

# 计算机体系结构课程实验

## 一、实验简介

本实验通过仿真器来观察 CPU 的运行状态，结合简单的应用程序观察处理器的行为。

实验主要用到的软件仿真器是 Gem5，其广泛用于计算机体系结构的研究中。可以仿真多种 CPU 模型 (Atomic, Timing, Out-of-Order 等)，支持多种 CPU 架构 (Alpha, ARM, SPARC, X86 等)。

关于 Gem5 可以参考以下链接：

[http://www.m5sim.org/Main\\_Page](http://www.m5sim.org/Main_Page)

<http://pages.cs.wisc.edu/~david/courses/cs752/Fall2015/gem5-tutorial/index.html>

## 二、配置实验环境：

你需要在 Linux 或 Mac OS X 上运行 Gem5，关于环境配置可以参考以下链接：

<http://www.m5sim.org/Dependencies>

[http://www.m5sim.org/Running\\_gem5](http://www.m5sim.org/Running_gem5)

## 三、实验要求：

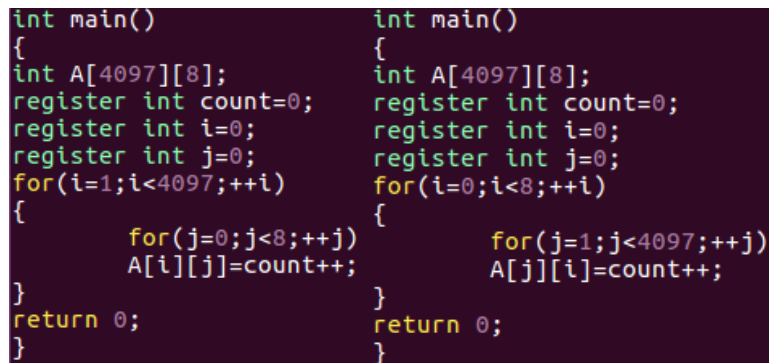
本实验不需要修改 Gem5，不需要配置 Python 脚本。你所需要做的有以下几点：

1. 根据题意编写一些简单的程序，使用 **GCC 静态编译**。
2. 使用 X86 架构，使用内置的 configs/example/se.py 脚本，并为其配置运行参数。  
你可以通过 ./build/X86/gem5.opt ./configs/example/se.py -h 查找可以使用的参数。
3. 在 m5out/stats.txt 中找到有用的结果，并分析。

## 四、实验内容：

### 实验 1 Cache (20%)

- 1) 假设一个单核 CPU，L1 Data cache 配置如下：大小为 1kB，cacheline 为 32B，**计算并实验测试**以下程序在直接映射的情况下的 Data cache 写入 miss 为多少次？并对比仿真的**总周期数**和 **ipc**。



```
int main()
{
    int A[4097][8];
    register int count=0;
    register int i=0;
    register int j=0;
    for(i=1;i<4097;++i)
    {
        for(j=0;j<8;++j)
            A[i][j]=count++;
    }
    return 0;
}

int main()
{
    int A[4097][8];
    register int count=0;
    register int i=0;
    register int j=0;
    for(i=0;i<8;++i)
    {
        for(j=1;j<4097;++j)
            A[j][i]=count++;
    }
    return 0;
}
```

- 2) 实验结果与计算结果不严格相等，可能的原因有哪些（至少两点）？

### 实验 2 分支预测 (40%)

考虑一个简单的分支执行 1000000 次，程序如下所示：

```
#include<stdio.h>
int main()
{
    char x=0,y=64;
    for(int i=0;i<1000000;++i)
    {
        if(x!=y)
            x=y;
    }
    return 0;
}
```

图 2-1

对于其中的 if 判断，可以用以下两种汇编实现：

方案一、使用条件转移指令

```
L4:
    test x,y
    jump to L3 if not equal (jnz)
    move y to x
L3:
    add 1 to i
L2:
    compare 1000000,i
    jump to L4 if less (jle)
```

图 2-2

方案二、使用条件 MOV 指令 (cmovz)

关于 cmovz 指令的特性可以参考：<http://yarchive.net/comp/linux/cmov.html>

```
L4:
    test x,y
    conditionally move y to x (cmovz)
    add 1 to i
L2:
    compare 1000000,i
    jump to L4 if less (jle)
```

图 2-3

你所要做的有以下四个工作：

- 1) 将图 2-1 的程序编译为方案一的汇编实现，保存源码为 branch.s
- 2) 在 branch.s 基础上修改为方案二的汇编实现，保存源码为 cmovz.s
- 3) 编译两个.s 文件，在 Gem5 的 X86 架构下使用**乱序流水线模型** (OoO) 进行仿真。罗列并对比两者的**总指令数**、**运行时间**、**ipc**、**分支预测**以及**流水线**信息。
- 4) 结合**汇编文件**和**cmovz 指令特性**来阐述 3) 中数值差异性的原因。

### 实验 3 单指令多操作 (40%)

现今的 X86 处理器支持单指令多操作(SIMD)。将一条指令的行为作用于多个操作数上，实现矢量化运算。以典型的 Intel SSE 指令集为例，支持 SSE 指令集的 CPU 含有 8 个大小为 16B 的矢量寄存器(xmm0-xmm7)，它们可以一次存储多个操作数，并通过特殊的指令对这些操作数进行矢量化运算。

关于 Intel 的 SIMD 指令可以参考：

<http://www.engr.uconn.edu/~zshi/course/cse5302/student/mmx.pdf>

考虑一个简单的数组操作

```
int A[1048576]
for(int i=0;i<1048576;++i)
{
    A[i]=A[i]+1;
}
```

附件 SIMD.s 与 Ori.s 分别对应着 SSE 指令集实现和一般实现。

你需要结合附件做以下四个工作：

- 1) 结合 SIMD.s 汇编代码，**阐述矢量化程序的运算流程**。
- 2) 在 X86 架构下使用**乱序流水线模型**进行仿真。对比两种实现方法的**运行时间，ipc，总指令数，访存请求个数**。
- 3) 在 SIMD.s 中的访存指令个数与仿真中的访存请求个数是否近似相等？如果相差很多，造成的原因是什么？
- 4) 结合实验数据，从体系结构的角度阐述 SIMD 性能更优的原因。

#### 五、上传要求及评分标准：

1. 包含实验报告（PDF 文件）和源码（branch.s、cmovz.s），打包为 rar 或 zip 文件，命名方式为：姓名\_学号，统一上传到助教邮箱：[cyh-shanghai@sjtu.edu.cn](mailto:cyh-shanghai@sjtu.edu.cn)
2. 报告包含必要的实验结果数据（采用截图或图表形式），报告的可读性会影响评分。