**深 圳 大 学 实 验 报 告**

**课程名称： 计算机系统(3)**

**实验项目名称： 处理器结构实验二：控制冒险与分支预测**

**学院： 计算机与软件学院**

**专业： 计算机与软件学院所有专业**

**指导教师： 李琰**

**报告人： 叶茂林 学号： 2021155015 班级： 腾班**

**实验时间： 2023年11月23日**

**实验报告提交时间： 2023年11月24日**

**教务部制**

|  |
| --- |
| 实验目的与要求： 实验目的 了解控制冒险分支预测的概念  了解多种分支预测的方法，动态分支预测更要深入了解  理解什么是BTB（Branch Target Buffer），并且学会用BTB来优化所给程序  利用BTB的特点，设计并了解在哪种状态下BTB无效  了解循环展开，并于BTB功能进行对比  对WinMIPS64的各个窗口和操作更加熟悉 |
| 方法、步骤： 二、实验步骤及相关说明 按照下面的实验步骤及说明，完成相关操作记录实验过程的截图：  首先，给出一段矩阵乘法的代码，通过开启BTB功能对其进行优化，并且观察流水线的细节，解释BTB在其中所起的作用；  其次，自行设计一段使得即使开启了BTB也无效的代码。  第三，使用循环展开的方法，观察流水因分支停顿的次数减少的现象，并对比采用BTB结构时流水因分支而停顿的次数。  （选做：在x86系统上编写C语言的矩阵乘法代码，用perf观察分支预测失败次数，分析其次数是否与你所学知识吻合。再编写前面第二部使用的令分支预测失败的代码，验证x86是否能正确预测，并尝试做解释） |
| 实验过程及内容： 三、实验内容 一、矩阵乘法及优化  在这一阶段，我们首先给出矩阵乘法的例子，接着将流水线设置为不带BTB功能（configure->enable branch target buffer）直接运行，观察结果进行记录；然后，再开启BTB功能再次运行，观察实验结果。将两次的实验结果进行对比，观察BTB是否起作用，如果有效果则进一步观察流水线执行细节并且解释BTB起作用原因。  矩阵乘法的代码如下：  .data  str: .asciiz "the data of matrix 3:\n"  mx1: .space 512  mx2: .space 512  mx3: .space 512  .text  initial: daddi r22,r0,mx1 #这个initial模块是给三个矩阵赋初值  daddi r23,r0,mx2  daddi r21,r0,mx3  input: daddi r9,r0,64  daddi r8,r0,0  loop1: dsll r11,r8,3  dadd r10,r11,r22  dadd r11,r11,r23  daddi r12,r0,2  daddi r13,r0,3  sd r12,0(r10)  sd r13,0(r11)  daddi r8,r8,1  slt r10,r8,r9  bne r10,r0,loop1  mul: daddi r16,r0,8  daddi r17,r0,0  loop2: daddi r18,r0,0 #这个循环是执行for(int i = 0, i < 8; i++)的内容  loop3: daddi r19,r0,0 #这个循环是执行for(int j = 0, j < 8; j++)的内容  daddi r20,r0,0 #r20存储在计算result[i][j]过程中每个乘法结果的叠加值  loop4: dsll r8,r17,6 #这个循环的执行计算每个result[i][j]  dsll r9,r19,3  dadd r8,r8,r9  dadd r8,r8,r22  ld r10,0(r8) #取mx1[i][k]的值  dsll r8,r19,6  dsll r9,r18,3  dadd r8,r8,r9  dadd r8,r8,r23  ld r11,0(r8) #取mx2[k][j]的值  dmul r13,r10,r11 #mx1[i][k]与mx2[k][j]相乘  dadd r20,r20,r13 #中间结果累加  daddi r19,r19,1  slt r8,r19,r16  bne r8,r0,loop4  dsll r8,r17,6  dsll r9,r18,3  dadd r8,r8,r9  dadd r8,r8,r21 #计算result[i][j]的位置  sd r20,0(r8) #将结果存入result[i][j]中  daddi r18,r18,1  slt r8,r18,r16  bne r8,r0,loop3  daddi r17,r17,1  slt r8,r17,r16  bne r8,r0,loop2  halt  **不设置BTB功能，运行该程序，观察Statistics窗口的结果截屏并记录下来。**  **接着，设置BTB功能（在菜单栏处选择Configure项，然后在下拉菜单中为Enable Branch Target Buffer选项划上钩）。并在此运行程序，观察Statistics窗口的结果并截屏记录下来。**  **在这里，我们仅仅观察比较Stalls中的最后两项------Branch Taken Stalls和Branch Misprediction Stalls。**  **接下来，对比其结果。我们就结合流水线执行细节分析造成这种情况发生的原因。**  **（30分，结果的获取10分，细节分析20分）**  将流水线设置为不带BTB功能，如图1所示。    图1  运行该程序，观察Statistics窗口的结果截屏并记录下来，如图2所示。    图2  设置BTB功能，在菜单栏处选择Configure项，然后在下拉菜单中为Enable Branch Target Buffer选项划上钩，如图3所示。    图3  运行该程序，观察Statistics窗口的结果截屏并记录下来，如图4所示。    图4  对比两次运行结果，可以看到，开启BTB之前的Branch Taken Stalls是574次，Branch Misprediction Stalls是0次，开启BTB后的Branch Taken Stalls和Branch Misprediction Stalls都是148次，如图5所示。      图5  在未开启BTB之前，分支跳转指令需要到执行阶段才能计算出要跳转的地址，因此会阻塞产生Branch Taken Stalls，如图6所示。  因为没有进行预测，所以Branch Misprediction Stalls为0。输入矩阵是一个 64 次的循环，因此会产生63次Branch Taken Stalls。而在矩阵乘法中计算 result[i][j]的部分每次循环结束会产生7次Branch Taken Stalls，重复 64 次，因此会产生 64\*7=448次 Branch Taken Stalls。而中间层 j++的循环结束会产生7次 Branch Taken Stalls，重复 8 次，因此会产生 8\*7=56 次 Branch Taken Stalls。而最外一层 i++的循环会产生 7 次 Branch Taken Stalls，重复1次，因此会产生7次Branch Taken Stalls。所以，不开启BTB总共会产生63+448+56+7=574次Branch Taken Stalls。因此 Branch Taken Stall 产生的阻塞周期数为574。    图6 未开启BTB  而开启BTB之后，它将分支指令对应的指令地址放到一个缓冲区保存起来，当下次在遇到相同的分支指令时，它预测将会跳转上一次跳转的指令地址，而一般循环中都会多次跳转到上一次跳转的指令地址，因此会大大减少Branch Taken Stalls的次数，如图7所示。    图7 开启BTB  而当要跳出循环的时候，BTB就会预测错误，产生Branch Misprediction Stalls，需要清除已取的指令，如图8所示。    图8 开启BTB  在循环开始时第一次执行分支指令会默认地按顺序取指令，此时要建立 BTB就会产生 1 次 Branch Taken Stalls。而在循环结束时，此时不需要跳转会产生 1 次 Branch Misprediction Stalls。因此，一个开启BTB的循环会产生 1 个 Branch Taken Stalls 和 1 个 Branch Misprediction Stalls。而每次 Branch Taken Stall 和 Branch Misprediction Stalls 阻塞的周期数为 2 个。因此就会产生 2 个 Branch Taken Stall 和 Branch Misprediction Stalls 的阻塞周期。代码中，在输入部分时有1个循环，在乘法部分有 64+8+1个循环，总共有74次循环，因此总共会产生 Branch Taken Stalls 和 Branch Misprediction Stalls 的阻塞周期数都为74\*2=148个。  二、设计使BTB无效的代码  **在这个部分，我们要设计一段代码，这段代码包含了一个循环。根据BTB的特性，我们设计的这个代码将使得BTB的开启起不到相应的优化作用，反而会是的性能大大降低。**  **提示：一定要利用BTB的特性，即它的跳转判定是根据之前跳转成功与否来决定的。**  **给出所用代码以及设计思路，给出运行结果的截屏证明代码实现了目标。**  **（30分，代码及思路20，获取结果并证明目标实现10分）**  我们可以设计一个循环，循环里面设计一个判断跳转，让每一次跳转的指令地址都和上一次跳转的不一样，这样BTB每一次预测分支都失败。  具体来说，先准备好一个元素为1010101010的数组，如图9所示，然后循环遍历数组元素，如果元素为0则跳转一个地址，否则跳转另一个地址，这样每次跳转的地址都和上一次跳转的地址不一样，BTB每次都预测错误。    图9  先关闭BTB，运行程序，结果如图10所示，总时钟周期数为106，其中Branch Taken Stalls有14次。    图10 关闭BTB  再开启BTB，运行程序，结果如图11所示，这次的总时钟周期数达到了114次，高于没开的106次，Branch Taken Stalls虽然只有12次，但是Branch Misprediction Stalls有10次，说明每次预测都失败了，一共产生了22次Stalls，高于没开的14次，说明我们设计的这个代码将使得BTB的开启起不到相应的优化作用，反而会使得性能大大降低。    图11 开启BTB  三、循环展开与BTB的效果比对  **首先，我们需要对循环展开这个概念有一定的了解。**  **什么是循环展开呢？所谓循环展开就是通过在每次迭代中执行更多的数据操作来减小循环开销的影响。其基本思想是设法把操作对象线性化，并且在一次迭代中访问线性数据中的一个小组而非单独的某个。这样得到的程序将执行更少的迭代次数，于是循环开销就被有效地降低了。**  **接下来，我们就按照这种思想对上述的矩阵乘法程序进行循环展开。要求将上述的代码通过循环展开将最里面的一个执行迭代8次的循环整个展开了，也就是说，我们将矩阵相乘的三个循环通过代码的增加，减少到了两个循环。**  **比较，通过对比循环展开（未启用BTB）、使用BTB（未进行循环展开）以及未使用BTB且未作循环展开的运行结果。比较他们的Branch Tanken Stalls和Branch Misprediction Stalls的数量，并尝试给出评判。**  我们将矩阵乘法中三层循环的内循环展开，即去掉原本代码中的loop4，将循环累加元素乘积改成直接累加，如图12所示，重复8次累加。    图12 循环展开  关闭BTB，重新运行程序，结果如图13，Branch Taken Stalls为126次，Branch Misprediction Stalls为0次。    图13 关闭BTB循环展开  而未使用BTB且未作循环展开的Branch Taken Stalls为574次，如图14所示，使用BTB（未进行循环展开）的次数为148次（Branch Take Stalls）+148次（Branch Misprediction Stalls），可见循环展开可以通过减少迭代次数来大大减少分支判断的开销，但是循环展开会使得代码变得冗长，可读性下降和维护困难。      图14 |

深圳大学学生实验报告用纸

|  |
| --- |
| 实验结论：  在本次实验中，我们首先通过开启BTB（分支目标缓冲器）功能来观察流水线的细节和分析BTB在其中所起的作用，BTB是一种硬件机制，它用于预测分支指令的目标地址，减少流水线的停顿。然后我们设计了一个循环代码，让每次跳转的指令地址都和上一次跳转的地址不同，从而使得BTB无效。最后我们使用了循环展开的方法，通过复制循环体内的代码，减少了分支指令的数量，从而减少了流水线因分支停顿而导致的效率损失。与使用BTB结构相比，循环展开可以更好地减少分支停顿次数，但也会增加代码的长度和复杂度。  通过本次实验，我们深入理解了BTB在优化矩阵乘法代码中的作用，以及在特定情况下BTB功能失效的影响。同时，我们也了解了循环展开对流水线分支停顿次数的影响。这对于进一步优化代码、提高程序性能以及理解计算机体系结构有着重要的指导意义。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。