**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**

**学 院： 计算机与软件学院**

**专 业： 软件工程/计算机科学与技术**

**指 导 教 师： 王 毅**

**报告人： 学号： 班级：**

**实 验 时 间： 2016年11月21日**

**实验报告提交时间： 2016年12月1日**

**教务处制**

## 一、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

## 二、实验内容

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）

## 三、实验环境

硬件：桌面PC

软件：Windows

## 四、实验步骤及说明

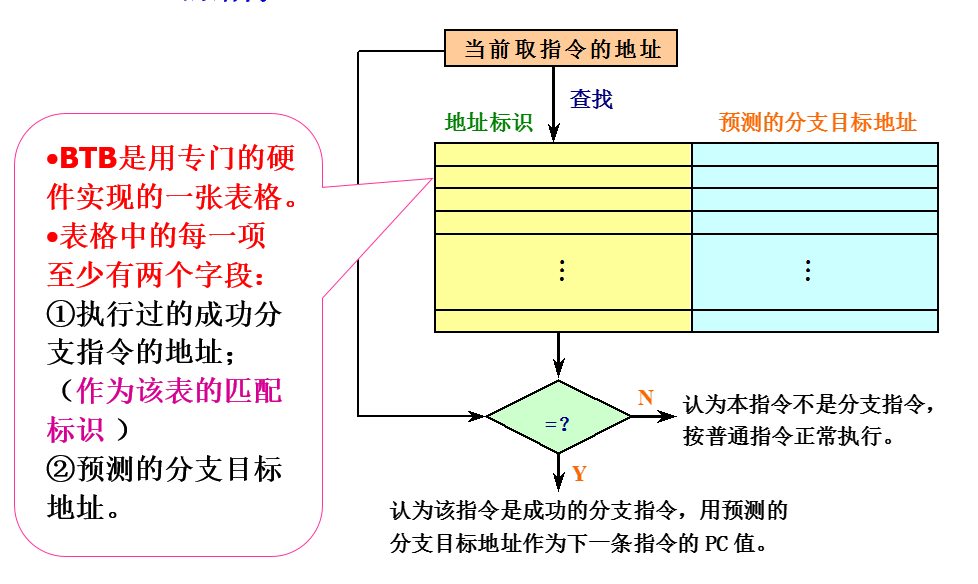
背景知识

在遇到跳转语句的时候，我们往往需要等到MEM阶段才能确定这条指令是否跳转（通过硬件的优化，可以极大的缩短分支的延迟，将分支执行提前到ID阶段，这样就能够将分支预测错误代价减小到只有一条指令），这种为了确保预取正确指令而导致的延迟叫控制冒险（分支冒险）。

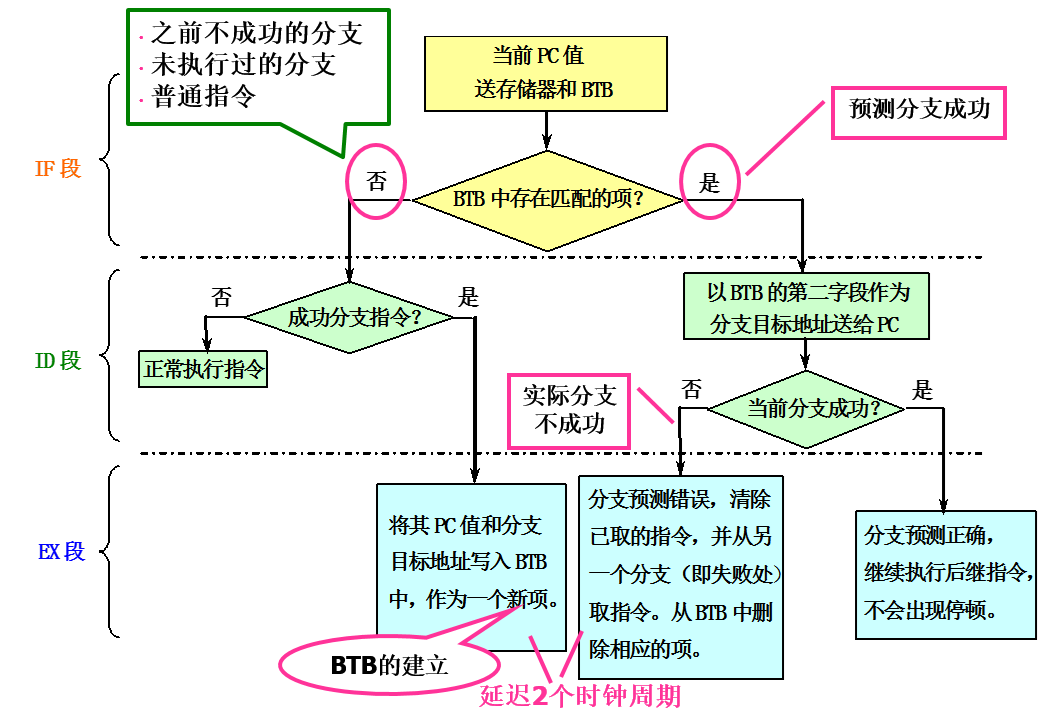
为了降低控制冒险所带来的性能损失，一般采用分支预测技术。分支预测技术包含编译时进行的静态分支预测，和执行时进行的动态分支预测。这里，我们着重介绍动态分支预测中的BTB（Branch Target Buffer）技术。

BTB即为分支目标缓冲器，它将分支指令（对应的指令地址）放到一个缓冲区中保存起来，当下次再遇到相同的指令（跳转判定）时，它将执行和上次一样的跳转（分支或不分支）预测。

一种可行的BTB结构示意图如下：



在采用了BTB之后，在流水线各个阶段所进行的相关操作如下：



注意，为了填写BTB，需要额外一个周期。

1. 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。

（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。

（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。

## 五、实验结果

## 六、实验总结与体会

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 王毅  年 月 日 |
| 备注： |