sim-outorder 分析调研报告

2014211304 班

D 组

史文翰 2014211218

林宇辰 2014211223

王剑督 2014211228

崔嘉伟 2014211233

杨　莹 2014211238

徐丹雅 2014211243

郝绍明 2014210123

分工情况

史文翰 2014211218：负责整理、分析官方 sim-outorder 文档，画全局流程图，解释部分参数、过程，整合最后文档

林宇辰 2014211223：运行并分析了 factorial 程序在模拟器下的行为和参数

王剑督 2014211228：运行并分析了 helloworld 程序在模拟器下的行为和参数

崔嘉伟 2014211233：运行并分析了 helloworld 程序在模拟器下的行为和参数

杨　莹 2014211238：分析了部分官方文档，并针对具体程序作了解析

徐丹雅 2014211243： 对模拟器的参数配置、资源配置和统计变量作了具体的分析

史文翰部分

一、 功能简介

生成一个非常详细的乱序发射过程的时间统计，这个统计基于有两级 cache 和主存的处理器内核。

二、 设置项

-fetch:ifqsize <size> - 指令队列的容量，以指令数量为单位
-fetch:mplat <cycles> - 额外的失败分支预测信息
-bpred <type> - 显式指定分支预测器
-decode:width <insts> - 指令译码器的带宽（单位为单位周期内解码的指令数）
-issue:width <insts> - RUU 带宽（单位同上）
-issue:inorder – 限制指令的发射顺序为定序的
-issue:wrongpath – 允许在预测失败之后继续进行指令相关的工作
-ruu:size <insts> - RUU 的容量（单位为指令数）
-lsq:size <insts> - load/store 指令队列的容量（单位同上）
-cache:dl1 <config> - L1 数据缓存的设置
-cache:dl1lat <cycles> - L1 数据缓存的延迟时间
-cache:dl2 <config> - L2 数据缓存的设置
-cache:dl2lat <cycles> - L2 数据缓存的命中延迟时间
-cache:il1 <config> - L1 指令缓存的配置
-cache:il1lat <cycles> - L1 指令缓存的命中延迟时间
-cache:il2 <config> - L2 指令缓存的配置
-cache:il2lat <cycles> - L2 指令缓存的命中延迟时间
-cache:flush – 清除系统调用的所有缓存
-cache:icompress – 将 64 位地址重新映射到 32 位地址
-mem:lat <1st> <next> - 显式声明数据存储器访问的延迟时间
-mem:width – 显式声明存储总线的带宽（以字节为单位）
-tlb:itlb <config> - 指令 TLB 设置，TLB 为相联存储器
-tlb:dtlb <config> - 数据 TLB 设置
-tlb:lat <cycles> - TLB 未命中时的处理延迟（以周期为单位）
-res:ialu – 指出整数 ALU 的数量
-res:imult – 指出整数乘法器/除法器的数量
-res:memports – 指出 L1 缓存的端口数量
-res:fpalu – 指出浮点数 ALU 的数量
-res:fpmult – 指出浮点除法器/乘法器的数量
-pcstat <stat> - 将静态地址 stat 记录为文本地址（符号地址）
-ptrace <file> <range> - 生成指令的流水线痕迹

三、 设置分支预测器

需要显式指出分支预测器的种类，利用如下命令

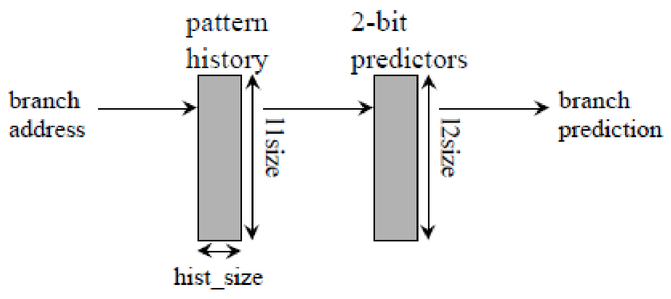
-bpred <type>

包括如下可以被选用的种类：
nottaken：总是预测不发生的分支
taken：总是预测发生的分支
perfect：完美的预测器，即每次都能正确预测分支
bimod：二元模式的预测器，用 2bit 表示预测状态，1bit 表示前一预测状态，1bit 表示当前预测状态
2lev：两级依赖的预测器（？）

其中 bimod 可以带有参数<size>，显式指出 BTB 的大小

设置两级依赖预测器可以显示指出 l1size, l2size 和 hist_size：
l1size：一级预测器的大小
l2size：二级预测器的大小
hist_size：历史预测宽度，即预测器的记忆能力

可用如下图表示：



四、 sim-outorder 流水线的追踪设置

sim-outorder 具有设置 pipetrace 的能力，它提供了所有指令的执行的历史细节，包括

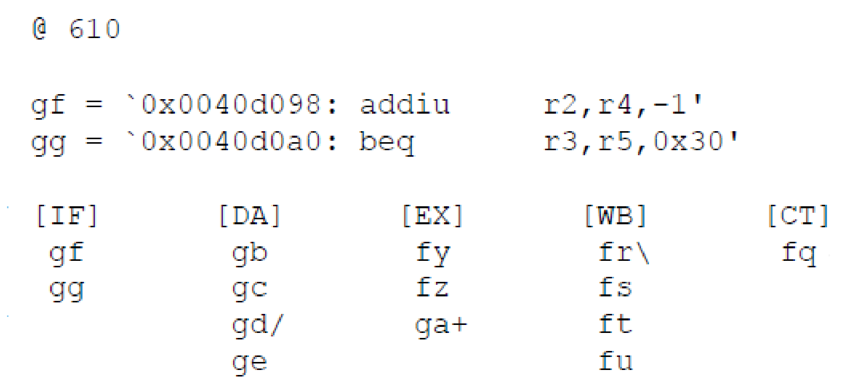
- 指令是如何被取出的
- 指令是如何被舍弃的（retirement）
- 指令的状态变化

利用如下指令显式指出生成流水线踪迹：
-ptrace <file> <range>

举例说明：
-ptrace FOO.trc : - trace entire execution to FOO.trc
表示打印 FOO.trc 记录文件中所有的执行的指令的踪迹
-ptrace BAR.trc 100:5000 - trace from inst 100 to 5000
只打印 BAR.trc 记录文件的 100-5000 条
-ptrace UXXE.trc :10000 - trace until instruction 10000
打印前 10000 条

simplescalar 还提供了 perl 脚本来打印这些踪迹，调用指令为：
pipeview.pl <ptrace_file>

举例说明：
sim-outorder -ptrace FOO.trc :1000 test-math
pipeview.pl FOO.trc
执行结果为：



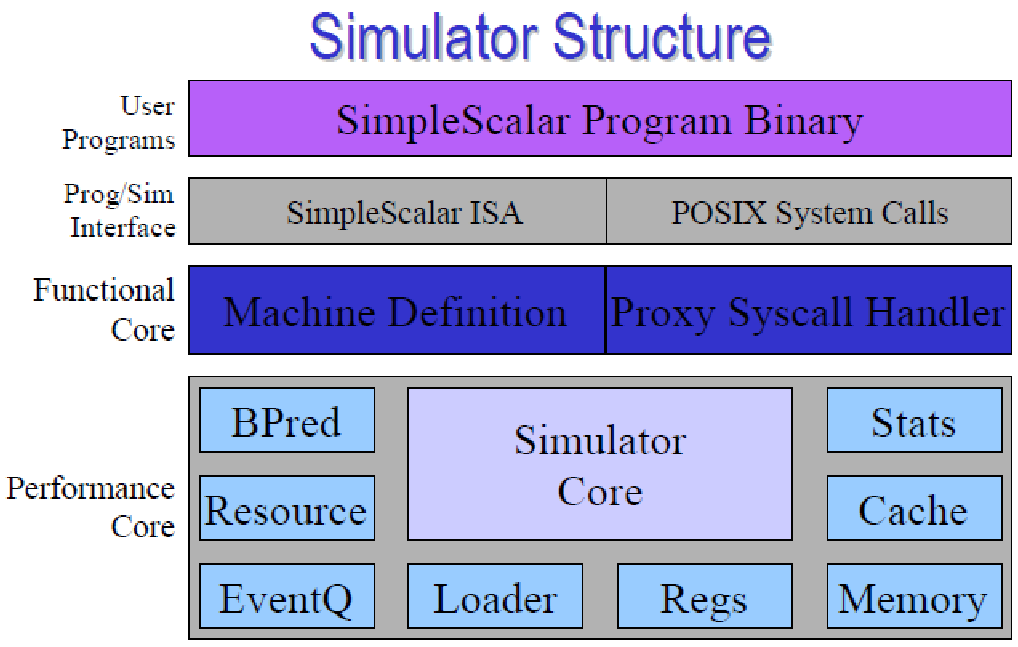
```
@ 610

gf = `0x0040d098: addiu     r2,r4,-1'
gg = `0x0040d0a0: beq       r3,r5,0x30'

[IF]      [DA]      [EX]      [WB]      [CT]
 gf        gb        fy        fr\       fq
 gg        gc        fz        fs
           gd/       ga+       ft
           ge                  fu
```

从横向看，这些结果被分为三个部分（从上至下）：
- 这是第几条指令（610）
- 这条指令的定义
- 当前流水线的状态

其中流水线状态的字段解释如下：
- IF：从指令队列中读出的指令，或正在指令队列中的指令（目前为 gf 和 gg 两条）
- DA：正在被译码，或者正在等待的指令
- EX：正在执行的指令
- WB：结果正在被写会 RUU 的指令，或者被正在被抛弃的指令
- CT：流水线事件（被检测出未成功预测分支）

五、 simplescalar 结构与 sim-outorder 结构

simplescalar 本身是模块化设计的，可以根据自己的需求来决定使用哪一个模块，模块可以高度定制化、可选化。



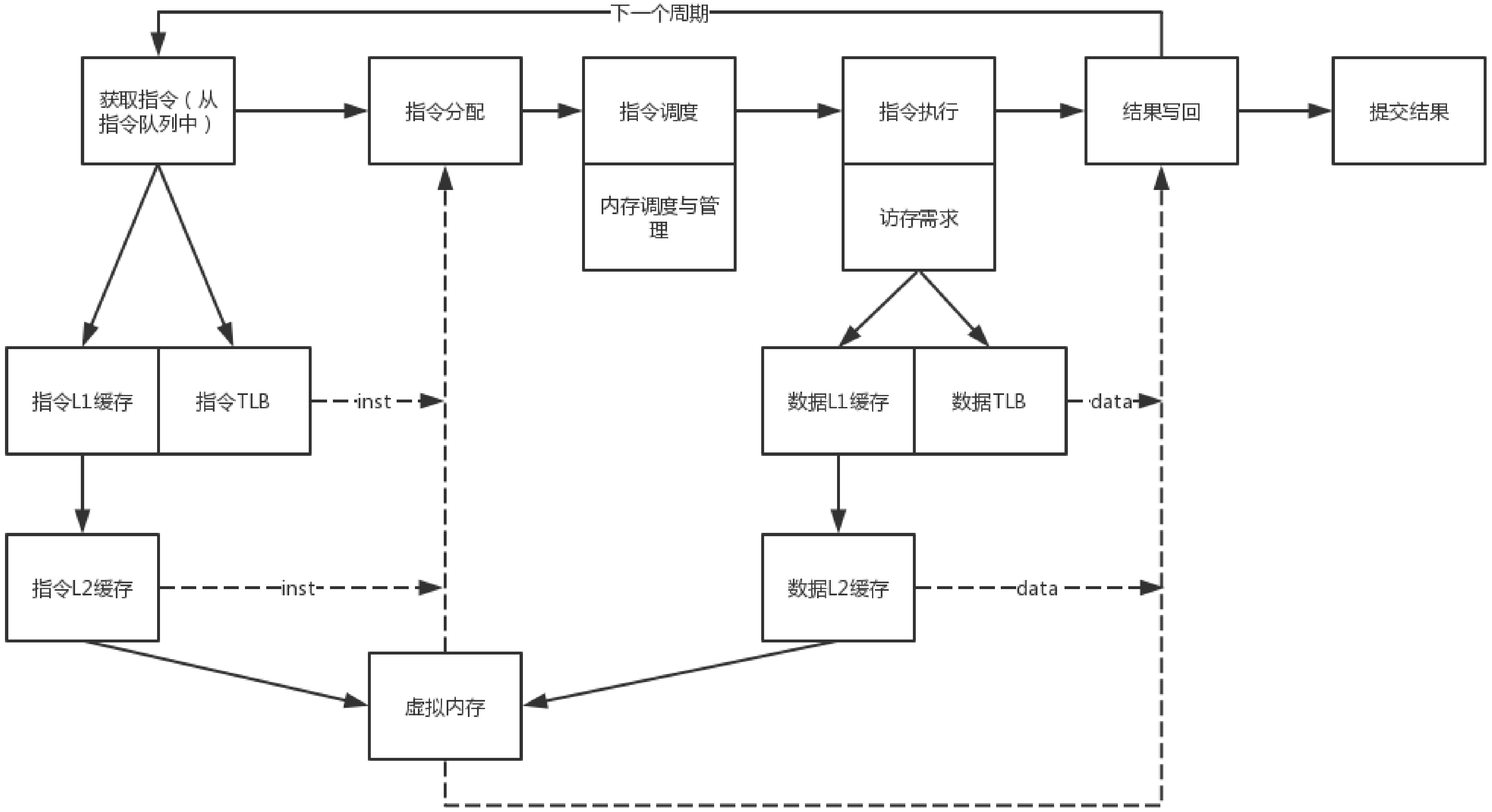
从上至下依次为：

- 用户程序：simplescalar 二进制文件，可以直接被模拟器执行
- 程序-模拟器接口：包括 simplescalar ISA 和一些系统调用的接口
- 功能模块核心：包括机器有关的定义和系统调用的处理
- 性能核心：包括 BPred（？），资源管理，事件队列，装载器，寄存器堆，内存，缓存，统计量和模拟核心

对于乱序发射的模拟过程，可以以如下的流程图来表示：

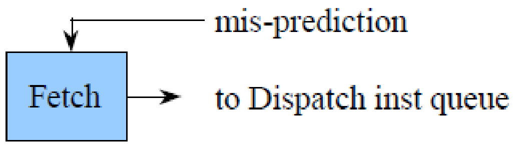
注：方块中的过程可以和之后的内容相对应

- *fetch-获取指令*
- *dispatch-指令分配*
- *schedule-指令调度*
- *execute-指令执行*
- *writeback-指令写回*
- *commit-结果提交*



下一个周期
获取指令（从指令队列中）
指令分配
指令调度
内存调度与管理
指令执行
访存需求
结果写回
提交结果
指令L1缓存
指令TLB
inst
数据L1缓存
数据TLB
data
指令L2缓存
inst
数据L2缓存
data
虚拟内存

六、 Fetch 过程



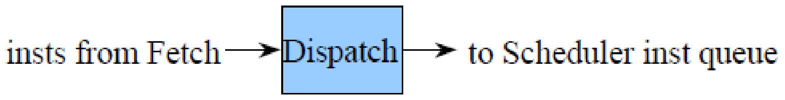
该过程在 ruu_fetch() 过程中实现

Fetch 过程的输入为：
- 程序计数器 PC
- 预测器的状态
- 未命中的检测结果（从分支预测器中得到）

输出为：从指令队列中取出的指令，将这些指令送往指令分配器

fetch 过程（每周期的执行内容）：
- 从 L1 指令缓存中取出指令，如果未命中，在未命中之前要一直阻塞该过程
- 将已经取出的指令排成指令队列，送往分配器 dispatcher
- 根据取指结果更新预测器的状态，为下一次循环做好准备

七、 Dispatch 过程



该过程在 ruu_dispatch() 实现

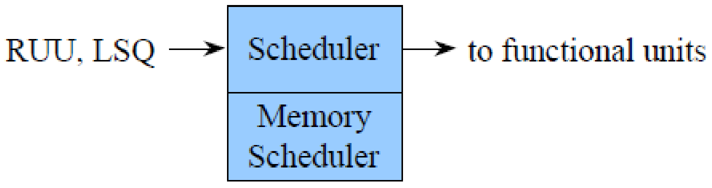
Dispatch 过程的输入为：
- 在 fetch 过程中输出的指令队列
- RUU（在 simplescalar 以一个结构实现）
- 重命名表
- 机器结构状态

输出为：
- 更新之后的 RUU，重命名表和机器状态

dispatch 过程：
- 从 dispatch 队列中取指
- 译码并执行该指令（允许进行数据依赖的优化，以及早期分支预测）
- 如果预测失败：将之前机器状态拷贝回机器状态结构中
- 如果预测成功：将指令链接并送入 RUU 和 load/store 队列中
- 链接过程由 RS_LINK 结构执行

- load/store 指令被分割为两个指令：ADD 和它本身
- 需要检查内存加速依赖

八、 Scheduler 过程



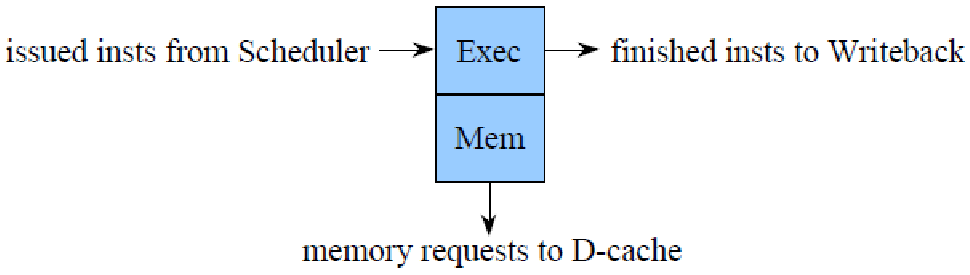
该过程在 ruu_issue()和 lsq_refresh()实现

输入：
RUU 和 LSQ

输出：
- 更新（重排）之后的 RUU 和 LSQ
- 更新之后的功能单元的状态

处理过程（每周期一次）：
- 用所有的寄存器输入信息来定位指令
- 用所有的内存输入信息来定位指令
  如果需要提前存储，但地址是未知的，则阻塞请求并且轮询该状态
  如果需要提前存储，但地址已经被绑定，则进行相应的存储操作
  否则，访问数据缓存

九、 Execute 过程


issued insts from Scheduler
Exec
finished insts to Writeback
Mem
memory requests to D-cache

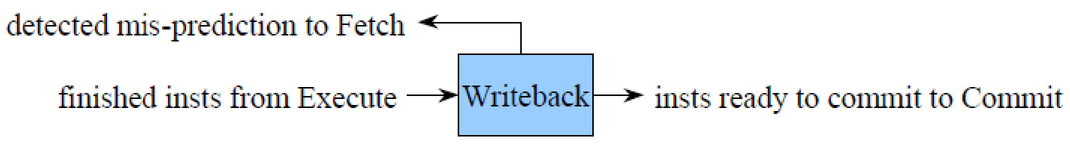
该规程在 ruu_issue()实现

输入：
- 已经 ready 的指令，且该指令已经通过 scheduler 调度
- 功能单元和数据缓存的状态

输出：
- 已被更新的功能单元和数据缓存状态
- 已被更新的事件队列，事件告知写回单元：该指令已经完成

过程（每周期被执行一次）：
- 得到已经就绪的指令
- 访问功能单元的状态，已得到许可信息
- 保留可被重复执行的单元（？）
- 产生写回事件，使用功能模块的操作延迟时间
  如果在数据缓存中找到了数据，则模拟数据缓存的访问延迟
  如果访问了数据 TLB，如果未命中，阻塞之后的过程
  数据 TLB 未命中的延迟时间是固定的

十、 Writeback 过程


detected mis-prediction to Fetch
finished insts from Execute
Writeback
insts ready to commit to Commit

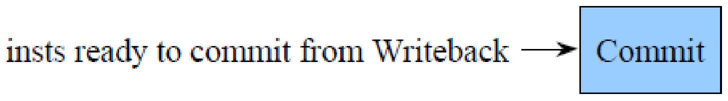
该过程在 ruu_writeback()中实现

输入：
- 由事件队列提供的已完成的指令信息
- RUU，LSQ 状态（为了实现唤醒功能）

输出：
- 被更新的事件队列
- 被更新的 RUU，LSQ，和 ready 队列
- 分支预测信息的更新

过程（每周期被执行一次）：
- 得到已经执行完成的指令（在事件队列中，可以得到这些指令的信息）
- 如果分支预测失败，需要恢复 RUU 和结构状态
- 唤醒遍历：唤醒某些阻塞于依赖的指令链

十一、 Commit 过程



该过程在 run_commit()实现

输入：
- 在 RUU 和 LSQ 中已完成的指令，这些指令等待被抛弃
- 数据缓存的状态

输出：
- 被更新的 RUU 和 LSQ
- 被更新的数据缓存状态

十二、 main 函数及其实现

在 sim_main 中被定义，该函数进行初始化，以及上述的所有工作。

```
ruu_init()
for (;;) {
  ruu_commit();
  ruu_writeback();
  lsq_refresh();
  ruu_issue();
  ruu_dispatch();
  ruu_fetch();
}
```

循环结构如上图所示，在每次循环中，上述操作最多被执行一次。

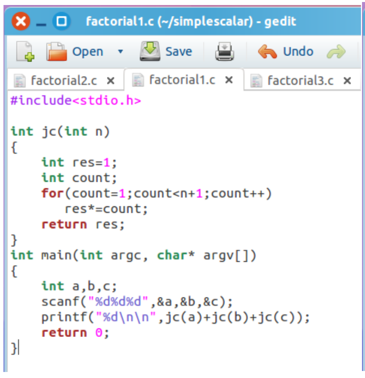
林宇辰部分

sim-outorder 分析

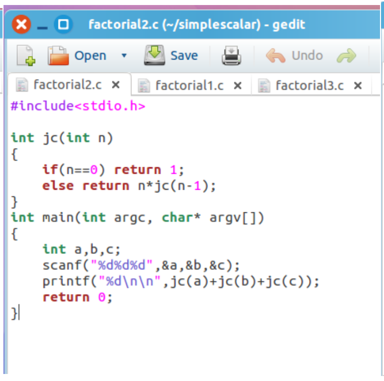
SimpleScalar 是一组体系结构仿真器工具，他可以模拟一个程序在某种体系结构机器上的具体执行过程，给出该体系结构的功能和性能参数。它包括的仿真器有：sim-fast ,sim – safe,sim-cache,sim-cheetah,sim-profile,sim-bpred,sim-eio 和 sim-outorder.

其中 sim-outorder 包括了几乎所有的仿真器参数，其他仿真器都是它的子集，现编写简单程序测试并对它的参数按类进行说明。

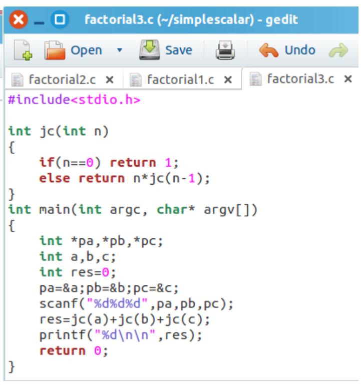
对 factorial1.c，factorial2.c 和 factorial2.c 三个求一个数字的阶乘的.c 程序进行编译：



```
#include<stdio.h>

int jc(int n)
{
    int res=1;
    int count;
    for(count=1;count<n+1;count++)
       res*=count;
    return res;
}
int main(int argc, char* argv[])
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("%d\n\n",jc(a)+jc(b)+jc(c));
    return 0;
}
```



```
#include<stdio.h>

int jc(int n)
{
    if(n==0) return 1;
    else return n*jc(n-1);
}
int main(int argc, char* argv[])
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("%d\n\n",jc(a)+jc(b)+jc(c));
    return 0;
}
```



```
#include<stdio.h>

int jc(int n)
{
    if(n==0) return 1;
    else return n*jc(n-1);
}
int main(int argc, char* argv[])
{
    int *pa,*pb,*pc;
    int a,b,c;
    int res=0;
    pa=&a;pb=&b;pc=&c;
    scanf("%d%d%d",pa,pb,pc);
    res=jc(a)+jc(b)+jc(c);
    printf("%d\n\n",res);
    return 0;
}
```

其中，三个小程序分别用循环、递归、指针方式实现目标。
生成能够在模拟器中运行的可执行文件 a.out，执行结果如下(以 factorial1.c 为例)：

sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.

All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use. No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

//版本声明

sim: simulation started @ Thu May 11 17:24:37 2017, options follow:
sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support. This simulator is a performance simulator, tracking the
latency of all pipeline operations.

//功能介绍

```
# -config                      # load configuration from a file
```
//从文件加载配置
```
# -dumpconfig                  # dump configuration to a file
```
//将配置转存
```
# -h                     false # print help message
```
//打印帮助信息
```
# -v                     false # verbose operation
```
//详细地显示操作列表
```
# -d                     false # enable debug message
```
//启用调试信息
```
# -i                     false # start in Dlite debugger
```
//启动 Dlite 调试器
```
-seed                        1 # random number generator seed (0 for timer seed)
```
//随机数发生器种子(0 用于计时器种子)
```
# -q                     false # initialize and terminate immediately
```
//进行初始化并立即终止
```
# -chkpt                <null> # restore EIO trace execution from <fname>
```
//从一个文件中恢复 EIO 跟踪程序
```
# -redir:sim            <null> # redirect simulator output to file (non-interactive only)
```
//将仿真器的输入写入一个文件中
```
# -redir:prog           <null> # redirect simulated program output to file
```
//重定向模拟程序到一个文件中
```
-nice                        0 # simulator scheduling priority
```
//仿真器的调度优先级
```
-max:inst                    0 # maximum number of inst's to execute
```
//仿真器最多执行的指令条数
```
-fastfwd                     0 # number of insts skipped before timing starts
```
//开始计时前跳过的指令的条数
```
# -ptrace               <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
-fetch:ifqsize               4 # instruction fetch queue size (in insts)
```
//指令获取队列的大小

```
-fetch:mplat                    3 # extra branch mis-prediction latency
```
//额外的分支预测失败的延迟
```
-fetch:speed                    1 # speed of front-end of machine relative to execution core
```
//前端相对于计算核心的速度
```
-bpred                                        bimod  #  branch  predictor  type
{nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod        2048 # bimodal predictor config (<table size>)
-bpred:2lev         1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb         1024 # combining predictor config (<meta_table_size>)
-bpred:ras                      8 # return address stack size (0 for no return stack)
-bpred:btb          512 4 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update            <null>  #  speculative  predictors  update  in  {ID|WB}  (default
non-spec)
```
//有关分支预测的一些设置
```
-decode:width                   4 # instruction decode B/W (insts/cycle)
```
//指令解码的带宽
```
-issue:width                    4 # instruction issue B/W (insts/cycle)
```
//发指令的带宽
```
-issue:inorder              false # run pipeline with in-order issue
```
//发指令顺序
```
-issue:wrongpath             true # issue instructions down wrong execution paths
```
//在预测错误时仍然允许发指令
```
-commit:width                   4 # instruction commit B/W (insts/cycle)
```
//指令提交的带宽
```
-ruu:size                      16 # register update unit (RUU) size
```
//寄存器更新部件的大小
```
-lsq:size                       8 # load/store queue (LSQ) size
```
//负载/存储队列（LSQ）的大小
```
-cache:dl1          dl1:128:32:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat                   1 # l1 data cache hit latency (in cycles)
-cache:dl2          ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat                   6 # l2 data cache hit latency (in cycles)
-cache:il1          il1:512:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1lat                   1 # l1 instruction cache hit latency (in cycles)
-cache:il2                    dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2lat                   6 # l2 instruction cache hit latency (in cycles)
-cache:flush                false # flush caches on system calls
-cache:icompress            false # convert 64-bit inst addresses to 32-bit inst equivalents
```
//数据缓存 cache 的设置
```
-mem:lat            18 2 # memory access latency (<first_chunk> <inter_chunk>)
```
//内存访问延迟
```
-mem:width                      8 # memory access bus width (in bytes)
```
//内存访问带宽
```
-tlb:itlb          itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
```

//指令 TLB 配置
-tlb:dtlb   dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
//数据 TLB 配置
-tlb:lat   30 # inst/data TLB miss latency (in cycles)
//指令、数据 TLB 的丢失延迟
-res:ialu   4 # total number of integer ALU's available
//可用整型 ALU 的总数
-res:imult   1 # total number of integer multiplier/dividers available
//可用整型乘法器/除法器的个数
-res:memport   2 # total number of memory system ports available (to CPU)
//可用内存的访问端口的总数
-res:fpalu   4 # total number of floating point ALU's available
//浮点型 ALU 的总数
-res:fpmult   1 # total number of floating point multiplier/dividers available
//浮点型乘法器/除法器的总数
# -pcstat   <null> # profile stat(s) against text addr's (mult uses ok)
//关于文字地址的档案统计信息
-bugcompat   false # operate in backward-compatible bugs mode (for testing only) **//操作在向后兼容的错误模式(仅供测试用)**

sim: ** starting performance simulation ****//开始性能模拟**
Please input two integers:2 10 **//输入**
The result is 1024. **//模拟运行结果**
sim: ** simulation statistics ** **//模拟统计**
sim_num_insn   10796 # total number of instructions committed
sim_num_refs   5080 # total number of loads and stores committed
sim_num_loads   1356 # total number of loads committed
sim_num_stores   3724.0000 # total number of stores committed
sim_num_branches   1866 # total number of branches committed
//以上 5 个参数分别为提交的 全部指令的条数，负载和存储的指令条数，负载条数，存储条数，存储条数，分支条数
sim_elapsed_time   5 # total simulation time in seconds
//模拟执行时间：5 秒
sim_inst_rate   2159.2000 # simulation speed (in insts/sec)
//每秒钟执行的指令数：大约每秒两千条
sim_total_insn   11923 # total number of instructions executed
sim_total_refs   5326 # total number of loads and stores executed
sim_total_loads   1523 # total number of loads executed
sim_total_stores   3803.0000 # total number of stores executed
sim_total_branches   2041 # total number of branches executed
//以上 5 个参数分别为执行的 全部指令的条数，负载和存储的指令条数，负载条数，存储条数，存储条数，分支条数
sim_cycle   22185 # total simulation time in cycles
//一个周期内的总模拟时间

```
sim_IPC                          0.4866 # instructions per cycle
//每个周期执行的指令数
sim_CPI                          2.0549 # cycles per instruction
//执行一条指令所需周期
sim_exec_BW                      0.5374 # total instructions (mis-spec + committed) per
cycle //每个周期实际执行的指令条数(包括分支预测错误时执行的无用的指令)
sim_IPB                          5.7856 # instruction per branch
//每条分支指令数
IFQ_count                         17723 # cumulative IFQ occupancy
IFQ_fcount                         3839 # cumulative IFQ full count
ifq_occupancy                    0.7989 # avg IFQ occupancy (insn's)
ifq_rate                         0.5374 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                      1.4865 # avg IFQ occupant latency (cycle's)
ifq_full                         0.1730 # fraction of time (cycle's) IFQ was full
// IFQ 的累计占用数，占用全数，平均占用率，平均发射速率 (指令/周期)，占用延迟，周
期内的占用时间
RUU_count                         62034 # cumulative RUU occupancy
RUU_fcount                          913 # cumulative RUU full count
ruu_occupancy                    2.7962 # avg RUU occupancy (insn's)
ruu_rate                         0.5374 # avg RUU dispatch rate (insn/cycle)
ruu_latency                      5.2029 # avg RUU occupant latency (cycle's)
ruu_full                         0.0412 # fraction of time (cycle's) RUU was full
// RUU 的累计占用数，占用全数，平均占用率，平均发射速率 (指令/周期)，占用延迟，周
期内的占用时间
LSQ_count                         26878 # cumulative LSQ occupancy
LSQ_fcount                         1660 # cumulative LSQ full count
lsq_occupancy                    1.2115 # avg LSQ occupancy (insn's)
lsq_rate                         0.5374 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                      2.2543 # avg LSQ occupant latency (cycle's)
lsq_full                         0.0748 # fraction of time (cycle's) LSQ was full
// LSQ 的累计占用数，占用全数，平均占用率，平均发射速率 (指令/周期)，占用延迟，周
期内的占用时间
sim_slip                         100999 # total number of slip cycles
//滑动周期的总数
avg_sim_slip                     9.3552 # the average slip between issue and retirement
//发退指令之间的平均滑差
bpred_bimod.lookups                2088 # total number of bpred lookups
bpred_bimod.updates                1866 # total number of updates
bpred_bimod.addr_hits              1389 # total number of address-predicted hits
bpred_bimod.dir_hits               1601 # total number of direction-predicted hits (includes
addr-hits)
bpred_bimod.misses                  265 # total number of misses
bpred_bimod.jr_hits                 101 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen                 111 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP                    1 # total number of address-predicted hits for
non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP                    4 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate         0.7444  #  branch  address-prediction  rate  (i.e.,  addr-
hits/updates)
bpred_bimod.bpred_dir_rate          0.8580  #  branch  direction-prediction  rate  (i.e.,  all-
hits/updates)
bpred_bimod.bpred_jr_rate        0.9099  #  JR  address-prediction  rate  (i.e.,  JR  addr-hits/JRs
seen)
bpred_bimod.bpred_jr_non_ras_rate.PP       0.2500  #  non-RAS  JR  addr-pred  rate  (ie,  non-
RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes                 135  #  total  number  of  address  pushed  onto  ret-
addr stack
bpred_bimod.retstack_pops                 115 # total number of address popped off of ret-addr
stack
bpred_bimod.used_ras.PP                 107 # total number of RAS predictions used
bpred_bimod.ras_hits.PP                 100 # total number of RAS hits
bpred_bimod.ras_rate.PP       0.9346 # RAS prediction rate (i.e., RAS hits/used RAS)
```

//分支预测的参数统计

```
il1.accesses                        12686 # total number of accesses
il1.hits                            11804 # total number of hits
il1.misses                            882 # total number of misses
il1.replacements                      449 # total number of replacements
il1.writebacks                          0 # total number of writebacks
il1.invalidations                       0 # total number of invalidations
```

//il1 访问，命中，丢失，替换，写回，无效的次数

```
il1.miss_rate                      0.0695 # miss rate (i.e., misses/ref)
il1.repl_rate                      0.0354 # replacement rate (i.e., repls/ref)
il1.wb_rate                        0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                       0.0000 # invalidation rate (i.e., invs/ref)
```

//il1 的丢失率，替换率，写回率，无效率

```
dl1.accesses                         5114 # total number of accesses
dl1.hits                             4661 # total number of hits
dl1.misses                            453 # total number of misses
dl1.replacements                        9 # total number of replacements
dl1.writebacks                          7 # total number of writebacks
dl1.invalidations                       0 # total number of invalidations
dl1.miss_rate                      0.0886 # miss rate (i.e., misses/ref)
dl1.repl_rate                      0.0018 # replacement rate (i.e., repls/ref)
dl1.wb_rate                        0.0014 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                       0.0000 # invalidation rate (i.e., invs/ref)
```

//dl1 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无效率

```
ul2.accesses                         1342 # total number of accesses
```

```
ul2.hits                       666 # total number of hits
ul2.misses                     676 # total number of misses
ul2.replacements                 0 # total number of replacements
ul2.writebacks                   0 # total number of writebacks
ul2.invalidations                0 # total number of invalidations
ul2.miss_rate               0.5037 # miss rate (i.e., misses/ref)
ul2.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
ul2.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

//ul2 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无效率

```
itlb.accesses                12686 # total number of accesses
itlb.hits                    12668 # total number of hits
itlb.misses                     18 # total number of misses
itlb.replacements                0 # total number of replacements
itlb.writebacks                  0 # total number of writebacks
itlb.invalidations               0 # total number of invalidations
itlb.miss_rate              0.0014 # miss rate (i.e., misses/ref)
itlb.repl_rate              0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
```

//itlb 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无效率

```
dtlb.accesses                 5142 # total number of accesses
dtlb.hits                     5133 # total number of hits
dtlb.misses                      9 # total number of misses
dtlb.replacements                0 # total number of replacements
dtlb.writebacks                  0 # total number of writebacks
dtlb.invalidations               0 # total number of invalidations
dtlb.miss_rate              0.0018 # miss rate (i.e., misses/ref)
dtlb.repl_rate              0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
```

//dtlb 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无效率

```
sim_invalid_addrs                0 # total non-speculative bogus addresses seen (debug
var)
```

//所有可见的非投机虚拟地址

```
ld_text_base            0x00400000 # program text (code) segment base
```

//程序的文本（代码段）的基地址

```
ld_text_size                 98640 # program text (code) size in bytes
```

//程序的文本（代码段）的字节大小

```
ld_data_base            0x10000000 # program initialized data segment base
```

//程序初始数据段的基地址

```
ld_data_size                 12288 # program init'ed `.data' and uninit'ed `.bss' size in
```

bytes

ld_stack_base   0x7fffc000 # program stack segment base (highest address in stack)**// 程序的堆栈的基地址(堆栈中的最高地址)**

ld_stack_size   16384 # program initial stack size

//程序的初始堆栈大小

ld_prog_entry   0x00400140 # program entry point (initial PC)

//程序的入口点(初始的 pc 值)

ld_environ_base   0x7fff8000 # program environment base address address

//程序环境的基地址

ld_target_big_endian   0 # target executable endian-ness, non-zero if big endian**//程序是否为大端，非 0 即是。**

mem.page_count   34 # total number of pages allocated

//分配的页面总数

mem.page_mem   136k # total size of memory pages allocated

//分配的内存页的大小

mem.ptab_misses   34 # total first level page table misses

//一级页表丢失数

mem.ptab_accesses   708272 # total page table accesses

//总页表访问数

mem.ptab_miss_rate   0.0000 # first level page table miss rate

//一级页表丢失率

Aborted

//模拟运行终止

分别观察如下三个程序的各个参数值，进行对比发现，绝大多数的参数值随着程序复杂度的增大而增加。

```
sim_total_stores             3823.0000# totalnumberofstoresexecuted
sim_total_branches                1666 # totalnumber of branchesexecuted
sim_cycle                        22231 # totalsimulation timeincycles
sim_IPC                         0.4260# instructionsper cycle
sim_CPI                         2.3475# cyclesperinstruction
sim_exec_BW                      0.4733# totalinstructions(mis-spec+committed) per cycle
sim_IPB                         6.2344# instruction per branch
IFQ_count                        15238 # cumulative IFQ occupancy
IFQ_fcount                        3270# cumulative IFQ full count
ifq_occupancy                  0.6854# avg IFQ occupancy(insn's)
ifq_rate                       0.4733# avg IFQ dispatch rate (insn/cycle)
ifq_latency                    1.4483# avg IFQ occupant latency(cycle's)
ifq_full                       0.1471# fraction of time (cycle's) IFQ wasfull
RUU_count                        50913 # cumulative RUU occupancy
RUU_fcount                         381# cumulative RUU full count
ruu_occupancy                   2.2902# avg RUU occupancy(insn's)
ruu_rate                        0.4733# avg RUU dispatch rate (insn/cycle)
ruu_latency                     4.8392# avg RUU occupant latency(cycle's)
ruu_full                        0.0171# fraction of time (cycle's) RUU wasfull
LSQ_count                        24851 # cumulative LSQ occupancy
LSQ_fcount                        1610# cumulative LSQ fullcount
lsq_occupancy                   1.1179# avg LSQ occupancy(insn's)
lsq_rate                        0.4733# avg LSQ dispatch rate (insn/cycle)
lsq_latency                     2.3620# avg LSQ occupant latency (cycle's)
lsq_full                        0.0724# fraction of time (cycle's) LSQ wasfull
sim_slip                         85193 # totalnumberofslipcycles
avg_sim_slip                    8.9961 # the average slip between issue and retirement
bpred_bimod.lookups               1710 # totalnumberofbpred lookups
bpred_bimod.updates               1519# total number of updates
bpred_bimod.addr_hits             1030 # totalnumber ofaddress-predicted hits
bpred_bimod.dir_hits                  1248 # total number of direction-predicted hits (includes
addr-hits)
bpred_bimod.misses                 271 # total number of misses
bpred_bimod.jr_hits                101# total number of address-predicted hits for JR's
bpred_bimod.jr_seen                113 # totalnumber of JR's seen
```

```
sim_total_stores             3818.0000# totalnumberofstoresexecuted
sim_total_branches                1672 # totalnumber of branchesexecuted
sim_cycle                        22270 # totalsimulation timeincycles
sim_IPC                         0.4251# instructionsper cycle
sim_CPI                         2.3521# cyclesperinstruction
sim_exec_BW                      0.4766# totalinstructions(mis-spec+committed) per cycle
sim_IPB                         6.2330# instruction per branch
IFQ_count                        15279 # cumulative IFQ occupancy
IFQ_fcount                        3300# cumulative IFQ full count
ifq_occupancy                  0.6861# avg IFQ occupancy(insn's)
ifq_rate                       0.4766# avg IFQ dispatch rate (insn/cycle)
ifq_latency                    1.4395# avg IFQ occupant latency(cycle's)
ifq_full                       0.1482# fraction of time (cycle's) IFQ wasfull
RUU_count                        51189 # cumulative RUU occupancy
RUU_fcount                         373# cumulative RUU full count
ruu_occupancy                   2.2986# avg RUU occupancy(insn's)
ruu_rate                        0.4766# avg RUU dispatch rate (insn/cycle)
ruu_latency                     4.8228# avg RUU occupant latency(cycle's)
ruu_full                        0.0167# fraction of time (cycle's) RUU wasfull
LSQ_count                        25103 # cumulative LSQ occupancy
LSQ_fcount                        1607# cumulative LSQ fullcount
lsq_occupancy                   1.1272# avg LSQ occupancy(insn's)
lsq_rate                        0.4766# avg LSQ dispatch rate (insn/cycle)
lsq_latency                     2.3651# avg LSQ occupant latency (cycle's)
lsq_full                        0.0722# fraction of time (cycle's) LSQ wasfull
sim_slip                         85398 # totalnumberofslipcycles
avg_sim_slip                    9.0196 # the average slip between issue and retirement
bpred_bimod.lookups               1719 # totalnumberofbpred lookups
bpred_bimod.updates               1519# total number of updates
bpred_bimod.addr_hits             1030 # totalnumber ofaddress-predicted hits
bpred_bimod.dir_hits                  1248 # total number of direction-predicted hits (includes
addr-hits)
bpred_bimod.misses                 271 # total number of misses
bpred_bimod.jr_hits                101# total number of address-predicted hits for JR's
bpred_bimod.jr_seen                113 # totalnumber of JR's seen
```

```
sim_total_stores             3825.0000 # total number of stores executed
sim_total_branches                1666 # total number of branches executed
sim_cycle                        21896 # total simulation time in cycles
sim_IPC                         0.4327 # instructions per cycle
sim_CPI                         2.3112 # cycles per instruction
sim_exec_BW                     0.4712 # total instructions (mis-spec + committed) per cycle
sim_IPB                         6.2370 # instruction per branch
IFQ_count                        15035 # cumulative IFQ occupancy
IFQ_fcount                        3219 # cumulative IFQ full count
ifq_occupancy                   0.6867 # avg IFQ occupancy (insn's)
ifq_rate                        0.4712 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                     1.4572 # avg IFQ occupant latency (cycle's)
ifq_full                        0.1470 # fraction of time (cycle's) IFQ was full
RUU_count                        50994 # cumulative RUU occupancy
RUU_fcount                         381 # cumulative RUU full count
ruu_occupancy                   2.3289 # avg RUU occupancy (insn's)
ruu_rate                        0.4712 # avg RUU dispatch rate (insn/cycle)
ruu_latency                     4.9422 # avg RUU occupant latency (cycle's)
ruu_full                        0.0174 # fraction of time (cycle's) RUU was full
LSQ_count                        24959 # cumulative LSQ occupancy
LSQ_fcount                        1610 # cumulative LSQ full count
lsq_occupancy                   1.1399 # avg LSQ occupancy (insn's)
lsq_rate                        0.4712 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                     2.4190 # avg LSQ occupant latency (cycle's)
lsq_full                        0.0735 # fraction of time (cycle's) LSQ was full
sim_slip                         85287 # total number of slip cycles
avg_sim_slip                    9.0022 # the average slip between issue and retirement
bpred_bimod.lookups               1710 # total number of bpred lookups
bpred_bimod.updates               1519 # total number of updates
bpred_bimod.addr_hits             1030 # total number of address-predicted hits
bpred_bimod.dir_hits              1249 # total number of direction-predicted hits (includes
addr-hits)
bpred_bimod.misses                 270 # total number of misses
bpred_bimod.jr_hits                101 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen                113 # total number of JR's seen
```

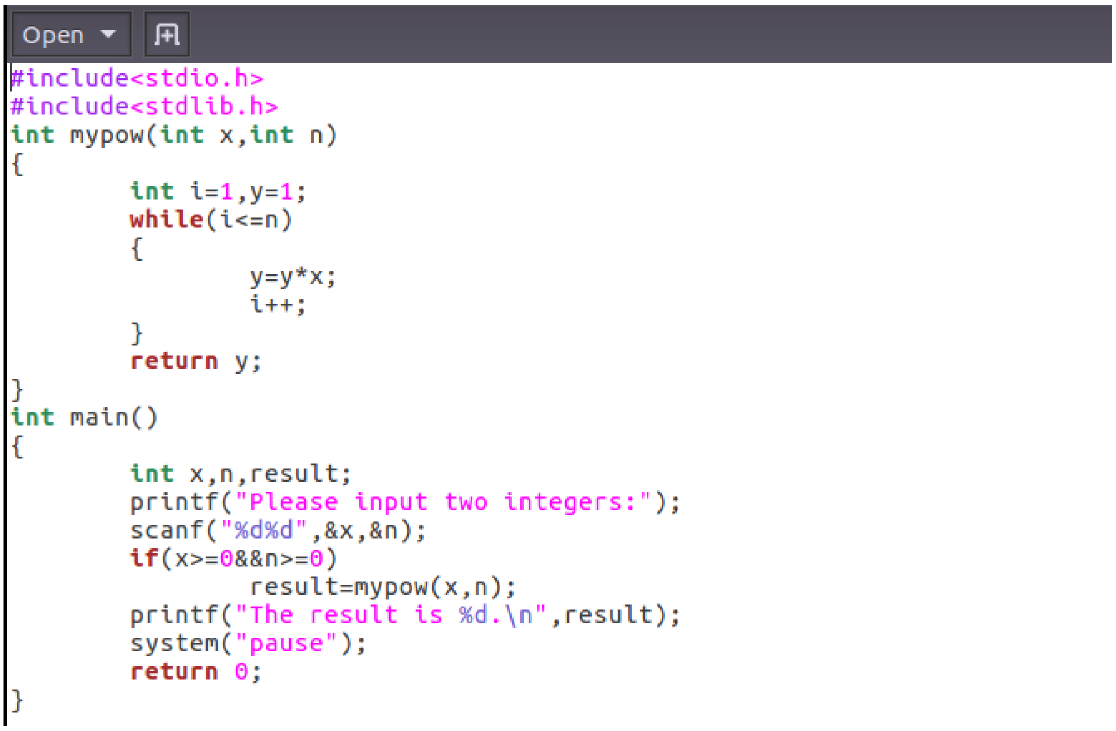
王剑督部分

一、利用 sim-outorder.out 运行 mypow.c，打印结果，根据程序分析输出结果

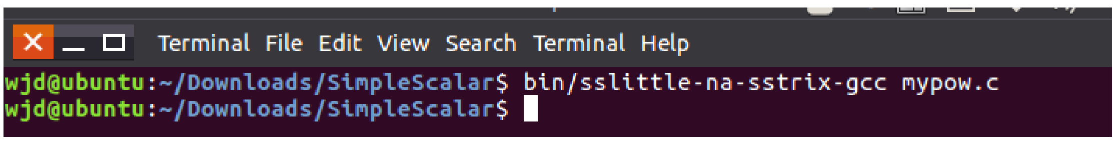
各种参数的意义

1. **编译源程序**

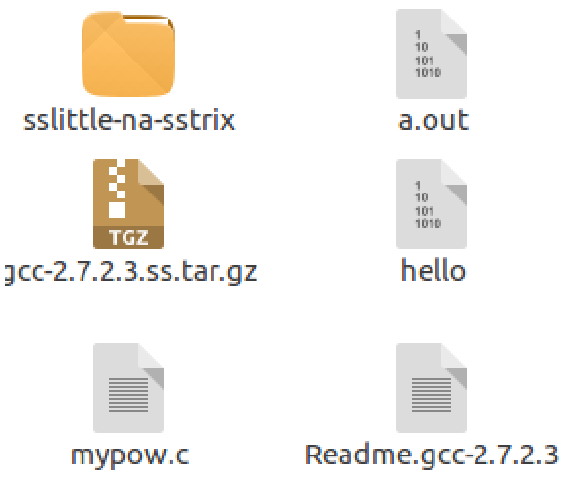
   用 simplescalar 的编译器对 mypow.c 进行编译，生成能够在模拟器中运行的可执行文件 a.out。其中，mypow.c 是计算 a 的 n 次方的运算程序，如下。



```c
#include<stdio.h>
#include<stdlib.h>
int mypow(int x,int n)
{
        int i=1,y=1;
        while(i<=n)
        {
                y=y*x;
                i++;
        }
        return y;
}
int main()
{
        int x,n,result;
        printf("Please input two integers:");
        scanf("%d%d",&x,&n);
        if(x>=0&&n>=0)
                result=mypow(x,n);
        printf("The result is %d.\n",result);
        system("pause");
        return 0;
}
```

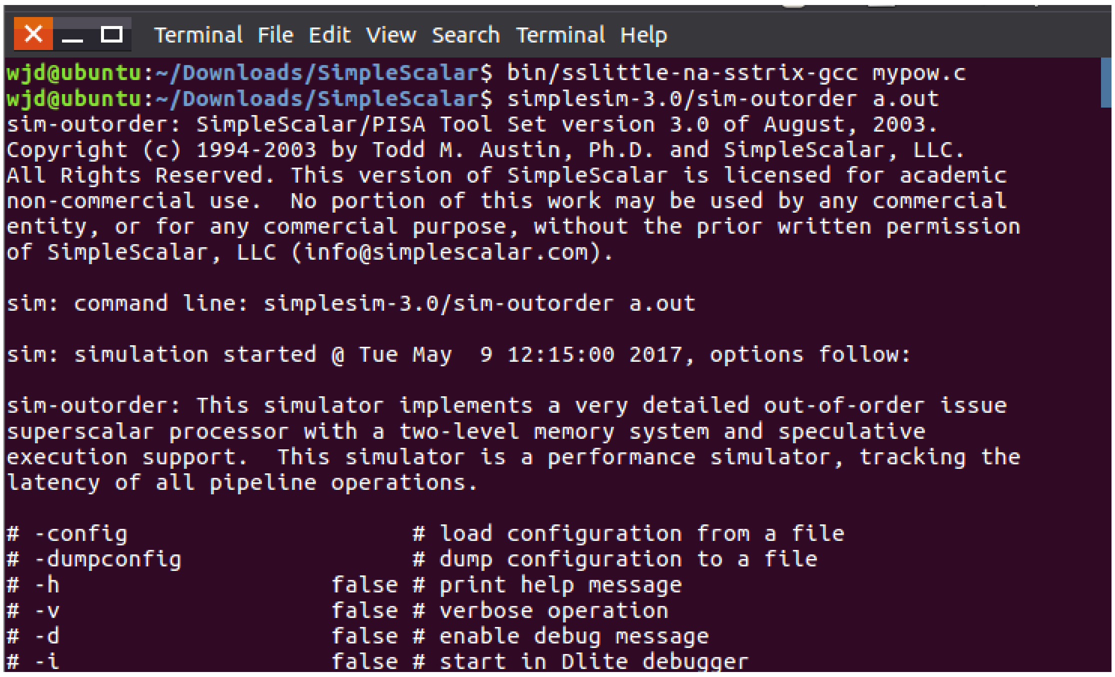


```
wjd@ubuntu:~/Downloads/SimpleScalar$ bin/sslittle-na-sstrix-gcc mypow.c
wjd@ubuntu:~/Downloads/SimpleScalar$ 
```

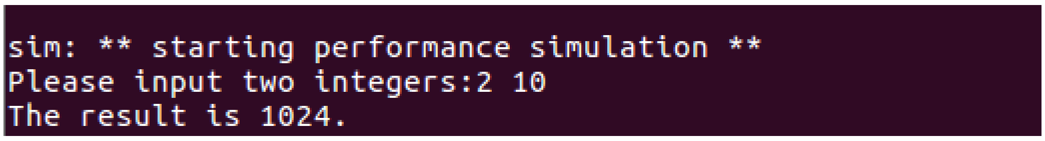


2. **模拟运行可执行文件 a.out**

用 simplescalar 中的模拟器 sim-outorder 对 a.out 进行模拟运行，结果如下。

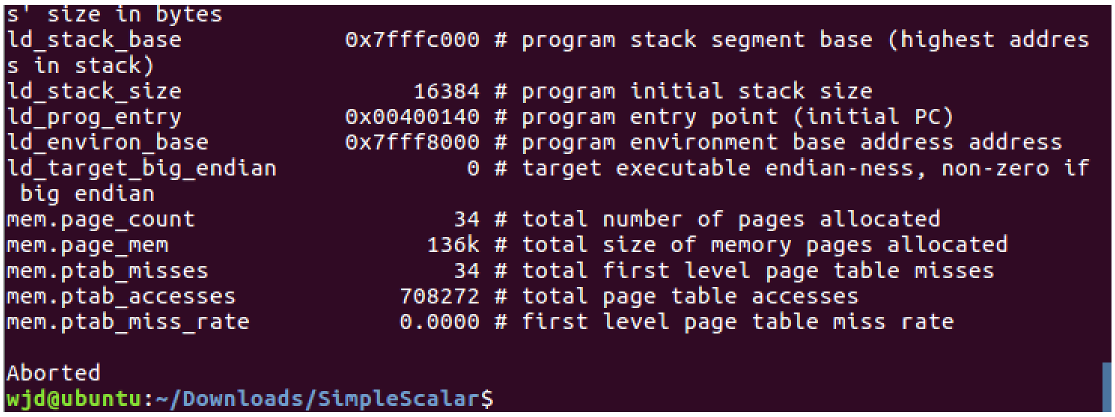


```
wjd@ubuntu:~/Downloads/SimpleScalar$ bin/sslittle-na-sstrix-gcc mypow.c
wjd@ubuntu:~/Downloads/SimpleScalar$ simplesim-3.0/sim-outorder a.out
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: simplesim-3.0/sim-outorder a.out

sim: simulation started @ Tue May  9 12:15:00 2017, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

# -config                         # load configuration from a file
# -dumpconfig                     # dump configuration to a file
# -h                        false # print help message
# -v                        false # verbose operation
# -d                        false # enable debug message
# -i                        false # start in Dlite debugger
```



```
sim: ** starting performance simulation **
Please input two integers:2 10
The result is 1024.
```

上图为函数 mypow（）的运行结果，计算了 2 的 10 次方,结果是 1024。



```
s' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest addres
s in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if
 big endian
mem.page_count                   34 # total number of pages allocated
mem.page_mem                   136k # total size of memory pages allocated
mem.ptab_misses                  34 # total first level page table misses
mem.ptab_accesses            708272 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate

Aborted
wjd@ubuntu:~/Downloads/SimpleScalar$
```

3.　分析用 sim-outorder 运行的输出结果中各参数的意义

sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.　No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).
//版本、版权所有、非商业用途的声明
sim: command line: simplesim-3.0/sim-outorder a.out **//命令行**

```
sim: simulation started @ Tue May  9 12:15:00 2017, options follow://模拟开始

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.//功能性能介绍

# -config                        # load configuration from a file
//从一个文件中加载配置
# -dumpconfig                    # dump configuration to a file
//将配置转储到一个文件中
# -h                       false # print help message
//打印帮助信息
# -v                       false # verbose operation
//详细地显示操作列表
# -d                       false # enable debug message
//启用调试信息
# -i                       false # start in Dlite debugger
//启动 Dlite 调试器
-seed                          1 # random number generator seed (0 for timer seed)
//随机数发生器种子(0 用于计时器种子)
# -q                       false # initialize and terminate immediately
//进行初始化并立即终止
# -chkpt                  <null> # restore EIO trace execution from <fname>
//从一个文件中恢复 EIO 跟踪程序
# -redir:sim              <null> # redirect simulator output to file (non-interactive only)
//将仿真器的输入写入一个文件中
# -redir:prog             <null> # redirect simulated program output to file
//重定向模拟程序到一个文件中
-nice                          0 # simulator scheduling priority
//仿真器的调度优先级
-max:inst                      0 # maximum number of inst's to execute
//仿真器最多执行的指令条数
-fastfwd                       0 # number of insts skipped before timing starts
//开始计时前跳过的指令的条数
# -ptrace                 <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
-fetch:ifqsize                 4 # instruction fetch queue size (in insts)
//指令获取队列的大小
-fetch:mplat                   3 # extra branch mis-prediction latency
//额外的分支预测失败的延迟
-fetch:speed                   1 # speed of front-end of machine relative to execution core
//前端相对于计算核心的速度
-bpred                     bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
```

```
-bpred:bimod          2048 # bimodal predictor config (<table size>)
-bpred:2lev           1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb           1024 # combining predictor config (<meta_table_size>)
-bpred:ras                        8 # return address stack size (0 for no return stack)
-bpred:btb            512 4 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update             <null> # speculative predictors update in {ID|WB} (default non-spec)
//有关分支预测的一些设置
-decode:width                     4 # instruction decode B/W (insts/cycle)
//指令解码的带宽
-issue:width                      4 # instruction issue B/W (insts/cycle)
//发指令的带宽
-issue:inorder                false # run pipeline with in-order issue
//发指令顺序
-issue:wrongpath               true # issue instructions down wrong execution paths
//在预测错误时仍然允许发指令
-commit:width                     4 # instruction commit B/W (insts/cycle)
//指令提交的带宽
-ruu:size                        16 # register update unit (RUU) size
//寄存器更新部件的大小
-lsq:size                         8 # load/store queue (LSQ) size
//负载/存储队列（LSQ）的大小
-cache:dl1            dl1:128:32:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat                     1 # l1 data cache hit latency (in cycles)
-cache:dl2            ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat                     6 # l2 data cache hit latency (in cycles)
-cache:il1            il1:512:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1lat                     1 # l1 instruction cache hit latency (in cycles)
-cache:il2                      dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2lat                     6 # l2 instruction cache hit latency (in cycles)
-cache:flush                  false # flush caches on system calls
-cache:icompress              false # convert 64-bit inst addresses to 32-bit inst equivalents
//数据缓存 cache 的设置
-mem:lat              18 2 # memory access latency (<first_chunk> <inter_chunk>)
//内存访问延迟
-mem:width                        8 # memory access bus width (in bytes)
//内存访问带宽
-tlb:itlb             itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
//指令 TLB 配置
-tlb:dtlb             dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
//数据 TLB 配置
-tlb:lat                         30 # inst/data TLB miss latency (in cycles)
//指令、数据 TLB 的丢失延迟
-res:ialu                         4 # total number of integer ALU's available
```

```
//可用整型 ALU 的总数
-res:imult                    1 # total number of integer multiplier/dividers available
//可用整型乘法器/除法器的个数
-res:memport                  2 # total number of memory system ports available (to CPU)
//可用内存的访问端口的总数
-res:fpalu                    4 # total number of floating point ALU's available
//浮点型 ALU 的总数
-res:fpmult                   1 # total number of floating point multiplier/dividers available
//浮点型乘法器/除法器的总数
# -pcstat                <null> # profile stat(s) against text addr's (mult uses ok)
//关于文字地址的档案统计信息
-bugcompat                false # operate in backward-compatible bugs mode (for testing only) //操作在向后兼容的错误模式(仅供测试用)

sim: ** starting performance simulation **//开始性能模拟
Please input two integers:2 10   //输入
The result is 1024.   //模拟运行结果
sim: ** simulation statistics ** //模拟统计
sim_num_insn                  10796 # total number of instructions committed
sim_num_refs                   5080 # total number of loads and stores committed
sim_num_loads                  1356 # total number of loads committed
sim_num_stores            3724.0000 # total number of stores committed
sim_num_branches               1866 # total number of branches committed
//以上 5 个参数分别为提交的 全部指令的条数，负载和存储的指令条数，负载条数，存储条数，存储条数，分支条数
sim_elapsed_time                  5 # total simulation time in seconds
//模拟执行时间：5 秒
sim_inst_rate             2159.2000 # simulation speed (in insts/sec)
//每秒钟执行的指令数：大约每秒两千条
sim_total_insn                11923 # total number of instructions executed
sim_total_refs                 5326 # total number of loads and stores executed
sim_total_loads                1523 # total number of loads executed
sim_total_stores          3803.0000 # total number of stores executed
sim_total_branches             2041 # total number of branches executed
//以上 5 个参数分别为执行的 全部指令的条数，负载和存储的指令条数，负载条数，存储条数，存储条数，分支条数
sim_cycle                     22185 # total simulation time in cycles
//一个周期内的总模拟时间
sim_IPC                      0.4866 # instructions per cycle
//每个周期执行的指令数
sim_CPI                      2.0549 # cycles per instruction
//执行一条指令所需周期
sim_exec_BW                  0.5374 # total instructions (mis-spec + committed) per cycle //每个周期实际执行的指令条数(包括分支预测错误时执行的无用的指令)
```

```
sim_IPB                        5.7856 # instruction per branch
```
//每条分支指令数
```
IFQ_count                       17723 # cumulative IFQ occupancy
IFQ_fcount                       3839 # cumulative IFQ full count
ifq_occupancy                  0.7989 # avg IFQ occupancy (insn's)
ifq_rate                       0.5374 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                    1.4865 # avg IFQ occupant latency (cycle's)
ifq_full                       0.1730 # fraction of time (cycle's) IFQ was full
```
// IFQ 的累计占用数，占用全数，平均占用率，平均发射速率 (指令/周期)，占用延迟，周期内的占用时间
```
RUU_count                       62034 # cumulative RUU occupancy
RUU_fcount                        913 # cumulative RUU full count
ruu_occupancy                  2.7962 # avg RUU occupancy (insn's)
ruu_rate                       0.5374 # avg RUU dispatch rate (insn/cycle)
ruu_latency                    5.2029 # avg RUU occupant latency (cycle's)
ruu_full                       0.0412 # fraction of time (cycle's) RUU was full
```
// RUU 的累计占用数，占用全数，平均占用率，平均发射速率 (指令/周期)，占用延迟，周期内的占用时间
```
LSQ_count                       26878 # cumulative LSQ occupancy
LSQ_fcount                       1660 # cumulative LSQ full count
lsq_occupancy                  1.2115 # avg LSQ occupancy (insn's)
lsq_rate                       0.5374 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                    2.2543 # avg LSQ occupant latency (cycle's)
lsq_full                       0.0748 # fraction of time (cycle's) LSQ was full
```
// LSQ 的累计占用数，占用全数，平均占用率，平均发射速率 (指令/周期)，占用延迟，周期内的占用时间
```
sim_slip                       100999 # total number of slip cycles
```
//滑动周期的总数
```
avg_sim_slip                   9.3552 # the average slip between issue and retirement
```
//发退指令之间的平均滑差
```
bpred_bimod.lookups              2088 # total number of bpred lookups
bpred_bimod.updates              1866 # total number of updates
bpred_bimod.addr_hits            1389 # total number of address-predicted hits
bpred_bimod.dir_hits             1601 # total number of direction-predicted hits (includes
addr-hits)
bpred_bimod.misses                265 # total number of misses
bpred_bimod.jr_hits               101 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen               111 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP             1 # total number of address-predicted hits for
non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP             4 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate    0.7444 # branch address-prediction rate (i.e., addr-
hits/updates)
bpred_bimod.bpred_dir_rate     0.8580 # branch direction-prediction rate (i.e., all-
hits/updates)
bpred_bimod.bpred_jr_rate        0.9099 # JR address-prediction rate (i.e., JR addr-hits/JRs
seen)
bpred_bimod.bpred_jr_non_ras_rate.PP       0.2500 # non-RAS JR addr-pred rate (ie, non-
RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes                135 # total number of address pushed onto ret-
addr stack
bpred_bimod.retstack_pops                115 # total number of address popped off of ret-addr
stack
bpred_bimod.used_ras.PP               107 # total number of RAS predictions used
bpred_bimod.ras_hits.PP               100 # total number of RAS hits
bpred_bimod.ras_rate.PP       0.9346 # RAS prediction rate (i.e., RAS hits/used RAS)
```
//分支预测的参数统计
```
il1.accesses                    12686 # total number of accesses
il1.hits                        11804 # total number of hits
il1.misses                        882 # total number of misses
il1.replacements                  449 # total number of replacements
il1.writebacks                      0 # total number of writebacks
il1.invalidations                   0 # total number of invalidations
```
//il1 访问，命中，丢失，替换，写回，无效的次数
```
il1.miss_rate                  0.0695 # miss rate (i.e., misses/ref)
il1.repl_rate                  0.0354 # replacement rate (i.e., repls/ref)
il1.wb_rate                    0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                   0.0000 # invalidation rate (i.e., invs/ref)
```
//il1 的丢失率，替换率，写回率，无效率
```
dl1.accesses                     5114 # total number of accesses
dl1.hits                         4661 # total number of hits
dl1.misses                        453 # total number of misses
dl1.replacements                    9 # total number of replacements
dl1.writebacks                      7 # total number of writebacks
dl1.invalidations                   0 # total number of invalidations
dl1.miss_rate                  0.0886 # miss rate (i.e., misses/ref)
dl1.repl_rate                  0.0018 # replacement rate (i.e., repls/ref)
dl1.wb_rate                    0.0014 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                   0.0000 # invalidation rate (i.e., invs/ref)
```
//dl1 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无效率
```
ul2.accesses                     1342 # total number of accesses
ul2.hits                          666 # total number of hits
ul2.misses                        676 # total number of misses
ul2.replacements                    0 # total number of replacements
ul2.writebacks                      0 # total number of writebacks
ul2.invalidations                   0 # total number of invalidations
ul2.miss_rate                  0.5037 # miss rate (i.e., misses/ref)
```

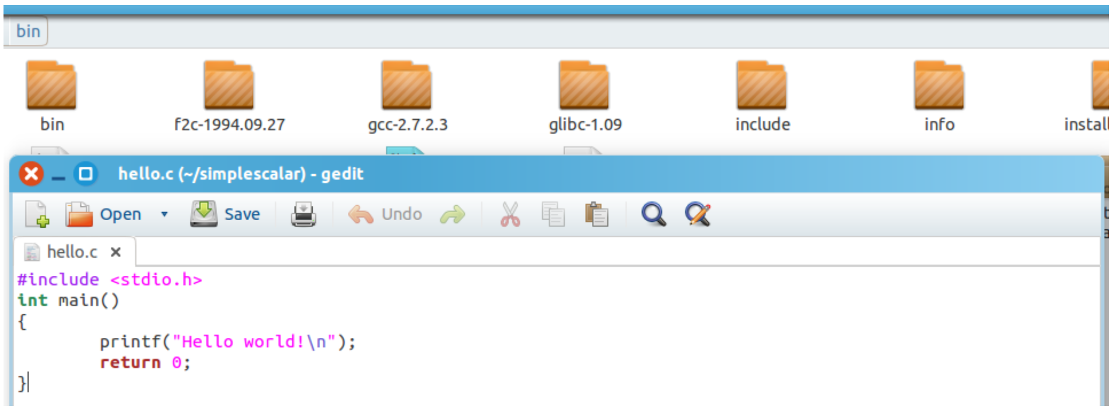
```
ul2.repl_rate                    0.0000 # replacement rate (i.e., repls/ref)
ul2.wb_rate                      0.0000 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                     0.0000 # invalidation rate (i.e., invs/ref)
//ul2 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无
效率
itlb.accesses                     12686 # total number of accesses
itlb.hits                         12668 # total number of hits
itlb.misses                          18 # total number of misses
itlb.replacements                     0 # total number of replacements
itlb.writebacks                       0 # total number of writebacks
itlb.invalidations                    0 # total number of invalidations
itlb.miss_rate                   0.0014 # miss rate (i.e., misses/ref)
itlb.repl_rate                   0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                     0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate                    0.0000 # invalidation rate (i.e., invs/ref)
//itlb 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，无
效率
dtlb.accesses                      5142 # total number of accesses
dtlb.hits                          5133 # total number of hits
dtlb.misses                           9 # total number of misses
dtlb.replacements                     0 # total number of replacements
dtlb.writebacks                       0 # total number of writebacks
dtlb.invalidations                    0 # total number of invalidations
dtlb.miss_rate                   0.0018 # miss rate (i.e., misses/ref)
dtlb.repl_rate                   0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                     0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate                    0.0000 # invalidation rate (i.e., invs/ref)
//dtlb 访问数，命中数，丢失数，替换数，写回数，无效数， 丢失率，替换率，写回率，
无效率
sim_invalid_addrs                     0 # total non-speculative bogus addresses seen (debug
var)//所有可见的非投机虚拟地址
ld_text_base                 0x00400000 # program text (code) segment base
//程序的文本（代码段）的基地址
ld_text_size                      98640 # program text (code) size in bytes
//程序的文本（代码段）的字节大小
ld_data_base                 0x10000000 # program initialized data segment base
//程序初始数据段的基地址
ld_data_size                      12288 # program init'ed `.data' and uninit'ed `.bss' size in
bytes
ld_stack_base                0x7fffc000 # program stack segment base (highest address in
stack)// 程序的堆栈的基地址(堆栈中的最高地址)
ld_stack_size                     16384 # program initial stack size
//程序的初始堆栈大小
ld_prog_entry                0x00400140 # program entry point (initial PC)
```

//程序的入口点(初始的 pc 值)
ld_environ_base 0x7fff8000 # program environment base address address
//程序环境的基地址
ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian**//程序是否为大端，非 0 即是。**
mem.page_count 34 # total number of pages allocated
//分配的页面总数
mem.page_mem 136k # total size of memory pages allocated
//分配的内存页的大小
mem.ptab_misses 34 # total first level page table misses
//一级页表丢失数
mem.ptab_accesses 708272 # total page table accesses
//总页表访问数
mem.ptab_miss_rate 0.0000 # first level page table miss rate
//一级页表丢失率
Aborted
//模拟运行终止

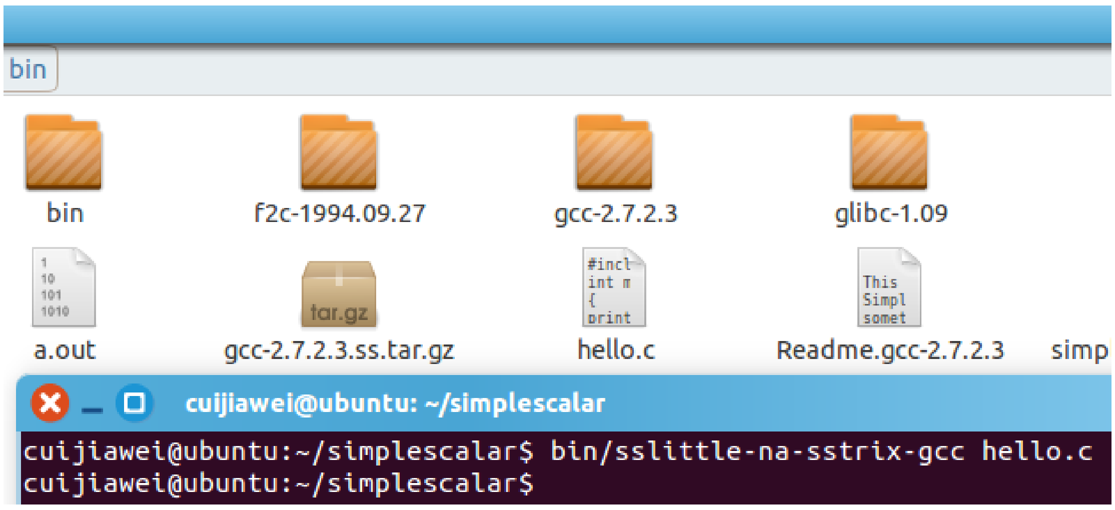
崔嘉伟部分

一．编译程序

用 simplescalar 的编译器对 hello.c 进行编译，以生成能够在模拟器中运行的可执行文件，此条命令将 hello.c 编译成 a.out.这种可执行文件并不是通常意义下的可执行文件，它的可执行性是相对于模拟器程序而已的。



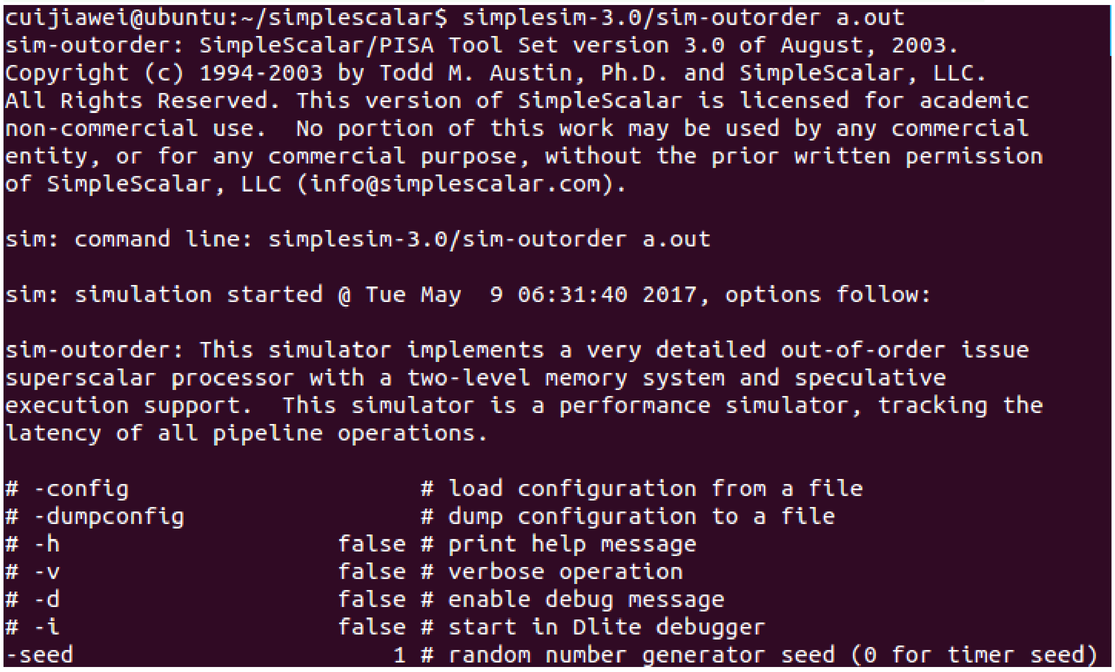
```
#include <stdio.h>
int main()
{
        printf("Hello world!\n");
        return 0;
}
```



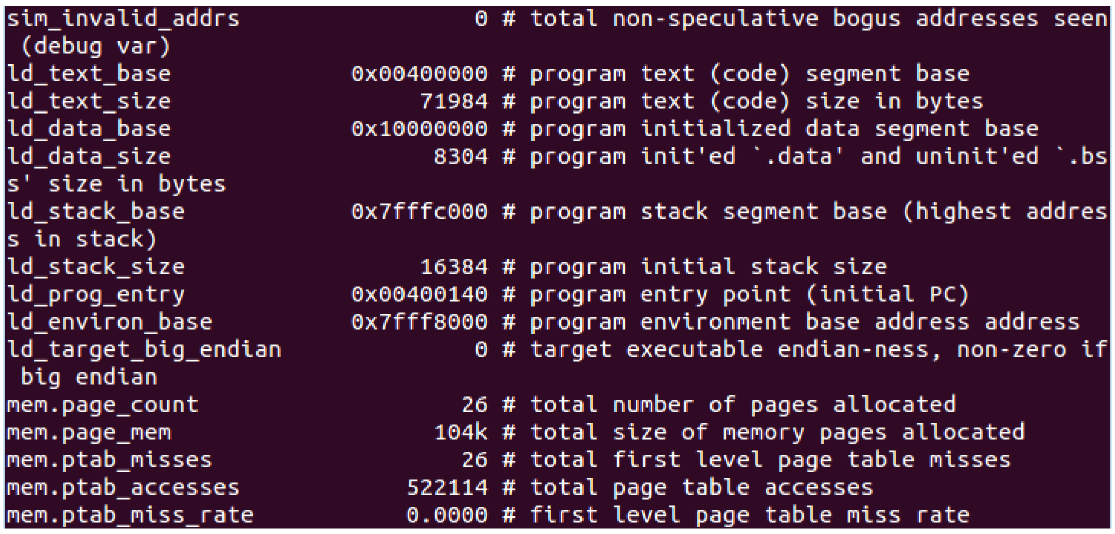
```
cuijiawei@ubuntu:~/simplescalar$ bin/sslittle-na-sstrix-gcc hello.c
cuijiawei@ubuntu:~/simplescalar$
```

二．模拟运行

用 sim-outorder 对 a.out 进行模拟运行。它是 simplescalar 中的一个模拟器。



```
cuijiawei@ubuntu:~/simplescalar$ simplesim-3.0/sim-outorder a.out
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: simplesim-3.0/sim-outorder a.out

sim: simulation started @ Tue May  9 06:31:40 2017, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

# -config                     # load configuration from a file
# -dumpconfig                 # dump configuration to a file
# -h                    false # print help message
# -v                    false # verbose operation
# -d                    false # enable debug message
# -i                    false # start in Dlite debugger
-seed                       1 # random number generator seed (0 for timer seed)
```



```
sim_invalid_addrs                 0 # total non-speculative bogus addresses seen
 (debug var)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  71984 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                   8304 # program init'ed `.data' and uninit'ed `.bs
s' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest addres
s in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if
 big endian
mem.page_count                   26 # total number of pages allocated
mem.page_mem                   104k # total size of memory pages allocated
mem.ptab_misses                  26 # total first level page table misses
mem.ptab_accesses            522114 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

三．仿真结果

以下为用 sim-outorder 对 a.out 进行模拟运行后的结果(删去部分内容)：

```
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: simplesim-3.0/sim-outorder a.out

sim: simulation started @ Tue May  9 06:31:40 2017, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

# -config                     # load configuration from a file
//加载配置文件
# -dumpconfig                 # dump configuration to a file
//将配置文件转储到文件中
# -h                    false # print help message
//打印 help 信息
# -v                    false # verbose operation
//详细操作
# -d                    false # enable debug message
//启用调试信息
# -i                    false # start in Dlite debugger
//在 Dlite 调试器中开始
-seed                       1 # random number generator seed (0 for timer seed)
//随机数生成器种子(0 用于计时器种子)

# -q                    false # initialize and terminate immediately
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim           <null> # redirect simulator output to file (non-interactive only)
# -redir:prog          <null> # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-max:inst                   0 # maximum number of inst's to execute
-fastfwd                    0 # number of insts skipped before timing starts
# -ptrace              <null> # generate pipetrace, i.e., <fname|stdout|stderr>
```

<range>
红色部分为 Sim-outorder 特有参数：
-fetch:ifqsize   4 # instruction fetch queue size (in insts)
//指令获取队列大小
-fetch:mplat   3 # extra branch mis-prediction latency
//额外的分支预测错误延迟
-fetch:speed   1 # speed of front-end of machine relative to execution Core
//前端相对于计算核心的速度
-bpred   bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod   2048 # bimodal predictor config (<table size>)
-bpred:2lev   1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb   1024 # combining predictor config (<meta_table_size>)
-bpred:ras   8 # return address stack size (0 for no return stack)
-bpred:btb   512 4 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update   <null> # speculative predictors update in {ID|WB} (default non-spec)
-decode:width   4 # instruction decode B/W (insts/cycle)
//指令解码 B / W(insts/周期)
-issue:width   4 # instruction issue B/W (insts/cycle)
-issue:inorder   false # run pipeline with in-order issue
-issue:wrongpath   true # issue instructions down wrong execution paths
-commit:width   4 # instruction commit B/W (insts/cycle)
-ruu:size   16 # register update unit (RUU) size
-lsq:size   8 # load/store queue (LSQ) size
-cache:dl1   dl1:128:32:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat   1 # l1 data cache hit latency (in cycles)
-cache:dl2   ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat   6 # l2 data cache hit latency (in cycles)
-cache:il1   il1:512:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1lat   1 # l1 instruction cache hit latency (in cycles)
-cache:il2   dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2lat   6 # l2 instruction cache hit latency (in cycles)
-cache:flush   false # flush caches on system calls
-cache:icompress   false # convert 64-bit inst addresses to 32-bit inst equivalents
-mem:lat   18 2 # memory access latency (<first_chunk> <inter_chunk>)
-mem:width   8 # memory access bus width (in bytes)
-tlb:itlb   itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb   dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}

```
-tlb:lat                   30 # inst/data TLB miss latency (in cycles)
-res:ialu                   4 # total number of integer ALU's available
-res:imult                  1 # total number of integer multiplier/dividers available
-res:memport                2 # total number of memory system ports available (to CPU)
-res:fpalu                  4 # total number of floating point ALU's available
-res:fpmult                 1 # total number of floating point multiplier/dividers available
# -pcstat              <null> # profile stat(s) against text addr's (mult uses ok)
-bugcompat              false # operate in backward-compatible bugs mode (for testing only)



sim: ** starting performance simulation **
Hello world!
sim: ** simulation statistics **
//仿真数据
sim_num_insn                   7942 # total number of instructions committed
//执行的全部指令的条数
sim_num_refs                   4337 # total number of loads and stores committed
//执行的负载和存储的指令条数
sim_num_loads                   791 # total number of loads committed
//提交的负载数量
sim_num_stores            3546.0000 # total number of stores committed
//提交的存储数量
sim_num_branches               1155 # total number of branches committed
//提交的分支数量
sim_elapsed_time                  1 # total simulation time in seconds
//模拟时间
sim_inst_rate             7942.0000 # simulation speed (in insts/sec)
//每秒钟执行的指令数，在这个测试程序中没有意义，因为指令数太少，大概可以达到每秒80万左右
sim_total_insn                 8715 # total number of instructions executed
sim_total_refs                 4581 # total number of loads and stores executed
sim_total_loads                 946 # total number of loads executed
sim_total_stores          3635.0000 # total number of stores executed
sim_total_branches             1289 # total number of branches executed
sim_cycle                     12143 # total simulation time in cycles
//周期内的总模拟时间
sim_IPC                      0.6540 # instructions per cycle
//每个周期指令数
```

```
sim_CPI                      1.5290 # cycles per instruction
//每条指令所需周期
sim_exec_BW                  0.7177 # total instructions (mis-spec + committed)
per cycle
//总指令数(错误配置+已提交)/周期
sim_IPB                      6.8762 # instruction per branch
//每条分支指令数
il1.accesses                   9173 # total number of accesses
//总访问数
il1.hits                       8738 # total number of hits
//命中数
il1.misses                      435 # total number of misses
//丢失数
il1.replacements                174 # total number of replacements
//替代数
il1.writebacks                    0 # total number of writebacks
//回写数
il1.invalidations                 0 # total number of invalidations
il1.miss_rate                0.0474 # miss rate (i.e., misses/ref)
il1.repl_rate                0.0190 # replacement rate (i.e., repls/ref)
il1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses                   4381 # total number of accesses
dl1.hits                       3951 # total number of hits
dl1.misses                      430 # total number of misses
dl1.replacements                  4 # total number of replacements
dl1.writebacks                    3 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0982 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0009 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0007 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses                    868 # total number of accesses
ul2.hits                        426 # total number of hits
ul2.misses                      442 # total number of misses
ul2.replacements                  0 # total number of replacements
ul2.writebacks                    0 # total number of writebacks
ul2.invalidations                 0 # total number of invalidations
ld_text_base             0x00400000 # program text (code) segment base
//程序文本(代码)段基
ld_text_size                  71984 # program text (code) size in bytes
//程序文本(代码)大小(以字节为单位)
ld_data_base             0x10000000 # program initialized data segment base
//程序初始化数据段基
```

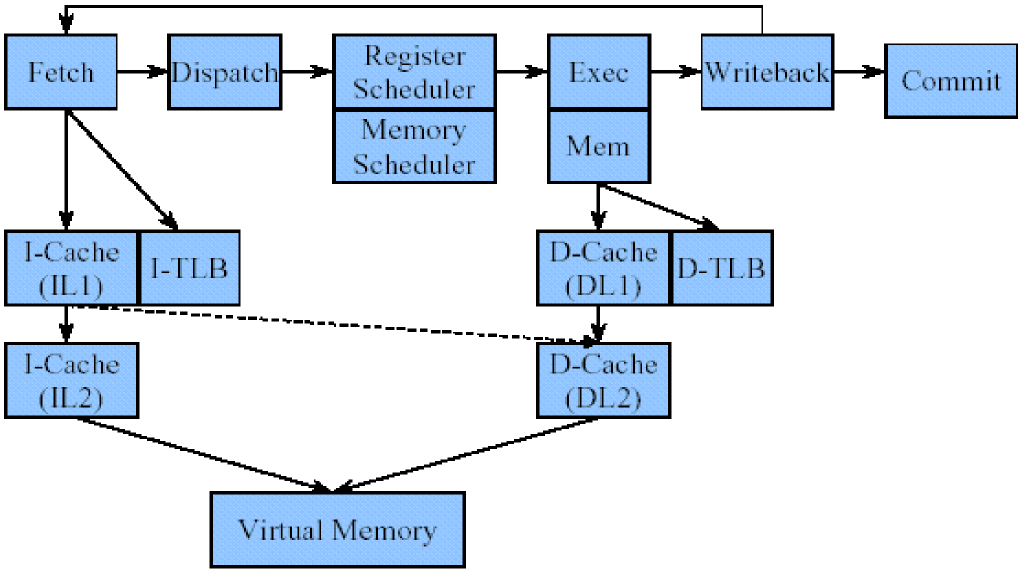
```
ld_data_size                     8304 # program init'ed `.data' and uninit'ed
`.bss' size in bytes
ld_stack_base               0x7fffc000 # program stack segment base (highest
address in stack)
//程序堆栈段基(堆栈中的最高地址)
ld_stack_size                   16384 # program initial stack size
//程序初始堆栈大小
ld_prog_entry           0x00400140 # program entry point (initial PC)
//程序入口点(初始 PC)
ld_environ_base         0x7fff8000 # program environment base address address
//程序环境基地址
ld_target_big_endian                0 # target executable endian-ness, non-zero
if big endian
mem.page_count                     26 # total number of pages allocated
//分配的页面总数
mem.page_mem                     104k # total size of memory pages allocated
//分配的内存页的大小
mem.ptab_misses                    26 # total first level page table misses
//一级页表丢失数
mem.ptab_accesses              522114 # total page table accesses
//总页表访问数
mem.ptab_miss_rate             0.0000 # first level page table miss rate
//一级页表丢失率
```

杨莹部分

sim-outorder 分析

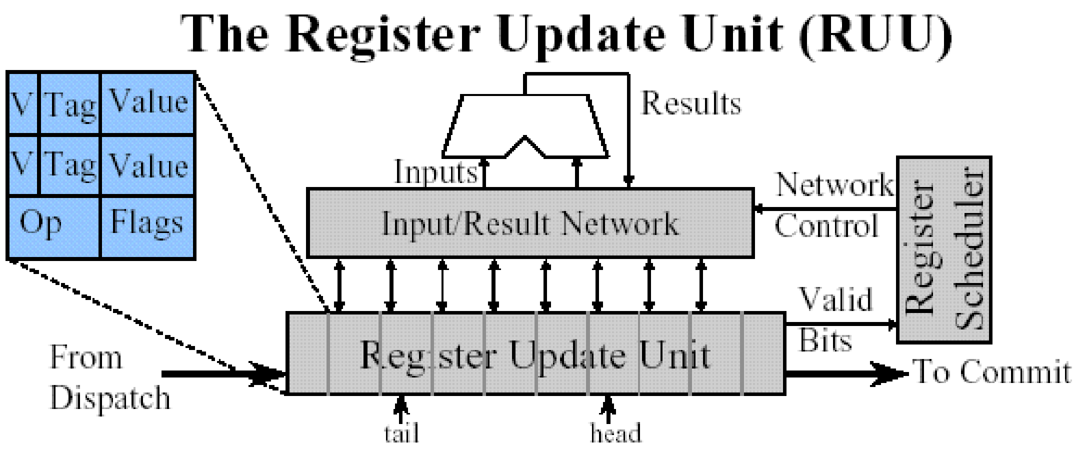
sim-outorder 实现了对一个非常详细的支持乱序发射，拥有一个二级的 memory 和推断执行的超标量处理器的仿真，本身拥有很高的性能，而且对整个程序执行期间流水线的状态都进行了记录，基本上包括了以上各种仿真器的全部功能。

一、sim-outorder的指令执行流程



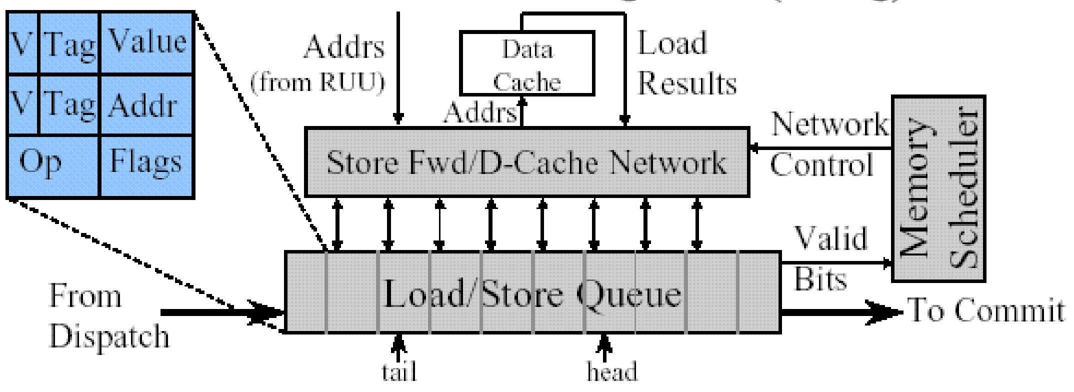
sim-outorder 的动态调度方法基本上是加上了分支预测的 Tomasulo 算法，它的关键部件是寄存器更新部件 RUU 和存取缓冲队列 LSQ。这两个部件的说明如下：

RUU:



RUU负责寄存器间的信息交换和同步，它包含保留站和再定序缓冲的功能，指令从Dispatch段进入，分配RUU，在commit段释RUU,作到乱序发射，顺序提交。RUU被组织成一个循环队列，之间通过总线交换信息。

LSQ:



The Load/Store Queue (LSQ)

LSQ 负责存储器间的同步和信息交换，它包含了 LOAD/STORE 指令，指令被分成有效地址计算指令和真正的存取原语指令，前者位于 RUU 中。

流水段描述：

取指段（fetch）：

从一级指令 cache 中取指令，并把它放入 IFQ 中。判断下一条指令是否是预测执行的指令，同时接收其他段送来的预测失败信息。

分派指令段(dispatch)：

从 LSQ 中取指令解码，重命名寄存器，分配 RUU 和 LSQ，支持早期的分支预测失败的探测。

发射段（schedule)

发射操作数已经准备好的指令到执行段。

执行段(execution/mem)：

指令执行，输出寄存器 / 存储器写回事件。

写回段（write-back)：

执行写回事件，发出操作数可用信息到总线，触发分支预测失败的恢复事件。

提交段（commit）：

指令结束，释放 RUU 和 LSQ。

指令执行时流水线状态描述：

利用仿真器自带的PERL程序可以看outorder的流水线中每个cycle的状态记录，方法如下：

在执行时加入-ptrace <file> <range>参数，将结果存入file指定的文件中，

用pipeview.pl <ptrace_file>来浏览。

二、sim-outorder 中的设置选项及参数描述

# -config		# load configuration from a file 从一个文件中载入设置
# -dumpconfig		# dump configuration to a file 将目前的设置写入文件中
# -h	false	# print help message 打印帮助信息
# -v	false	# verbose operation 详细地显示执行的所有指令的列表
# -d	false	# enable debug message 显示调试信息
# -i	false	# start in Dlite debugger 启动 Dlite 调试器
-seed	1	# random number generator seed (0 for timer seed) 设置产生随机数的种子

```
# -q                          false # initialize and terminate immediately
                                   进行初始化并立即结束
# -chkpt                     <null> # restore EIO trace execution from <fname>
                                   从一个文件中恢复 EIO 跟踪程序的执行
# -redir:sim                 <null> # redirect simulator output to file (non-interactive only)
                                  将仿真器的输入写入一个文件中
# -redir:prog                <null> # redirect simulated program output to file
                                  将 benchmark 程序的输入写入到一个文件中
-nice                            0 # simulator scheduling priority
                                   设置仿真器的调度优先级
-max:inst                        0 # maximum number of inst's to execute
                                   设置仿真器最多执行的指令条数
-fastfwd                         0 # number of insts skipped before timing starts
                                   开始计时前跳过的指令的条数
# -ptrace                    <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
                                   生成流水线的状态信息
-fetch:ifqsize                   4 # instruction fetch queue size (in insts)
                                 取指队列的长度
-fetch:mplat                     3 # extra branch mis-prediction latency
                                  分支预测失败的延迟开销
-fetch:speed                     1 # speed of front-end of machine relative to execution core
-bpred                                          bimod  #  branch  predictor  type
{nottaken|taken|perfect|bimod|2lev|comb}                    分支预测的类型
-bpred:bimod         2048 # bimodal predictor config (<table size>)
                          设置 bimodal 类型的分支预测表的大小
-bpred:2lev          1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
                              2 级分支预测设置
-bpred:comb          1024 # combining predictor config (<meta_table_size>)
                            采用复合分支预测方法的设置
-bpred:ras                       8 # return address stack size (0 for no return stack)
                                   设置返回地址栈的大小（0 表示不设立该栈）
-bpred:btb           512 4 # BTB config (<num_sets> <associativity>)设置 BTB
#  -bpred:spec_update          <null>  #  speculative  predictors  update  in  {ID|WB}  (default
non-spec)                                 设置推断预测方法的更新时间
-decode:width                    4 # instruction decode B/W (insts/cycle)
                                    设置译码器的带宽（单位为字节）
-issue:width                     4 # instruction issue B/W (insts/cycle)
                                   设置指令发射的带宽（单位为字节）
-issue:inorder              false # run pipeline with in-order issue 顺序发射指令
-issue:wrongpath             true # issue instructions down wrong execution paths
                                   在预测错误时仍然允许发射指令
-commit:width                    4 # instruction commit B/W (insts/cycle)
                                    设置确认的带宽（单位为字节）
-ruu:size                       16 # register update unit (RUU) size  寄存器更新部件的大小
```

-lsq:size 8 # load/store queue (LSQ) size

load/store 队列（LSQ）的大小

-cache:dl1 dl1:128:32:4:l # l1 data cache config, i.e., {<config>|none}

设置 level1 数据 cache

-cache:dl1lat 1 # l1 data cache hit latency (in cycles)

level1 数据 cache 的命中时间

-cache:dl2 ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}

设置 level2 数据 cache

-cache:dl2lat 6 # l2 data cache hit latency (in cycles)

level2 数据 cache 的命中时间

-cache:il1 il1:512:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}

level1 指令 cache 设置

-cache:il1lat 1 # l1 instruction cache hit latency (in cycles)

level1 指令 cache 的命中时间

-cache:il2 dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}

level2 指令 cache 设置

-cache:il2lat 6 # l2 instruction cache hit latency (in cycles)

level2 指令 cache 的命中时间

-cache:flush false # flush caches on system calls

在系统调用的时候写回所有 cache

-cache:icompress false # convert 64-bit inst addresses to 32-bit inst equivalents

将 64-bit 的指令转换成相应的 32-bit 的指令

-mem:lat 18 2 # memory access latency (<first_chunk> <inter_chunk>)

内存访问时间

-mem:width 8 # memory access bus width (in bytes)

内存访问带宽（单位为字节）

-tlb:itlb itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}

设置指令 TLB

-tlb:dtlb dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}

设置数据 TLB

-tlb:lat 30 # inst/data TLB miss latency (in cycles)

指令、数据 TLB 的失效开销（周期）

-res:ialu 4 # total number of integer ALU's available

设置整数 ALU 的最大个数

-res:imult 1 # total number of integer multiplier/dividers available

设置整数 乘法器/除法 的最大个数

-res:memport 2 # total number of memory system ports available (to CPU)

设置内存的访问端口的最大数目

-res:fpalu 4 # total number of floating point ALU's available

设置浮点数 ALU 的最大个数

-res:fpmult 1 # total number of floating point multiplier/dividers available

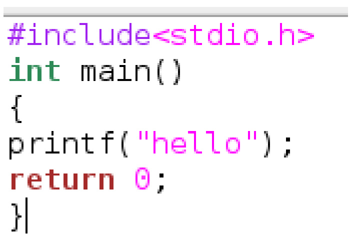
设置浮点数 乘法器/触发器的最大个数

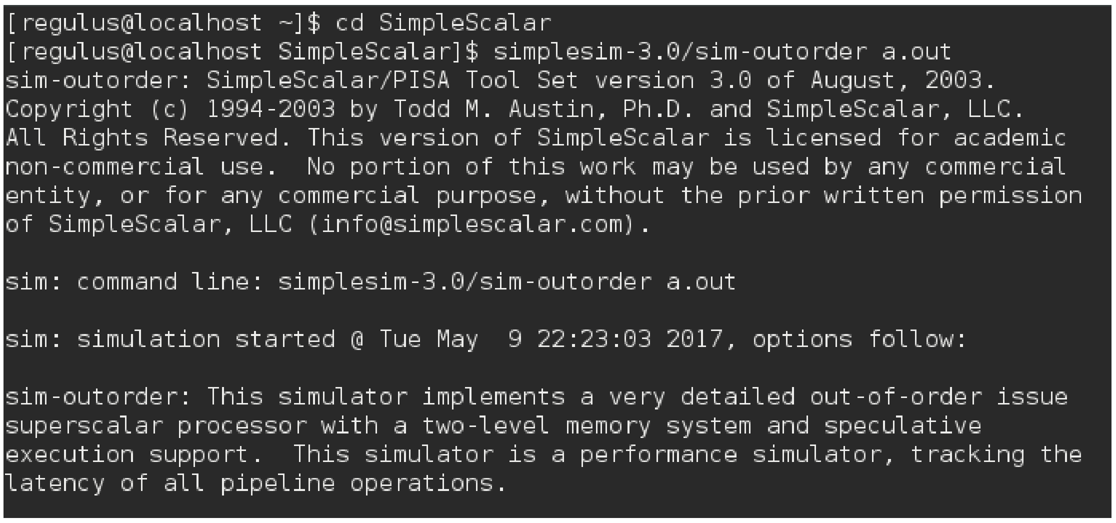
# -pcstat <null> # profile stat(s) against text addr's (mult uses ok)

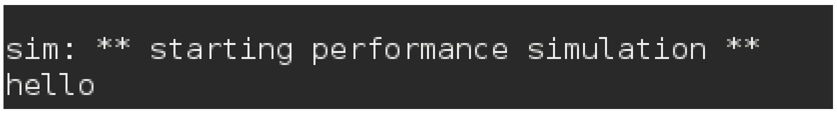
显示代码的状态而不是地址

-bugcompat   false # operate in backward-compatible bugs mode (for testing only)   设置是否开启向后兼容错误的工作模式（仅为测试使用）

三、sim-outorder 的程序执行统计参数描述


```
#include<stdio.h>
int main()
{
printf("hello");
return 0;
}
```


```
[regulus@localhost ~]$ cd SimpleScalar
[regulus@localhost SimpleScalar]$ simplesim-3.0/sim-outorder a.out
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: simplesim-3.0/sim-outorder a.out

sim: simulation started @ Tue May  9 22:23:03 2017, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.
```


```
sim: ** starting performance simulation **
hello
```

sim_num_insn	7159 # total number of instructions committed 确认(committed)的指令条数
sim_num_refs	4042 # total number of loads and stores committed 确认的 load/store 的个数
sim_num_loads	576 # total number of loads committed 确认的 load 操作的个数
sim_num_stores	3466.0000 # total number of stores committed 确认的 store 操作的个数
sim_num_branches	965 # total number of branches committed 确认的分支操作的个数

sim_elapsed_time	1 # total simulation time in seconds
	程序执行的时间(以秒为单位)

sim_inst_rate	7159.0000 # simulation speed (in insts/sec)
	程序执行的速度(条指令/秒)

sim_total_insn	7931 # total number of instructions executed 一共执行的指令的条数

```
sim_total_refs                    4280 # total number of loads and stores executed
                                         一共执行的 load/store 的次数
sim_total_loads                    728 # total number of loads executed
                                         一共执行的 load 个数
sim_total_stores             3552.0000 # total number of stores executed
                                         一共执行的 store 个数
sim_total_branches                1091 # total number of branches executed
                                         一共执行的 branch 个数
sim_cycle                        12109 # total simulation time in cycles
                                         一共执行的周期数
sim_IPC                         0.5912 # instructions per cycle
                                         平均每个周期执行的指令条数
sim_CPI                         1.6914 # cycles per instruction
                                         平均每条指令执行的周期数
sim_exec_BW                     0.6550 # total instructions (mis-spec + committed) per
cycle           平均每个周期实际执行的指令条数(包括分支预测错误时执行的无用的指令)
sim_IPB                         7.4187 # instruction per branch
                                         平均多少条指令中有一条分支指令
IFQ_count                        11701 # cumulative IFQ occupancy
IFQ_fcount                        2664 # cumulative IFQ full count
ifq_occupancy                   0.9663 # avg IFQ occupancy (insn's)
ifq_rate                        0.6550 # avg IFQ dispatch rate (insn/cycle)
                                         IFQ 的平均发射速率（指令/周期)
ifq_latency                     1.4753 # avg IFQ occupant latency (cycle's)
ifq_full                        0.2200 # fraction of time (cycle's) IFQ was full
RUU_count                        35860 # cumulative RUU occupancy
RUU_fcount                         149 # cumulative RUU full count
ruu_occupancy                   2.9614 # avg RUU occupancy (insn's)
ruu_rate                        0.6550 # avg RUU dispatch rate (insn/cycle)
                                          RUU 的平均发射速率（指令/周期)
ruu_latency                     4.5215 # avg RUU occupant latency (cycle's)
ruu_full                        0.0123 # fraction of time (cycle's) RUU was full
LSQ_count                        20470 # cumulative LSQ occupancy
LSQ_fcount                        1610 # cumulative LSQ full count
lsq_occupancy                   1.6905 # avg LSQ occupancy (insn's)
lsq_rate                        0.6550 # avg LSQ dispatch rate (insn/cycle)
                                         LSQ 的平均发射速率（指令/周期)
lsq_latency                     2.5810 # avg LSQ occupant latency (cycle's)
lsq_full                        0.1330 # fraction of time (cycle's) LSQ was full
sim_slip                         64755 # total number of slip cycles
avg_sim_slip                    9.0453 # the average slip between issue and retirement
bpred_bimod.lookups               1134 # total number of bpred lookups
                                           一共进行了多少次分支预测
bpred_bimod.updates                965 # total number of updates
```

```
                                  分支预测表更新的次数
bpred_bimod.addr_hits               708 # total number of address-predicted hits
                                  分支目标地址预测命中的次数
bpred_bimod.dir_hits               829 # total number of direction-predicted hits (includes
addr-hits)                        分支目标方向预测命中的次数
bpred_bimod.misses                  136 # total number of misses
                                  分支预测错误的次数
bpred_bimod.jr_hits                 59 # total number of address-predicted hits for JR's
                                  JR 型分支目标地址预测命中的次数
bpred_bimod.jr_seen                  67 # total number of JR's seen
                                  JR 型分支出现的次数
bpred_bimod.jr_non_ras_hits.PP              0 # total number of address-predicted hits for
non-RAS JR's                          non_RAS JR 型分支目标地址预测命中的次数
bpred_bimod.jr_non_ras_seen.PP               0 # total number of non-RAS JR's seen
                                       non_RAS JR 型分支出现的次数
bpred_bimod.bpred_addr_rate      0.7337 # branch address-prediction rate (i.e., addr-
hits/updates)                       分支目标地址预测的命中率
bpred_bimod.bpred_dir_rate       0.8591 # branch direction-prediction rate (i.e., all-
hits/updates)                       分支目标方向预测的命中率
bpred_bimod.bpred_jr_rate     0.8806 # JR address-prediction rate (i.e., JR addr-hits/JRs
seen)                             JR 型分支目标地址预测的命中率
bpred_bimod.bpred_jr_non_ras_rate.PP <error: divide by zero> # non-RAS JR addr-pred rate
(ie, non-RAS JR hits/JRs seen)               non_RAS JR 型分支目标地址预测的命中率
bpred_bimod.retstack_pushes               94 # total number of address pushed onto ret-
addr stack                            向 RAS 中压入的地址的个数
bpred_bimod.retstack_pops              70 # total number of address popped off of ret-addr
stack                               从 RAS 中弹出的地址的个数
bpred_bimod.used_ras.PP               67 # total number of RAS predictions used
                                  RAS  预测使用的总次数
bpred_bimod.ras_hits.PP              59 # total number of RAS hits
                                 RAS  预测命中的次数
bpred_bimod.ras_rate.PP       0.8806 # RAS prediction rate (i.e., RAS hits/used RAS)
                              RAS  预测的命中率
il1.accesses                      8356 # total number of accesses
                                il1 访问的次数
il1.hits                         7917 # total number of hits
                                il1 命中的次数
il1.misses                        439 # total number of misses
                                il1 不命中的次数
il1.replacements                  177 # total number of replacements
                                il1 替换的次数
il1.writebacks                      0 # total number of writebacks
                                il1 写回的次数
il1.invalidations                  0 # total number of invalidations
```

```
                                        il1 无效的次数
il1.miss_rate                 0.0525 # miss rate (i.e., misses/ref)
                                       il1 失效率(不命中次数/访问次数)
il1.repl_rate                 0.0212 # replacement rate (i.e., repls/ref)
                                       il1 替换率(替换次数/访问次数)
il1.wb_rate                   0.0000 # writeback rate (i.e., wrbks/ref)
                                       il1 写回率(写回次数/访问次数)
il1.inv_rate                  0.0000 # invalidation rate (i.e., invs/ref)
                                       il1 无效率(无效次数/访问次数)
dl1.accesses                    4085 # total number of accesses
                                         dl1 访问的次数
dl1.hits                        3654 # total number of hits
                                         dl1 命中的次数
dl1.misses                       431 # total number of misses
                                         dl1 不命中的次数
dl1.replacements                   2 # total number of replacements
                                         dl1 替换的次数
dl1.writebacks                     1 # total number of writebacks
                                         dl1 写回的次数
dl1.invalidations                  0 # total number of invalidations
                                       dl1 无效的次数
dl1.miss_rate                 0.1055 # miss rate (i.e., misses/ref)
                                       dl1 失效率(不命中次数/访问次数)
dl1.repl_rate                 0.0005 # replacement rate (i.e., repls/ref)
                                       dl1 替换率(替换次数/访问次数)
dl1.wb_rate                   0.0002 # writeback rate (i.e., wrbks/ref)
                                        dl1 写回率(写回次数/访问次数)
dl1.inv_rate                  0.0000 # invalidation rate (i.e., invs/ref)
                                        dl1 无效率(无效次数/访问次数)
ul2.accesses                     871 # total number of accesses
                                        ul2 访问的次数
ul2.hits                         421 # total number of hits
                                        ul2 命中的次数
ul2.misses                       450 # total number of misses
                                        ul2 不命中的次数
ul2.replacements                   0 # total number of replacements
                                        ul2 替换的次数
ul2.writebacks                     0 # total number of writebacks
                                        ul2 写回的次数
ul2.invalidations                  0 # total number of invalidations
                                        ul2 无效的次数
ul2.miss_rate                 0.5166 # miss rate (i.e., misses/ref)
                                        ul2 失效率(不命中次数/访问次数)
ul2.repl_rate                 0.0000 # replacement rate (i.e., repls/ref)
```

```
                                      ul2 替换率(替换次数/访问次数)
ul2.wb_rate                      0.0000 # writeback rate (i.e., wrbks/ref)
                                      ul2 写回率(写回次数/访问次数)
ul2.inv_rate                     0.0000 # invalidation rate (i.e., invs/ref)
                                      ul2 无效率(无效次数/访问次数)
itlb.accesses                      8356 # total number of accesses
                                      itlb 访问的次数
itlb.hits                          8346 # total number of hits
                                      itlb 命中的次数
itlb.misses                          10 # total number of misses
                                      itlb 不命中的次数
itlb.replacements                     0 # total number of replacements
                                      itlb 替换的次数
itlb.writebacks                       0 # total number of writebacks
                                      itlb 写回的次数
itlb.invalidations                    0 # total number of invalidations
                                      itlb 无效的次数
itlb.miss_rate                   0.0012 # miss rate (i.e., misses/ref)
                                      itlb 失效率(不命中次数/访问次数)
itlb.repl_rate                   0.0000 # replacement rate (i.e., repls/ref)
                                      itlb 替换率(替换次数/访问次数)
itlb.wb_rate                     0.0000 # writeback rate (i.e., wrbks/ref)
                                      itlb 写回率(写回次数/访问次数)
itlb.inv_rate                    0.0000 # invalidation rate (i.e., invs/ref)
                                      itlb 无效率(无效次数/访问次数)
dtlb.accesses                      4090 # total number of accesses
                                      dtlb 访问的次数
dtlb.hits                          4082 # total number of hits
                                      dtlb 命中的次数;
dtlb.misses                           8 # total number of misses
                                      dtlb 不命中的次数
dtlb.replacements                     0 # total number of replacements
                                      dtlb 替换的次数
dtlb.writebacks                       0 # total number of writebacks
                                      dtlb 写回的次数
dtlb.invalidations                    0 # total number of invalidations
                                      dtlb 无效的次数
dtlb.miss_rate                   0.0020 # miss rate (i.e., misses/ref)
                                      dtlb 失效率(不命中次数/访问次数)
dtlb.repl_rate                   0.0000 # replacement rate (i.e., repls/ref)
                                      dtlb 替换率(替换次数/访问次数)
dtlb.wb_rate                     0.0000 # writeback rate (i.e., wrbks/ref)
                                      dtlb 写回率(写回次数/访问次数)
dtlb.inv_rate                    0.0000 # invalidation rate (i.e., invs/ref)
```

```
                                   dtlb 无效率(无效次数/访问次数)
sim_invalid_addrs                  0 # total non-speculative bogus addresses seen (debug
var)
ld_text_base              0x00400000 # program text (code) segment base
                                   程序的代码段的基地址
ld_text_size                   71968 # program text (code) size in bytes
                                   程序的代码段的大小
ld_data_base              0x10000000 # program initialized data segment base
                                   程序的数据段的基地址
ld_data_size                    8304 # program init'ed `.data' and uninit'ed `.bss' size in
bytes                               开始时程序的数据端的大小
ld_stack_base             0x7fffc000 # program stack segment base (highest address in
stack)                             程序的堆栈的基地址(高端地址)
ld_stack_size                  16384 # program initial stack size
                                   开始时程序的堆栈的大小
ld_prog_entry             0x00400140 # program entry point (initial PC)
                                   程序的入口点(初始的 pc 值)
ld_environ_base           0x7fff8000 # program environment base address address
                                   程序环境的基地址
ld_target_big_endian               0 # target executable endian-ness, non-zero if big
endian                           benchmark 程序是小尾端还是大尾端(非零表示大围端)
mem.page_count                    26 # total number of pages allocated
                                   一共分配的页数
mem.page_mem                    104k # total size of memory pages allocated
                                   程序运行过程中分配内存的大小
mem.ptab_misses                   26 # total first level page table misses
                                    访问第一级页表的失效的次数
mem.ptab_accesses             515978 # total page table accesses
                                    访问页表的总次数
mem.ptab_miss_rate            0.0001 # first level page table miss rate
                                    第一级页表的失效率
```

徐丹雅部分

sim-outorder 与 SimpleScalar

SimpleScalar 模拟器是一个超标量、5 级流水的 RISC(Reduced Instruction Set Computing)体系结构模拟器，提供了从最简单的功能模拟到超标量乱序发射的不同的模拟程序。

SimpleScalar 模拟器在功能级上实现了执行驱动、解释执行，在行为级上实现了流水线模拟。该工具集提供了一个以 GCC 为主的编译器以及相关组件，能够产生基于 SimpleScalar 体系结构的目标代码，然后在 SimpleScalar 模拟器上运行。

运行模拟器时，主程序 main( )做所有的初始化工作，并将二进制目标码载入内存，然后调用 sim_main()，sim_main()在每个模拟器中单独说明，预先译码整个正文段，加快模拟。然后开始目标程序的模拟：
sim-outorder 是一个具有完整功能的模拟程序，在 sim-outorder 中使用了几乎所有的模拟资源。

具体乱序执行策略

有五种很重要的功能单元支持 sim-outorder 对指令序列的乱序执行：保留站与重定序缓冲（RUU）、Load/Store 队列（LSQ）、取指队列、输入输出相关链和寄存器忙闲表。它们在 simplescalar 中是通过五种数据结构来实现的。

RUU 单元实现寄存器的同步和通讯功能，它将再定序缓冲和保留站统一起来，作为一个循环队列来管理。RUU 队列记录了指令的操作类型、源操作数、数据有效性标识。其中的数据项在指令发射时分配，在提交时回收；当寄存器数据和存储器数据相关性满足时，实现乱序流出；

Load/Store 队列处理存储器的相关性问题。如果 store 操作是猜测执行的，其值就被放入队列中。当所有之前的写入地址都已知之后，Load 操作就可以访存。如果地址匹配，load 操作可以在存储系统或者 Load/Store 队列中以前的 store 值的允许下进行。

取指队列是由取指段建立，在调度段译码并调度的指令队列；没有被调度的指令仍留在其中。它是用一个结构数组来实现的。

所谓输入输出相关链，即是用来记录前一条指令的输出数据（结果操作数）与后几条指令的输入数据（源操作数）的相关性的链表。

所谓寄存器忙闲表，即是用来记录当前各个寄存器被哪一条指令占用的结构数组。

Sim-outorder 的具体的乱序过程如下：

A.取指段：根据配置的各种参数的要求，从一级指令 Cache 里预取指令，加入到取指队列里。如果在一级 cache 里找不到指令，同时配置了二级 cache，就试图从二级 cache 里再找，否则就从存储器里寻找。

1. 根据分支预测的要求、cache 容量的支持、事先配置的取指队列的大小，确定预取多少条指令。
2. 在地址有效的条件下，取出指令，并根据一级指令 CACHE 的延时和一级指令 TLB 的延时计算出其取指延时的大小。
3. 若是分支指令，则要根据事先配置的分支预测策略预测下一条指令地址；若不是，指令地址自加一。
4. 把这一条指令加入取指队列里，更新取指队列。

B.调度段：从取指队列调度指令。指令首先被译码，然后为其分配 RUU 资源，判断是否存在数据相关性。如果不存在就可以发射出去，存在的话仍旧留在 RUU 队列里等待发射。若是访存指令则分配 LSQ 资源，最后更新输入输出的相关链。

C.发射执行段：检查从调度段发射出来的指令所需的功能部件是否可用（结构相关性），如果可用则将其发射执行。

1. 查看指令所需数据、功能部件是否准备好；
2. 如果是 store 指令，执行之。由于数据可先存在 LSQ 队列中，执行时间为零，实际的访存操作在 ruu_commit() 中执行。其他指令则需先查看无功能部件。

3. 如果是 load 指令，要确定 cache 访问的延时，先扫描 LSQ 队列看其前是否无访存地址相同的 store 指令。如果确实没有，那么 store 指令存的数据就是 load 指令要取的数据，因而访存延时为一周期；如果没就并行访问数据 cache 和数据 TLB，访存延时为二者中较大者。

4. 如果是非访存指令，操作时间为其功能部件的执行时间；不需功能部件的指令，操作时间为一个周期。如果是空指令操作时间为零。

D.LSQ 队列更新：此过程是找出下一条数据相关性被满足了的指令，并将其发射。而这是通过检查 LSQ 队列，查找存储器阻塞的情况来实现的。

E.写回段：完成把功能部件的输出数据（结果操作数）写入 RUU（registerupdate unit） 的任务。就这点来说，模拟器根据正在完成的指令的输出数据，确定取指队列中的后续指令的输入数据是否与其相关，如果是这样，将把这条指令从取指队列中调度出来进行发射。

F.提交段：这个阶段把已经完成的结果从 RUU 和 LSQ 提交到寄存器文件中，并且 LSQ 中的 store 指令将把其存储数据提交到数据 cache 中。

1. RUU 和 LSQ 中结果可提交，就执行提交。
2. 让 LSQ 中的 store 指令把其存储数据提交到数据 cache 中并计算其操作时间，其中要考虑 TLB 的延时。
3. 按序把已经完成的结果从 RUU 和 LSQ 提交到寄存器文件中，并更新 RUU 和 LSQ。

sim-outorder 程序中的参数配置

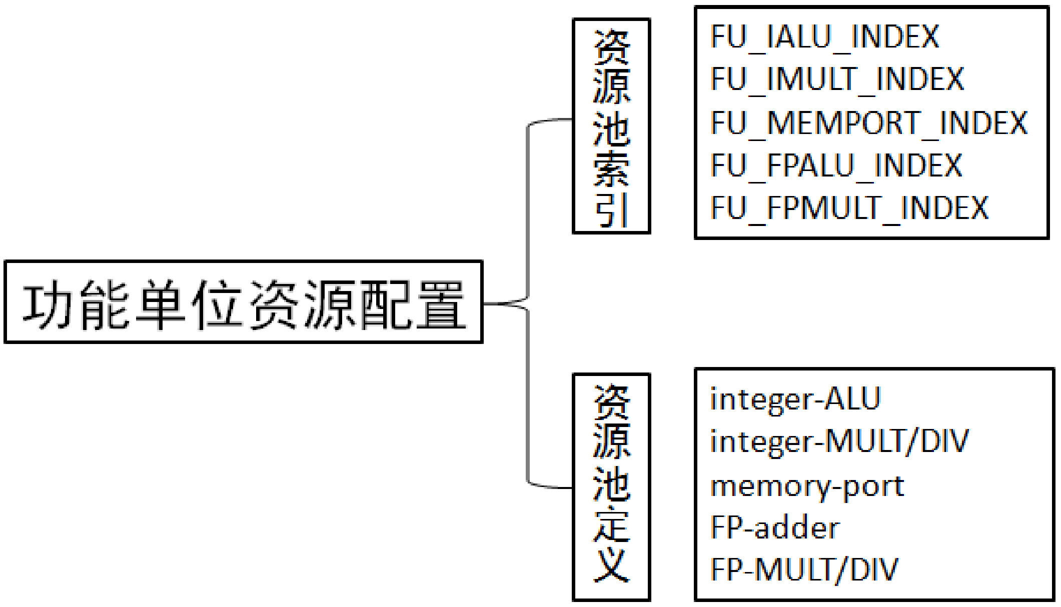
以模拟器、功能单位资源和模拟器为例：

模拟器选项

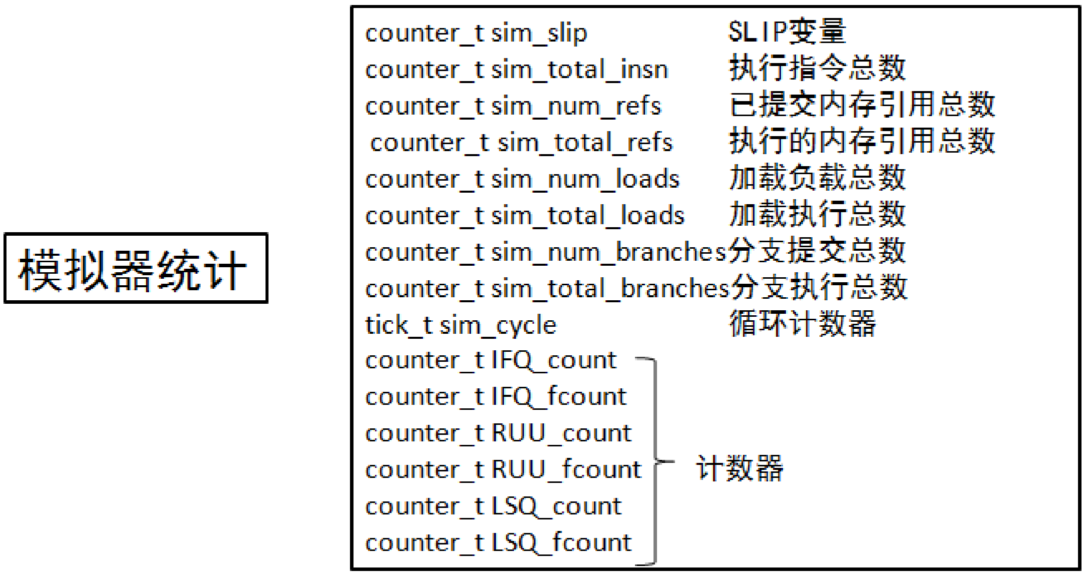
static unsigned int max_insts;	要执行的最大 inst 数
static int fastfwd_count;	定时开始之前跳过的 insts 数
static int ptrace_nelt = 0;	管道轨迹范围
static char *ptrace_opts[2];	输出文件名
static int ruu_ifq_size;	指令提取队列大小
static int ruu_branch_penalty;	额外的分支误预测延迟
static int fetch_speed;	机器相对于执行核心的前端速度
static char *pred_type;	分支预测器类型
static int bimod_nelt = 1;	双模式预测器配置
static int twolev_nelt = 4;	2 级预测器配置
static int comb_nelt = 1;	组合预测器配置
static int ras_size = 8;	返回地址堆栈（RAS）大小
static int btb_nelt = 2;	BTB 预测器配置
static int ruu_decode_width;	指令解码 B / W
static int ruu_issue_width;	指令问题 B / W
static int ruu_inorder_issue;	运行管道与按顺序问题
static int ruu_include_spec = TRUE;	发出指令错误的执行路径

static int ruu_commit_width;	指令提交 B / W（insts / cycle）
static int RUU_size = 8;	寄存器更新单元（RUU）大小
static int LSQ_size = 4;	加载/存储队列（LSQ）大小
static char *cache_dl1_opt;	l1 数据缓存配置
static int cache_dl1_lat;	l1 数据缓存命中延迟（以周期为单位）
static char *cache_il1_opt;	l1 指令缓存配置
static int cache_il1_lat;	l1 指令缓存命中延迟（以周期为单位）
static int flush_on_syscalls;	在系统调用时刷新缓存
static int compress_icache_addrs;	将 64 位本地地址转换为 32 位本体等价
static int mem_nelt = 2;	内存访问延迟
static int mem_bus_width;	内存访问总线宽度（以字节为单位）
static char *itlb_opt;	指令 TLB 配置，即{<config> | none}
static char *dtlb_opt;	数据 TLB 配置，即{<config> | none}
static int tlb_miss_lat;	inst /数据 TLB 未命中延迟（以周期为单位）
static int res_ialu;	ALU 可用的总数
static int res_imult;	整数倍数/分频器总数可用
static int res_memport;	可用的内存系统端口总数（到 CPU）
static int res_fpalu;	浮点 ALU 总数可用
static int res_fpmult;	可用的浮点乘数/分频器总数
#define MAX_PCSTAT_VARS 8	基于文本的统计资料

功能单位资源配置



模拟器统计：



counter_t sim_slip	SLIP变量
counter_t sim_total_insn	执行指令总数
counter_t sim_num_refs	已提交内存引用总数
counter_t sim_total_refs	执行的内存引用总数
counter_t sim_num_loads	加载负载总数
counter_t sim_total_loads	加载执行总数
counter_t sim_num_branches	分支提交总数
counter_t sim_total_branches	分支执行总数
tick_t sim_cycle	循环计数器
counter_t IFQ_count	
counter_t IFQ_fcount	
counter_t RUU_count	
counter_t RUU_fcount	计数器
counter_t LSQ_count	
counter_t LSQ_fcount	