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：计算机与软件学院所有专业

指导教师：刘　刚

报告人：林宪亮　学号：2022150130　班级：国际班

实验时间：2024 年 11 月 21 日-2024 年 11 月 26 日

实验报告提交时间：2024 年 11 月 26 日

教务处制

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。

一、实验目的

——控制冒险与分支预测

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）。

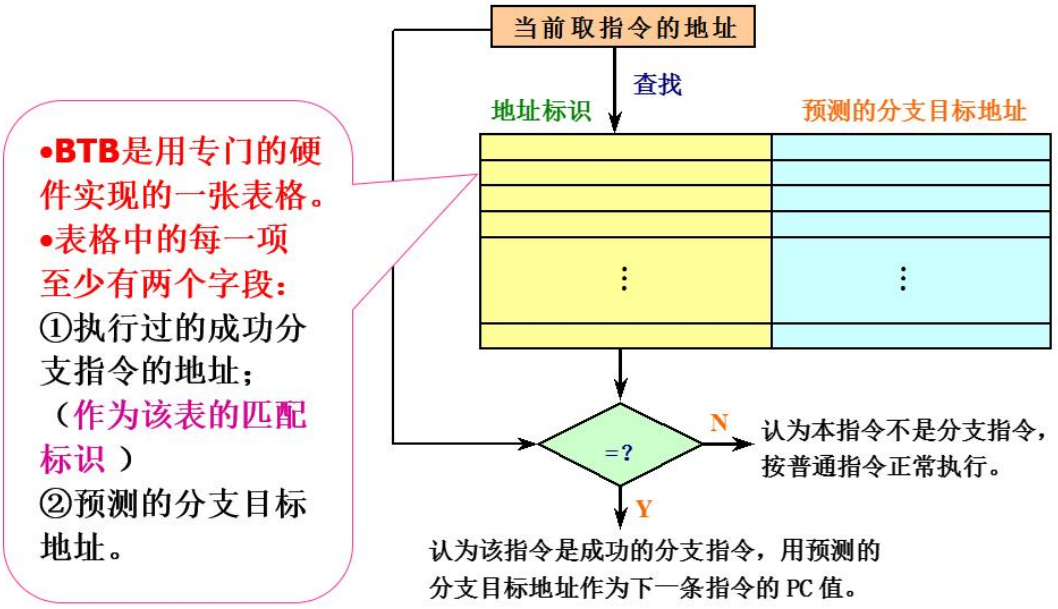
为了降低控制冒险所带来的性能损失，一般采用分支预测技术。分支预测技术包含编译时进行的静态分支预测，和执行时进行的动态分支预测。这里，我们着重介绍动态分支预测中的 BTB（Branch Target Buffer）技术。

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。

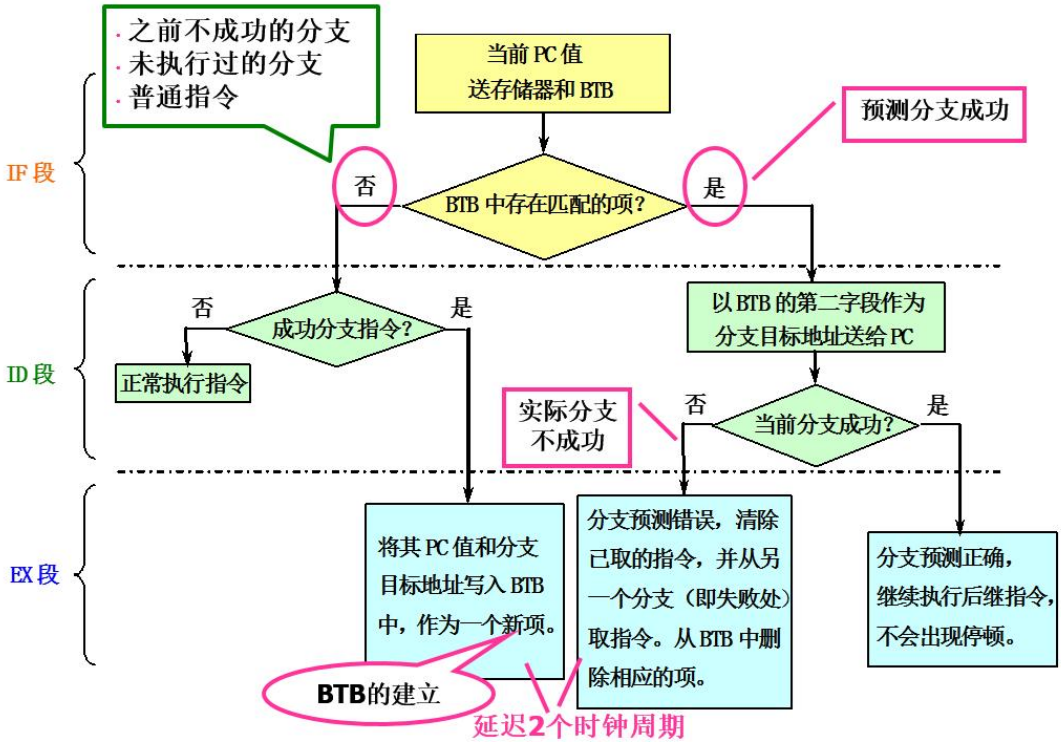
一种可行的 BTB 结构示意图如下：

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。



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。

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。

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
```

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。

```
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）

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。

1. 首先，将提供的代码复制，阅读理解

```
.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
```

图 1 代码片段 1

```
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
```

图 2 代码片段 2

将这段代码整理对应的 c 语言代码，以便于后续关于 BTS 的计算：
Init 部分：

```
for (i = 0; i < N*N; i++) {
    mx1[i] = 2;
    mx2[i] = 3;
}
```

图 3 代码片段 3

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。

矩阵乘法部分：

```
//mx3 = mx1 * mx2
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        long sum = 0;
        for (k = 0; k < N; k++) {
            sum += mx1[i*N + k] * mx2[k*N + j];
        }
        mx3[i*N + j] = sum;
    }
}
```

图 4 矩阵乘法部分

2. 运行程序，记录 Statistics 窗口的结果

```
Execution
13810 Cycles
9000 Instructions
1.534 Cycles Per Instruction (CPI)

Stalls
4232 RAW Stalls
0 WAW Stalls
0 WAR Stalls
512 Structural Stalls
574 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
188 Bytes
```

图 5 结果 1

分析：

初始化部分（Init）

- 循环次数：

  共有 N × N = 8 × 8 = 64 次迭代。

- 分支取情况：

  - 循环条件检查：每次迭代结束后都会检查条件 i < 64。当条件为真时，分支被取，循环继续；当条件为假时，分支不被取，循环结束。

  - 分支被取次数计算：循环从 i = 0 到 i = 63，共计 64 次迭代，其中有 63 次条件为真，分支被取；最后 1 次条件为假，分支不被取。

- 总分支被取次数：初始化部分的分支被取次数为 63 次。

乘法部分（Multiplication）

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。

外层循环（变量 i）
- 循环次数：N = 8 次。
- 分支被取次数：8 - 1 = 7 次。

中间循环（变量 j）
- 每次外层循环的迭代中，中间循环执行 N 次，单次循环分支被取次数为 8 - 1 = 7 次。
- 总迭代次数：N × N = 8 × 8 = 64 次。
- 总分支被取次数：7 次（每次中间循环） × 8 次（外层循环次数） = 56 次。

内层循环（变量 k）
- 每次中间循环的迭代中，内层循环执行 N 次，单次循环分支被取次数为 8 - 1 = 7 次。
- 总迭代次数：N × N × N = 8 × 8 × 8 = 512 次。
- 总分支被取次数：7 次（每次内层循环） × 64 次（中间循环总次数） = 448 次。
总分支被取次数
汇总各部分的分支被取次数：
- 初始化部分：63 次
- 外层循环：7 次
- 中间循环：56 次
- 内层循环：448 次

总计：63 + 7 + 56 + 448 = 574

理论分析结果与实际预期完全一致，表明分支被取次数的计算准确无误。

3. 设置 BTB 功能，记录 Statistics 窗口的结果

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。

如下图启用 BTB 后，重新载入上述程序，观察到 Statistics 窗口中的输出如下图所示，观察到发生 148 次 Branch Taken Stalls（下缩写 BTS ），同时发生 148 次 Branch Misprediction Stalls（下缩写 BMS ）.

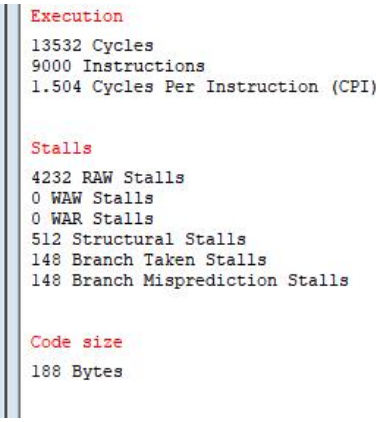


图 6 结果 2

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。

4. 对比结果，分析原因

148 次 BTS 的原因：

（1） init 部分：一层循环，发生 1 次 BTS 和 1 次 BMS

（2） 矩阵乘法部分：

a) 第一层循环，执行 8＊8＝64 轮，每轮发生 1 次 BTS 和 1 次 BMS，共计各 64 次.

b) 第二层循环，执行 8 轮，每轮发生 1 次 BTS 和 1 次 BMS，共计各 8 次.

c) 第三层循环，执行 1 轮，发生 1 次 BTS 和 1 次 BMS，共计各 1 次.

总次数 = (1 + 64 + 8 + 1) ＊ 2 = 148，与输出一致，其中 ＊ 2 是因为流水线阻塞一次会推迟 2 个时钟周期.

对比循环次数和阻塞次数可以发现开启 BTB 使程序得到了优化。

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）

思路 1:减少每层循环的次数，让 BMS 的次数增加超过打开 BTB 对 BTS 减少的增益，即可使 BTB 无效

思路 2：可以改变相邻循环的跳转地址，比如奇数跳转到地址 1，偶数跳转到地址 2，这样打开 BTB 后，BMS 的次数就会大大增加，使性能降低。

代码实现：

1. 矩阵初始化部分

初始化矩阵基地址：

使用 daddi 指令将矩阵 mx1、mx2 和 mx3 的基地址存入寄存器 $r22、$r23 和 $r21 中。

赋初值：

通过两个嵌套的循环（iloop1 和 iloop2）为 mx1 和 mx2 赋初值。

使用左移操作 dsll 计算矩阵元素的位置偏移（行偏移和列偏移）。

为每个位置赋固定的值（mx1 赋值为 2，mx2 赋值为 3）。

2. 矩阵乘法部分

三重循环（loop1、loop2、loop3）：

通过三重嵌套的循环进行矩阵乘法运算。

计算每个矩阵乘积 mx3[i][j] 的值，通过三个循环变量 i、j 和 k 分别实现矩阵的行、列和中间乘积部分。

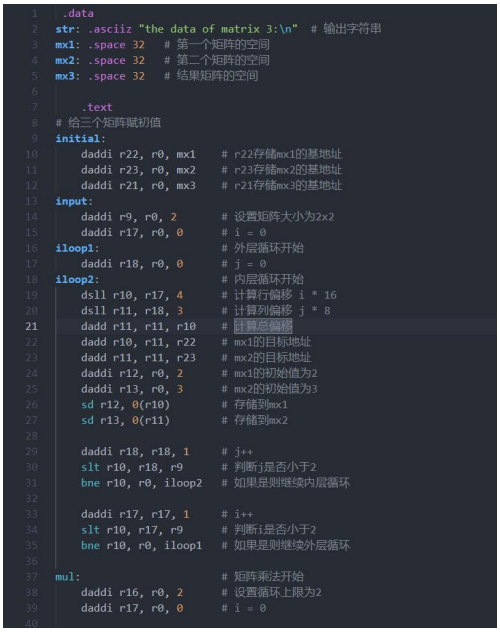
矩阵元素计算：

对于每个元素，首先通过左移计算对应的矩阵元素的地址，然后使用 ld（加载双字）将元素值加载到寄存器中。

使用 dmul 指令进行乘法操作，然后累加结果，最后使用 sd（存储双字）将计算结果存入目标矩阵 mx3 中。

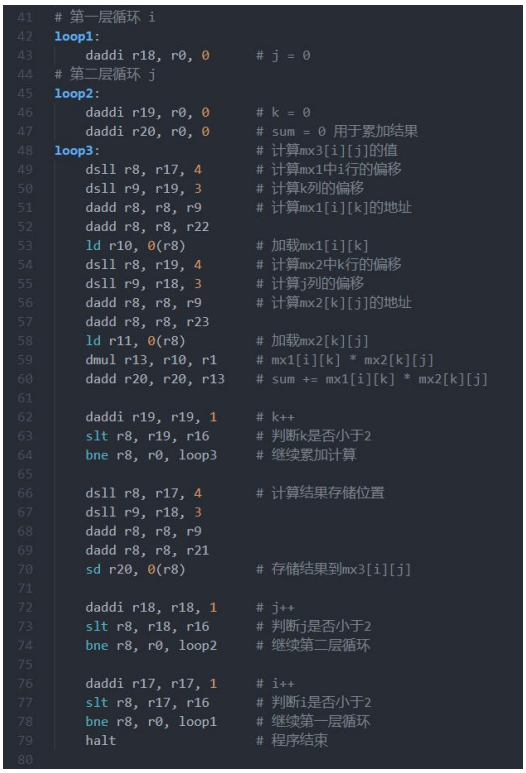
注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

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。



```
.data
str: .asciiz "the data of matrix 3:\n"  # 输出字符串
mx1: .space 32   # 第一个矩阵的空间
mx2: .space 32   # 第二个矩阵的空间
mx3: .space 32   # 结果矩阵的空间

    .text
# 给三个矩阵赋初值
initial:
    daddi r22, r0, mx1    # r22存储mx1的基地址
    daddi r23, r0, mx2    # r23存储mx2的基地址
    daddi r21, r0, mx3    # r21存储mx3的基地址
input:
    daddi r9, r0, 2       # 设置矩阵大小为2x2
    daddi r17, r0, 0      # i = 0
iloop1:                   # 外层循环开始
    daddi r18, r0, 0      # j = 0
iloop2:                   # 内层循环开始
    dsll r10, r17, 4      # 计算行偏移 i * 16
    dsll r11, r18, 3      # 计算列偏移 j * 8
    dadd r11, r11, r10    # 计算总偏移
    dadd r10, r11, r22    # mx1的目标地址
    dadd r11, r11, r23    # mx2的目标地址
    daddi r12, r0, 2      # mx1的初始值为2
    daddi r13, r0, 3      # mx2的初始值为3
    sd r12, 0(r10)        # 存储到mx1
    sd r13, 0(r11)        # 存储到mx2

    daddi r18, r18, 1     # j++
    slt r10, r18, r9      # 判断j是否小于2
    bne r10, r0, iloop2   # 如果是则继续内层循环

    daddi r17, r17, 1     # i++
    slt r10, r17, r9      # 判断i是否小于2
    bne r10, r0, iloop1   # 如果是则继续外层循环

mul:                      # 矩阵乘法开始
    daddi r16, r0, 2      # 设置循环上限为2
    daddi r17, r0, 0      # i = 0
```

图 7 代码 1



```
# 第一层循环 i
loop1:
    daddi r18, r0, 0      # j = 0
# 第二层循环 j
loop2:
    daddi r19, r0, 0      # k = 0
    daddi r20, r0, 0      # sum = 0 用于累加结果
loop3:                    # 计算mx3[i][j]的值
    dsll r8, r17, 4       # 计算mx1中i行的偏移
    dsll r9, r19, 3       # 计算k列的偏移
    dadd r8, r8, r9       # 计算mx1[i][k]的地址
    dadd r8, r8, r22
    ld r10, 0(r8)         # 加载mx1[i][k]
    dsll r8, r19, 4       # 计算mx2中k行的偏移
    dsll r9, r18, 3       # 计算j列的偏移
    dadd r8, r8, r9       # 计算mx2[k][j]的地址
    dadd r8, r8, r23
    ld r11, 0(r8)         # 加载mx2[k][j]
    dmul r13, r10, r1     # mx1[i][k] * mx2[k][j]
    dadd r20, r20, r13    # sum += mx1[i][k] * mx2[k][j]

    daddi r19, r19, 1     # k++
    slt r8, r19, r16      # 判断k是否小于2
    bne r8, r0, loop3     # 继续累加计算

    dsll r8, r17, 4       # 计算结果存储位置
    dsll r9, r18, 3
    dadd r8, r8, r9
    dadd r8, r8, r21
    sd r20, 0(r8)         # 存储结果到mx3[i][j]

    daddi r18, r18, 1     # j++
    slt r8, r18, r16      # 判断j是否小于2
    bne r8, r0, loop2     # 继续第二层循环

    daddi r17, r17, 1     # i++
    slt r8, r17, r16      # 判断i是否小于2
    bne r8, r0, loop1     # 继续第一层循环
    halt                  # 程序结束
```

图 8 代码 2

将上述程序（BTB_hacker.s）载入 WinMIPS64,开启 BTB 并运行，观察到 Statistics 窗口的输入如下图所示,发生了 20 次 BTS 和 20 次 BMS。

20 次 BTS 的计算:

(1) 初始化：3 次.

(2) 矩阵乘法：7 次.

(3) 总次数 = (3 + 7) * 2 = 20 次，与输出一致.

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。

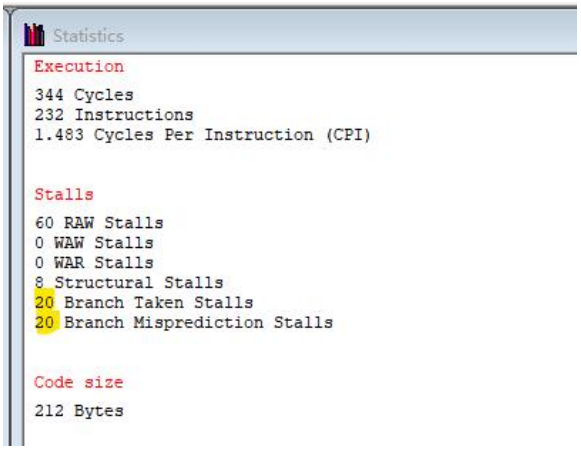


图 9 结果 1

将上述程序重新载入，关闭 BTB 并运行，观察到 Statistics 窗口的输入如下图所示，发生了 10 次 BTS ，性能约提升一倍.

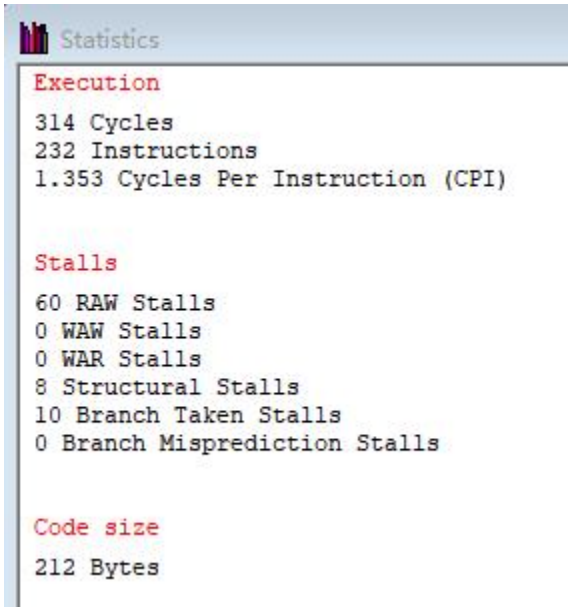


图 10 结果 2

该实验表明：并非开启 BTB 一定能提高性能，还与程序的实现有关，尤其是循环的层数和每轮的执行次数。

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。

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。

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。

（30 分，循环展开代码及思路 20 分，评判 10 分）

思路：

矩阵初始化部分：

初始化矩阵 mx1 和 mx2，并赋初值。

使用循环将矩阵的每个元素赋固定的初始值（mx1 赋值为 2，mx2 赋值为 3）。

矩阵乘法部分：

通过三重循环实现矩阵乘法运算。

使用了循环展开优化，手动展开了内层的乘法累加计算，减少循环控制开销。

```
.data
str: .asciiz "the data of matrix 3:\n"  # 输出字符串
mx1: .space 512   # 第一个矩阵的空间(8x8)
mx2: .space 512   # 第二个矩阵的空间(8x8)
mx3: .space 512   # 结果矩阵的空间(8x8)

    .text
initial:
    daddi r22, r0, mx1    # r22存储mx1的基地址
    daddi r23, r0, mx2    # r23存储mx2的基地址
    daddi r21, r0, mx3    # r21存储mx3的基地址
input:
    daddi r9, r0, 64      # 设置矩阵大小为64(8x8)
    daddi r8, r0, 0       # 循环计数器初始化为0
loop1:                    # 初始化矩阵的循环
    dsll r11, r8, 3       # 计算偏移量(乘以8)
    dadd r10, r11, r22    # mx1的目标地址
    dadd r11, r11, r23    # mx2的目标地址
    daddi r12, r0, 2      # mx1的初始值为2
    daddi r13, r0, 3      # mx2的初始值为3
    sd r12, 0(r10)        # 存储到mx1
    sd r13, 0(r11)        # 存储到mx2

    daddi r8, r8, 1       # 循环计数器加1
    slt r10, r8, r9       # 判断是否小于64
    bne r10, r0, loop1    # 如果小于64则继续循环

mul:                      # 矩阵乘法开始
    daddi r16, r0, 8      # 设置循环上限为8
    daddi r17, r0, 0      # i = 0

# 第一层循环 i
loop2:
    daddi r18, r0, 0      # j = 0
```

图 11 代码 1

```
# 第二层循环 j
loop3:
    daddi r19, r0, 0      # k = 0
    daddi r20, r0, 0      # r20存储累加结果

# 循环展开开始
    # k = 0
    dsll r8, r17, 6       # 计算mx1中i行的偏移(i*64)
    dsll r9, r19, 3       # 计算k列的偏移(k*8)
    dadd r8, r8, r9       # 计算总偏移
    dadd r8, r8, r22      # 计算mx1[i][k]的地址
    ld r10, 0(r8)         # 加载mx1[i][k]
    dsll r8, r19, 6       # 计算mx2中k行的偏移(k*64)
    dsll r9, r18, 3       # 计算j列的偏移(j*8)
    dadd r8, r8, r9       # 计算总偏移
    dadd r8, r8, r23      # 计算mx2[k][j]的地址
    ld r11, 0(r8)         # 加载mx2[k][j]
    dmul r13, r10, r11    # mx1[i][k] * mx2[k][j]
    dadd r20, r20, r13    # 累加结果
    daddi r19, r19, 1     # k++

    # k = 1 (以下7个块与k=0的计算过程相同)
```

图 12 代码 2

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。

关闭BTB 并运行，Statistics 窗口的输出如下图所示，发生了 126 次 BTS。

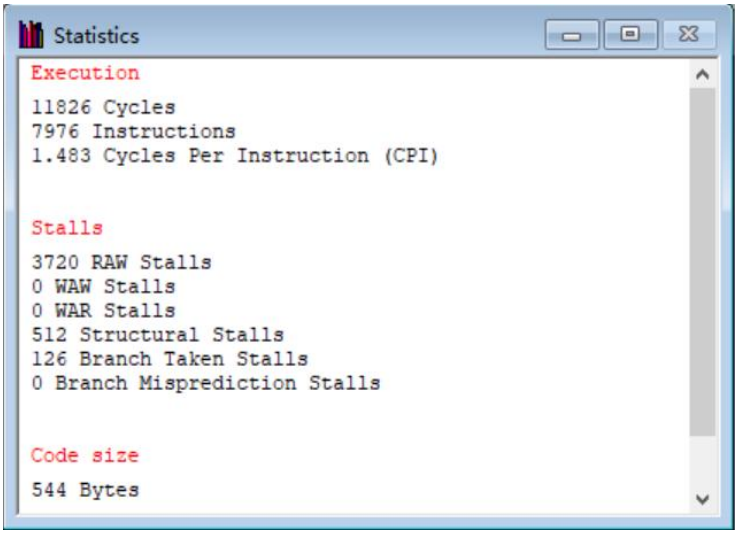


图13 结果1

循环展开前发生 574 次 BTS ，循环展开后减小了 448 次 BTS ，减量恰为矩阵乘法的第三层循环发生的 BTS 的次数。

将原程序重新载入 WinMIPS64 后，打开BTB 并运行，Statistics窗口的输出如下图所示，发生了 20 次 BTS 和 20 次 BMS。

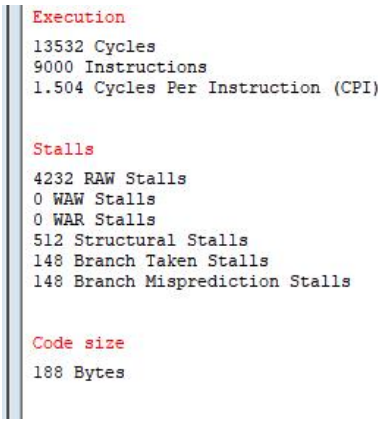


图14 结果2

循环展开前发生 148次 BTS ，循环展开后减小了 120 次 BTS ，减量恰为矩阵乘法的第三层循环发生的 BTS 的次数。

本次实验表明：循环次数较大时，开启 BTB 可提高性能。但循环层数过多时，内层循环会发生多次 BTS ，导致性能下降，此时可将循环次数较小的循环展开 以减小分支预测错误的次数，提高性能。

四、结束语

写下对于这次试验的所得与感想。

1. BTB 对代码优化的作用

提升循环结构的性能：

分支目标缓冲器（BTB）对规则循环结构具有显著的优化作用。通过缓存分支目标地址，BTB 能够有效预测嵌套循环中的分支行为，从而减少因错误预测引发的流水线清空与重新

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。

加载。
矩阵乘法的优化：
在矩阵乘法代码中，BTB 能够准确预测外层和内层循环的分支行为，保持流水线的顺畅执行。这样一来，分支停顿的次数大幅减少，整体性能显著提高。
2. 使 BTB 无效的代码设计
随机或频繁变化的分支模式：
通过设计随机分支行为或频繁变化的分支模式，可以使 BTB 的分支预测能力失效。在这种情况下，BTB 的命中率会显著下降，导致流水线频繁清空，程序执行效率随之降低。
BTB 的局限性：
当分支行为不可预测或具有较强随机性时，BTB 的优势难以发挥，甚至可能增加预测错误的开销。这种情况下，BTB 不仅无法优化性能，反而可能引发额外的性能损失。
3. 循环展开对流水线性能的影响
减少分支指令：
循环展开是一种有效的优化技术，能够减少分支指令数量，从而降低分支预测错误率，显著减少流水线停顿。在矩阵乘法代码中，如果将内层循环完全展开，分支控制指令的数量将大幅减少，流水线性能随之提升。
BTB 的协同作用：
与未展开的代码相比，循环展开后的代码在 BTB 的辅助下表现出更优的性能。这是因为展开后的代码减少了对 BTB 的依赖，流水线执行更加连贯，进一步提升了执行效率。
结论
BTB 的优势与局限性：
BTB 在规则性循环中通过减少错误预测和流水线停顿，显著提高了代码执行性能。然而，当面对不可预测的分支行为时，BTB 的优化效果会显著下降，甚至可能导致额外的性能开销。
循环展开的效果：
循环展开能够有效减少分支指令的频率，降低流水线停顿的可能性。当与 BTB 优化协同使用时，循环展开可以进一步提升流水线的执行效率。
优化策略选择：
在编写和优化代码时，应根据分支的可预测性选择合适的优化策略。通过合理利用 BTB 和循环展开等技术，可以最大限度减少分支开销，确保程序性能的最优表现。

（报告撰写质量 10 分）

五、实验结果

1. 矩阵乘法优化

```
Execution
13810 Cycles
9000 Instructions
1.534 Cycles Per Instruction (CPI)

Stalls
4232 RAW Stalls
0 WAW Stalls
0 WAR Stalls
512 Structural Stalls
574 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
188 Bytes
```

图 15 结果 1

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。

分析：

初始化部分（Init）

- 循环次数：

  共有 N × N = 8 × 8 = 64 次迭代。

- 分支取情况：

  - 循环条件检查：每次迭代结束后都会检查条件 i < 64。当条件为真时，分支被取，循环继续；当条件为假时，分支不被取，循环结束。

  - 分支被取次数计算：循环从 i = 0 到 i = 63，共计 64 次迭代，其中有 63 次条件为真，分支被取；最后 1 次条件为假，分支不被取。

- 总分支被取次数：初始化部分的分支被取次数为 63 次。

乘法部分（Multiplication）

外层循环（变量 i）

- 循环次数：N = 8 次。
- 分支被取次数：8 - 1 = 7 次。

中间循环（变量 j）

- 每次外层循环的迭代中，中间循环执行 N 次，单次循环分支被取次数为 8 - 1 = 7 次。
- 总迭代次数：N × N = 8 × 8 = 64 次。
- 总分支被取次数：7 次（每次中间循环） × 8 次（外层循环次数） = 56 次。

内层循环（变量 k）

- 每次中间循环的迭代中，内层循环执行 N 次，单次循环分支被取次数为 8 - 1 = 7 次。
- 总迭代次数：N × N × N = 8 × 8 × 8 = 512 次。
- 总分支被取次数：7 次（每次内层循环） × 64 次（中间循环总次数） = 448 次。

总分支被取次数

汇总各部分的分支被取次数：

- 初始化部分：63 次
- 外层循环：7 次
- 中间循环：56 次
- 内层循环：448 次

总计：63 + 7 + 56 + 448 = 574

理论分析结果与实际预期完全一致，表明分支被取次数的计算准确无误。

```
Execution

13532 Cycles
9000 Instructions
1.504 Cycles Per Instruction (CPI)


Stalls

4232 RAW Stalls
0 WAW Stalls
0 WAR Stalls
512 Structural Stalls
148 Branch Taken Stalls
148 Branch Misprediction Stalls


Code size

188 Bytes
```

图 16 结果 2

148 次 BTS 的原因：

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。

（3） init 部分：一层循环，发生 1 次 BTS 和 1 次 BMS

（4） 矩阵乘法部分：

a) 第一层循环，执行 8＊8＝64 轮，每轮发生 1 次 BTS 和 1 次 BMS，共计各 64 次.

b) 第二层循环，执行 8 轮，每轮发生 1 次 BTS 和 1 次 BMS，共计各 8 次.

c) 第三层循环，执行 1 轮，发生 1 次 BTS 和 1 次 BMS，共计各 1 次.

总次数 = (1 + 64 + 8 + 1) ＊ 2 = 148，与输出一致，其中 ＊ 2 是因为流水线阻塞一次会推迟 2 个时钟周期.

对比循环次数和阻塞次数可以发现开启 BTB 使程序得到了优化。

2. 设计使 BTB 无效的代码

思路 1:减少每层循环的次数，让 BMS 的次数增加超过打开 BTB 对 BTS 减少的增益，即可使 BTB 无效

思路 2：可以改变相邻循环的跳转地址，比如奇数跳转到地址 1，偶数跳转到地址 2，这样打开 BTB 后，BMS 的次数就会大大增加，使性能降低。

实验表明：并非开启 BTB 一定能提高性能，还与程序的实现有关，尤其是循环的层数和每轮的执行次数，跳转条件。

3. 循环展开与 BTB 的效果对比

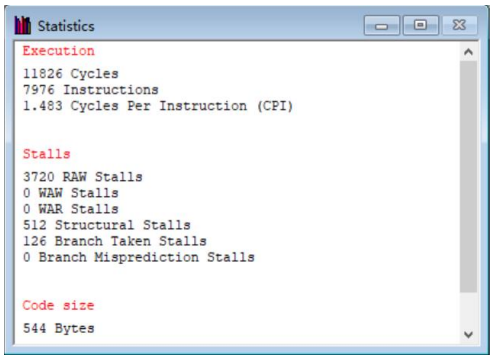


图17 结果3

循环展开前发生 574 次 BTS，循环展开后减小了 448 次 BTS，减量恰为矩阵乘法的第三层循环发生的 BTS 的次数。

将原程序重新载入 WinMIPS64 后,打开BTB 并运行，Statistics窗口的输出如下图所示，发生了 20 次 BTS 和 20 次 BMS。

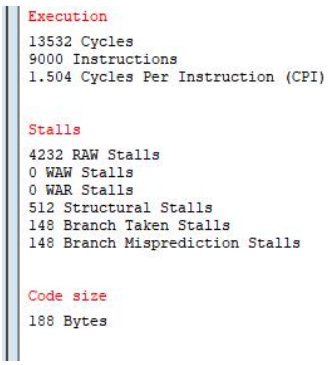


图18 结果4

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。

循环展开前发生 148次 BTS ，循环展开后减小了 120 次 BTS ，减量恰为矩阵乘法的第三层循环发生的 BTS 的次数。

本次实验表明：循环次数较大时，开启 BTB 可提高性能。但循环层数过多时，内层循环会发生多次 BTS ，导致性能下降，此时可将循环次数较小的循环展开 以减小分支预测错误的次数，提高性能。

五、实验总结与体会

1. BTB 对代码优化的作用

提升循环结构的性能：

分支目标缓冲器（BTB）对规则循环结构具有显著的优化作用。通过缓存分支目标地址，BTB 能够有效预测嵌套循环中的分支行为，从而减少因错误预测引发的流水线清空与重新加载。

矩阵乘法的优化：

在矩阵乘法代码中，BTB 能够准确预测外层和内层循环的分支行为，保持流水线的顺畅执行。这样一来，分支停顿的次数大幅减少，整体性能显著提高。

2. 使 BTB 无效的代码设计

随机或频繁变化的分支模式：

通过设计随机分支行为或频繁变化的分支模式，可以使 BTB 的分支预测能力失效。在这种情况下，BTB 的命中率会显著下降，导致流水线频繁清空，程序执行效率随之降低。

BTB 的局限性：

当分支行为不可预测或具有较强随机性时，BTB 的优势难以发挥，甚至可能增加预测错误的开销。这种情况下，BTB 不仅无法优化性能，反而可能引发额外的性能损失。

3. 循环展开对流水线性能的影响

减少分支指令：

循环展开是一种有效的优化技术，能够减少分支指令数量，从而降低分支预测错误率，显著减少流水线停顿。在矩阵乘法代码中，如果将内层循环完全展开，分支控制指令的数量将大幅减少，流水线性能随之提升。

BTB 的协同作用：

与未展开的代码相比，循环展开后的代码在 BTB 的辅助下表现出更优的性能。这是因为展开后的代码减少了对 BTB 的依赖，流水线执行更加连贯，进一步提升了执行效率。

结论

BTB 的优势与局限性：

BTB 在规则性循环中通过减少错误预测和流水线停顿，显著提高了代码执行性能。然而，当面对不可预测的分支行为时，BTB 的优化效果会显著下降，甚至可能导致额外的性能开销。

循环展开的效果：

循环展开能够有效减少分支指令的频率，降低流水线停顿的可能性。当与 BTB 优化协同使用时，循环展开可以进一步提升流水线的执行效率。

优化策略选择：

在编写和优化代码时，应根据分支的可预测性选择合适的优化策略。通过合理利用 BTB 和循环展开等技术，可以最大限度减少分支开销，确保程序性能的最优表现。

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。

指导教师批阅意见： **成绩评定：** 指导教师签字： 2024 年 12 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。