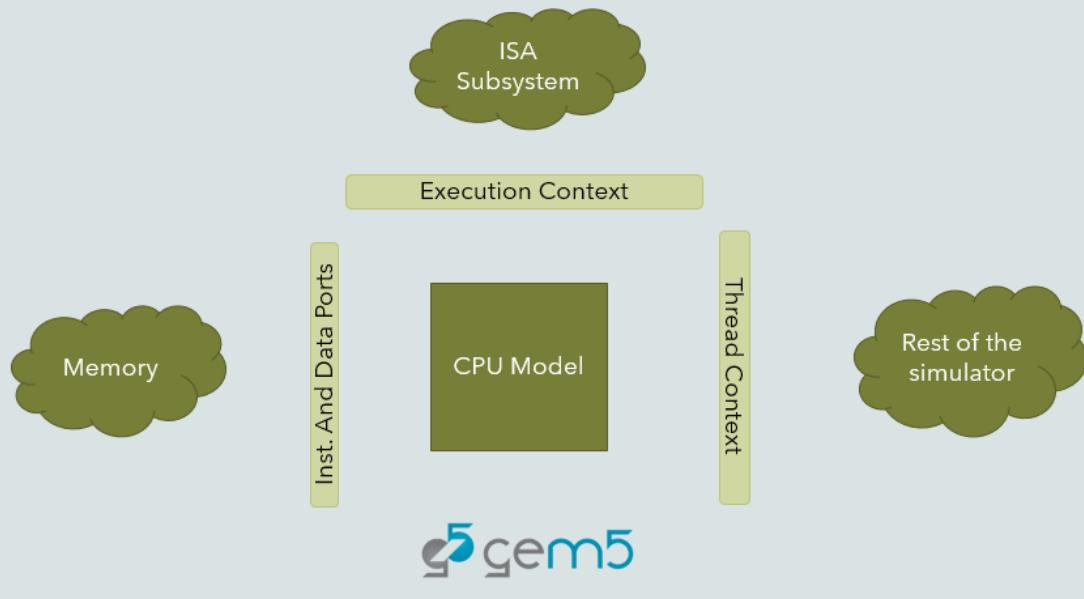
configs: 用于测试的常规配置。

test-progs:每个ISA的“Hello world”二进制文件，其他二进制文件则分别下载。

quick,long:快速和长期回归输入，参考输出以及测试特定的配置文件（按测试排列）。

util:实用程序脚本，程序和有用的文件不是gem5二进制文件的一部分，但在使用gem5时通常很有用。

Interaction of CPU model with other parts of gem5



A screenshot of a file explorer window showing the directory structure of the gem5 source code. The tree view shows:

- GEM5 (root)
 - > build_opts
 - > build_tools
 - > configs
 - > ext
 - > include
 - > site_scons
- < src (selected)
 - > arch
 - > base
 - > cpu
 - > dev
 - > doc
 - > doxygen
 - > gpu-compute
 - > kalm
 - > kern
 - > learning_gem5
 - > mem
 - > proto
 - > python
 - > sim
 - > sst
 - > systemc
 - Doxyfile
 - SConscript
 - system
 - tests
 - util
- & .git-blame-ignore-revs
- & .gitignore
- & .mailmap
- & CODE-OF-CONDUCT.md
- & CONTRIBUTING.md
- & COPYING
- & LICENSE
- ! MAINTAINERS.yaml
- ① README
- ▼ RELEASE-NOTES.md
- ◆ SConstruct
- & .TERMINAL
- > OUTLINE

构建的gem5模拟环境里包含了一个虚拟的计算机系统，用于模拟真实的计算机系统。

具体来说，gem5模拟环境里可能包含以下组件：

处理器模型：模拟真实处理器的指令集、流水线、执行引擎等功能，用于执行虚拟机器指令。

内存模型：模拟真实内存的存储空间、访问速度、缓存机制等特性，用于提供处理器运行时所需的数据和程序。

输入/输出设备模型：模拟真实输入/输出设备的功能、速度、协议等特性，用于实现处理器与外部世界的交互。

操作系统模型：模拟真实操作系统的内核、进程、线程、文件系统等功能，用于管理处理器、内存、输入/输出设备等组件，并实现对应用程序的支持。

2.2 Gem5 模拟对象

EventManager 类	负责调度、管理、执行事件。EventManager 类是对 EventQueue 类的包装，SimObject 对象中所有的事件实际都由 EventQueue 队列管理。该队列以二维的单链表的形式管理着所有事件，事件以触发时间点从近到远排列。
Serializable 类	负责对象的序列化。SimObjects 可通过 <code>SimObject::serializeAll()</code> 函数自动完成序列化，写入到自己的 sections 中。Serializable 类根据 SimObject 类对象的名字以及对象间的包含关系，帮助用户构建起了层次化的序列化模型，并使用该模型完成 SimObject 的序列化，以 ini 文件格式输出
Drainable 类	负责 drain 对象。DrainManager 类以单例的方式管理整个模拟器的 drain 过程。只有系统中所有的对象都被 drained，才能开始序列化、更改模型等操作。完成后需要使用 <code>DrainManager::resume()</code> 函数将系统回归到正常运行状态。
statistics::Group 类	负责运行过程中统计、管理数据。Group 对象之间可组成树状层次，从而反应出 SimObject 对象间的树状层次。
Name 类	负责给 SimObject 起名。

(1) SimObject

gem5采用了一种结构清晰的模块化设计，使得各个模拟组件之间能够良好地协作。在gem5中，SimObject是一个基础类，用于创建和管理模拟对象。

- `SimObject::init()` 只有当 C++ SimObject 对象被创建，且所有接口都被连上后，该函数会被调用
- `SimObject::regStats()` 本是 Group 类的回调函数，用于设置需要复杂参数的统计信息。
(例如，分布)
 - `SimObject::initState()` 若 SimObject 不是从检查点恢复时，需要调用该函数。该函数标记了状态的初始点，仅在冷启动时会被使用，让 simobject 回到初始状态。
 - `SimObject::loadState()` 若从检查点恢复，调用该函数。其默认实现是调用 `unserialize()` 函数。因为从检查点恢复的过程就如同序列化后，装载之前保存的状态的过程。
- `SimObject::resetStats()` 重置统计数据。

- `SimObject::startup()` 是模拟前的最终的启动函数。此时所有状态都已初始化（包括未序列化的状态，如果有的话，如 `curTick()` 的值），因此这是调度初始事件的合适时间点。
- `Drainable::drainResume()` 如果从检查点恢复。

以上这些函数（除 `loadState()` 有默认非空的实现）都需要继承 `SimObject` 类的派生类来实现。`SimObject` 类只是搭建了模拟对象的运行框架，规定了对象的运行步骤。

参考链接 <https://dingfen.github.io/cpp/2022/03/13/gem5-3.html>

(2) SimObjects

`SimObjects` 则是 `gem5` 中用于管理模拟对象的组件。`SimObjects` 组件维护了一个模拟对象的列表，并且为模拟对象提供了一些基本的管理功能。例如，可以使用 `SimObjects` 组件查询模拟对象列表中的对象、添加新的模拟对象或者删除已有的模拟对象等。通常，`gem5` 中的模拟对象都会继承自 `SimpleObject` 类，并且通过 `SimObjects` 组件来进行管理。

(3) SimpleObject

`SimpleObject` 是 `SimObject` 类的一个派生类，它在 `SimObject` 类的基础上增加了一些额外的功能，例如支持静态和动态的内存分配等。它提供了一些简单的功能，可用于创建和管理模拟中的对象。`SimpleObject` 类提供了一些基本的接口，例如可以用来控制对象的生命周期、获取对象的名称和控制对象的 debug 信息等。通常，在 `gem5` 中创建新的模拟对象时，都会从 `SimpleObject` 类进行派生，以便获得这些基本的功能。

(4) SimpleObject/SimObject/SimObjects 的关系

`SimObject`、`SimObjects` 和 `SimpleObject` 之间存在着一种父子关系，即 `SimObject` 是 `SimpleObject` 的基类，`SimObjects` 则用于管理由 `SimObject` 和 `SimpleObject` 派生而来的模拟对象。通常，在 `gem5` 中创建新的模拟对象时，都会从 `SimpleObject` 类进行派生，并通过 `SimObjects` 组件进行

1. 如何构建模拟对象

`gem5` 中的模拟对象是由两部分组成的：

一部分是 `python` 代码，用于定义模拟对象的接口和参数；

另一部分是 `C++` 代码，用于实现模拟对象的具体功能。为了让模拟对象能够正常工作，需要将这两部分代码进行注册，以便它们之间能够相互通信。

注册模拟对象的过程如下：

Step1. 创建一个 `python` 文件，用于定义模拟对象的接口和参数。该文件的格式如下：

```

1 class <SimObjectName>(SimObject):
2     type = '<SimObjectName>'
3     cxx_class = '<C++ClassName>'
4     cxx_header = '<C++HeaderFile>'
```

Step2. 创建一个 `C++` 文件，用于实现模拟对象的具体功能。该文件的格式如下：

```
1 #include "<C++HeaderFile>"  
2 namespace gem5 {  
3     class <C++ClassName> : public SimObject {  
4         public: <C++ClassName>(<const SimObjectName>Params &p);  
5     };  
6 } // namespace gem5
```

Step3. 创建一个SConScript文件，用于指定模拟对象的配置信息。该文件的格式如下：

```
1 Import('*')  
2 SimObject('<SimObjectName>.py', sim_objects=['<SimObjectName>'])  
3 Source('<C++ClassName.cc'')
```

Step4. 重构Gem5模拟环境

```
1 scons build/RISCV/gem5.opt
```

Step5: 在py文件中创建指定的模拟对象，并将其添加到gem5的模拟环境中

```
1 import m5  
2 from m5.objects import *  
3 root = Root(full_system = False)  
4 root.hello = SimObjectName()  
5 m5.instantiate()  
6 print("Beginning simulation!")  
7 exit_event = m5.simulate()  
8 print('Exiting @ tick {} because {}'.format(m5.curTick(), exit_event.getCause()))
```

Step6: 使用以下命令构建和运行gem5模拟环境：

```
1 build/RISCV/gem5.opt configs/learning_gem5/part2/run_hello.py(your_name_file.py)
```

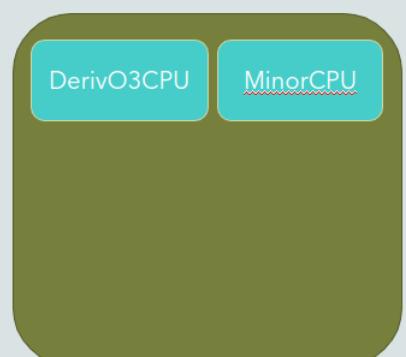
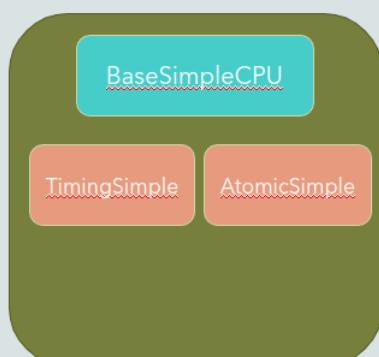
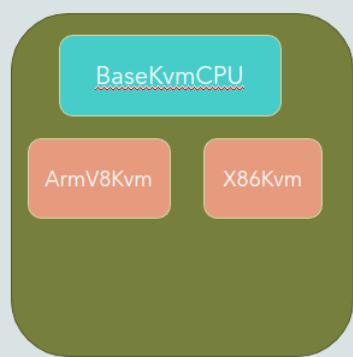
2.3 Gem5 CPU 模型

05-gem5-cpus-tutorial 2.pptx

- (1) AtomicSimpleCPU,
- (2) TimingSimpleCPU,
- (3) O3CPU,
- (4) MinorCPU,
- (5) KvmCPU

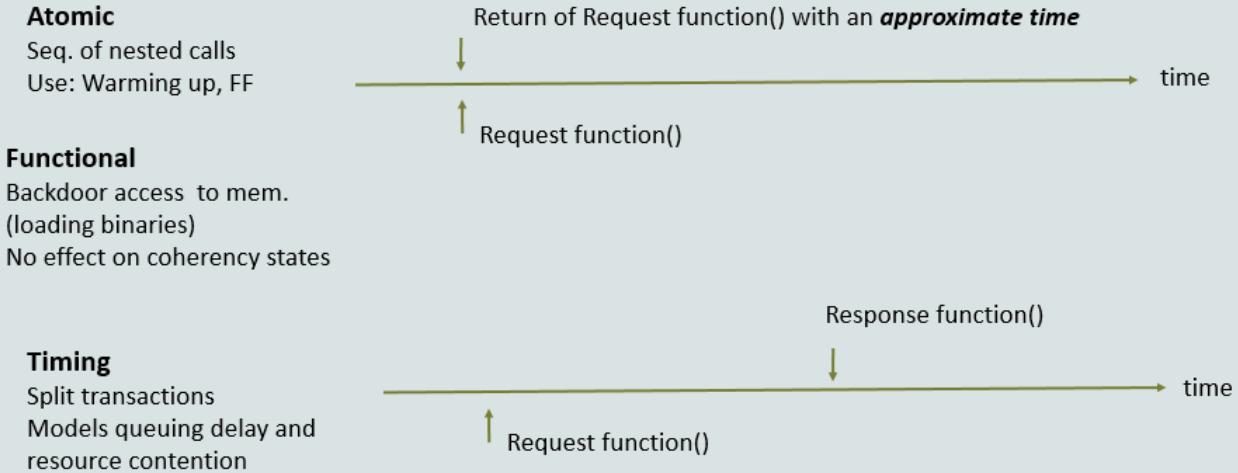
Summary of gem5 CPU Models

BaseCPU



gem5

Memory Access Types in gem5



gem5

AtomicSimpleCPU

Uses **Atomic** memory accesses
no resource contentions or queuing delay
Mostly used for fast-forwarding and warming of caches

TimingSimpleCPU

Uses **Timing** memory accesses
Execute non-memory operations in one cycle
Models the timing of memory accesses in detail

gem5

O3CPU (Out of Order CPU Model)

Timing memory accesses

execute-in-execute semantics

Time buffers between stages

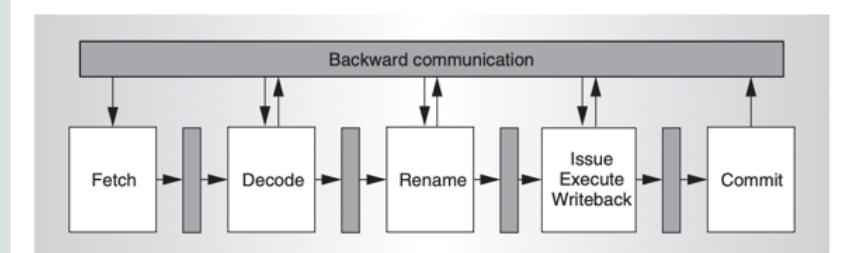


Figure 2. O3CPU pipeline. Shaded boxes represent time buffers.



O3CPU Model Parameters (very configurable)

`src/cpu/o3/BaseO3CPU.py`

Inter-stage delay params

```
decodeToFetchDelay = Param.Cycles(1, "Decode to fetch delay")
renameToFetchDelay = Param.Cycles(1, "Rename to fetch delay")
issueToFetchDelay = Param.Cycles(1, "Issue/Execute/Writeback to fetch "
    "delay")
commitToFetchDelay = Param.Cycles(1, "Commit to fetch delay")
```

stage width params

```
fetchWidth = Param.Unsigned(8, "Fetch width")
fetchBufferSize = Param.Unsigned(64, "Fetch buffer size in bytes")
fetchQueueSize = Param.Unsigned(32, "Fetch queue size in micro-ops "
    "per-thread")
```

configurable
buffer sizes



O3CPU Model Parameters (very configurable)

`src/cpu/o3/BaseO3CPU.py`

LQ/SQ Buffers

```
LQEntries = Param.Unsigned(32, "Number of load queue entries")
SQEntries = Param.Unsigned(32, "Number of store queue entries")
LSQDepCheckShift = Param.Unsigned(4,
    "Number of places to shift addr before check")
LSQCheckLoads = Param.Bool(True,
    "Should dependency violations be checked for "
    "loads & stores or just stores")
store_set_clear_period = Param.Unsigned(250000,
    "Number of load/store insts before the dep predictor "
    "should be invalidated")
LFSTSize = Param.Unsigned(1024, "Last fetched store table size")
SSITSize = Param.Unsigned(1024, "Store set ID table size")

numRobs = Param.Unsigned(1, "Number of Reorder Buffers");

numPhysIntRegs = Param.Unsigned(256,
    "Number of physical integer registers")
numPhysFloatRegs = Param.Unsigned(256, "Number of physical floating point "
    "registers")
numPhysVecRegs = Param.Unsigned(256, "Number of physical vector "
    "registers")
numPhysVecPredRegs = Param.Unsigned(32, "Number of physical predicate "
    "registers")
# most ISAs don't use condition-code regs, so default is 0
numPhysCCRegs = Param.Unsigned(0, "Number of physical cc registers")
numIQEntries = Param.Unsigned(64, "Number of instruction queue entries")
numROBEntries = Param.Unsigned(192, "Number of reorder buffer entries")
```

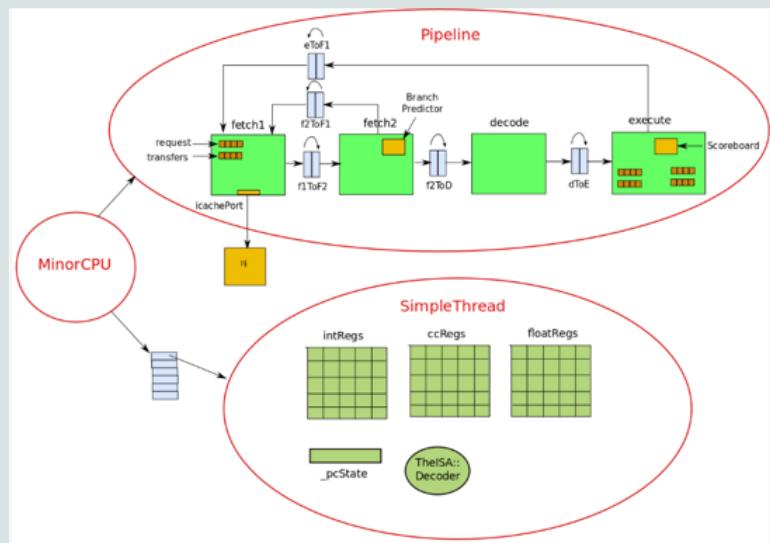
Number of Physical regs.

ROB size

Reservation station size

MinorCPU

[1]



[1] <https://nitish2112.github.io/post/gem5-minor-cpu/>

SimpleCPU	Functional, In-Order CPU Models
O3CPU	Out-of-order

Minor	
KVM	Kernel-based virtual machine

```

115     response_latency = 20
116     mshrs = 20
117     tgts_per_mshr = 12
118
119     SimpleOpts.add_option('--l2_size', help='L2 Cache size')
120
121     def __init__(self, opts=None):
122         super(L2Cache, self).__init__()
123         if not opts or not opts.l2_size:
124             return
125         self.size = opts.l2_size
126
127     def connectCPUSSideBus(self, bus):
128         self.cpu_side = bus.mem_side_ports
129
130     def connectMemSSideBus(self, bus):
131         self.mem_side = bus.cpu_side_ports
132

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~/code/gem5\$ build_opts CODE-OF-CONDUCT.md CONTRIBUTING.md examples/configs COPYING infrastructure

- 系统模型：SE (System-callemulation) 、 FS (Full System)

- 存储模型：Classic、Ruby

Material to use

gem5-bootcamp-env/materials/using-gem5/05-cpu-models/
 cpu-models.py
 IntMM/
 finished-material/

2.4 Gem5 内存模型

gem5: MSI example cache protocol

Outline

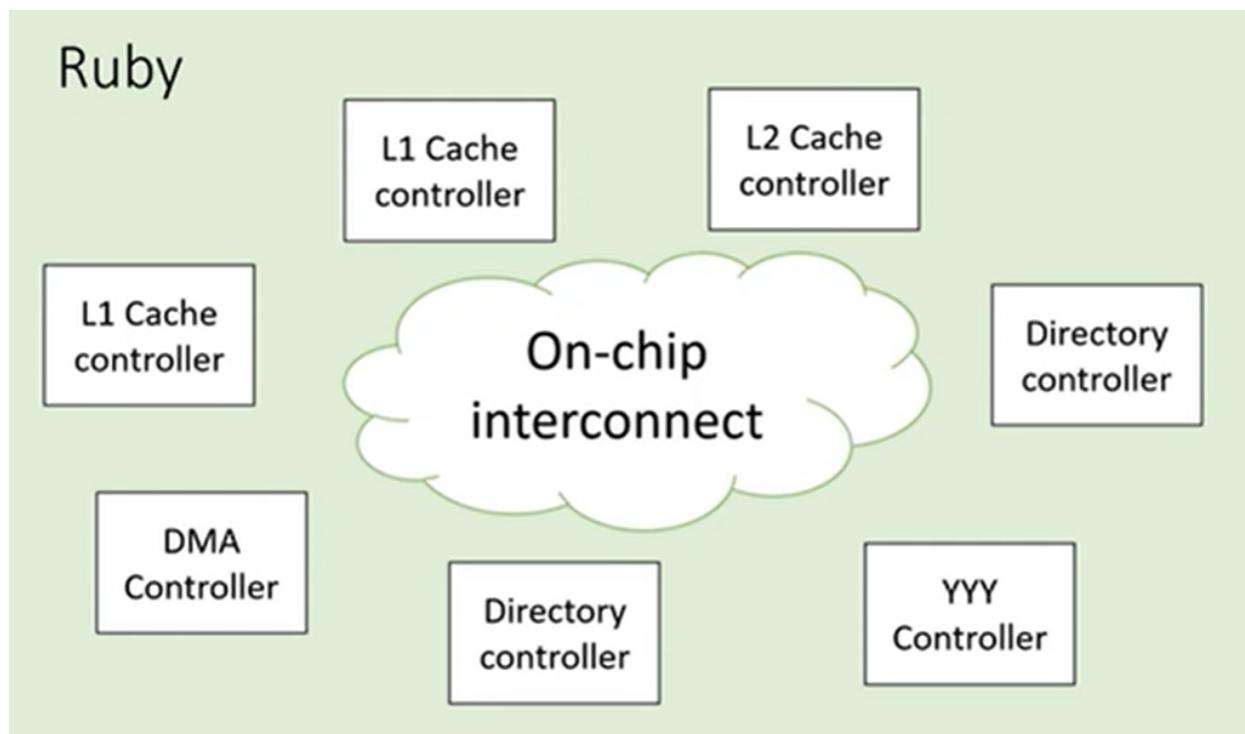
Ruby overview

SLICC controller details

Configuring Ruby

A few other small things

(1) Ruby



Ruby实现了一个更细节的内存子系统的模拟模型。它建模了inclusive/exclusive cache，包含多种替换策略、一致性协议、互连网络、DMA和内存控制器，以及多种用于初始化内存请求和处理响应的排序器。

特点：

关注点分离。例如，一致性协议规范与替换策略和缓存索引映射是分开的，网络拓扑结构与实现也是分开指定的。

丰富的可配置性。几乎所有影响内存层次结构的功能和时间都可以控制。

快速的原型。使用一种高级规范语言SLICC来指定各种控制器的功能。

(2) SLICC + 一致性协议

SLICC: Specification Language for Implementing Cache Coherence, 是一种特定领域的语言，用于指定缓存一致性协议。

Controller Models

Implemented in SLICC

Code for controllers is “generated” via SLICC compiler

SLICC: Specification Language including Cache Coherence

```
Your branch is up-to-date with 'origin/master'.
[jlp@amarillo ~/tutorial/gem5 <master>
$ scons -j8 build/X86/gem5.opt PROTOCOL=MSI
```

缓存一致性协议就像一个状态机，SLICC用于指定状态机的行为。

3. 与协议无关的内存组件

(1) 排序器 (Sequencer)

Sequencer类负责为内存子系统（包括cache和片外存储）装载来自处理器的load/store/atomic访存请求。每一个在内存子系统中完成的访存请求都会通过Sequencer向处理器发送响应。

系统中模拟的每个硬件线程（或核心）都有一个Sequencer。

(2) cache memory

Cache state machine outline

Parameters:

Cache memory: Where the data is stored

Message buffers: Sending/receiving messages from network

State declarations: The stable and transient states

Event declarations: State machine events that will be “triggered”

Other structures and functions: Entries, TBEs, get/setState, etc.

In ports: Trigger events based on incoming messages

Actions: Execute *single* operations on cache structures

Transitions: Move from *state* to *state* and execute *actions*

Cache memory



See src/mem/ruby/structures/CacheMemory

Stores the cache data (Entry) and the state (State)

cacheProbe() returns the replacement address if cache is full

Important!
Must call setMRU on each access!

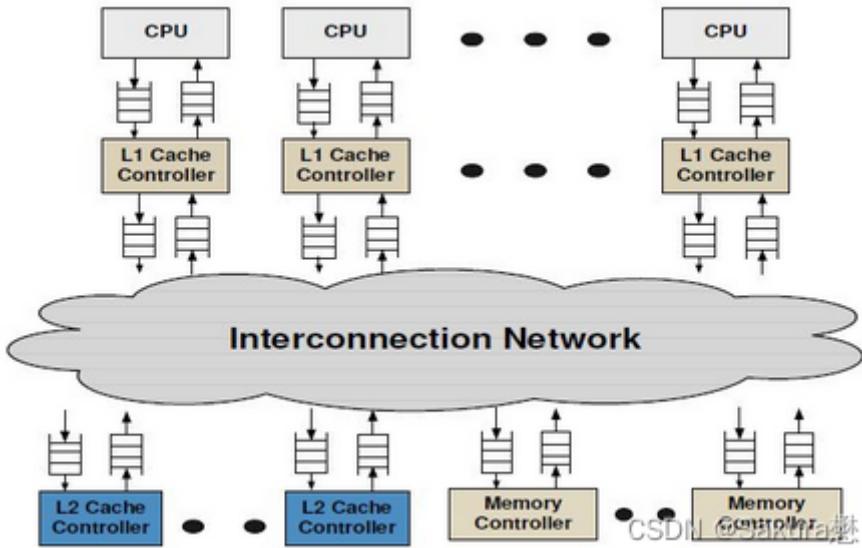
(3) 替换策略

(4) 内存控制器

内存控制器负责模拟和服务在模拟系统的所有片上cache中错过的任何请求。还建模了DRAM禁令争用、DRAM刷新和DRAM缓冲的close-page策略。

4. 互连网络

互连网络将内存层次结构的各个组件(缓存、内存、DMA控制器)连接在一起。



关键组件：

拓扑

路由

流量控制

路由器微架构

5. Ruby中一个内存请求的完整过程

Message buffers

Declaring is confusing!



```
MessageBuffer * requestToDir, network="To", virtual_network="0", vnet_type="request";
MessageBuffer * forwardFromDir, network="From", virtual_network="1", vnet_type="forward";
```

peek(): Get the head message

pop(): Remove head message (don't forget this!)

isReady(): Is there a message?

recycle(): Move the head to the tail (better perf., but unrealisitic)

stallAndWait(): Move (stalled) message to different buffer



State declarations



```
state_declaration(State, desc="Cache states") {  
    I,      AccessPermission:Invalid, desc="Not present/Invalid";  
  
    // States moving out of I  
    IS_D,   AccessPermission:Invalid, desc="Invalid, moving to S, waiting for data";  
    IM_AD,  AccessPermission:Invalid, desc="Invalid, moving to M, waiting for acks and data";  
    IM_A,   AccessPermission:Busy,     desc="Invalid, moving to M, waiting for acks";  
  
    S,      AccessPermission:Read_Only, desc="Shared. Read-only, other caches may have the block";  
  
    . . .  
}
```

State declarations



```
state_declaration(State, desc="Cache states") {  
    I,      AccessPermission:Invalid, desc="Not present/Invalid";  
  
    // States moving out of I  
    IS_D,   AccessPermission:Invalid, desc="Invalid, moving to S, waiting for data";  
    IM_AD,  AccessPermission:Invalid, desc="Invalid, moving to M, waiting for acks and data";  
    IM_A,   AccessPermission:Busy,     desc="Invalid, moving to M, waiting for acks";  
  
    S,      AccessPermission:Read_Only, desc="Shared. Read-only, other caches may have the block";  
  
    . . .  
}
```

AccessPermission: Used for functional accesses

IS_D -> Read: “Invalid transitioning to Shared waiting for Data”

Event declarations



```
enumeration(Event, desc="Cache events") {  
    // From the processor/sequencer/mandatory queue  
    Load,          desc="Load from processor";  
    Store,         desc="Store from processor";  
  
    // Internal event (only triggered from processor requests)  
    Replacement,   desc="Triggered when block is chosen as victim";  
  
    // Forwarded request from other cache via dir on the forward network  
    FwdGetS,        desc="Directory sent us a request to satisfy Gets. ";  
                    "We must have the block in M to respond to this. ";  
    FwdGetM,        desc="Directory sent us a request to satisfy GetM. ";  
    . . .
```

Other structures and functions



Entry: Declare the data structure for each entry

Block data, block state, sometimes others (e.g., tokens)

TBE/TBETable: Transient Buffer Entry

Like an MSHR, but not exactly (allocated more often)

Holds data for blocks in *transient* states

get/set State, AccessPermissions, functional read/write

Required to implement AbstractController

Usually just copy-paste from examples

+ 关注

Ports/Message buffers



Not gem5 ports!

out_port: “Rename” the message buffer and declare message type

in_port: Much of the SLICC “magic” here.

Called every cycle

Look at head message

Trigger events

In ports



```
in_port(forward_in, RequestMsg, forwardToCache) {  
    if (forward_in.isReady(clockEdge())) {  
        peek(forward_in, RequestMsg) {  
            Entry cache_entry := getCacheEntry(in_msg.addr);  
            TBE tbe := TBEs[in_msg.addr];  
            if (in_msg.Type == CoherenceRequestType:GetS) {  
                trigger(Event:FwdGetS, in_msg.addr, cache_entry, tbe);  
            } else  
            . . .
```

In ports

```
in_port(forward_in)           Weird syntax!
                                Automatically populates "in_msg"
                                in the following block
if (forward_in.isReady(clockEdge())) {
    peek(forward_in, RequestMsg) {
        Entry cache_entry := getCacheEntry(in_msg.addr);
        TBE tbe := TBEs[in_msg.addr];
        if (in_msg.Type == CoherenceRequestType:GetS) {
            trigger(Event:FwdGetS, in_msg.addr, cache_entry, tbe);
        } else
    ...
}
```

Trigger() looks for a *transition*. It also ensures resources available.

Actions

```
action(sendGetM, "gM", desc="Send GetM to the directory") {
    enqueue(request_out, RequestMsg, 1) {
        out_msg.addr := address;
        out_msg.Type := CoherenceRequestType:GetM;
        out_msg.Destination.add(mapAddressToMachine(address,
                                                      MachineType:Directory));
        out_msg.MessageSize := MessageSizeType:Control;
        out_msg.Requestor := machineID;
    }
}
```

内存请求在ruby中的生命周期

- (1) gem5中发出的请求(gem5包的形式)通过 **RubyPort::recvTiming** (位于 src/mem/ruby/system/RubyPort.hh/cc中) 接口进入ruby的范围。 rubyport实例化的数目和gem5硬件线程数目是一致的
- (2) RubyPort将这个请求转换为RubyRequest object的形式，将其发送到 **Sequencer::makeRequest** 接口 (Sequencer类本身是RubyPort类的派生类) 。
- (3) 当请求到达Sequencer::makeRequest 接口后，会给他分配资源并执行统计，将请求在 **mandatory queue** (变量名为m_mandatory_q_ptr) 中排队，最后将他发送到ruby cache 层次结构中。
- (4) L1 cache的控制器从mandatory queue中取出请求并查找L1 cache，进行必要的一致性状态转换，如果未命中需要通过类 **MessageBuffer** 将请求送到下一级cache。一旦请求命中，则通过 MessageBuffer类层层递进，向上推进。其中不同类型的cache控制器和组件通过MessageBuffer类 (src/mem/ruby/buffers/MessageBuffer.cc/hh) 进行交流沟通。

MessageBuffers还充当片上互连的一致性消息的入口点。

- (5) 一旦请求在L1 cache中命中， **L1 cache controller**会利用 **Sequencer object**对象的 readCallback或者writeCallback方法通知对应的Sequencer object。

(6) 最后，Sequencer清除相应请求的统计信息，然后调用RubyPort :: ruby_hit_callback方法,将结果发送到gem5的线程或者核心中

目录结构

```
1 src/mem/
```

protocols: 一致性协议的SLICC规范

slicc: 实现SLICC解析器和代码生成器

common: 常用的数据结构，如：地址(带位操作方法)、直方图、数据块

filters: 各种Bloom过滤器(来自GEMS的陈旧代码)

network: 互连实现、示例拓扑规范、网络功率计算、用于连接控制器的消息缓冲区

profiler: cache事件、内存控制器事件的分析

recorder: cache预热和访问跟踪记录

slicc_interface: 消息数据结构，各种映射(例如：地址到目录节点)，实用函数(例如：地址和int之间的转换，将地址转换为cache line地址)

structures: 协议无关的内存组件—CacheMemory, DirectoryMemory

system: glue组件—Sequencer, RubyPort, RubySystem

4. 配置Ruby

Ruby config scripts

1. Instantiate the controllers

Here is where you pass all of the options from the *.sm file

2. Create a *Sequencer* for each CPU

More details in a moment

3. Create and connect all of the network routers

[如何在gem5中配置ruby协议_gem5 ruby_Sakura懋的博客-CSDN博客](#) (参考链接)

2.5 Gem5 输出

```
1 ./build/ISA_Name/gem5.opt configs/example/se.py --help
```

1. 运行gem5时，输出文件

运行gem5时，会根据配置文件和程序的不同，生成不同的文件。常见的文件包括：

- **config.ini**: gem5的主配置文件，包含了各种参数和设置，用于定义模拟器如何运行。
- **config.json**: JSON格式的配置文件，用于覆盖config.ini中的设置。
- **stats.txt**: 存储模拟器运行时的性能统计信息，用于分析和研究系统性能行为。
- system.prv: 存储模拟器运行时的指令和事件信息，可以用于进一步分析系统的行为。
- system.pim: 存储模拟器运行时的状态和内存信息，可以用于进一步分析系统的行为。
- system.map: 存储模拟器运行时的地址映射信息，可以用于进一步分析系统的行为。
- system.ptx: 存储模拟器运行时的指令流信息，可以用于进一步分析系统的行为。

这些文件的作用各不相同

The slide has a title 'Understanding gem5 output' and the Gem5 logo in the top right. It shows a terminal session:

```
> ls m5out
config.ini  config.json  stats.txt
```

Below the terminal, there are three callout boxes:

- config.ini**: Dumps all of the parameters of all SimObjects. This shows exactly what you simulated.
- config.json**: Same as config.ini, but in json format.
- stats.txt**: Detailed statistic output. Each SimObject defines and updates statistics. They are printed here at the end of simulation.

At the bottom left is the name 'JCDAVIS' and at the bottom right is the number '21'.

2. Tick

在gem5中，**tick是指模拟器模拟的时间单位。它用于指定模拟器每次迭代执行的时间间隔。**这个时间间隔是可以通过配置文件来设定的。例如，如果在config.ini中将tick设定为1ns，那么每次迭代模拟器就会执行1ns的时间。

tick在gem5中扮演着重要的角色，因为它可以用来控制模拟的精度和速度。如果将**tick设定的更小，那么模拟器就会更加精确，但同时也会更慢**。反之，如果将tick设定的更大，那么模拟器就会更快，但同时也会更不精确。因此，在使用gem5时，需要合理地调整tick的值，以达到最佳的模拟精度和速度。

3. stats.txt中的输出含义

```

1 ----- Begin Simulation Statistics -----
2 simSeconds                      0.000057
3 # Number of seconds simulated (Second)
4 simTicks                         57467000
5 # Number of ticks simulated (Tick)
6 finalTick                        57467000
7 # Number of ticks from beginning of simulation
8 (restored from checkpoints and never reset) (Tick)
9 simFreq                           10000000000000
10 # The number of ticks per simulated second ((Tick/Second))
11 hostSeconds                      0.03
12 # Real time elapsed on the host (Second)
13 hostTickRate                     2295882330
14 # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
15 hostMemory                       665792
16 # Number of bytes of host memory used (Byte)
17 simInsts                          6225
18 # Number of instructions simulated (Count)
19 simOps                            11204
20 # Number of ops (including micro ops) simulated (Count)
21 hostInstRate                      247382
22 # Simulator instruction rate (inst/s) ((Count/Second))
23 hostOpRate                        445086
24 # Simulator op (including micro ops) rate (op/s) ((Count/Second))

```

- **simSeconds** 表示模拟的秒数。这个值表示模拟的时钟的累计时间。
- **simTicks** 表示模拟的时钟滴答数，可以看作时钟周期。滴答（tick）是模拟中使用的最小时间单位。
- **finalTick** 表示模拟从开始到结束的时钟滴答数，即总的时钟周期数。注意，这个值不会重置，即使模拟从检查点恢复。
- **simFreq** 表示模拟时钟的频率。它表示每秒钟有多少滴答（时钟周期）。
- **hostSeconds** 指在宿主机上，表示真实时间中的秒数。这个值表示主机上实际流逝的时间。
- **hostTickRate** 指在宿主机上，表示每秒钟模拟的时钟周期数。它反映了模拟的速度。
- **hostMemory** 表示模拟器在宿主机上使用了多少字节的内存。
- **simInsts**：指模拟器模拟了多少条指令。指令（instruction）是计算机中执行的最小单位。这个数值越大，表明模拟器的性能越好。
- **simOps**：指模拟器模拟了多少个操作（包括微操作）。这个数值越大，表明模拟器的性能越
- **hostInstRate**：指模拟器的指令模拟速率，单位是指令/秒。这个数值越大，表明模拟器的性能越好。

- **hostOpRate**：指模拟器的操作模拟速率（包括微操作），单位是操作/秒。这个数值越大，表明模拟器的性能越好。

4. SimObjects' statistics 统计信息含义

(1) 以CPU为例

```

1 system.clk_domain.clock           1000
2 # Clock period in ticks (Tick)
3 system.clk_domain.voltage_domain.voltage   1
4 # Voltage in Volts (Volt)
5 system.cpu.numCycles            57467
6 # Number of cpu cycles simulated (Cycle)
7 system.cpu.numWorkItemsStarted    0
8 # Number of work items this cpu started (Count)
9 system.cpu.numWorkItemsCompleted  0
10 # Number of work items this cpu completed (Count)
11 system.cpu.dcache.demandHits::cpu.data      1941
12 # number of demand (read+write) hits (Count)
13 system.cpu.dcache.demandHits::total        1941
14 # number of demand (read+write) hits (Count)
15 system.cpu.dcache.overallHits::cpu.data     1941
16 # number of overall hits (Count)

```

上面这些输出是一组性能指标，每个指标都对应着一个数字。比如，

system.clk_domain.clock 表示时钟周期，它对应的数字是 1000，这意味着时钟的频率是 1000 ticks/second。另一个例子是 **system.cpu.dcache.demandHits::cpu.data**，它表示 **CPU 数据高速缓存的需求命中次数**，它对应的数字是 1941，这意味着在模拟过程中，CPU 数据高速缓存有 1941 次需求命中。用户可以比较不同模型的吞吐量、延迟、命中率等参数来评估模型的好坏。

(2) 以存储器为例

```

1 system.mem_ctrl.bytesReadWrQ          0
2 # Total number of bytes read from write queue (Byte)
3 system.mem_ctrl.bytesReadSys         23168
4 # Total read bytes from the system interface side (Byte)
5 system.mem_ctrl.bytesWrittenSys       0
6 # Total written bytes from the system interface side (Byte)
7 system.mem_ctrl.avgRdBWSys          403153113.96105593
8 # Average system read bandwidth in Byte/s ((Byte/Second))
9 system.mem_ctrl.avgWrBWSys          0.00000000
10 # Average system write bandwidth in Byte/s ((Byte/Second))
11 system.mem_ctrl.totGap              57336000
12 # Total gap between requests (Tick)
13 system.mem_ctrl.avgGap              158386.74

```

```

14 # Average gap between requests ((Tick/Count))
15 system.mem_ctrl.requestorReadBytes::cpu.inst      14656
16 # Per-requestor bytes read from memory (Byte)
17 system.mem_ctrl.requestorReadBytes::cpu.data      8512
18 # Per-requestor bytes read from memory (Byte)
19 system.mem_ctrl.requestorReadRate::cpu.inst 255033323.472601681948
20 system.mem_ctrl.requestorReadRate::cpu.data 148119790.488454252481
21 system.mem_ctrl.requestorReadAccesses::cpu.inst    229
22 # Per-requestor read serviced memory accesses (Count)
23 system.mem_ctrl.requestorReadAccesses::cpu.data      133
24 # Per-requestor read serviced memory accesses (Count)
25 system.mem_ctrl.requestorReadTotalLat::cpu.inst     6234000
26 # Per-requestor read total memory access latency (Tick)
27 system.mem_ctrl.requestorReadTotalLat::cpu.data     4141000
28 # Per-requestor read total memory access latency (Tick)
29 system.mem_ctrl.requestorReadAvgLat::cpu.inst      27222.71
30 # Per-requestor read average memory access latency ((Tick/Count))
31 system.mem_ctrl.requestorReadAvgLat::cpu.data      31135.34
32 # Per-requestor read average memory access latency ((Tick/Count))
33 system.mem_ctrl.dram.bytesRead::cpu.inst      14656
34 # Number of bytes read from this memory (Byte)

```

上面这些输出是一组性能指标，每个指标都对应着一个数字。

例如，**system.mem_ctrl.avgRdBWSys** 表示系统接口侧读取的**平均带宽**，它对应的数字是 403153113.96105593，这意味着在模拟过程中，系统接口侧的读取带宽平均为 403153113.96105593 Bytes/second。

system.mem_ctrl.requestorReadAccesses::cpu.data，它表示 CPU 数据请求方的读取访问次数，它对应的数字是 133，这意味着 CPU 数据请求方在模拟过程中有 133 次读取访问。

5. 分析统计信息

gem5的输出可以通过分析**统计信息和调试信息**来评估模拟计算机系统的性能。用户可以通过比较不同模型的统计信息和调试信息来判断哪种模型的性能更好。

在分析统计信息时，用户可以**比较不同模型的吞吐量、延迟、命中率等参数**，以评估模型的性能。例如，用户可以通过比较不同模型的吞吐量来评估模型的处理能力；可以通过比较不同模型的延迟来评估模型的响应能力；可以通过比较不同模型的命中率来评估模型的存储能力。在分析调试信息时，用户可以**比较不同模型的调度策略、内存管理策略等参数**，以评估模型的性能。

2.5 Gem5 事件驱动

事件驱动的核心思想是，模拟器会按照事件的发生顺序来处理事件。

模拟器会根据事件的时间戳来决定处理哪个事件，并且会按照事件的顺序来处理它们。

在 gem5 中，EventFunctionWrapper 是一个用于封装事件处理函数的类模板。事件处理函数是 gem5 中用于响应特定事件的函数，例如计时器超时、中断请求或网络数据到达。

EventFunctionWrapper 类模板提供了一种方便的方法来封装这些处理函数，使它们能够被 gem5 框架调用。EventFunctionWrapper 类模板接受事件处理函数的参数类型和返回值类型作为模板参数，并提供了一个 operator() 符号重载函数，用于调用封装的事件处理函数。例如，下面是一个使用 EventFunctionWrapper 类模板封装事件处理函数的示例代码：

```
1 void handleEvent(int event_id)
2 {
3     // Code to handle the event with the given event_id.
4 }
5
6 EventFunctionWrapper<void, int> event_handler(&handleEvent);
```

1. 创建一个简单的事件

(1) 定义一个新的 C++ 类，并继承自 SimObject 抽象基类

```
1 class HelloObject : public SimObject
2 {
3     private:
4     void processEvent();
5
6     EventFunctionWrapper event;
7
8     public:
9     HelloObject(HelloObjectParams *p);
10
11    void startup();
12 };
```

(2) 实现必要的构造函数和初始化函数

```
1 HelloObject::HelloObject(HelloObjectParams *params) :
2     SimObject(params), event([this]{processEvent();}, name())
3 {
4     DPRINTF(Hello, "Created the hello object\n");
5 }
```

这代码定义了 gem5::HelloObject 类的一个构造函数。它接受一个指针类型的参数，类型为 HelloObjectParams，表示该构造函数需要一个 HelloObjectParams 类型的参数来构造新的 HelloObject 对象。构造函数首先调用父类的构造函数，然后使用 C++11 的 lambda 表达式定义了一个事件处理函数，该函数将在特定事件发生时被调用。该事件处理函数将调用 gem5::HelloObject 类中的名为 processEvent 的成员函数。最后，构造函数使用 gem5 框架的 DPRINTF 宏打印了一条消息。

代码解析：

```
1 event([this]{processEvent();}, name())
```

在上面的代码中，event 是一个 gem5::HelloObject 类中的成员变量。它是 gem5 框架中的 Event 类型，用于表示一个事件。Event 类提供了一个构造函数，可以接受一个事件处理函数和一个事件名称作为参数。

在上面的代码中，event 的构造函数调用了两个参数：一个 lambda 表达式和一个调用 gem5::HelloObject 类的 name 函数的结果。lambda 表达式定义了一个匿名函数，该函数在被调用时将执行 gem5::HelloObject 类中的 processEvent 成员函数。name 函数返回 gem5::HelloObject 对象的名称，它将作为该事件的名称来使用。因此，event 是一个指向一个名称为 gem5::HelloObject 对象的名称的匿名函数的指针。该匿名函数的实现由 lambda 表达式指定，在被调用时将执行 gem5::HelloObject 类中的 processEvent 成员函数。

详解event中的this的含义

在 C++ 中，this 指针指向一个对象的实例本身。它是一个隐式参数，由编译器在需要时自动插入到函数或方法中。在上面的代码中，[this]{processEvent();} 表示定义一个捕获 this 指针的匿名函数。捕获 this 指针的匿名函数允许在函数体内访问该对象的成员变量和成员函数。在上面的代码中，processEvent 是 gem5::HelloObject 类的成员函数，匿名函数的函数体中调用了该成员函数。

```
1 event([this]{processEvent();}, name())
```

因此，event 是一个指向一个捕获 this 指针的匿名函数的指针。该匿名函数的实现是执行 gem5::HelloObject 类的 processEvent 成员函数。在匿名函数中使用 this 指针可以方便地访问 gem5::HelloObject 对象的成员变量和成员函数。

event是指向一个捕获 this 指针的匿名函数的指针说明

在 C++ 中，this 指针指向一个对象的实例本身。它是一个隐式参数，由编译器在需要时自动插入到函数或方法中。在上面的代码中，event 是一个指向一个捕获 this 指针的匿名函数的指针，意思是 event 是一个指针，它指向一个匿名函数，该匿名函数捕获了指向 gem5::HelloObject 类实例的 this 指针。这里的“捕获 this 指针”指的是，在匿名函数的函数体中可以使用 this 指针来访问

gem5::HelloObject 类实例的成员变量和成员函数。在上面的代码中，lambda 表达式中的 processEvent 方法就是通过 this 指针访问的。因此，event 是指向一个捕获 this 指针的匿名函数的指针，意思是 event 指向一个能够访问 gem5::HelloObject 类实例的成员变量和成员函数的匿名函数。

综上，event([this]{processEvent();}, name()) 的作用

它在创建 HelloObject 的时候被执行。这条语句创建了一个名为 event 的对象，它被初始化为一个调用 processEvent() 方法的匿名函数。

在 C++11 中，可以使用 lambda 表达式创建匿名函数。该语句使用了 lambda 表达式，并将 this 作为参数传递给了匿名函数。这样，匿名函数中就可以访问 HelloObject 类的成员变量和成员函数。

另外，在 event 的构造函数中，还会传入 name() 方法的返回值。这个方法会返回当前 HelloObject 对象的名称。这样，当 event 对象被调用时，它就可以使用这个名称来执行一些特定的操作。

这条语句创建了一个名为 event 的对象，该对象持有一个调用 processEvent() 方法的匿名函数，并且可以使用当前 HelloObject 对象的名称来执行一些特定的操作。

(3) 实现 processEvent 函数

gem5 中的 processEvent 是一个函数，用于处理事件队列中的事件。该函数会按照事件的时间戳来决定处理哪个事件，并且会按照事件的顺序来处理它们。例如，如果一个事件的时间戳比另一个事件的时间戳更早，那么这个事件将会优先被处理。通常，gem5 中的 processEvent 函数会在每次模拟器执行一个时间片之后被调用，以便处理所有已经到达的事件。

```
1 void
2 HelloObject::processEvent()
3 {
4     DPRINTF(Hello, "Hello world! Processing the event!\n");
5 }
```

2. 事件驱动的原理

(1) EventBase 类

Gem5 是一个事件驱动的模拟器。所谓事件（Event），可以理解为系统的读写、信号的到达等等一系列改变系统状态的行为，也可以包括程序员自己定义的回调函数。事件的公共父类是 EventBase 类，其内部定义了很多类的静态常量，方便事件之间共享标志位和优先级等定义。

EventBase 类定义的静态常量中，很大一部分是事件优先级（Priority）。这是用于区分在同一 cycle 中事件被处理的先后顺序的重要方式，事件包括 CPU 切换上下文、延迟写、DVFS 更新等。值得注意的是，优先级数值越小，事件的优先级越高。

事件	描述	优先级
Debug_Enable_Pri	调试启动的优先级，用于追踪可能 cycle 动作	-101
Debug_Break_Pri	breakpoints 的优先级，应该非常高否则会丢失信息	-100
CPU_Switch_Pri	CPU 上下文切换优先级，切换需要先完成再处理其他事件	-31
Delayed_Writeback_Pri	延迟写回，在常规写回之前	-1
Default_Pri	默认的优先级	0
DVFS_Update_Pri	DVFS 更新，会调用 stat dump	31
Serialize_Pri	序列化操作优先级	32
CPU_Tick_Pri	CPU 下一拍优先级，必须在所有CPU事件后触发	50
CPU_Exit_Pri	CPU 退出线程优先级	64
Stat_Event_Pri	输出统计信息的优先级，肯定在所有实质事件之后	90
Progress_Event_Pri	模拟的进程事件	95
Sim_Exit_Pri	标记模拟结束的事件	100

标志位	描述	常值
PublicRead	可被公开读的标志位	0x003f
PublicWrite	可被公开写的标志位	0x001d
Squashed	已经 squashed	0x0001
Scheduled	已经被调度	0x0002
Managed	Use life cycle manager	0x0004
AutoDelete	process() 后自动删除	0x0004
IsExitEvent	special exit event	0x0010
IsMainQueue	on main event queue	0x0020
Initialized	somewhat random bits	0x7a40
InitMask	mask for init bits	0xffc0

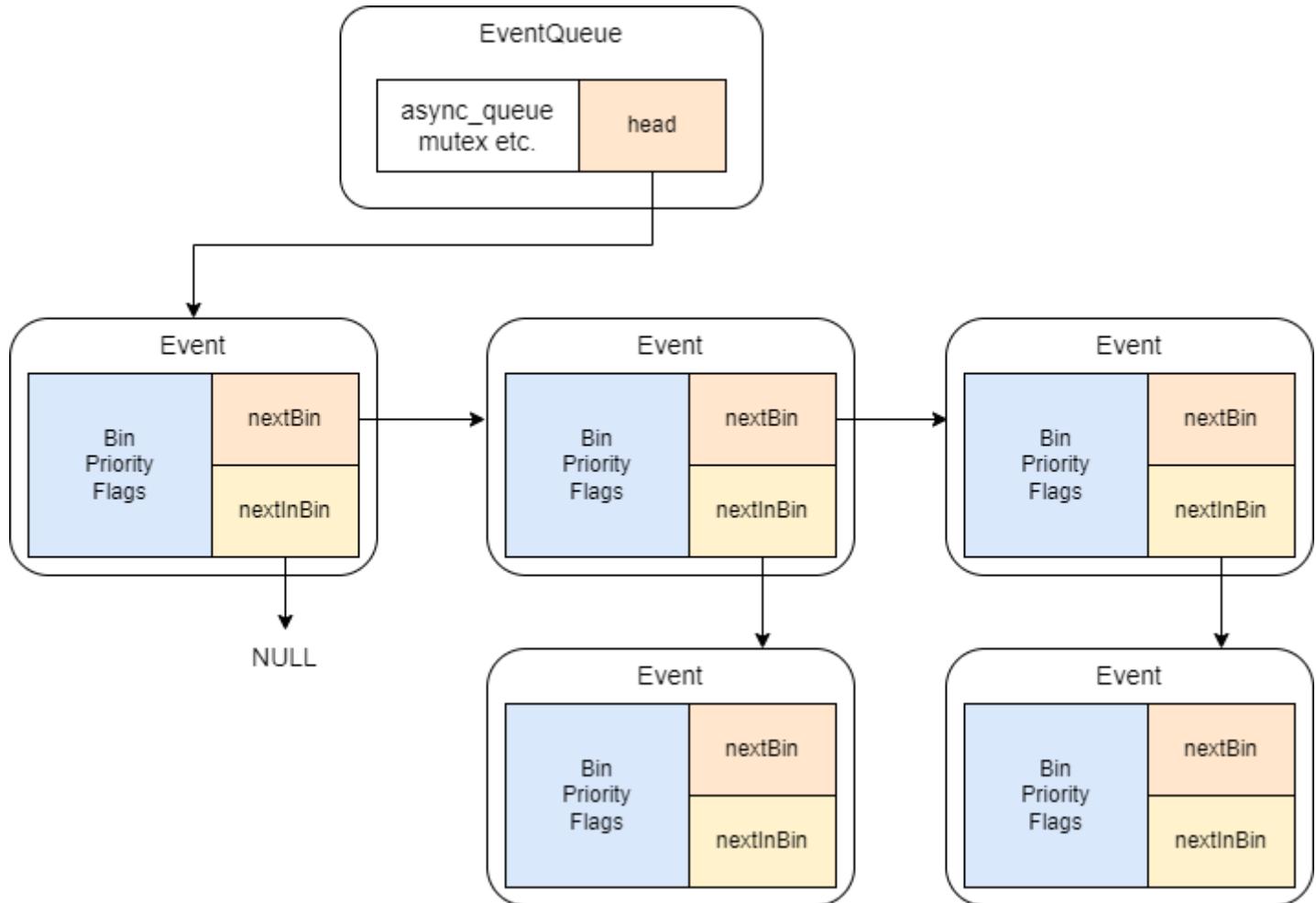
(2) Event 类

```

1 class Event : public EventBase, public Serializable {
2     Event *nextBin; // Bin defined as when+priority
3     Event *nextInBin;
4     Tick _when;    // timestamp when event should be processed
5     Priority _priority; // !< event priorityFlags
6     flags;
7     Counter instance; // event unique IDEventQueue *queue;}
```

事件队列 EventQueue 实质上是一个二级链表，因此 Event 中存在两个指针：nextBin 指向链表中下一项，而 nextInBin 指向 Bin 相同的下一项事件。此外还有时间戳（记录应当处理该事件的时间）、

优先级、标志位、ID和指向队列的指针。



在 Gem5 中，每个事件在被处理时都有一个回调函数 `process()`。每个继承了 Event 类的子类都必须实现。在处理事件时，还需要注意 EventBase 中的标志位，并随时注意调整 cycle 周期。此外，事件获取 `acquire()` 和释放 `release()` 时的动作也可以被子类重载实现。使用 `setwhen()` 可以设置事件被触发的时间，并被指定的队列。

```
1 void setWhen(Tick when, EventQueue *q) {
2     _when = when; #ifndef NDEBUG
3     queue = q; #endif
4     #ifdef EVENTQ_DEBUG
5     whenScheduled = curTick(); #endif
6 }
```

Event 类的使用场景多见于 EventQueue 类内的实现。若用户要直接使用 Event 类包装另一个类的对象，可能 EventWrapper 类或者 EventFunctionWrapper 更加合适：

```
1 template <class T, void (T::* F)()>
```

```

2 class EventWrapper : public Event {
3 private:T *object;
4 public:EventWrapper(T *obj, bool del = false, Priority p = Default_Pri):
5 Event(p), object(obj) {
6 if (del)setFlags(AutoDelete);
7 }
8 void process() { (object->*F)(); } // ...
9 class EventFunctionWrapper :
10 public Event {
11 private:
12 std::function<void(void)> callback;
13 std::string _name;
14 public:
15 EventFunctionWrapper(const std::function<void(void)> &callback,
16 const std::string &name, bool del = false, Priority p = Default_Pri):
17 Event(p), callback(callback), _name(name) {
18 if (del)setFlags(AutoDelete);
19 }
20 void process() { callback(); }};

```

(3) EventQueue

事件队列 (EventQueue) 是管理系统事件的重要载体，每个线程都会维护一个局部的事件队列，事件会在队列中被调度（即插入到队列中）。

```

1 class EventQueue {
2 Event *head;
3 Tick _curTick;
4 //! Mutex to protect async queue.
5 UncontendedMutex async_queue_mutex;
6 //! List of events added by other threads to this event queue.
7 std::list<Event*> async_queue;
8 // taken when servicing events
9 UncontendedMutex service_mutex;
10 }

```

async_queue 用于处理异步事件的队列，而真正的链表头是 **head**。此外，EventQueue 类重点还是在于 **schedule()** 函数，该函数负责事件调度，其参数是将要被执行的事件 event 和具体执行时间 when，global 参数用于判断队列是否被另一个线程运行，进而分成两个执行路径：

同步事件：调用 **insert()** 函数，此时 global 参数为 false，这是很简单的链表插入动作，插入在插入点之前，符合惯例。之前提到，EventQueue 队列是一个二级链表，是由一串串单链表的头节点组成的链表。从代码可知，在 in Bin 链表中，使用头插法插入了新节点，因此二级链表是 LIFO 的。

```

1 //! Current mode of execution: parallel / serial
2 extern bool inParallelMode;
3 void schedule(Event *event, Tick when, bool global=false) {
4     event->setWhen(when, this);
5     // a. A thread schedules local events on other queues through the asyncq.
6     // b. A thread schedules global events on the asyncq, whether or not
7     //      this event belongs to this eventq. This is required to maintain
8     //      a total order amongst the global events. See global_event.{cc,hh}
9     //      for more explanation.
10    if (inParallelMode && (this != curEventQueue() || global)) {
11        asyncInsert(event);
12    } else {
13        insert(event);
14    }
15    event->flags.set(Event::Scheduled);
16    event->acquire();
17    if (debug::Event)
18        event->trace("scheduled");
19 }

```

跨线程的事件调度：要么采用异步事件的调度方案，要么直接获取目的线程的事件队列锁。此时 global 参数设置为 true，采取第一种做法时，因为事件队列被另一个线程运行，为防止冲突，事件会被暂时插入到异步队列 async_queue 中。

最后在每个 simulation quantum 的最后时刻，async_queue 会被合入到真正的事件队列中（见 `handleAsyncInsertions()` 函数）。而若通过直接获取目的线程的事件队列锁的方式实现，那么为避免死锁情况，线程需要先自动放弃自己的事件队列锁，才能获得新事件队列锁，这样每个线程至多获取一个事件队列锁，从而避免了死锁。该功能由 `ScopedMigration` 类负责，将东西暂时移植到另一个队列中。

在 `schedule()` 函数的基础之上，还添加了 `reschedule()` 和 `deschedule()` 等函数，方便事件队列的调度管理。其中 `deschedule()` 函数牵扯到链表的删除。

当调用 `ServiceOne()` 函数时，默认是处理 `EventQueue` 中的第一个事件，取出第一个事件 `event` 后，在判断其是否 `squashed`（被压缩、合并），若不是被压缩的，那么直接将时间调整至事件处理时刻（事件驱动），随后调用 `event->process()`，若是，那么仅需要清除标志位即可。

根据代码，`SimObject` 类继承了 `EventManager` 类，`EventManager` 类中包装了一个 `EventQueue` 对象。因此，在 `startup()` 函数中使用的 `schedule()` 函数本质就是调用了 `EventQueue::schedule()` 函数：将事件 `event` 放入事件队列中，等待被调度。

另外，`EventFunctionWrapper` 类是 `Event` 类的子类，该类建立起了回调函数与事件对象的联系，当事件被触发后，回调函数会被执行。

2.7 Gem5 Drain

暂时未用到，没有仔细看

<https://dingfen.github.io/cpp/2022/03/08/gem5-2.html>

2.8 Gem5 Debug

1. Gem5中的多种debug模式

gem5模拟器提供了多种debug模式。具体取决于gem5的版本和安装，不同版本可能提供的debug模式不同。一些常用的debug模式包括：

DRAM：调试内存访问。

RoiOnly：只运行特定区域（ROI）内的代码，用于提高模拟速度。

SwitchCPU：调试多处理器系统中的CPU切换。

Checkpoint：调试检查点功能。

Checker：启用内存访问检查器。

Interrupt：调试中断处理。

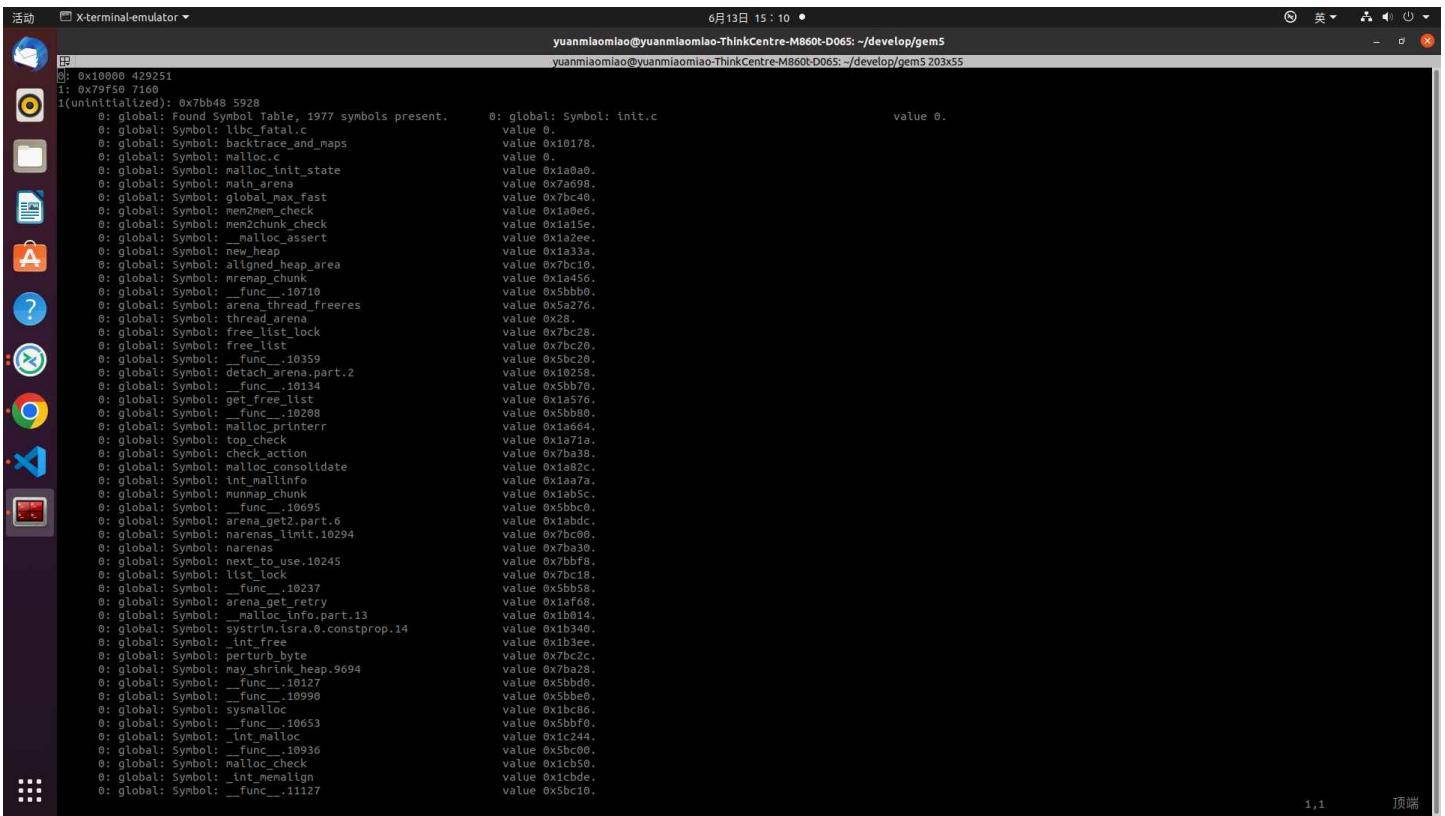
不同的debug模式可以提供不同的调试功能。

例如，DRAM模式可以帮助用户调试内存访问问题，而RoiOnly模式可以帮助用户优化模拟速度。

用户可以根据自己的需要选择适当的debug模式，并通过命令行参数启用它们。例如，可以使用--debug-flags=DRAM来启用DRAM模式。

```
1 ./build/RISCV/gem5.opt --debug-flags=All configs/example/se.py
2 -c tests/test-progs/hello/bin/riscv/linux/hello > hello_dbg
3 NOTE: 一定要指定一个二进制的可执行文件，否则会报错
4 command line: ./build/RISCV/gem5.opt --debug-flags=All configs/example/se.py
5 No workload specified. Exiting!
6
```

正确执行结果示例如下：



活动 X-terminal-emulator • 6月13日 15:10 •
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065: ~/develop/gem5
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065: ~/develop/gem5.203x55

```
0x10000 429251
I: 0x79750 71d0
l(uninitialized): 0x7bb48 5928
0: global: Found Symbol Table, 1977 symbols present.
0: global: Symbol: init.c
0: global: value 0.
0: global: Symbol: lIBC_fatal.c
0: global: value 0x1b178.
0: global: Symbol: backtrace_and_maps
0: global: value 0.
0: global: Symbol: malloc
0: global: value 0x1a080.
0: global: Symbol: malloc_init_state
0: global: value 0x7a698.
0: global: Symbol: global_max_fast
0: global: value 0x7bc40.
0: global: Symbol: mem2mem_check
0: global: value 0x1a066.
0: global: Symbol: mem2chunk_check
0: global: value 0x1a15e.
0: global: Symbol: __malloc_assert
0: global: value 0x1a2ee.
0: global: Symbol: new_heap
0: global: value 0x1a33a.
0: global: Symbol: aligned_heap_area
0: global: value 0x7bc10.
0: global: Symbol: mrenap_chunk
0: global: value 0x1a456.
0: global: Symbol: __func_.10710
0: global: value 0x5bb00.
0: global: Symbol: arena_thread_freeares
0: global: value 0x5a276.
0: global: Symbol: thread_arena
0: global: value 0x28.
0: global: Symbol: free_list_lock
0: global: value 0x7bc28.
0: global: Symbol: free_llist
0: global: value 0x7bc20.
0: global: Symbol: __func_.10359
0: global: value 0x10258.
0: global: Symbol: detach_arena.part.2
0: global: value 0x5bb70.
0: global: Symbol: __func_.10134
0: global: value 0x1a576.
0: global: Symbol: get_free_list
0: global: value 0x5bb80.
0: global: Symbol: __func_.10208
0: global: value 0x1a664.
0: global: Symbol: malloc_printer
0: global: value 0x1a71a.
0: global: Symbol: __op_check
0: global: value 0x1a73c.
0: global: Symbol: check_action
0: global: value 0x1a838.
0: global: Symbol: malloc_consolidate
0: global: value 0x1a87c.
0: global: Symbol: int_mallocinfo
0: global: value 0x1a97c.
0: global: Symbol: malloc_chunk
0: global: value 0x1bb5c.
0: global: Symbol: __func_.10695
0: global: value 0x5bbc0.
0: global: Symbol: arena_get2.part.6
0: global: value 0x1abd4.
0: global: Symbol: narenas_limit.10294
0: global: value 0x7bc00.
0: global: Symbol: narenas
0: global: value 0x7ba30.
0: global: Symbol: nos_to_use.10245
0: global: value 0x7bf8.
0: global: Symbol: list_lock
0: global: value 0x7bc18.
0: global: Symbol: __func_.10237
0: global: value 0x5bb58.
0: global: Symbol: arena_get_retry
0: global: value 0x1af08.
0: global: Symbol: __malloc_info.part.13
0: global: value 0x1b014.
0: global: Symbol: syslrm_isra.0.constprop.14
0: global: value 0x1b340.
0: global: Symbol: __in_free
0: global: value 0x1b3ee.
0: global: Symbol: perturb_byte
0: global: value 0x7bc2c.
0: global: Symbol: may_shrink_heap.9694
0: global: value 0x7ba28.
0: global: Symbol: __func_.10127
0: global: value 0x5bbd0.
0: global: Symbol: __func_.10990
0: global: value 0x5bbe0.
0: global: Symbol: sysnalloc
0: global: value 0x1bc86.
0: global: Symbol: __func_.10653
0: global: value 0x5bbf0.
0: global: Symbol: __int_malloc
0: global: value 0x1c244.
0: global: Symbol: __func_.10936
0: global: value 0x5bc00.
0: global: Symbol: malloc_check
0: global: value 0x1cb50.
0: global: Symbol: __int_menalign
0: global: value 0x1cbde.
0: global: Symbol: __func_.11127
```

2. DPRINTF

DPRINTF 是一种宏，它用于打印调试信息。它的语法如下：

```
1 DPRINTF(Class, format, args...)
```

其中，Class 是一个调试信息类别，是一个 Debug Flag，需要提前声明，format 是一个字符串，包含要打印的信息的格式。如果宏被调用，那么会将 format 中的信息格式化后输出到控制台上。例如，如果执行上面的语句，那么会在控制台上打印出：

Created the hello object

这条语句的目的是将一条调试信息输出到控制台上，用于提示程序创建了一个名为 hello 的对象。

注意：DPRINTF 宏只有在程序被编译成调试模式时才会生效，在发布模式下不会输出任何信息。

3. 如何添加新的Debug Flag

```
1 build/RISCV/gem5.opt --debug-flags=Exec
2 configs/learning_gem5/part1/simple.py | head -n 50
```

执行结果如下：

活动 x-terminal-emulator • 6月13日 15:20 • 英 •

```

yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065: ~/develop/gem5
build/RISCV/ren/mem/interface.cc:794: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065: ~/develop/gem5
o->system.remote_gdb: listening for remote gdb on port 7000
build/RISCV/sim/simulate.cc:94: Info: Entering event queue @ 0. Starting simulation...
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 21.2.1.0
gem5 compiled Jun 5 2023 21:34:54
gem5 started Jun 13 2023 15:20:06
gem5 executing on yuanmiaomiao-ao-ThinkCentre-M860t-D065, pid 70031
command line: ./build/RISCV/gem5.opt --debug-flags=Exec configs/learning_gem5/part1/simple.py

Beginning simulation!
77000: system.cpu: T0 : 0x102a4 @ start : jal ra, 44      : IntAlu : D=0x0000000000000000102a8
126000: system.cpu: T0 : 0x102d0 @ start+44 : addi gp, sp, -480   : IntAlu : D=0x0000000000007c2d0
175000: system.cpu: T0 : 0x102d4 @ start+48 : addi gp, sp, -480   : IntAlu : D=0x00000000000000007c0f0
224000: system.cpu: T0 : 0x102d8 @ start+52 : c_mv a5, a0      : IntAlu : D=0x000000000000000000000000
273000: system.cpu: T0 : 0x102a8 @ start+4 : c_mv a5, a0      : IntAlu : D=0x000000000000000000000000
371000: system.cpu: T0 : 0x102a6 @ start+6 : addi a0, a0, 232    : IntAlu : D=0x000000000000000000000000
469000: system.cpu: T0 : 0x102a4 @ start+10 : addi a0, a0, 232    : IntAlu : D=0x000000000000000000000000
518000: system.cpu: T0 : 0x102b2 @ start+16 : c_ldsp a1, 0(sp)   : MemRead : D=0x000000000000000000000001 A=0x7fffffffffffffed0
638000: system.cpu: T0 : 0x102b4 @ start+16 : c_addl4spn a2, sp, 8  : IntAlu : D=0x7fffffffffffe0
728000: system.cpu: T0 : 0x102b8 @ start+16 : addi sp, sp, 1844674407378955160 : IntAlu : D=0x7fffffffffffe0
782000: system.cpu: T0 : 0x102b4 @ start+20 : addi a2, a3, 1504   : IntAlu : D=0x000000000000000000000000
938000: system.cpu: T0 : 0x102b2 @ start+26 : addi a2, a3, 1504   : IntAlu : D=0x000000000000000000000000
1036000: system.cpu: T0 : 0x102c2 @ start+30 : addi a4, a4, 1640    : IntAlu : D=0x000000000000000000000000
1134000: system.cpu: T0 : 0x102c6 @ start+34 : addi a4, a4, 1640    : IntAlu : D=0x000000000000000000000000
1183000: system.cpu: T0 : 0x102ca @ start+34 : c_mv a6, sp      : IntAlu : D=0x7fffffffffffe0
1232000: system.cpu: T0 : 0x102cc @ start+40 : ? : f21912, 230    : IntAlu :
1281000: system.cpu: T0 : 0x103b2 @ libc_start_main : c_addl16sp sp, -320  : IntAlu : D=0x7fffffffffffe0
1330000: system.cpu: T0 : 0x103b0 @ libc_start_main+2 : lui t1, 0     : IntAlu : D=0x0000000000000000
1379000: system.cpu: T0 : 0x103b0 @ libc_start_main+6 : c_sdsp s0, 304(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffffec0
1449000: system.cpu: T0 : 0x103b0 @ libc_start_main+8 : c_sdsp s1, 296(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffffeb8
1518000: system.cpu: T0 : 0x103b0 @ libc_start_main+10 : c_sdsp s2, 288(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffe0
1589000: system.cpu: T0 : 0x103c0 @ libc_start_main+12 : c_sdsp r0, 2812(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffe0
1658000: system.cpu: T0 : 0x103c0 @ libc_start_main+14 : addi t1, a7, 0    : IntAlu : D=0x0000000000000000
17708000: system.cpu: T0 : 0x103c4 @ libc_start_main+18 : c_sdsp a0, 24(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffd8
1778000: system.cpu: T0 : 0x103c6 @ libc_start_main+20 : c_sdsp a1, 8(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffd8
1862000: system.cpu: T0 : 0x103c8 @ libc_start_main+22 : c_sdsp a2, 16(sp) : MemWrite : D=0x0000000000000000 A=0x7fffffffffffd8
1932000: system.cpu: T0 : 0x103ca @ libc_start_main+24 : c_mv s2, a5      : IntAlu : D=0x0000000000000000
1981000: system.cpu: T0 : 0x103ca @ libc_start_main+26 : c_mv s0, a3      : IntAlu : D=0x0000000000000000
2030000: system.cpu: T0 : 0x103ca @ libc_start_main+28 : c_mv s1, a4      : IntAlu : D=0x0000000000000000
2079000: system.cpu: T0 : 0x103d0 @ libc_start_main+30 : c_ll a5, 0      : IntAlu : D=0x0000000000000000
217000: system.cpu: T0 : 0x103d2 @ libc_start_main+32 : beq t1, zero, 12  : IntAlu :
2326000: system.cpu: T0 : 0x103d4 @ libc_start_main+44 : addi a4, a4, 8(sp) : MemRead : D=0x0000000000000001 A=0x7fffffffffffd98
2324000: system.cpu: T0 : 0x103d8 @ libc_start_main+48 : addi a4, a4, 1    : IntAlu : D=0x0000000000000002
2373800: system.cpu: T0 : 0x103e4 @ libc_start_main+50 : c_ldsp a4, 10(sp) : MemRead : D=0x7fffffffffffe0 A=0x7fffffffffffd98
2471000: system.cpu: T0 : 0x103e6 @ libc_start_main+52 : c_sllv a0, 3       : IntAlu : D=0x0000000000000010
2526000: system.cpu: T0 : 0x103e8 @ libc_start_main+54 : c_add a0, a4       : IntAlu : D=0x7fffffffffffe0
2618000: system.cpu: T0 : 0x103e8 @ libc_start_main+56 : sw a5, -1576(gp)   : MemWrite : D=0x0000000000000000 A=0x7bac8
2737000: system.cpu: T0 : 0x103e8 @ libc_start_main+60 : sw a0, -1192(gp)   : MemWrite : D=0x7fffffffffffe0 A=0x7bc48
2870000: system.cpu: T0 : 0x103f2 @ libc_start_main+64 : lut a5, 12     : IntAlu : D=0x0000000000007a000
build/RISCV/sim/syscall_emul.hh:1014: warn: linkline() called on '/proc/self/exe' may yield unexpected results in various settings.
    Returning '/home/yuanmiaomiao/Hello/bin/riscv/Hello'
build/RISCV/sim/mem_state.cc:443: Info: Increasing stack size by one page.
Exception ignored: In <_IO_TextIOWrapper name='stdout' mode='w' encoding='utf-8'>
BrokenPipeError [Errno 32] Broken pipe
yuanmiaomiao@yuanmiaomiao-ao-ThinkCentre-M860t-D065: ~/develop/gem5 ./build/RISCV/gem5.opt --debug-flags=Exec configs/learning_gem5/part1/simple.py | head -n 50

```

具体解析如下：

Exec 是 gem5 中的一个调试标志，它允许用户在模拟执行过程中跟踪执行流程。它可以帮助用户更好地理解系统是如何执行给定的指令的，并且可以帮助用户发现潜在的问题和错误。不过，由于它会增加模拟执行的时间和复杂度，因此在实际应用中通常不会使用 exec 标志。

如果要在gem5中添加一个新的debug flag，可以按照以下步骤操作：

Step1.希望使用自己的debug flag，首先要在 **SConscript** 声明，比如以我们刚刚建立的hello object为基础，在 **learning_gem5/** 下的 **SConscript** 添加如下声明：

- Path: `src/learning_gem5/part2/SConscript`

```

1 Import('*')
2
3 SimObject('SimpleObject.py', sim_objects=['SimpleObject'])
4 SimObject('HelloObject.py', sim_objects=['HelloObject', 'GoodbyeObject'])
5
6 Source('simple_object.cc')
7 Source('hello_object.cc')
8
9 DebugFlag('HelloExample', "For Learning gem5 Part 2. Simple example debug flag")
10
11

```

这一句代表声明了一个叫' **HelloExample**' 的debug flag，声明后一个调试用的头文件就会默认在该目录生成，因此只需要在hello_object.cc里添加上引入头文件声明就好：

```
1 #include "debug/Hello.hh"
```

最后就能替换std::out了：

```
1 DPRINTF(Hello, "Created the hello object\n");
```

DPRINTF是一个C++的宏，第一个参数就是debug flag，其余的参数可以是任何你想传递给printf的东西。

测试有效

其他方法：

Step1. 在gem5源代码中找到**debug/Debug.py**文件。(没有找到)

Step2. 打开该文件，在类**DebugFlags**中定义一个新的debug flag。例如：

```
1 class DebugFlags:
2     def __init__(self):
3         self.my_new_flag = False
```

Step3. 在同一个文件中，找到**parse_options()**函数。该函数用于解析命令行参数，并设置debug flag的值。在该函数中，添加一行代码来解析新增的debug flag，并将其设置为True或False。例如：

```
1 def parse_options(self, options=None):
2     ...
3     if options.my_new_flag:
4         self.my_new_flag = True
```

Step4. 保存文件，并使用scons编译gem5源代码。

Step5. 在运行gem5模拟器时，使用--debug-flags=my_new_flag命令行参数来启用新增的debug flag。

通过上述步骤，可以在gem5中添加一个新的debug flag，并在模拟运行时启用它。此外，还可以使用该debug flag来控制模拟运行时的输出。

4. Debug Ouput

```
1 gem5 Simulator System. http://gem5.org
2 gem5 is copyrighted software; use the --copyright option for details.
3 gem5 executing on chinook, pid 29078
4 command line: build/RISCV/gem5.opt --debug-flags=HelloExample
5 configs/learning_gem5/part2/run_hello.py
6 Global frequency set at 1000000000000 ticks per second
7      0: hello: Created the hello object
8 Beginning simulation!
9 info: Entering event queue @ 0. Starting simulation...
10 Exiting @ tick 18446744073709551615 because simulate() limit reached
11
```

该输出是自定义的Debug flag，通过调用DPRINTF得到，输出有三个部分组成，

- 1: “0” 即执行DPRINTF时的tick（时钟周期）
- 2: “hello” SimObject的名字，在python配置文件中写明的，通过name()函数返回
- 3: “Created the hello object” 你自己传递到DPRINTF函数中的字符串

2.9 Power and Thermal

暂时没有用到，没有时间看

<https://zhuanlan.zhihu.com/p/404925882>

3 GEM5 RVV支持

```
1 //NOTE: 修改的路径在src/arch/riscv/*, 不是build/arch/riscv/*, build 下的会在编译时被
2 //NOTE: generate 目录下的内容是通过模板生成的, 不要改动
3 //NOTE: 新增的内容, 一定要添加到src/arch/riscv/isa/includes.isa, 否则编译不通过
```

3.1 参数定义

```
1 SConstruct
```

```
1 global_vars = Variables(global_vars_file, args=ARGUMENTS)
2 global_vars.AddVariables(
3 // .....
4     ('RISCV_VLEN', "VLEN for RVV", environ.get('RISCV_VLEN', '1024')),
5     ('RISCV_ELEN', "ELEN for RVV", environ.get('RISCV_ELEN', '128')),
6 )
7
```

```
1 Path
2 util/crosstool-ng/riscv64-unknown-linux-gnu.defconfig
```

```
1 CT_KERNEL_LINUX=y
2 CT_BINUTILS_PLUGINS=y
3 CT_CC_LANG_CXX=y
4 CT_DEBUG_GDB=y
5 # CT_GDB_CROSS PYTHON is not set
6 # CT_GDB_GDBSERVER is not set
```

3.2 指令格式

指令比特字段定义

```
1 src/arch/riscv/isa/bitfields.isa
```

```
1 // Vector instructions
2 def bitfield VFUNC_6      <31:26>;
3 def bitfield VFUNC_5      <31:27>;
4 def bitfield VFUNC_3      <27:25>;
5 def bitfield VFUNC_2      <26:25>;
6
7 def bitfield VS3          <11:7>;
8 def bitfield VS2          <24:20>;
9 def bitfield VS1          <19:15>;
10 def bitfield VD           <11:7>;
11
12 def bitfield NF           <31:29>;
13 def bitfield MEW          <28:28>;
```

```

14 def bitfield MOP          <27:26>;
15 def bitfield VM           <25>;
16 def bitfield LUMOP        <24:20>;
17 def bitfield SUMOP        <24:20>;
18 def bitfield WIDTH         <14:12>;
19
20 def bitfield SIMM_3       <17:15>;
21 def bitfield BIT31        <31>;
22 def bitfield BIT30        <30>;
23 def bitfield ZIMM11        <30:20>;
24 def bitfield ZIMM10        <29:20>;
25 def bitfield UIMM5         <19:15>;

```

1 Path src/riscv/insts/vector_static_inst.hh

```

1 //向量指令字段定义
2 class VectorInsn : public VectorStaticInst
3 {
4 public:
5     VectorInsn(const char *mnem, MachInst _machInst, OpClass __opClass):
6         VectorStaticInst(mnem, __opClass),
7         machInst(_machInst),
8         mnemo(mnem){}
9     ~VectorInsn() {}
10
11    std::string getName() const override { return mnemo; }
12
13    std::string regName(RegIndex reg) const;
14
15    RegIndex rs1() const { return (RegIndex)bits(machInst, 19, 15); }
16    RegIndex rs2() const { return (RegIndex)bits(machInst, 24, 20); }
17    RegIndex rd() const { return (RegIndex)bits(machInst, 11, 7); }
18
19    RegIndex vs1() const { return (RegIndex)bits(machInst, 19, 15); }
20    RegIndex vs2() const { return (RegIndex)bits(machInst, 24, 20); }
21    RegIndex vs3() const { return (RegIndex)bits(machInst, 11, 7); }
22    RegIndex vd() const { return (RegIndex)bits(machInst, 11, 7); }
23
24    bool vm() const { return bits(machInst, 25, 25); }
25
26    uint32_t vtypei() const { return (bits(machInst, 31, 31) == 0) ? bits
27
28    uint8_t uimm5() const { return bits(machInst, 19, 15); }

```

```
29
30     const RiscvISA::ExtMachInst machInst;
31
32 private:
33     const char *mnemo;
34 };
```

向量访存指令继承VectorInsn

```
1 class VectorMemInst : public VectorInsn
2 {
3     protected:
4         Request::Flags memAccessFlags;
5
6     VectorMemInst(const char *mnem, ExtMachInst _machInst, OpClass __opClass)
7         : VectorInsn(mnem, _machInst, __opClass)
8     {}
9 };
```

操作数定义

```
1 src/arch/riscv/isa/operands.isa
```

```
1     'Vd': ('VecReg', 'vc', 'VD', 'IsVector', 1),
2     'Vs1': ('VecReg', 'vc', 'VS1', 'IsVector', 2),
3     'Vs2': ('VecReg', 'vc', 'VS2', 'IsVector', 3),
4     'Vs3': ('VecReg', 'vc', 'VS3', 'IsVector', 4),
```

3.2 字段定义

(1) 向量配置寄存器定义

```
1 src/arch/riscv/isa.cc
```

```
1 [MISCREG_VSTART] = "VSTART",
```

```
2 [MISCREG_VXSAT]          = "VXSAT",
3 [MISCREG_VXRM]           = "VXRM",
4 [MISCREG_VCSR]           = "VCSR",
5 [MISCREG_VL]              = "VL",
6 [MISCREG_VTYPE]           = "VTYPE",
7 [MISCREG_VLENB]           = "VLENB",
```

(2) 寄存器各字段定义

```
1 src/arch/riscv/regs/vec.hh
```

```
1 #ifndef __ARCH_RISCV_REGS_VEC_HH__
2 #define __ARCH_RISCV_REGS_VEC_HH__
3
4 #include "arch/generic/vec_pred_reg.hh"
5 #include "arch/generic/vec_reg.hh"
6 #include "arch/riscv/types.hh"
7 #include "arch/riscv/vec_params.hh"
8 #include "base/bitunion.hh"
9
10 namespace gem5
11 {
12
13 namespace RiscvISA
14 {
15
16 constexpr uint32_t VLENB = RISCV_VLEN / 8;
17 constexpr uint32_t ELEN = RISCV_ELEN;
18
19 constexpr size_t VecRegSizeBytes = VLENB;
20 using VecElem = uint8_t;
21 constexpr unsigned NumVecElemPerVecReg = VLENB / sizeof(VecElem);
22
23 using VecRegContainer =
24     gem5::VecRegContainer<NumVecElemPerVecReg * sizeof(VecElem)>;
25
26 // Not applicable to RISC-V
27 using VecPredRegContainer = ::gem5::DummyVecPredRegContainer;
28
29 const int NumVecRegs = 32;
30
```

```

31 const std::vector<std::string> VectorRegNames ={
32     "v0",    "v1",    "v2",    "v3",    "v4",    "v5",    "v6",    "v7",
33     "v8",    "v9",    "v10",   "v11",   "v12",   "v13",   "v14",   "v15",
34     "v16",   "v17",   "v18",   "v19",   "v20",   "v21",   "v22",   "v23",
35     "v24",   "v25",   "v26",   "v27",   "v28",   "v29",   "v30",   "v31"
36 };
37
38 /**
39 * These fields are specified in the RISC-V Vector Extension Manual.
40 */
41 BitUnion32(VTYPE)
42     Bitfield<31> vill;
43     Bitfield<7> vma;
44     Bitfield<6> vta;
45     Bitfield<5, 3> vsew;
46     Bitfield<2, 0> vlmul;
47 EndBitUnion(VTYPE)
48
49 } // namespace RiscvISA
50 } // namespace gem5
51
52 #endif

```

3.3 指令分类

主要根据操作类型和指令功能进行分类进行分类，基本类包括以下几类：

具体定义路径

1 path: src/arch/riscv/vector.hh

	A	B
1	Class	说明
2	VectorVRXUNARY0Op	// vd[0] = rs1
3	VectorVRFUNARY0Op	// vd[0] = fs1
4	VectorVWFUNARY0Op	// vd[0] = Widen *fs
5	VectorVFUNARY0Op	
6	VectorVMUNARY0Op	
7	VectorVMUNARYVs20Op	
8	VectorVWXUNARY0Op	//red = vs2[0]
9	VectorOPIVIOp	
10	VectorVdVs2Rs1Op	// op vd, rs1, vs2
11	VectorVdVs2Vs1Op	// op vd, vs1, vs2
12	VectorNarrowingVIOp	
13	VectorNarrowingVVOp	
14	VectorNarrowingWXOp	
15	VectorIntegerExtensionOp	
16	VectorMaskRegisterVdVs2Vs1Op	
17	VectorMaskRegisterVdVs2Op	
18	VectorWholeRegisterMoveOp	
19	VectorWideningVXOp	
20	VectorWideningWVOp	
21	VectorWideningVVOp	
22	VectorUnitStrideMemLoadOp	
23	VectorStridedMemLoadOp	
24	VectorIndexedMemLoadOp	
25	VectorIndexedMemStoreOp	
26	VectorUnitStrideMemStoreOp	
27	VectorStridedMemStoreOp	
28	VectorCfgOp	设置配置寄存器
29	VectorVdVs2Fs1Op	
30	VectorWideningVdVs2Op	

以上类型， gem5 设置了指令构造模板,主要设置以下参数

- (1) 操作数类型
- (2) 源寄存器的索引和目的寄存器的索引
- (3) 操作元素的大小
- (4) 操作类型

(5) 不同指令类型的执行模板

```
1 path:src/arch/riscv/isa/formats/vector.isa
```

以向量load-index 为例，Constructor 模板如下：

(1) %(class_name), %(base_class), %(mnemonic)为模板替换

(2) 根据指令的FUNCT3字段进行EEW,SEW 的计算

(3) 根据指令需要的源寄存器和目的寄存器的个数设置对应的索引

(4) 设置对应的flag, VectorIndexedMemLoad 设置为

```
flags[IsLoad] = true;
flags[IsVector] = true;;
```

```
1 def template VectorIndexedMemLoadConstructor {{
2     %(class_name)s::%(class_name)s(MachInst machInst, RiscvISA::VTYPE machVtype,
3         : %(base_class)s("%(mnemonic)s", machInst, VectorIndexedMemLoadOpClass)
4     {
5         %(set_reg_idx_arr)s;
6
7         _numSrcRegs = 0;
8         _numDestRegs = 0;
9         _numFPDestRegs = 0;
10        _numVecSrcRegs = 0;
11        _numVecDestRegs = 0;
12        _numVecElemDestRegs = 0;
13        _numVecPredDestRegs = 0;
14        _numIntDestRegs = 0;
15        _numCCDestRegs = 0;
16
17        // Example:
18        // vlxuei32.v      nf 28=0 27..26=1 vm vs2 rs1 14..12=0x6 vd 6..0=0x07
19        setSrcRegIdx(_numSrcRegs++, RegId(IntRegClass, RS1));
20
21        uint32_t eewb = 0;
22        switch (FUNCT3) {
23            case 0:
24                eewb = 1;
25                break;
26            case 5:
27                eewb = 2;
28                break;
29            case 6:
```

```

30             eewb = 4;
31             break;
32         case 7:
33             eewb = 8;
34             break;
35     }
36
37     // The data vector register group has EEW=SEW, EMUL=LMUL,
38     // while the offset vector register group has EEW
39     // encoded in the instruction and EMUL=(EEW/SEW)*LMUL.
40     _numVecSrcRegs =
41         ceil((float) machVl / (VecRegSizeBytes/eewb));
42     assert((VS2 % alignToPowerOfTwo(_numVecSrcRegs)) == 0);
43
44     uint32_t sewb = getSew(machVtype.vsew) / 8;
45     _numVecDestRegs =
46         ceil((float) machVl / (VecRegSizeBytes/sewb));
47     assert((VD % alignToPowerOfTwo(_numVecDestRegs)) == 0);
48
49     for (int i = 0; i < _numVecDestRegs; ++i) {
50         setDestRegIdx(_numDestRegs++, RegId(VecRegClass, VD + i));
51     }
52
53     for (int i = 0; i < _numVecSrcRegs; ++i) {
54         setSrcRegIdx(_numSrcRegs++, RegId(VecRegClass, VS2 + i));
55     }
56
57     if (VM == 0) {
58         // Masked instruction.
59         setSrcRegIdx(_numSrcRegs++, RegId(VecRegClass, 0));
60         ++_numVecSrcRegs;
61     }
62
63     flags[IsLoad] = true;
64     flags[IsVector] = true;;
65 }
66 }};

```

Execute 模板

- (1) 读 VL, Vtype 寄存器
- (2) 根据Vtype 中SEW 的值和操作码字段FUNCT3 确定EEW
- (3) 根据Vtype 中LMUL的值, VL , EEW 计算操作元素的个数
- (4) 根据VM 的值确定目的寄存器的VMask 比特

(5) 根据EEW，读出向量寄存器的值，计算偏移量

(6) 译码时，通过脚本将 (code)s 的内容替换，调用readMemAtomicLE获取load_value

(7) 将load_value 的值放到对应的目的寄存器里

```
1 def template VectorIndexedMemLoadExecute {{
2     Fault
3     %(class_name)s::execute(ExecContext *xc,
4         Trace::InstRecord *traceData) const
5     {
6         Fault fault = NoFault;
7         uint32_t vl = xc->readMiscReg(MISCREG_VL);
8         VTYPE vtype = xc->readMiscReg(MISCREG_VTYPE);
9         size_t sewb = getSew(vtype.vsew) / 8;
10
11         uint32_t eewb = 0;
12         switch (FUNCT3) {
13             case 0:
14                 eewb = 1;
15                 break;
16             // ...
17             default:
18                 std::string error = csprintf(
19                     "Illegal eew value: 0x%x\n", FUNCT3);
20                 return std::make_shared<IllegalInstFault>(error, machInst);
21                 break;
22         }
23
24         float vemul = ((float) eewb / sewb) * getVflmul(vtype.vlmul);
25
26         if ((vemul < 0.125) || (vemul > 8)) {
27             std::string error = csprintf(
28                 "Illegal vemul value: %f\n", vemul);
29             return std::make_shared<IllegalInstFault>(error, machInst);
30         }
31
32         %(op_decl)s;
33
34         Rs1 = xc->readIntRegOperand(this, 0);
35
36         size_t num_elements_in_src_reg = (VecRegSizeBytes/eewb);
37         uint32_t srcVecRegID = 0;
38         uint32_t srcVecRegElemID = 0;
39
40         for (uint32_t destVecRegID = 0; destVecRegID < _numVecDestRegs; ++destVe
41             TheISA::VecRegContainer& Vd_container =
```

```

42         xc->getWritableVecRegOperand(this, destVecRegID);
43
44         uint32_t vmask = UINT32_MAX;
45         if (VM == 0) {
46             const TheISA::VecRegContainer& Vmask_container =
47                 xc->readVecRegOperand(this, _numSrcRegs - 1);
48             switch(sewb) {
49                 case 1:
50                     vmask = Vmask_container.as<uint32_t>()[destVecRegID];
51                     break;
52                 case 2:
53                     vmask = Vmask_container.as<uint16_t>()[destVecRegID];
54                     break;
55                 ....
56             }
57         }
58
59         size_t num_elements_in_dest_reg =
60             (VecRegSizeBytes/sewb) > vl ? vl : (VecRegSizeBytes/sewb);
61
62         if (fault == NoFault) {
63             if (sewb == 1) {
64                 auto Vd = Vd_container.as<uint8_t>();
65                 for (uint32_t regElemID = 0; regElemID < num_elements_in_des
66                     if (bits(vmask, regElemID, regElemID) == 0) {
67                         continue;
68                     }
69
70                     const TheISA::VecRegContainer& Vs2_container =
71                         xc->readVecRegOperand(this, srcVecRegID + 1);
72
73                     uint64_t offset = 0;
74                     switch (eewb) {
75                         case 1:
76                             offset = Vs2_container.as<uint8_t>()[srcVecRegEl
77                             break;
78                         case 2:
79                             ...
80                         }
81                     uint8_t load_value;
82                     %(code)s;
83
84                     ++srcVecRegElemID;
85                     if (srcVecRegElemID == num_elements_in_src_reg) {
86                         srcVecRegElemID = 0;
87                         ++srcVecRegID;
88                     }

```

```

89             }
90         } else {
91             std::string error = csprintf(
92                 "Unexpected sewb value in instruction");
93             return std::make_shared<IllegalInstFault>(error, machInst);
94         }
95
96         if (fault == NoFault) {
97             if (traceData) {
98                 // TODO: Print all destination registers
99                 traceData->setData(Vd_container);
100            }
101        }
102
103         vl = vl - num_elements_in_dest_reg;
104     }
105 }
106
107 return fault;
108 }
109 };

```

readMemAtomicLE的定义在以下文件中

1 path: src/arch/generic/memhelpers.hh

该文件主要声明了一些相关的存储操作，主要包括初始化、读写、原子操作、字节序（大头优先还是小头优先）以及对应的时序模式

```

1 {
2     template <class XC>
3     Fault
4     initiateMemRead(XC *xc, Addr addr, std::size_t size,
5                     Request::Flags flags,
6                     const std::vector<bool> &byte_enable)
7     {
8         return xc->initiateMemRead(addr, size, flags, byte_enable);
9     }
10
11 // Initiate a read from memory in timing mode. Note that the 'mem'
12 // parameter is unused; only the type of that parameter is used
13 // to determine the size of the access.
14 template <class XC, class MemT>

```

```

15 Fault
16 initiateMemRead(XC *xc, Trace::InstRecord *traceData, Addr addr,
17                     MemT &mem, Request::Flags flags)
18 {
19     static const std::vector<bool> byte_enable(sizeof(MemT), true);
20     return initiateMemRead(xc, addr, sizeof(MemT),
21                             flags, byte_enable);
22 }
23
24 /// Extract the data returned from a timing mode read.
25 template <ByteOrder Order, class MemT>
26 void
27 getMem(PacketPtr pkt, MemT &mem, Trace::InstRecord *traceData)
28 {
29     mem = pkt->get<MemT>(Order);
30     if (traceData)
31         traceData->setData(mem);
32 }
33
34 template <class MemT>
35 void
36 getMemLE(PacketPtr pkt, MemT &mem, Trace::InstRecord *traceData)
37 {
38     getMem<ByteOrder::little>(pkt, mem, traceData);
39 }
40
41 template <class MemT>
42 void
43 getMemBE(PacketPtr pkt, MemT &mem, Trace::InstRecord *traceData)
44 {
45     getMem<ByteOrder::big>(pkt, mem, traceData);
46 }
47
48 /// Read from memory in atomic mode.
49 template <class XC>
50 Fault
51 readMemAtomic(XC *xc, Addr addr, uint8_t *mem,
52                 std::size_t size, Request::Flags flags,
53                 const std::vector<bool> &byte_enable)
54 {
55     return xc->readMem(addr, mem, size, flags, byte_enable);
56 }
57
58 /// Read from memory in atomic mode.
59 template <ByteOrder Order, class XC, class MemT>
60 Fault
61 readMemAtomic(XC *xc, Trace::InstRecord *traceData, Addr addr, MemT &mem,

```

```

62     Request::Flags flags)
63 {
64     memset(&mem, 0, sizeof(mem));
65     static const std::vector<bool> byte_enable(sizeof(MemT), true);
66     Fault fault = readMemAtomic(xc, addr, (uint8_t*)&mem,
67                                 sizeof(MemT), flags, byte_enable);
68     if (fault == NoFault) {
69         mem = gtoh(mem, Order);
70         if (traceData)
71             traceData->setData(mem);
72     }
73     return fault;
74 }
75
76 template <class XC, class MemT>
77 Fault
78 readMemAtomicLE(XC *xc, Trace::InstRecord *traceData, Addr addr, MemT &mem,
79                  Request::Flags flags)
80 {
81     return readMemAtomic<ByteOrder::little>(
82         xc, traceData, addr, mem, flags);
83 }
84
85 template <class XC, class MemT>
86 Fault
87 readMemAtomicBE(XC *xc, Trace::InstRecord *traceData, Addr addr, MemT &mem,
88                  Request::Flags flags)
89 {
90     return readMemAtomic<ByteOrder::big>(xc, traceData, addr, mem, flags);
91 }
92
93
94 ....
95
96 }

```

Declare 示例

```

1 def template VectorIndexedMemLoadDeclare {{
2     //
3     // Static instruction class for "%(mnemonic)s".
4     // vlxr ei32.v      nf 28=0 27..26=1 vm vs2 rs1 14..12=0x6  vd 6..0=0x07
5     class %(class_name)s : public %(base_class)s
6     {

```

```

7     private:
8         // Example:
9         //
10        // Sources: Rs1 + up to 8 vector registers starting at Vs2
11        //           + 1 mask register.
12        RegId srcRegIdxArr[1 + 8 + 1];
13        // Destinations: up to 8 vector regs starting at Vd
14        RegId destRegIdxArr[8];
15
16    public:
17        /// Constructor.
18        %(class_name)s(MachInst machInst, RiscvISA::VTYPE vtype, uint32_t vl);
19        Fault execute(ExecutionContext *, Trace::InstRecord *) const override;
20        using %(base_class)s::generateDisassembly;
21    };
22 }};

```

Format示例

```

1 def format VectorIndexedMemLoadOp(code, *opt_flags) {{
2     iop = InstObjParams(name, Name, 'VectorIndexedMemLoadOp', code, opt_flags)
3     header_output = VectorIndexedMemLoadDeclare.subst(iop)
4     decoder_output = VectorIndexedMemLoadConstructor.subst(iop)
5     decode_block = VectorDecode.subst(iop)
6     exec_output = VectorIndexedMemLoadExecute.subst(iop)
7 }};

```

上述模板函数，通过isa_parse.py解析

(1) 向量操作数

```
1 Path: build/RISCV/arch/isa_parser/operand_types.py
```

```

1
2 class VecElemOperand(Operand):
3     reg_class = 'VecElemClass'
4
5     def isReg(self):
6         return 1
7

```

```

8     def isVecElem(self):
9         return 1
10
11    def makeDecl(self):
12        if self.is_dest and not self.is_src:
13            return '\n\t%s %s;' % (self.ctype, self.base_name)
14        else:
15            return ''
16
17    def makeConstructor(self, predRead, predWrite):
18        c_src = ''
19        c_dest = ''
20
21        numAccessNeeded = 1
22
23        if self.is_src:
24            c_src = ('\n\tsetSrcRegIdx(_numSrcRegs++, RegId(%s, %s, %s));' %
25                      (self.reg_class, self.reg_spec, self.elem_spec))
26
27        if self.is_dest:
28            c_dest = ('\n\tsetDestRegIdx(_numDestRegs++, RegId(%s, %s, %s));' %
29                      (self.reg_class, self.reg_spec, self.elem_spec))
30            c_dest += '\n\t_numVecElemDestRegs++;'
31
32        return c_src + c_dest
33
34    def makeRead(self, predRead):
35        c_read = 'xc->readVecElemOperand(this, %d)' % self.src_reg_idx
36
37        if self.ctype == 'float':
38            c_read = 'bitsToFloat32(%s)' % c_read
39        elif self.ctype == 'double':
40            c_read = 'bitsToFloat64(%s)' % c_read
41
42        return '\n\t%s %s = %s;\n' % (self.ctype, self.base_name, c_read)
43
44    def makeWrite(self, predWrite):
45        if self.ctype == 'float':
46            c_write = 'floatToBits32(%s)' % self.base_name
47        elif self.ctype == 'double':
48            c_write = 'floatToBits64(%s)' % self.base_name
49        else:
50            c_write = self.base_name
51
52        c_write = ('\n\txc->setVecElemOperand(this, %d, %s);' %
53                   (self.dest_reg_idx, c_write))
54
55        return c_write

```

3.4 指令译码

1 Path: src/arch/riscv/isa/decoder isa

`isa_parse.py` 解析后缀名为`*.isa` 的文件，自动生成可执行的C 程序

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Visual Studio Code - decoder.isa - gem5 - Visual Studio Code
- File Explorer (Left):** Shows the project structure under the GEMS folder, including files like se.py, decoder.cc, decoder.h, faults.cc, faults.h, and fp_inst.h.
- Editor (Center):** Displays the content of the decoder.isa file. The code is a C++ class definition for a RISC-V ISA decoder. It includes sections for decoding OPCODEs (0x3, 0x03), FUNCT3 (0x00, 0x01, 0x02, 0x03), MOP (0x0, 0x6, 0x7), and LUMOP. It also handles compressed store operations and various memory load operations (VectorIndexedMemLoadOp, VectorStridedMemLoadOp).
- Right Side:** Includes a vertical navigation bar with icons for Generate, Simulate, and other tools, along with a detailed code search and navigation interface.

File Edit Selection View Go Run Terminal Help

decoder.isa - gem5 - Visual Studio Code

```
format FPROp { - }
```

```
0x15: decode FUNCT3 { - }
```

```
    0x26: VectorVdVs2S1Op::vmlahs_vx();
```

```
    0x27: VectorVdVs2S1Op::vmulh_vx({ - }
```

```
        _int128 t src1 = sext(Vs2[regElemID], sewb * 8);
```

```
        _int128 t src2 = sext(Rs1_ud, sewb * 8);
```

```
        Vd_vc[regElemID] =
```

```
            (_int128_t)src1 * src2) >> (sewb * 8);
```

```
});
```

```
0x29: VectorVdVs2S1Op::vmmadd_vx({ - }
```

```
    Vd_vc[regElemID] = Vd_vc[regElemID] * Rs1_ud + Vs2[regElemID];
```

```
});
```

```
0x2b: VectorVdVs2S1Op::vmmsub_vx({ - }
```

```
    Vd_vc[regElemID] = -(Rs1_ud + Vd_vc[regElemID]) + Vs2_vc[regElemID];
```

```
});
```

```
0x2d: VectorVdVs2S1Op::vnmacc_vx({ - }
```

```
    Vd_vc[regElemID] = Rs1_ud * Vs2[regElemID] + Vd_vc[regElemID];
```

```
});
```

```
0x2f: VectorVdVs2S1Op::vnmssac_vx({ - }
```

```
    Vd_vc[regElemID] = -(Rs1_ud + Vs2_vc[regElemID]) + Vd_vc[regElemID];
```

```
});
```

```
0x30: VectorWideningVXOp::vwadd_vx({ - }
```

```
    Vd_vc[destRegElemID] = Rs1_ud + Vs2_vc[srcRegElemID];
```

```
});
```

```
0x31: VectorWideningVXOp::vwadd_vx();
```

```
0x32: VectorWideningVXOp::vwsub_vx({ - }
```

```
    Vd_vc[destRegElemID] = Vs2_vc[srcRegElemID] - Rs1_ud;
```

```
});
```

```
0x33: VectorWideningVXOp::vwsub_vx();
```

```
0x34: VectorWideningVXOp::vwaddu_wx();
```

```
0x35: VectorWideningVXOp::vwadd_wx();
```

```
0x36: VectorWideningVXOp::vwsubu_wx();
```

```
0x37: VectorWideningVXOp::vwsub_wx();
```

```
0x38: VectorWideningVXOp::vmmulu_vx({ - }
```

```
    Vd_vc[destRegElemID] = Rs1_ud * Vs2[srcRegElemID];
```

```
});
```

```
0x3a: VectorWideningVXOp::vmmulsu_vx({ - }
```

```
    _int128 t src1 = sext(Vs2[srcRegElemID], sewb * 8);
```

```
    Vd_vc[destRegElemID] = Rs1_ud * src1;
```

```
});
```

```
0x3b: VectorWideningVXOp::vmmul_vx({ - }
```

```
    _int128 t src1 = sext(Vs2[srcRegElemID], sewb * 8);
```

```
    _int128 t src2 = sext(Rs1_ud, sewb * 8);
```

```
    Vd_vc[destRegElemID] = src2 * src1;
```

```
});
```

```

1 // vle8_v 译码示例
2     0x01: decode FUNCT3 {
3         0x0: decode MOP {
4             0x0: decode LUMOP {
5                 0x0: VectorUnitStrideMemLoadOp::vle8_v({{
6                     uint64_t target_address = Rs1_ud + offset;
7                     if (target_address % eewb) {
8                         // Alignment fault.
9                         return std::make_shared<AddressFault>(target_address
10                            ExceptionCode::L
11                        }
12
13                     fault =
14                         readMemAtomicLE(xc,
15                             traceData,
16                             target_address,
17                             Vd_vc[regElemID],
18                             memAccessFlags);
19                     if (fault != NoFault) {
20                         return fault;
21                     }
22                 }});

```

3.5 指令生成

RW ISA 实现: formats

```

def format VIntOp(code, exception_code={}, *opt_flags) {{
    iop = InstObjParams(name, Name, 'VOp', {'code':code,
    'exception_code':exception_code}, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop)
    exec_output = VectorIntExecute.subst(iop)
}};

```

```

    0x0: decode FUNCT6 {
        format VIntOp {
            0x0: vadd vv({{
                for (unsigned i = 0; i < eCount; i++) {
                    Vd_vi[i] = Vs1_vi[i] + Vs2_vi[i];
                }
            }});

```

```

def template VectorIntExecute {{
    Fault %(class_name)s::execute(ExecutionContext *xc,
        Trace::InstRecord *traceData) const
    {
        Fault fault = NoFault;

        unsigned eCount = getVecLen(xc->tcBase());
        unsigned eWidth = getVecSew(xc->tcBase());

        switch (eWidth) {
            case 8: {
                using vi = int8_t;
                %(op_decl)s;
                %(op_rd)s;
                if (fault != NoFault) return fault;
                %(code)s;
                if (fault != NoFault) return fault;
                %(op_wb)s;
                break;
            }
            case 16: {
                using vi = int16_t;
                %(op_decl)s;
                %(op_rd)s;
                if (fault != NoFault) return fault;
                %(code)s;
                if (fault != NoFault) return fault;
                %(op_wb)s;
                break;
            }
            case 32: {

```

RW ISA 实现：自动生成的c++代码

```
... Fault::Vadd_vv::execute(ExecutionContext *xc,
...     Trace::InstRecord *traceData) const
...
... {
...     Fault::fault = NoFault;
...
...     if (fault != NoFault) return fault;
...
...     unsigned eCount = getVecLen(xc->tcBase());
...     unsigned eWidth = getVecSew(xc->tcBase());
...
...     switch (eWidth) {
...         case 8: {
...             using vi = int8_t;
...
...             TheISA::VecRegContainer& tmp_d0 = xc->getWritableDatabaseRegOperand(this, 0);
...             auto Vd = tmp_d0.as<vi>();
...             const TheISA::VecRegContainer& tmp_s0 = xc->readVecRegOperand(this, 0);
...             auto Vs1 = tmp_s0.as<vi>();
...             const TheISA::VecRegContainer& tmp_s1 = xc->readVecRegOperand(this, 1);
...             auto Vs2 = tmp_s1.as<vi>();
...
...             if (fault != NoFault) return fault;
...
...             for (unsigned i = 0; i < eCount; i++) {
...                 Vd[i] = Vs1[i] + Vs2[i];
...             }
...         }
...     }
... }
```

3.6 指令执行

1 Path: `src/arch/riscv/insts/vector.cc`

通过脚本自动生成的可执行文件

1 build/RISCV/arch/riscv/generated/exec-ns.cc.inc

注意：该文件下的内容不能改

3.6.1 配置指令

配置指令译码

1 path: `src/arch/riscv/decoder.hh`

```
1 public:
2     Decoder(const RiscvDecoderParams &p) : InstDecoder(p, &machInst)
3     {
4         reset();
```

```

5      }
6
7      void reset() override;
8
9      inline bool compressed(ExtMachInst inst) { return (inst & 0x3) < 0x3; }
10
11     //Use this to give data to the decoder. This should be used
12     //when there is control flow.
13     void moreBytes(const PCStateBase &pc, Addr fetchPC) override;
14
15     void setVl(uint32_t new_vl) { mach_vl = new_vl; }
16     void setVtype(RiscvISA::VTYPE new_vtype) { mach_vtype = new_vtype; }
17
18     uint32_t getVl() { return mach_vl; }
19     RiscvISA::VTYPE getVtype() { return mach_vtype; }
20 };

```

1 Path: src/arch/riscv/decoder.cc

```

1 bool isVConfigInst(ExtMachInst inst)
2 {
3     // Looks for VSETVLI
4     uint64_t opcode = bits(inst, 6, 0);
5     uint64_t width = bits(inst, 14, 12);
6     return opcode == 0b1010111u && width == 0b111u;
7 }

```

执行过程

1 Path: src/arch/riscv/insts/vector.cc

```

1 float
2 getVflmul(uint32_t vlmul_encoding) {
3     int vlmul = int8_t(vlmul_encoding << 5) >> 5;
4     float vflmul = vlmul >= 0 ? 1 << vlmul : 1.0 / (1 << -vlmul);
5     return vflmul;
6 }

```

```

7
8 uint32_t
9 getVlmax(VTYPE vtype, uint32_t vlen) {
10    uint32_t sew = getSew(vtype.vsew);
11    uint32_t vlmax = (vlen/sew) * getVflmul(vtype.vlmul);
12    return vlmax;
13 }

```

```

1 uint32_t
2 setVsetvlCSR(ExecContext *xc,
3                 uint32_t rd_bits,
4                 uint32_t rs1_bits,
5                 uint32_t requested_vl,
6                 uint32_t requested_vtype) {
7     VTYPE new_vtype = requested_vtype;
8
9     uint32_t vlen = xc->readMiscReg(MISCREG_VLENB) * 8;
10
11    uint32_t vlmax = getVlmax(xc->readMiscReg(MISCREG_VTYPE), vlen);
12
13    auto tc = xc->tcBase();
14
15    if (xc->readMiscReg(MISCREG_VTYPE) != new_vtype) {
16        vlmax = getVlmax(new_vtype, vlen);
17
18        float vflmul = getVflmul(new_vtype.vlmul);
19
20        uint32_t sew = getSew(new_vtype.vsew);
21
22        uint32_t new_vill =
23            !(vflmul >= 0.125 && vflmul <= 8) ||
24            sew > std::min(vflmul, 1.0f) * ELEN ||
25            bits(requested_vtype, 30, 8) != 0;
26        if (new_vill) {
27            vlmax = 0;
28            new_vtype = 0;
29            new_vtype.vill = 1;
30        }
31
32        xc->setMiscReg(MISCREG_VTYPE, new_vtype);
33
34        tc->getDecoderPtr()->as<Decoder>().setVtype(new_vtype);
35    }

```

```

36
37 // Set vl
38 uint32_t current_vl = xc->readMiscReg(MISCREG_VL);
39 uint32_t vl = 0;
40 if (vlmax == 0) {
41     vl = 0;
42 } else if (rd_bits == 0 && rs1_bits == 0) {
43     vl = current_vl > vlmax ? vlmax : current_vl;
44 } else if (rd_bits != 0 && rs1_bits == 0) {
45     vl = vlmax;
46 } else if (rs1_bits != 0) {
47     vl = requested_vl > vlmax ? vlmax : requested_vl;
48 }
49
50 xc->setMiscReg(MISCREG_VL, vl);
51 tc->getDecoderPtr()->as<Decoder>().setVl(vl);
52
53 return vl;
54 }
55

```

3.6.2 运算指令

(1) 向量舍入模式，分为无符号和有符号两种

xrm 为 RVV 定义的舍入模式

result 为运算结果

gb

```

1 void roundUnsignedInteger(__uint128_t &result, uint32_t xrm, int gb) {
2     do {
3         const uint64_t lsb = 1UL << (gb);
4         const uint64_t lsb_half = lsb >> 1;
5         switch (xrm) {
6             case VectorRoundingMode::RoundToNearestUp:
7                 result += lsb_half;
8                 break;
9             case VectorRoundingMode::RoundToNearestEven:
10                if ((result & lsb_half) && ((result & (lsb_half - 1)) || (result & lsb)))
11                    result += lsb;
12                break;
13            case VectorRoundingMode::RoundDown:
14                break;
15            case VectorRoundingMode::RoundToOdd:
16                break;
17        }
18    }
19}

```

```

17     if (result & (lsb - 1)) {
18         result |= lsb;
19     }
20     break;
21 default:
22     printf("error: unknown vector rounding mode\n");
23     exit(1);
24 }
25 } while (0);
26 }
27

```

```

1 void roundSignedInteger(__int128_t &result, uint32_t xrm, int gb) {
2     do {
3         const uint64_t lsb = 1UL << (gb);
4         const uint64_t lsb_half = lsb >> 1;
5         switch (xrm) {
6             case VectorRoundingMode::RoundToNearestUp:
7                 result += lsb_half;
8                 break;
9             case VectorRoundingMode::RoundToNearestEven:
10                if ((result & lsb_half) && ((result & (lsb_half - 1)) || (result & lsb)))
11                    result += lsb;
12                }
13                break;
14             case VectorRoundingMode::RoundDown:
15                 break;
16             case VectorRoundingMode::RoundToOdd:
17                 if (result & (lsb - 1)) {
18                     result |= lsb;
19                 }
20                 break;
21             default:
22                 printf("error: unknown vector rounding mode\n");
23                 exit(1);
24             }
25 } while (0);
26 }

```

以上函数会被向量需要做舍入的指令调用

3.6.3 Widen/Narrow指令

3.6.4 Stride 访存

3.6.5 Index 访存

- (1) 读 VL, Vtype 寄存器
- (2) 根据Vtype 中SEW 的值和操作码字段FUNCT3 确定EEW
- (3) 根据Vtype 中LMUL的值, VL , EEW 计算操作元素的个数
- (4) 根据VM 的值确定目的寄存器的VMask 比特
- (5) 根据EEW, 读出向量寄存器的值, 计算偏移量
- (6) 译码时, 通过脚本将 (code)s 的内容替换, 调用readMemAtomicLE获取load_value
- (7) 将load_value 的值放到对应的目的寄存器里

```
1 def template VectorIndexedMemLoadExecute {{
2     Fault
3     %(class_name)s::execute(ExecContext *xc,
4         Trace::InstRecord *traceData) const
5     {
6         Fault fault = NoFault;
7
8         uint32_t vl = xc->readMiscReg(MISCREG_VL);
9         VTYPE vtype = xc->readMiscReg(MISCREG_VTYPE);
10        size_t sewb = getSew(vtype.vsew) / 8;
11
12        uint32_t eewb = 0;
13        switch (FUNCT3) {
14            case 0:
15                eewb = 1;
16                break;
17            case 5:
18                eewb = 2;
19                break;
20            case 6:
21                eewb = 4;
22                break;
23            case 7:
24                eewb = 8;
25                break;
26            default:
27                std::string error = csprintf(
28                    "Illegal eew value: 0x%x\n", FUNCT3);
29                return std::make_shared<IllegalInstFault>(error, machInst);
30                break;
31        }
```

```

32
33     float vemul = ((float) eewb / sewb) * getVflmul(vtype.vlmul);
34
35     if ((vemul < 0.125) || (vemul > 8)) {
36         std::string error = csprintf(
37             "Illegal vemul value: %f\n", vemul);
38         return std::make_shared<IllegalInstFault>(error, machInst);
39     }
40
41     %(op_decl)s;
42
43     Rs1 = xc->readIntRegOperand(this, 0);
44
45     size_t num_elements_in_src_reg = (VecRegSizeBytes/eewb);
46     uint32_t srcVecRegID = 0;
47     uint32_t srcVecRegElemID = 0;
48
49     for (uint32_t destVecRegID = 0; destVecRegID < _numVecDestRegs; ++destVe
50         TheISA::VecRegContainer& Vd_container =
51             xc->getWritableDatabaseVecRegOperand(this, destVecRegID);
52
53     uint32_t vmask = UINT32_MAX;
54     if (VM == 0) {
55         const TheISA::VecRegContainer& Vmask_container =
56             xc->readVecRegOperand(this, _numSrcRegs - 1);
57         switch(sewb) {
58             case 1:
59                 vmask = Vmask_container.as<uint32_t>()[destVecRegID];
60                 break;
61             case 2:
62                 vmask = Vmask_container.as<uint16_t>()[destVecRegID];
63                 break;
64             case 4:
65                 vmask = Vmask_container.as<uint8_t>()[destVecRegID];
66                 break;
67             case 8:
68                 vmask = Vmask_container.as<uint8_t>()[destVecRegID/2];
69                 vmask = (vmask >> (destVecRegID % 2));
70                 break;
71         }
72     }
73
74     size_t num_elements_in_dest_reg =
75         (VecRegSizeBytes/sewb) > vl ? vl : (VecRegSizeBytes/sewb);
76
77     if (fault == NoFault) {
78         if (sewb == 1) {

```

```

79             auto Vd = Vd_container.as<uint8_t>();
80             for (uint32_t regElemID = 0; regElemID < num_elements_in_des
81                 if (bits(vmask, regElemID, regElemID) == 0) {
82                     continue;
83                 }
84
85             const TheISA::VecRegContainer& Vs2_container =
86                 xc->readVecRegOperand(this, srcVecRegID + 1);
87
88             uint64_t offset = 0;
89             switch (eewb) {
90                 case 1:
91                     offset = Vs2_container.as<uint8_t>()[srcVecRegEl
92                     break;
93                 case 2:
94                     offset = Vs2_container.as<uint16_t>()[srcVecRegE
95                     break;
96                 case 4:
97                     offset = Vs2_container.as<uint32_t>()[srcVecRegE
98                     break;
99                 case 8:
100                     offset = Vs2_container.as<uint64_t>()[srcVecRegE
101                     break;
102                 }
103
104             uint8_t load_value;
105
106             %(code)s;
107
108             ++srcVecRegElemID;
109             if (srcVecRegElemID == num_elements_in_src_reg) {
110                 srcVecRegElemID = 0;
111                 ++srcVecRegID;
112             }
113         }
114     } else if (sewb == 2) {
115         auto Vd = Vd_container.as<uint16_t>();
116         for (uint32_t regElemID = 0; regElemID < num_elements_in_des
117             if (bits(vmask, regElemID, regElemID) == 0) {
118                 continue;
119             }
120
121             const TheISA::VecRegContainer& Vs2_container =
122                 xc->readVecRegOperand(this, srcVecRegID + 1);
123
124             uint64_t offset = 0;
125             switch (eewb) {

```

```

126             case 1:
127                 offset = Vs2_container.as<uint8_t>()[srcVecRegEl
128                     break;
129             case 2:
130                 offset = Vs2_container.as<uint16_t>()[srcVecRegE
131                     break;
132             case 4:
133                 offset = Vs2_container.as<uint32_t>()[srcVecRegE
134                     break;
135             case 8:
136                 offset = Vs2_container.as<uint64_t>()[srcVecRegE
137                     break;
138     }
139
140     uint16_t load_value;
141
142     %(code)s;
143
144     ++srcVecRegElemID;
145     if (srcVecRegElemID == num_elements_in_src_reg) {
146         srcVecRegElemID = 0;
147         ++srcVecRegID;
148     }
149 }
150 } else if (sewb == 4) {
151     auto Vd = Vd_container.as<uint32_t>();
152     for (uint32_t regElemID = 0; regElemID < num_elements_in_des
153         if (bits(vmask, regElemID, regElemID) == 0) {
154             continue;
155         }
156
157         const TheISA::VecRegContainer& Vs2_container =
158             xc->readVecRegOperand(this, srcVecRegID + 1);
159
160         uint64_t offset = 0;
161         switch (eewb) {
162             case 1:
163                 offset = Vs2_container.as<uint8_t>()[srcVecRegEl
164                     break;
165             case 2:
166                 offset = Vs2_container.as<uint16_t>()[srcVecRegE
167                     break;
168             case 4:
169                 offset = Vs2_container.as<uint32_t>()[srcVecRegE
170                     break;
171             case 8:
172                 offset = Vs2_container.as<uint64_t>()[srcVecRegE

```

```

173                         break;
174                     }
175
176                     uint32_t load_value;
177
178                     %(code)s;
179
180                     ++srcVecRegElemID;
181                     if (srcVecRegElemID == num_elements_in_src_reg) {
182                         srcVecRegElemID = 0;
183                         ++srcVecRegID;
184                     }
185                 }
186             } else if (sewb == 8) {
187                 auto Vd = Vd_container.as<uint64_t>();
188                 for (uint32_t regElemID = 0; regElemID < num_elements_in_des
189                     if (bits(vmask, regElemID, regElemID) == 0) {
190                         continue;
191                     }
192
193                     const TheISA::VecRegContainer& Vs2_container =
194                         xc->readVecRegOperand(this, srcVecRegID + 1);
195
196                     uint64_t offset = 0;
197                     switch (eewb) {
198                         case 1:
199                             offset = Vs2_container.as<uint8_t>()[srcVecRegEl
200                             break;
201                         case 2:
202                             offset = Vs2_container.as<uint16_t>()[srcVecRegE
203                             break;
204                         case 4:
205                             offset = Vs2_container.as<uint32_t>()[srcVecRegE
206                             break;
207                         case 8:
208                             offset = Vs2_container.as<uint64_t>()[srcVecRegE
209                             break;
210                     }
211
212                     uint64_t load_value;
213
214                     %(code)s;
215
216                     ++srcVecRegElemID;
217                     if (srcVecRegElemID == num_elements_in_src_reg) {
218                         srcVecRegElemID = 0;
219                         ++srcVecRegID;

```

```

220                     }
221                 }
222             } else {
223                 std::string error = csprintf(
224                     "Unexpected sewb value in instruction");
225                 return std::make_shared<IllegalInstFault>(error, machInst);
226             }
227
228         if (fault == NoFault) {
229             if (traceData) {
230                 // TODO: Print all destination registers
231                 traceData->setData(Vd_container);
232             }
233         }
234
235         vl = vl - num_elements_in_dest_reg;
236     }
237 }
238
239     return fault;
240 }
241 }};

```

3.6.6 Whole-register 访问

3.6.7 特殊指令

4 GEM5 程序执行

4.1 单元测试

[如何运行 gem5 单元测试? | 那些遇到过的问题](#)

4.2 SE 模式

(1) se.py 测试

SE模式需要注意，使用静态编译，单线程

step 1 新建一个自己的文档

然后在文件夹中写自己想要测试的程序

```
1 #include <stdio.h>
2
3 int main(){
4     int x,y,count;
5     x = 3; y = 4;
6     count = x + y;
7     printf("today, your lucky number is %d\n",count);
8 }
```

step 2 静态编译自己的.c文件

```
1 gcc -o hello hello.c -static
2 //NOTE:
3 //if you would like to test RISCV at SE mode
4 //pls install risc-v compiler: gcc or llvm
```

-o为指定输出文件的名字，如果不带-o，命令则变为gcc hello.c，默认会生成名为a.out的可执行文件

-static表示静态编译，如果不使用静态编译，那么gcc就会优先使用动态库进行编译，动态编译则会动态链接使用的库文件

动态编译和静态编译的区别：动态在程序运行时被链接，静态库直接在编译时所用到的库文件链接进了可执行文件，所以静态编译出来的复杂程序会大很多

step 3 测试自己的.c文件

输入

```
1 sudo build/X86/gem5.opt configs/example/se.py -c filepath/hello
```

输出结果: 此处结果直接使用了已经用riscv 编译器生成好的二进制可执行文件

```
1 yuanmiaomiao@yuanmiaomiao-ThinkCentre-M860t-D065:~/develop/gem5$ 
2 ./build/RISCV/gem5.opt configs/example/se.py -c
3 tests/test-progs/hello/bin/riscv/linux/hello -o 'abcd'
4 gem5 Simulator System. http://gem5.org
5 gem5 is copyrighted software; use the --copyright option for details.
6
```

```

7 gem5 version 21.2.1.0
8 gem5 compiled Jun  5 2023 21:34:54
9 gem5 started Jun 11 2023 00:11:50
10 gem5 executing on yuanmiaomiao-ThinkCentre-M860t-D065, pid 299262
11 command line: ./build/RISCV/gem5.opt configs/example/se.py
12 -c tests/test-progs/hello/bin/riscv/linux/hello -o abcd
13
14 Global frequency set at 1000000000000 ticks per second
15 warn: No dot file generated. Please install pydot to generate the dot file and p
16 build/RISCV/mem/mem_interface.cc:791: warn:
17 DRAM device capacity (8192 Mbytes) does not match the address range assigned
18 (512 Mbytes)
19 0: system.remote_gdb: listening for remote gdb on port 7000
20 **** REAL SIMULATION ****
21 build/RISCV/sim/simulate.cc:194: info: Entering event queue @ 0.
22 Starting simulation...
23 build/RISCV/sim/mem_state.cc:443: info: Increasing stack size by one page.
24 Hello world!
25

```

(2) cpu 模型测试

```

1 /build/RISCV/gem5.opt tests/gem5/cpu_tests/test.py
2 gem5 Simulator System. http://gem5.org
3 gem5 is copyrighted software; use the --copyright option for details.
4
5 gem5 version 21.2.1.0
6 gem5 compiled Jun  5 2023 21:34:54
7 gem5 started Jun 11 2023 00:30:50
8 gem5 executing on yuanmiaomiao-ThinkCentre-M860t-D065, pid 300232
9 command line: ./build/RISCV/gem5.opt tests/gem5/cpu_tests/test.py
10
11 ModuleNotFoundError: No module named 'testlib'
12
13 At:
14     tests/gem5/cpu_tests/test.py(44): <module>
15

```

4.3 FS 模式

