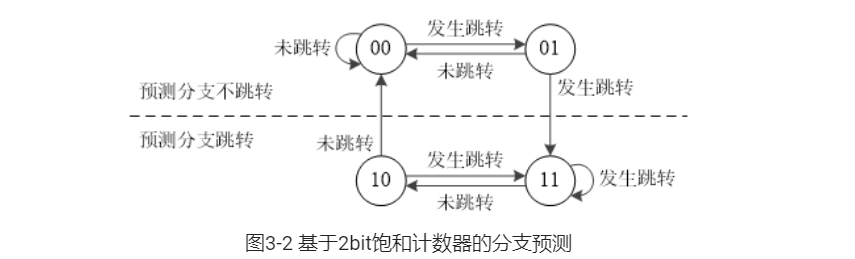
实验二：分支预测器设计

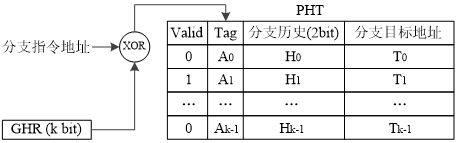
1. 题目分析
   1. 基于BHT的分支预测

BHT分支预测，主要结构是分支历史表，分支历史表是一个个可索引的条目，索引tag是指令的一部分，实际实验中是通过取地址的低L位，根据索引查表，得到2bit的分支历史，对分支跳转进行预测；

2bit饱和计数器的结构很简单，当val小于等于1的时候预测不跳转，大于1的时候预测跳转，饱和计数器的更新按照对应的更新规则进行更新，其实际意义是，如果这个地址在最近几次分支都是跳转，分支预测倾向于选择跳转，不跳转也是一样，根据最近分支跳转结果的情况进行调整。



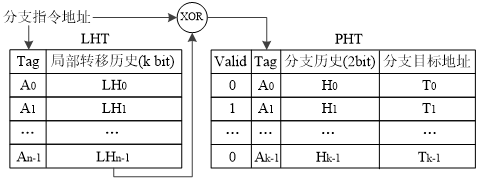
* 1. 基于全局历史的分支预测

基于BHT的分支预测方法忽视了分支指令之间的关联性。为此，基于全局历史的分支预测方法在BHT的基础上增加了GHR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，并使用PHT来记录各分支指令的分支历史。其中，PHT的结构类似于BHT，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： 

当分支指令的实际跳转行为被确定时，GHR通过移位的方式进行更新——若指令跳转，则GHR = (GHR << 1) | 1，否则GHR = (GHR << 1) | 0。

* 1. 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来代替全局历史预测中的GHR。LHT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。

预测时，首先用分支指令的地址查LHT，得到分支指令的局部转移历史LHT[i]；然后将分支指令的地址和LHT[i]进行异或，再用异或操作的结果来查PHT；最后根据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。

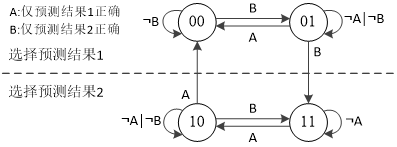
当分支指令的实际跳转行为被确定时，LHT[i]和GHR一样，也通过移位的方式进行更新——若指令跳转，则LHT[i] = (LHT[i] << 1) | 1，否则LHT[i] = (LHT[i] << 1) | 0。

* 1. 锦标赛

选取两个预测器，这里选择了基于全局和局部的分支预测，选择方法是基于全局选择历史的选择方法；

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。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，按如图3-7所示的规则更新：若只有子预测器1预测正确，则GSHR进行“减1”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。

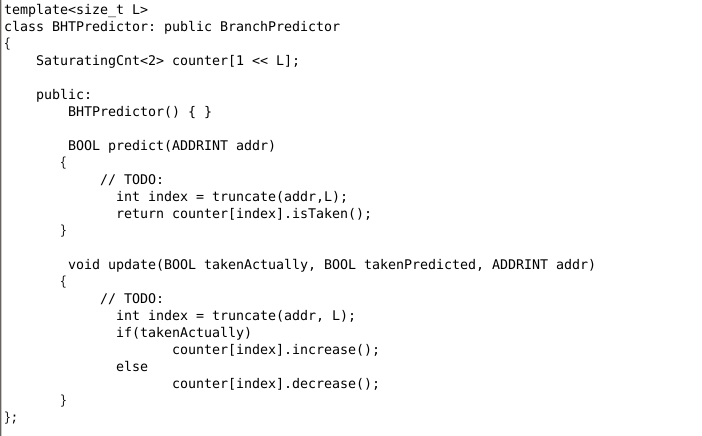


1. 关键实现
   1. 基于BHT的分支预测

主要数据结构都已经给出，这里需要完善预测和更新的函数

Truncate(a,b)函数的作用是将a的后b位返回，可以进行地址到tag的计算，预测时只要得到tag，然后用counter数组找到对应的表项，得到预测结果；

更新时应该按照饱和计数器的具体值进行相应的更新，但是相比之下，在预测正确时对其加一，否则减一的准确率稍微高一些：



* 1. 基于全局历史的分支预测

基于全局历史的分支预测相比于BHT多了一个GHR历史记忆单元，需要先通过GHR的值和地址进行异或才能得到饱和计数器的tag，这里L是tag的位数，需要对地址和GHR的异或取后L位来作为tag，更新时还要将GHR进行更新，也就是整体左移一位，最后一位添加上这次分支预测的结果。

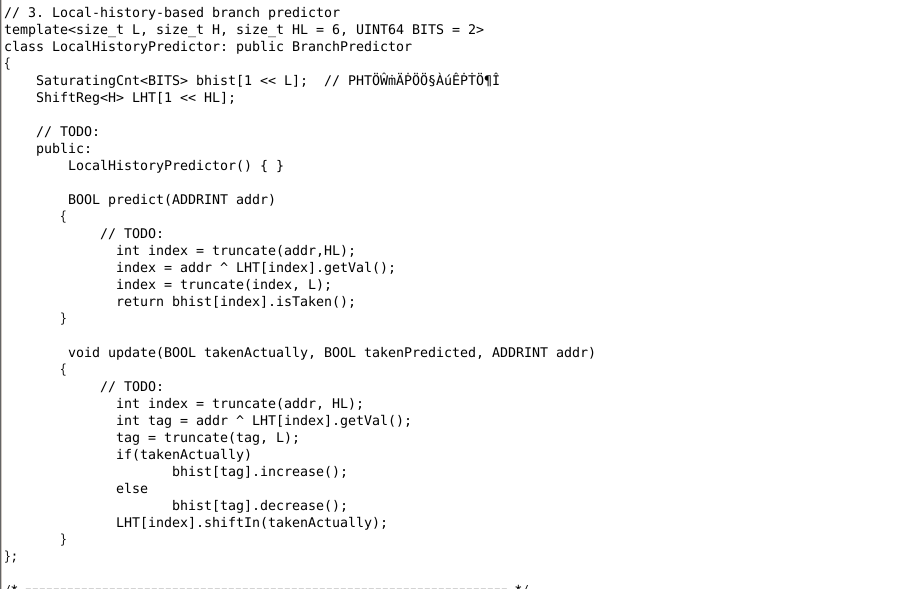


* 1. 基于局部历史的分支预测

基于局部历史的分支预测不同于全局历史的地方是，GHR单元变成了一个数组，包含HL个条目，每个条目记录一个局部历史信息；

获取tag时，需要先根据地址的后HL位作为LHT单元的tag，取出对应的历史记录信息，然后和地址进行异或操作，最后以此为新的地址取饱和计数器中找到对应的预测结果；

更新时和全局历史类似，需要注意的是每一步teg的选取需要和对应tag规定的位数相同，也就是用truncate进行截取，最后记忆单元只更新当前地址索引的记忆记录：

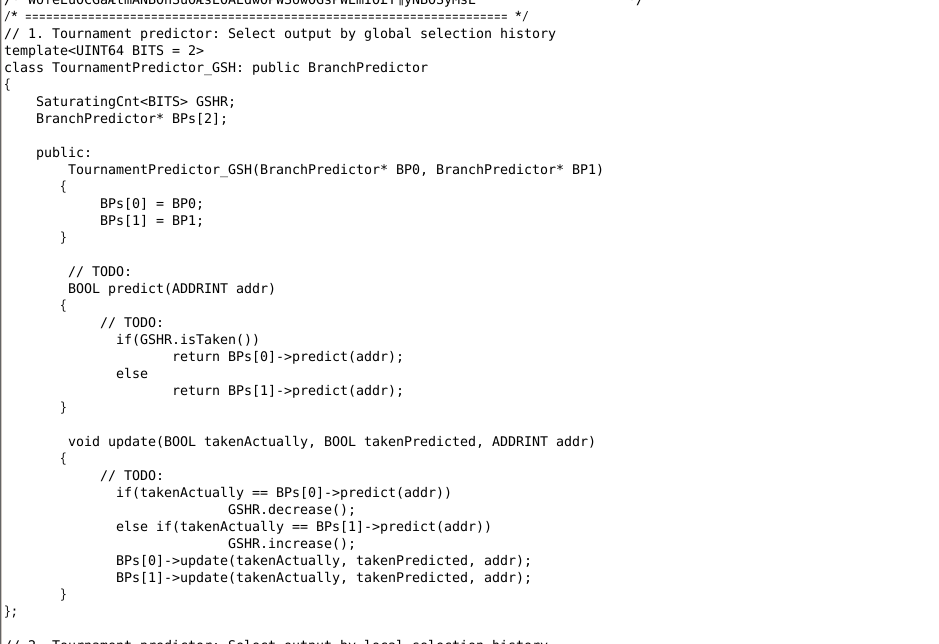


* 1. 锦标赛

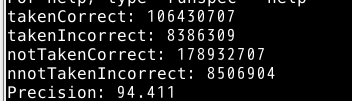
创建实例的时候会传入两个预测器实例；

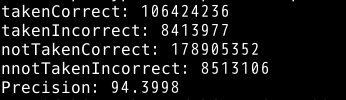
预测结果的确定是通过GSHR来确定的，GSHR饱和计数器大于一就输出预测器1的结果，否则输出预测器2的结果，而GSHR.isTaken()输出真时就是大于一，否则小于等于一，因此直接根据其值输出相应的预测器的预测结果；

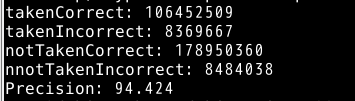
更新时，更新规则是只有预测器1预测成功时GSHR减一，只有预测器2预测成功时GSHR加一，其他情况不进行更新，因此要注意更新的判断条件，最后需要对两个预测器也进行更新。

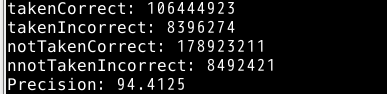


1. 测试结果及分析
   1. BHT的测试结果如下（依次为bzip2、sjeng、wrf、sphinx3）:



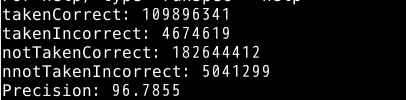


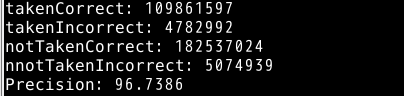


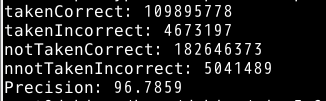


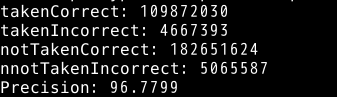
整体都达到了94.4左右的准确度，比不用策略（60左右）提高了很多，可见历史信息对分支的预测很有用，推测一般的循环体会预测准确率很高。

* 1. Glabol策略的结果如下（依次为bzip2、sjeng、wrf、sphinx3）：



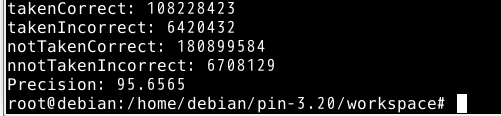


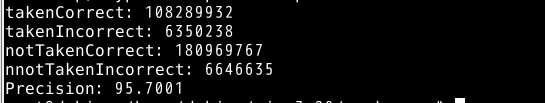


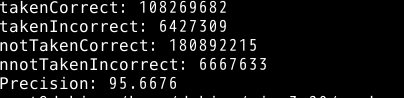


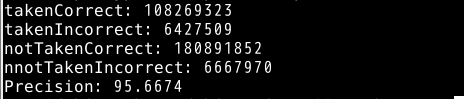
全局历史的方法是所有方法中效果最好的，当然也依赖于参数的选择，这里选取的全局历史记录是16位，相对来说效果较好，取8位时只有95.5左右。

* 1. Local策略的结果如下（依次为bzip2、sjeng、wrf、sphinx3）：









局部历史的方法的效果稍微逊色于全局历史，经过测试，发现局部历史记录位数为3时效果最好，也就是上图所示，达到了95.6，但还是小于全局的最优效果，相对来说，局部历史所囊括的范围要比全局历史小很多，因此其历史记录的位数应该小很多，这里为3时效果最好就是体现，取1时就和BHT几乎一样的结果。

* 1. 锦标赛测试结果及子预测器结果对比：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Bzip2 | Sjeng | Wrf | Sphinx3 |
| BHT | 94.411 | 94.3998 | 94.424 | 94.4125 |
| 预测器1（全局） | 96.7855 | 96.7386 | 96.7859 | 96.7799 |
| 预测器2（局部） | 95.6565 | 95.7001 | 95.6676 | 95.6674 |
| 锦标赛 | 95.6094 | 95.5849 | 95.6765 | 95.6565 |

锦标赛的结果并不比两个子预测器的结果好。