

（深圳）

实验报告

开课学期： 春季

课程名称： 计算机体系结构(实验)

实验名称： 分支预测器设计

实验性质： 综合设计型

实验时间： 地点： T

学生班级：

学生学号：

学生姓名： 陈妮娜

评阅教师：

报告成绩：

实验与创新实践教育中心印制

年月

# . 实验内容

*简述实验内容*

本实验将基于Pin插桩分析的工作机理，使用C/C++实现分支预测器的软件模型，从而在进一步熟悉插桩工具使用的同时，加深对分支预测原理的理解。

参考Pin工具包的插桩API文档，编写锦标赛分支预测器（至少需要使用指导书中的种动态分支预测方法）。所编写的分支预测器只需预测分支是否跳转，不需预测分支跳转的目标地址。

# . 设计与实现

## 2.1 题目分析

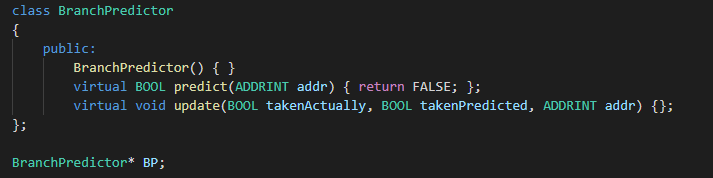
*描述个人对实验题目、实验内容和实验要求的分析和理解，并得出相应的结论，或根据分析得出设计方案。*

按照分支预测的工作方式，可将分支预测方法分为静态和动态两种。静态分支预测根据指令的操作码和寻址方式进行预测，而动态分支预测则是根据指令执行时的分支行为历史进行预测。静态分支预测的优点是实现简单、硬件开销较小、预测速度较快，但预测准确度相对较低，不能满足通用处理器对性能日益增加的需求，所以在本次实验中要完成的分支预测属于动态分支预测。

动态分支预测根据分支指令的过去表现来预测其将来的行为。如果分支行为发生变化，那么分支预测的结果也相应地发生改变。因此，动态分支预测与静态分支预测相比具有更好的预测准确率和适应性。

* 基于BHT的分支预测：使用分支历史表来记录分支指令的历史行为。随着预测的过程不断更新该表。
* 基于全局历史的分支预测：在BHT的基础上增加了GHR(全局历史寄存器)来将所有分支指令关联起来，在预测的过程中更新GHR。
* 基于局部历史的分支预测：使用LHT(局部历史表)来代替全局预测中的GHR。这里的LHT表只记录条记录。
* 锦标赛分支预测：是一种博采众长的分支预测方法，其基本原理是将两个或以上的分支预测方法进行结合，充分发挥各预测方法的优势，以进一步提高分支预测的准确度。

根据实验所提供的代码模板，是针对每种方法的特点，仿照静态分支预测的代码进行预测和更新的操作。

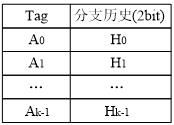


## 2.2 实验过程

*图文并茂地展示实验分析、优化和求解的过程。对于关键实现，可用图表叙述（必要时可贴关键代码，但禁止大段粘贴代码）。*

* 基于BHT的分支预测

使用分支历史表来记录分支指令的历史行为。BHT的基础结构如下图：

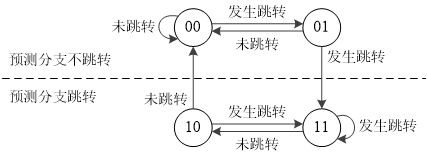


其中Tag字段是指令地址的一部分，其长度取值对分支预测的精确度有很大的影响，是优化中需要重点关注的部分。分支历史由bit饱和计数器组成。预测时，取分支指令的地址去查BHT的Tag字段，然后根据BHT当前行的分支历史，对分支的跳转方向进行预测。

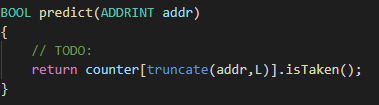
Tag在代码中的定义：



基于bit饱和计数器进行分支预测的原理如下图：



完整的分支预测过程包含预测和更新两个步骤，在代码中的体现如下：



预测时，用分支指令的地址查BHT，获得相应的饱和计数器值。若饱和计数器的最高位为，预测分支跳转，否则预测不跳转。

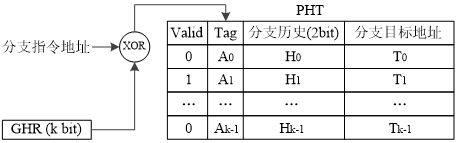


当分支指令的实际跳转方向被确定时，不管预测是否正确，都根据更新规则对BHT中的饱和计数器进行更新。

* 基于全局历史的分支预测

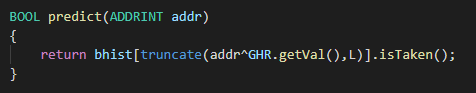
基于全局历史的分支预测方法在BHT的基础上增加了GHR(Global History Register,全局历史寄存器)来将所有分支指令关联起来。

基于全局历史的分支预测方法使用一个k比特的GHR来记录所有最近k条分支指令的历史跳转方向，本次实验中选用的k值为，对于GHR的更新规则是：若指令跳转，GHR的末位置为，否则置。

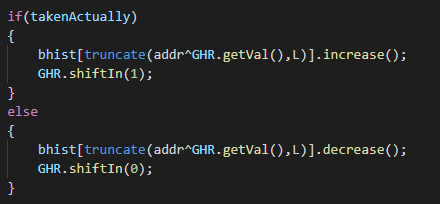


预测时，首先将分支指令的地址和GHR进行异或，再用异或操作的结果来查上表PHT，然后根据PHT当前行的分支历史和分支目标地址，对该分支指令的分支跳转方向和分支目标地址进行预测。

所以该预测器与BHT表的区别在于Tag需要异或获得，并且每次预测跳转过后都要及时更新GHR。代码如下：



输入的指令地址addr与GHR进行异或后得到Tag，然后读取PHT表返回预测结果。



发生真实的跳转之后，GHR逻辑右移后末位置为，否则置。

* 基于局部历史的分支预测

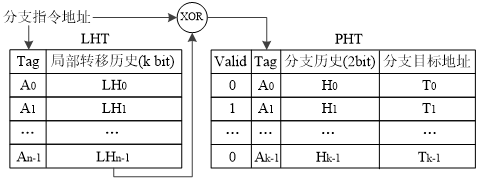
基于全局历史的分支预测方法将所有分支指令都关联到一起。然而事实上，并非所有的分支指令都具有关联性。为此，基于局部历史的分支预测方法使用LHT(Local History Table, 局部历史表)来代替全局历史预测中的GHR。

LHT一般具有64条记录，每条记录均包含Tag和局部转移历史2个字段。其中，Tag字段是分支指令地址的一部分，局部转移历史字段则是k比特的移位寄存器，其作用等同于GHR。

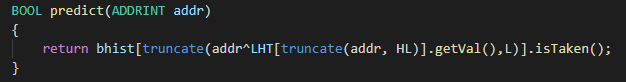
根据上述特点，代码中LHT的定义如下：



H取20，表示是20比特的移位寄存器，可以记录最近20条指令是否发生跳转。

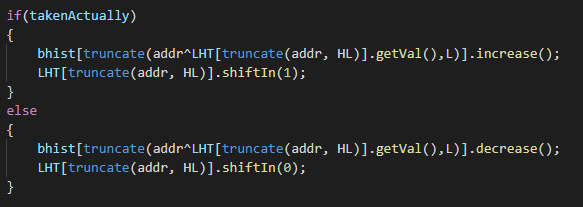


PHT中分支历史2bit的更新和预测方式仍然保持BHT中的规则。因此可以写出下述代码：



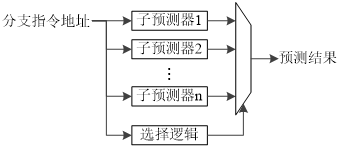
预测是通过读PHT表，输入的Tag值为地址与LHT表中读出的Tag异或得到的结果。

更新的规则仍然与上面两种预测器相似，不过增添了LHT中Tag索引的部分，要根据索引找出需要被更新的LHT。



* 锦标赛分支预测——基于全局选择历史的选择方法

锦标赛分支预测是一种博采众长的分支预测方法，其基本原理是将两个或以上的分支预测方法进行结合，充分发挥各预测方法的优势，以进一步提高分支预测的准确度。

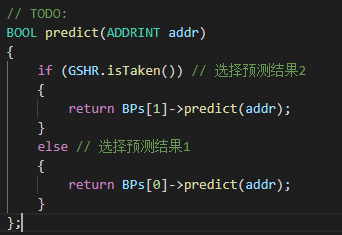


本次实验只选择含有2个子预测器的锦标赛分支预测器进行测试。

基于全局选择历史的选择方法使用一个2bit的GSHR(Global Selection History Register,全局选择历史寄存器)来记录子预测器预测结果的历史选择情况。

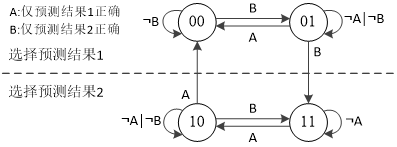
预测时，若GSHR的最高位为0，则输出子预测器1的预测结果；否则输出子预测器2的预测结果。

代码体现如下：

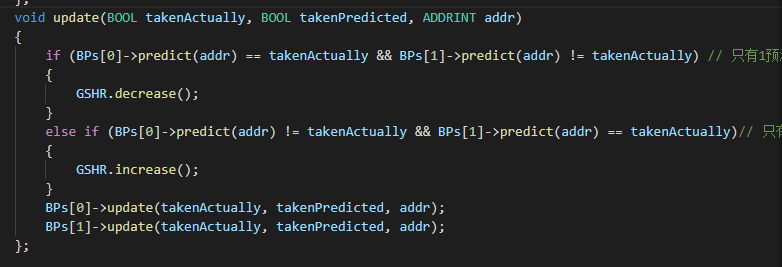


GSHR.isTake()返回1则表示GSHR的最高位为1。

当分支指令的实际跳转行为被确定时，需要同时对子预测器和GSHR进行更新。对于子预测器，根据分支指令的实际跳转行为和锦标赛预测结果等信息，使用子预测器自身的更新策略进行更新。对于GSHR，更新规则如下：若只有子预测器1预测正确，则对GSHR进行“减一”操作；若只有子预测器2预测正确，则对GSHR进行“加1”操作；其余情况下不更新GSHR。



更新过程代码如下：



先根据子预测器的预测情况更新GSHR，只有1预测正确时就decrease, 只有2预测正确时就increase。然后再更新子预测器的相关数据，如2bit的分支历史。如果先更新子预测器再更新GSHR则会降低精确度，因为GSHR首先要根据子预测器目前的预测结果进行更新，所以子预测器应该要先保持旧值。

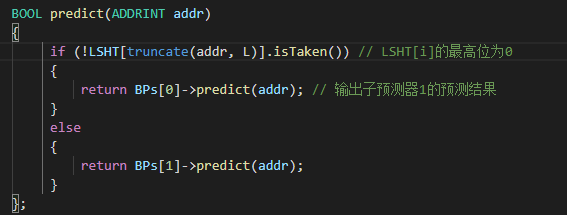
* 锦标赛分支预测——基于局部选择历史的选择方法

基于局部选择历史的选择方法使用LSHT(Local Selection History Table, 局部选择历史表)来记录子预测器预测结果的历史选择情况。

LSHT一般有4096条记录，每条记录均包含Tag和局部选择历史2个字段。其中，Tag字段是分支指令地址的一部分，局部选择历史字段则是2bit的饱和计数器，其作用等同于GSHR。

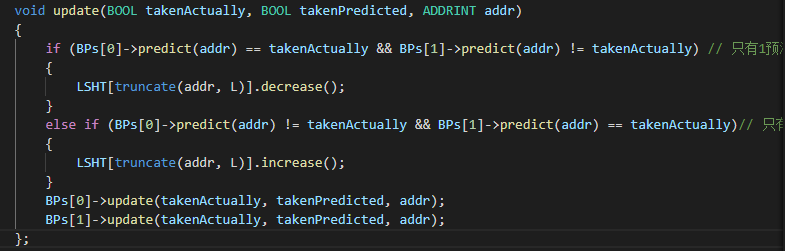
预测时，先取分支指令的地址查LSHT，得到相应的选择历史LSHT[i]。若LSHT[i]的最高位为0，则输出子预测器1的预测结果；否则输出子预测器2的预测结果。

代码体现如下：



当分支指令的实际跳转行为被确定时，需要同时对子预测器和LSHT[i]进行更新。对于子预测器，根据分支指令的实际跳转行为和锦标赛预测结果等信息，使用子预测器自身的更新策略进行更新。对于LSHT[i]，与GSHR的更新规则一样。

代码结构与基于全局的选择代码相似：



同样也是需要在更新LSHT之后再更新子预测器，因为更新LSHT时需要用到子预测器的旧值，若更新顺序交换，则测试的精确度会降低。

*\*如果有做附加题，需按照指导书要求，在报告中添加相应的分析、设计过程和运行结果截图和对比等内容。*

## 2.3 实验结果及分析

*描述测试方法，列出实验结果，并进行相应的分析。*

*如果进行了优化，需给出优化前后的结果对比截图及文字分析（如果优化没有效果，也需分析原因）。*

* 测试方法：

本实验使用SPEC 标准的测试集中的bzip(压缩算法)、sjeng(国际象棋算法)、wrf(天气预报算法)和sphinx(语音识别算法)对分支预测器模型进行测试。其中bzip和sjeng是定点测试程序，wrf和sphinx是浮点测试程序。

测试时，打开终端，用cd命令进入到~/pin-./workspace目录，然后在终端执行如下指令分别对个Benchmark进行测试。

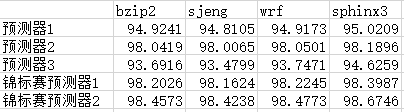


最后一个bzip2可以更换为其他三个测试代码名称。

* 测试结果分析：

可以优化的部分集中在分支预测器中Tag的大小以及用于记录历史跳转的字段长度，起初将这两个值都设置为，最高的准确率不过~左右，后来调整为，最高的准确度达到了，起初拿BHT分支预测器调试和两个参数，只有在bzip才高于，而其他三个测试代码都低于，因此觉得更佳。后来反复思考，觉得记录的指令信息更多，按理来说应该准确度更高才对，所以把其他所有的预测器都用测了一遍，发现其他预测器跑个测试代码的预测精确度都比的高。因此最后选定了这个参数。

* 测试结果展示：



其中预测器表示的是基于BHT的分支预测器，预测器是基于全局历史的分支预测器，预测器是基于局部历史的分支预测器，锦标赛预测器是采用基于全局选择历史的选择方法，其中个子预测器分别是基于全局历史的分支预测器和基于局部历史的分支预测器，锦标赛预测器是采用局部选择历史的选择方法，其中个子预测器也分别是基于全局历史的分支预测器和基于局部历史的分支预测器。

之所以选择这两个子预测器，是因为在一个程序执行的过程中，对于普通的跳转而言，可以大多数取决于全局分支预测器，而对于个别循环之类的代码，则大概率需要取决于局部分支预测器做出的预测。将二者结合就能集合两者的优点做出更好的锦标赛预测。

# . 总结和感想

*请填写实验过程中遇到的问题及解决方案、学习本课程的收获和反思，并对课程提出合理的建议。*

* 遇到的问题及解决方案：

起初看到这个实验题目感到比较陌生，对着代码模板也是一脸懵，在老师和同学的讲解之后，有了比较深入的认识，因此能模仿着代码模板给出来的静态分支预测器例子编写这些动态分支预测器和锦标赛分支预测器。写代码的过程中遇到的降低精确度的问题就是在锦标赛预测器的更新环节，提前更新了子预测器再更新GSHR或LSHT，对结果产生了一定不好的影响，后来改正后预测精确度高了。

* 收获与反思：

通过这次的体系结构实验，加深了对分支预测原理的理解，掌握了分支预测器的软件实现方法；进一步熟悉了程序插桩的原理，掌握了Pin插桩工具的编写和使用方法；了解了使用Pin插桩工具进行体系结构分析的方法。同时熟悉了C++代码与语法。在未来的理论和实验课程中，还将继续认真学习理论知识和夯实代码能力。感谢在本次实验中提供帮助的老师助教和同学们！