实验一 安装 gem5

操作系统 Ubuntu 16.04 以上
依赖库

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev
libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-
perftools-dev python-dev python
```

下载源码

```
git clone https://gem5.googlesource.com/public/gem5
```

或 `git clone` https://github.com/gem5/gem5.git
编译

```
scons build/X86/gem5.opt -j9
```

X86 代表 cpu 类型
-j9 用 9 个线程去编译，自己计算机有几个核心就设几个
如果正常，应该显示如下结果

```
Checking for C header file Python.h... yes
Checking for C library pthread... yes
Checking for C library dl... yes
Checking for C library util... yes
Checking for C library m... yes
Checking for C library python2.7... yes
Checking for accept(0,0,0) in C++ library None... yes
Checking for zlibVersion() in C++ library z... yes
Checking for GOOGLE_PROTOBUF_VERIFY_VERSION in C++ library
protobuf... yes
Checking for clock_nanosleep(0,0,NULL,NULL) in C library
None... yes
Checking for timer_create(CLOCK_MONOTONIC, NULL, NULL) in C
library None... no
Checking for timer_create(CLOCK_MONOTONIC, NULL, NULL) in C
library rt... yes
Checking for C library tcmalloc... yes
Checking for backtrace_symbols_fd((void*)0, 0, 0) in C library
None... yes
Checking for C header file fenv.h... yes
Checking for C header file linux/kvm.h... yes
Checking size of struct kvm_xsave ... yes
Checking for member exclude_host in struct
perf_event_attr...yes
```

```
Building in /local.chinook/gem5/gem5-tutorial/gem5/build/X86
Variables file /local.chinook/gem5/gem5-
tutorial/gem5/build/variables/X86 not found,
  using defaults in /local.chinook/gem5/gem5-
tutorial/gem5/build_opts/X86
scons: done reading SConscript files.
scons: Building targets ...
 [ISA DESC] X86/arch/x86/isa/main.isa -> generated/inc.d
 [NEW DEPS] X86/arch/x86/generated/inc.d -> x86-deps
 [ENVIRONS] x86-deps -> x86-environs
 [     CXX] X86/sim/main.cc -> .o
 ....
 .... <lots of output>
 ....
 [   SHCXX] nomali/lib/mali_midgard.cc -> .os
 [   SHCXX] nomali/lib/mali_t6xx.cc -> .os
 [   SHCXX] nomali/lib/mali_t7xx.cc -> .os
 [      AR]  -> drampower/libdrampower.a
 [   SHCXX] nomali/lib/addrspace.cc -> .os
 [   SHCXX] nomali/lib/mmu.cc -> .os
 [  RANLIB]  -> drampower/libdrampower.a
 [   SHCXX] nomali/lib/nomali_api.cc -> .os
 [      AR]  -> nomali/libnomali.a
 [  RANLIB]  -> nomali/libnomali.a
 [     CXX] X86/base/date.cc -> .o
 [    LINK]  -> X86/gem5.opt
scons: done building targets.
```

实验二 创建基本 gem5 配置脚本并运行

首先创建脚本文件

```
mkdir configs/tutorial
touch configs/tutorial/simple.py
```

导入 gem5 包

```
import m5
from m5.objects import *
```

创建系统对象

```
system = System()
```

设置主频等参数

```
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()
```

设置内存大小

```
system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]
```

创建 CPU， *TimingSimpleCPU* 是最简单的 CPU 模型，该模型每个时钟周期执行一条指令

```
system.cpu = TimingSimpleCPU()
```

创建内存总线

```
system.membus = SystemXBar()
```

创建 cache

system.cpu.icache_port = system.membus.slave

system.cpu.dcache_port = system.membus.slave

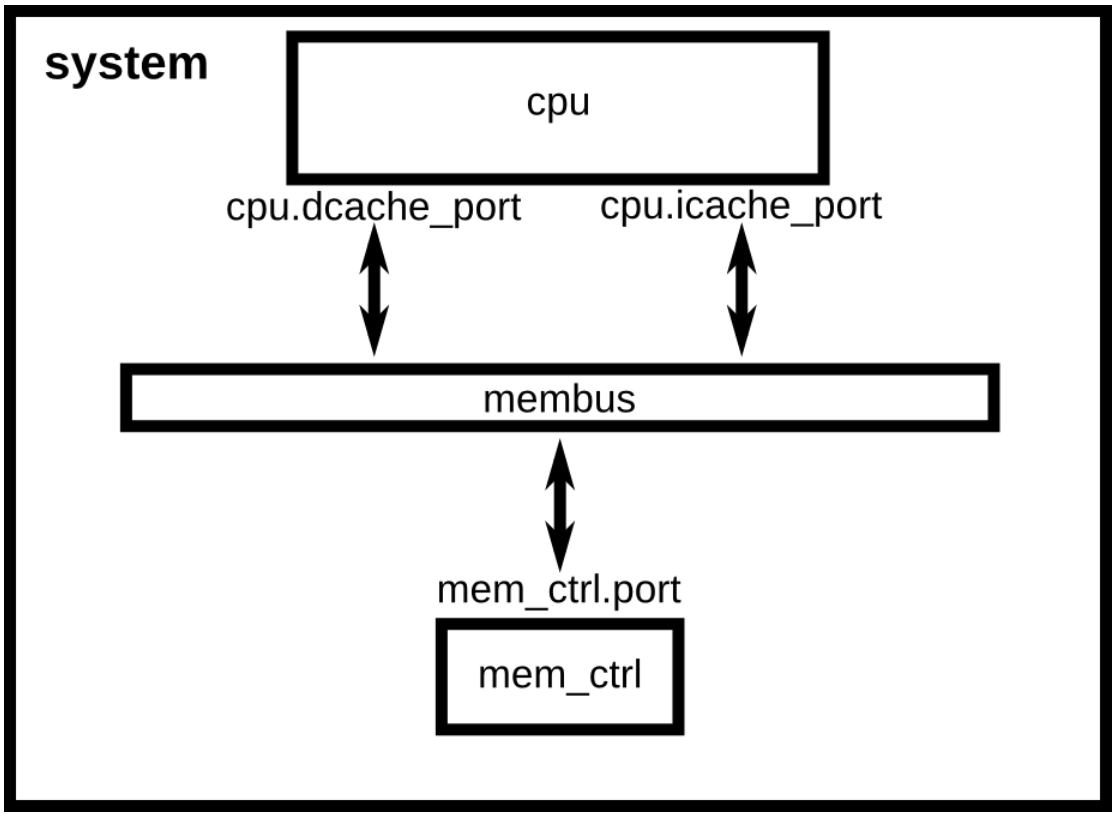
创建 I/O 控制器等，连接内存总线

```
system.cpu.createInterruptController()
system.cpu.interrupts[0].pio = system.membus.master
system.cpu.interrupts[0].int_master = system.membus.slave
system.cpu.interrupts[0].int_slave = system.membus.master
system.system_port = system.membus.slave
```

创建内存控制器

```
system.mem_ctrl = DDR3_1600_8x8()
system.mem_ctrl.range = system.mem_ranges[0]
system.mem_ctrl.port = system.membus.master
```

做完上述步骤后，我们创建了如下图所示的计算机系统结构



通过系统调用模式执行测试程序，
tests/test-progs/hello/bin/x86/linux/hello 是已编译好的测试程序

```
process = Process()
process.cmd = ['tests/test-progs/hello/bin/x86/linux/hello']
system.cpu.workload = process
system.cpu.createThreads()
```

最后，初始化系统

```
root = Root(full_system = False, system = system)
m5.instantiate()
```

执行仿真程序

```
print("Beginning simulation!")
exit_event = m5.simulate()
```

仿真结束后，输出状态信息

```
print('Exiting @ tick {} because {}'
      .format(m5.curTick(), exit_event.getCause()))
```

运行上述仿真程序

```
build/X86/gem5.opt configs/tutorial/simple.py
```

应该得到以下运行结果

```
gem5 Simulator System.  http://gem5.org
```

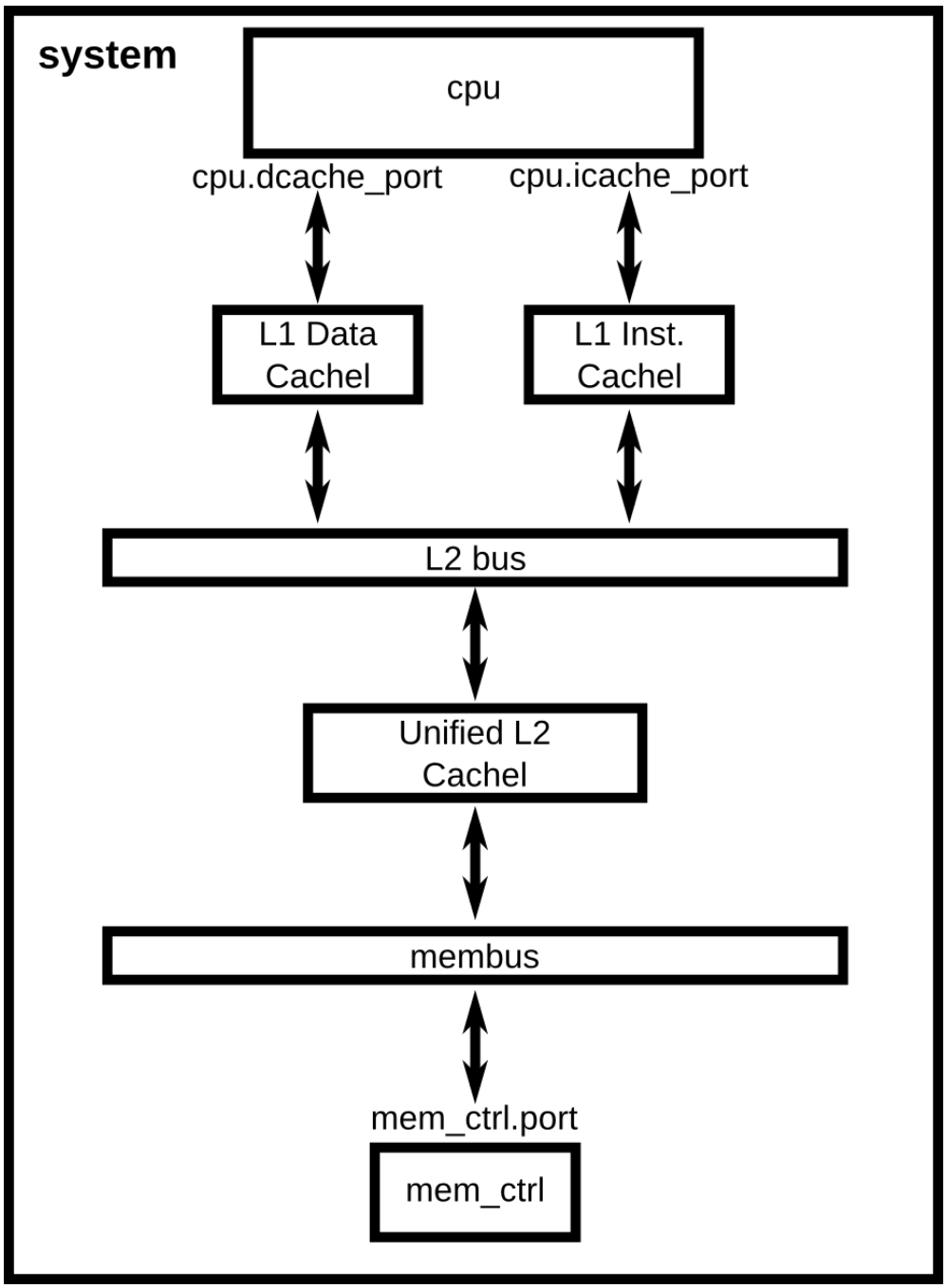
```
gem5 is copyrighted software; use the --copyright option for
details.

gem5 compiled Mar 16 2018 10:24:24
gem5 started Mar 16 2018 15:53:27
gem5 executing on amarillo, pid 41697
command line: build/X86/gem5.opt configs/tutorial/simple.py

Global frequency set at 1000000000000 ticks per second
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!
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 507841000 because exiting with last active
thread context
```

实验内容：将 hello world 替换为生成 10000 以内的斐波那契数列。

实验三 构建 L2 Cache



创建如上图所示包含 L2cache 计算机系统，代码可参考 gem5 目录下 configs/learning_gem5/part1/two_level.py

实验内容：实现 L3，三级 cache

实验四 Full system 模拟

在 gem5 目录下创建目录 full_system_images

将 linux 镜像下载到 full_system_images 下

下载地址 http://www.m5sim.org/dist/current/x86/x86-system.tar.bz2

解压文件 tar -jxvf x86-system.tar.bz2

修改环境变量

vim ~/.bashrc

在最后添加

export "M5_PATH=$M5_PATH:/home/wlx/workspace/gem5/full_system_images/"

（要设置为你自己的路径）

:wq(保存退出)

source ~/.bashrc（使设置生效）

echo $M5_PATH 查看是否修改成功

启动全仿真

build/X86/gem5.opt configs/example/fs.py --kernel=x86_64-vmlinux-2.6.22.9 --

disk-image=linux-x86.img

以上只是启动了全系统仿真，下来还要利用 term 进入该系统内

切换目录 cd util/term/

编译 make

sudo make install
进入系统

m5term 127.0.0.1 3456

实验五 综合设计

自己针对某种计算机架构进行仿真，并编写有实际意义的测试程序。(例如，ARM 架构手机系统；x86 架构服务器系统）(根据实现程度给分）

复现任一篇论文实验代码 http://www.m5sim.org/Publications#2018（可得 A+）