***2018***



**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 题 目： | 计算机系统结构实验报告 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1503 |
| 学 号： | U201514559 |
| 姓 名： | 周铭昊 |
| 电 话： | 15802740273 |
| 邮 件： | 630212894@qq.com |
| 完成日期： | 2018-05-03 周四晚上 |

目 录

[1 实验概述 2](#_Toc513180342)

[1.1 实验目的 2](#_Toc513180343)

[1.2 实验内容 2](#_Toc513180344)

[1.3 实验环境 2](#_Toc513180345)

[1.4 实验要求 3](#_Toc513180346)

[2 实验方案设计 5](#_Toc513180347)

[2.1 编写Cache模拟器 5](#_Toc513180348)

[2.2 优化矩阵转置操作 6](#_Toc513180349)

[3 实验过程与调试 8](#_Toc513180350)

[3.1 测试用例 8](#_Toc513180351)

[3.2 测试结果与性能分析 8](#_Toc513180352)

[3.3 主要故障与调试 8](#_Toc513180353)

[4 实验总结与心得 11](#_Toc513180354)

[4.1 实验总结 11](#_Toc513180355)

[4.2 实验心得 11](#_Toc513180356)

[参考文献 13](#_Toc513180357)

# 实验概述

## 实验目的

理解cache工作原理；

加深Cache缓存组成结构对C程序性能的影响的理解。

## 实验内容

第一部分：编写一个200-300行的C程序来模拟Cache缓存的行为；

第二部分：在参考Cache实现的基础上，优化一个矩阵转置函数，以最小化缓存不命中（cache miss）的数量。

## 实验环境

Linux 64-bit + valgrind软件包（第二个实验需要），C语言

## 实验要求

实验内容一：

* 任务：在csim.c提供的程序框架中，编写实现一个Cache模拟器：
* 输入：内存访问轨迹
* 操作：模拟缓存相对内存访问轨迹的命中/缺失行为
* 输出：命中、缺失和（缓存行）淘汰/驱逐的总数
* 具体要求：完成的csim.c文件应能接受与参考缓存模拟器csim-ref相同的命令行参数并产生一致的输出结果。完成的csim.c文件应能接受与参考缓存模拟器csim-ref相同的命令行参数并产生一致的输出结果。
* 编程要求：
  + 模拟器必须在输入参数s、E、b设置为任意值时均能正确工作——即需要使用malloc函数（而不是代码中固定大小的值）来为模拟器中数据结构分配存储空间。
  + 由于实验仅关心数据Cache的性能，因此模拟器应忽略所有指令cache访问（即轨迹中“I”起始的行）
  + 假设内存访问的地址总是正确对齐的，即一次内存访问从不跨越块的边界——因此可忽略访问轨迹中给出的访问请求大小
  + main函数最后必须调用printSummary函数输出结果，并如下传之以命中hit、缺失miss和淘汰/驱逐eviction的总数作为参数：

printSummary(hit\_count, miss\_count, eviction\_count);

实验内容二：

* 任务：在trans.c中编写实现一个矩阵转置函数transpose\_submit，要求其在参考Cache模拟器csim-ref上运行时对不同大小的矩阵均能最小化缓存缺失的数量

char transpose\_submit\_desc[] = "Transpose submission";

void transpose\_submit(int M, int N, int A[N][M], int B[M][N]);

* 实现要求：
  + 限制对栈的引用——在转置函数中最多定义和使用12个int类型的局部变量，同时不能使用任何long类型的变量或其他位模式数据以在一个变量中存储多个值。
    - 原因：实验测试代码不能/不应计数栈的引用访问，而应将注意力集中在对源和目的矩阵的访问模式上
  + 不允许使用递归。如果定义和调用辅助函数，在任意时刻，从转置函数的栈帧到辅助函数的栈帧之间最多可以同时存在12个局部变量。
    - 例如，如果转置函数定义了8个局部变量，其中调用了一个使用4个局部变量的函数，而其进一步调用了一个使用2个局部变量的函数，则栈上总共将有14个变量，则违反了本规则。
  + 转置函数不允许改变矩阵A，但可以任意操作矩阵B。
  + 不允许在代码中定义任何矩阵或使用malloc及其变种。

# 实验方案设计

## 编写Cache模拟器

修改cism.c文件，首先理解每个变量的含义和每个函数的作用,采用LRU算法，用结构体数组来保存cache数据。

### 变量及结构体定义

表2.1 变量及结构体定义

| 变量名 | 说明 |
| --- | --- |
| Cache\_line | Char valid：有效位，标识这块cache是否有效 |
| mem\_addr\_t tag ：标识位，与内存相对应 |
| unsigned long long int lru：LRU计数位 |
| cache\_set\_t | 一组cache，其中包含多行cache\_line |
| cache\_t | 所有的cache，其中包含多组cache\_set\_t |
| s | 组索引位数 |
| S | 组数，等于2的s次幂 |
| b | 块内偏移位 |
| B | 块大小，等于2的b次幂 |
| E | 关联度，每组cache包含的行数 |
| miss\_count | 未命中计数 |
| hit\_count | 命中计数 |
| eviction\_count | LRU替换计数 |
| eviction\_lru | 被替换的cache的lru计数值 |
| eviction\_line | 被替换的cache行号 |
| isHit | 命中标志位 |
| isFull | Cache已满标志位（一组中所有cache都有效） |
| lru\_counter | Lru计数器，每次访存加一 |

### 算法思想

Cache模拟器的整个运行过程是：先读取命令行的内容，然后设定s，E和b的值，调用initial函数进行初始化。

Initial函数用于cache的初始化。先判断S是否有效，若s<0则cache组数为0，输入无效直接退出。然后初始化一个二维数组cache[S][E]，其中包含S组，每组包含E行cache。每个cache的valid初始化为’n’，表示无效，tag和lru都初始化为0，同时计算组索引掩码，在计算组号时要用到。初始化完成后进入读取函数，读取轨迹。

replayTrace函数用于循环读取轨迹文件。先根据命令行输入的文件名trace\_fn，将其按行读取到buf，每次读取过程判断具体操作是S、L还是M，如果是前两者只需要访存一次，后者需要访存两次，即调用两次accessData函数。循环读取直到文件尾，完成后关闭文件。

AccessData函数用于判断每次访存的cache命中情况。首先计算出被访问的cache组号和内存标识号，然后通过组号选定这一组cache\_set。在判断访存情况之前，先把lru\_counter计数器+1，越大代表越新。随后判断是否命中，如果成功命中，则把这块cache的lru赋值为lru\_counter的值，跳出。如果未命中，则还需要判断这组cache是否还有空位，如果不满，则直接找到某一行没有使用的cache进行载入，并把这行cache的lru赋值为lru\_counter；如果满了，则需要找到lru最小的cache，即最久没有访问的cache进行替换，替换完成后，把这组cache的lru赋值为lru\_counter。所有的访存判断都结束以后，replayTrace函数也结束读文件。

Freecache函数用于释放在初始化时申请的cache空间。

printSummary函数用于打印三个计数器（命中、未命中、替换）的值。

## 优化矩阵转置操作

### 局部变量定义

两个索引变量i，j；用于控制for循环。

八个中间变量tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8；用于缓存数组值。

### 32×32矩阵转置的优化策略

由于cache的规格是： s=5, 组索引5位, 共有2^5=32组

E=1, 每组包含1行

b=5, 块内地址5位, 块大小为2^5=32 Bytes

一个cache块 可以包含8个int型变量，由于数组是连续存储的，那么访问A[0][0]时，cache中保存了A[0][0]及其相邻变量的值，其中可能包含A[0][1],A[0][2]---A[0][7]。因此按照顺序访问，可以提高cache命中率。

读取时每次连续读取8个的相邻的int，通过八个中间变量赋值给数组B。

### 64×64矩阵转置的优化策略

### 61×67矩阵转置的优化策略

# 实验过程与调试

## 测试用例

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

### 测试用例1

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

### 测试用例1

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 测试结果与性能分析

## 主要故障与调试

### XXX故障

理想流水线： 接口处数据传输问题。

**故障现象：**执行halt指令时控制信号无法通过ID/EX接口。

**原因分析：**如图 4.1，寄存器设置为上升沿刷新，但当D端有输入且clk变化一个周期后寄存器中的数据仍然没有改变，检查时注意到此时ALUControl端输入的值为不确定，这是由于在控制器电路中未给halt指令相应的ALUControl值，为设计控制器时的失误。如图 4.1，寄存器设置为上升沿刷新，但当D端有输入且clk变化一个周期后寄存器中的数据仍然没有改变，检查时注意到此时**ALUControl**端输入的值为不确定，这是由于在控制器电路中未给halt指令相应的ALUControl值，为设计控制器时的失误。

**解决方案：**在控制器中给halt指令（OP为12）一个ALUControl信号0000一边让控制信号在接口处顺利传递。



图 4.1无法向寄存器中写入数据示意图

### XXX故障

Verilog实现动态分支预测：状态机转移错误。

**故障现象：**如图 4.2所示，当判断信号judge为10b，即判断错误，predict\_f信号为10b，即判断指令不发生跳转，可判断指令最终发生了跳转，状态state[1]应向11b方向转移，即由01b转移至10b，可此时状态并未发生相应转移。



图 4.2程序故障图

**原因分析：**添加测试信号signal用来判断程序在此时进入了判断状态的case语句中的哪一个状态分支，结果显示，程序竟然进入了状态为00b的分支，再通过仔细分析可以得到，此模块的敏感变量表内为judge、predict\_t两个信号，在编写时采用的是电平触发，可能会产生毛刺，并最终导致状态机状态转移错误。

**解决方案：**修改动态分支预测器的敏感变量表，将此模块修改为时序控制，使用时钟的下降沿进行控制，在上一个时钟上升沿，分支指令进入EX段，并且判断出分支预测结果的正误，随后的时钟下降沿，使用判断结果作为依据进行状态机的状态转移，便可实现状态机状态的正确转移。

### XXX故障

Verilog实现动态分支预测：预测错误之后重新计算地址错误。

**故障现象：**如图 4.3，judge判断信号为10b，表示预测错误，nop信号，即分支跳转信号为0，表示分支指令没有发生跳转，此时情况为分支指令并未发生跳**转而**动态分支预测器预测指令会跳转，此时需要删去误取的指令，并将不跳转的地址重新计算，应为0x0000316c，然而此时计算出的不跳转地址为0x00003158，错误。

**原因分析：**分析地址计算模块发现代码编写错误。当预测器预测指令跳转而实际指令并未跳转时，应计算出此条分支指令的下一条指令的地址，即在此**分支指令**的地址基础上加上4，而之前的代码在计算时，直接在如今处于的IF段的指令的地址上加上4，必然导致程序出错。



图 4.3程序故障图

**解决方案：**修改地址计算模块**，**当预测器预测指令跳转而实际指令并未跳转时，因为此时那条被判断的分支指令处于EX段，于是应当使用EX段指令的地址作为基础加上4，便可以得到修改错误预测之后的下一条指令的正确地址**。**

# 实验总结与心得

## 实验总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1. 完成方案总结（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）。
2. 功能总结（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）。
3. 其他需要总结的内容，（自行修订扩充）。

## 实验心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。两个星期从早到晚的不懈努力以及国庆节假期的辛苦加班才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课程设计刚刚开始的时候，第一个任务是使用Logism设计单周期CPU，该任务和上学期实验课的任务类似，所以整个过程还算比较迅速。然而，在将该电路使用Verilog语言进行描述时，麻烦接踵而至。因为Logism对于电路是完全可视化的，连接过程清晰明了，但是使用Verilog语言进行数据通路时，因为各个部件的接口在定义时起名不是很规范，又因为只能使用各个wire型变量对于关键部件进行连接，而这些变量的名字在定义时同样没有做到一目了然，从而导致了我在完成了数据通路的连接后，使用了大量的时间对于整个电路进行排错和检查，最终还是将所有的接口和连线规范化的起了名字，才终于解决了问题。这也就说明了工程化、规范化对于电路设计或是程序设计的重要性。

紧接着，理想流水线CPU的设计并没有什么难度，但是使用插入气泡、数据重定向技术对于流水线CPU进行冒险处理时，因为这些方法书本上并没有，老师提供的PPT上也只有简单的一些描述，这就要求我不断地在网上搜索相关的知识内容，和小组内的成员进行相关探讨。随后的动态分支预测的设计难度更高，老师并没有提供任何相关的内容，这又使得我不断地去网上搜寻资料文献，阅读全英文的学术论文。从这一切也可以看出团队在一个工作中的重要性，以及自我学习能力的必要性。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计按照分组的机制，但是因为所有人所需要完成的工作相同，分组也只是更便于组内的同学在设计过程中进行互相交流，分组的作用并不是非常明显。分组内的同学们进度完全不同，且有些同学因为对于课程内容的掌握不是很好，逐渐边缘化，分组并不能调动所有同学加入到课程设计任务中。本课程设计的难度很大，但是时间又有一点不够充分，可以适当简化一下任务（并不是降低最终目标），使得在有限的时间内，在完成全部课程设计目标的基础上，适当减少任务中其中重复的繁杂的工作，以提高整个课程设计的效率。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011年.
4. 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.
5. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |