计算机体系结构



**分支预测实验报告**

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 专 业： | 计算机科学与技术 |
| 指导教师： | 吴为民 |
| 班 级： | 计科1602 |
| 姓 名： | 麻锦涛 |
| 学 号： | 16281262 |

2019年 5月 26日

1. 实验目的

理解分支预测的原理。

1. 实验内容
2. 理解转移延迟槽(branch slot)的作用
   1. 将WinMIPS64设置成允许Enable Forwarding和不允许Enable Branch Delay Slot，运行 incre1.s，记录运行结果。运行结果包括 Data 窗口和 Statistics 窗口的结果。
   2. 考虑带一个延迟槽的情况下，对 incre1.s 进行调度。请仔细考虑，教材上给出的三种策略中，哪种策略适合这个程序。将 WinMIPS64 设置成允许 Enable

Forwarding 和允许 Enable Branch Delay Slot，运行你调度后的程序并记录运行结果。运行结果包括 Data 窗口和 Statistics 窗口的结果。

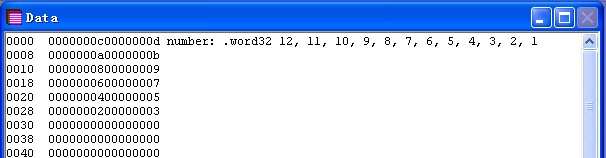
* 1. 比较①和②两种情况下的运行结果。首先要确定两种情况下 Data 窗口中的运行结果是一致的。其次，比较两种情况下 Statistics 窗口的性能结果，分析一下为什么调度后运行周期减少了。

1. 理解分支目标缓冲器(BTB)的作用
   1. 将WinMIPS64设置成允许Enable forwarding 和不允许Enable Branch Target buffer，运行 incre1.s 程序，记录 Statistics 窗口的统计结果。
   2. 将 WinMIPS64 设置成允许 Enable forwarding 和允许 Enable Branch Target buffer，运行 incre1.s 程序，记录 Statistics 窗口的统计结果。
   3. 比较①和②两种情况下的运行结果，解释为什么允许 BTB 就减少了运行周期数。为便于解释，建议按教材中图 3.49 的 BTB 表的形式，给出在程序运行过程中 BTB 的变化情况。
2. 实验过程及结果
   1. 理解转移延迟槽**(branch slot)**的作用

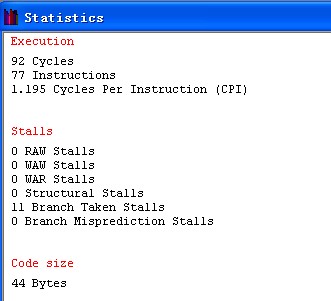
**3.1.1** 允许 **Enable Forwarding** 和不允许 **Enable Branch Delay Slot**

此条件下，运行 incre1.s 程序，运行结果如下:

* Data 窗口：



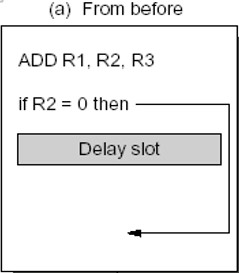
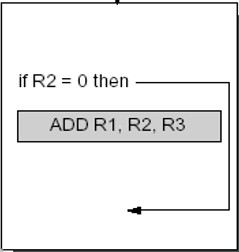
* Statistics 窗口：



**3.1.2** 允许 **Enable Forwarding** 和允许 **Enable Branch Delay Slot**

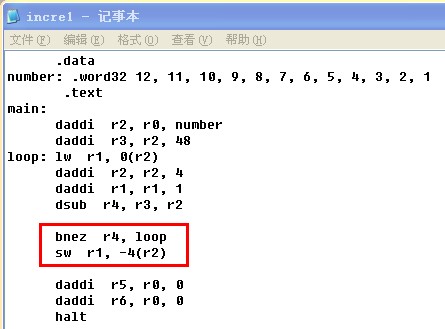
三种分支延迟的调度方法分别为：

* 来自以前

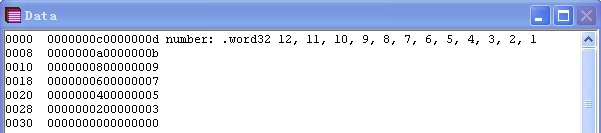
 变成 

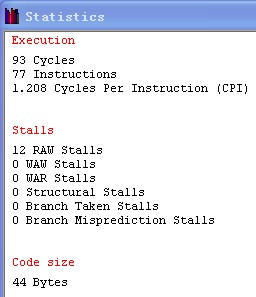
调度策略：将转移指令之前并且是没有数据依赖关系的语句放在转移指令后的第一条语句的位置，这样就不会产生因转移语句的跳转而重新取值造成的 stall.这种调度策略是最好的选择。

使用此方法进行调度，下图中红色框内为代码改动部分，

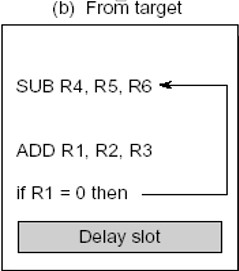
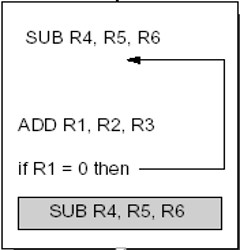


Data 窗口和 Statistics 窗口运行结果如下：



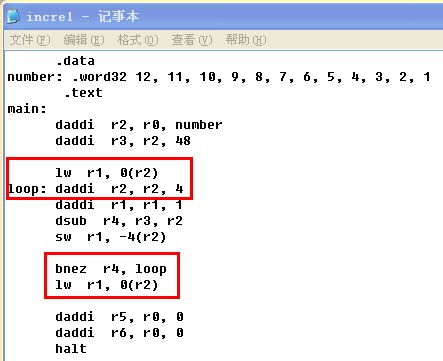


* 拷贝自目标

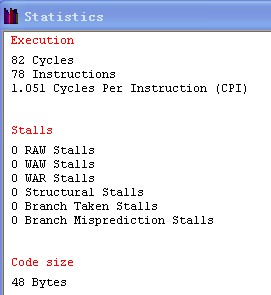
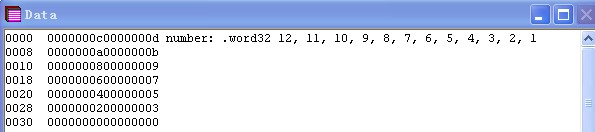
 变成 

调度策略：延迟槽中拷贝目标跳转语句，这种策略在概率较大的分支中比较适用，如 loop 循环。

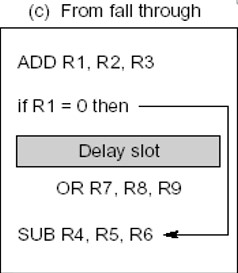
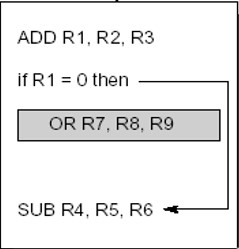
使用此方法进行调度，下图中红色框内为代码改动部分，



Data 窗口和 Statistics 窗口运行结果如下：

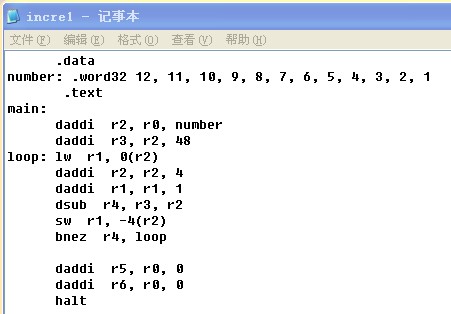


* 来自转移不成功分支

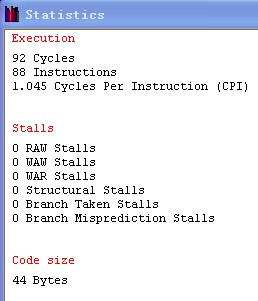
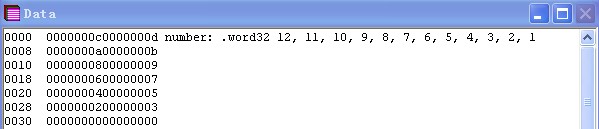
 变成 

调度策略：延迟槽中拷贝转移不成功分支，这种方法与上一种拷贝目标的共同点是虽然工作是浪费的，但程序仍将正确执行。

使用此方法进行调度，下图中红色框内为代码改动部分，



Data 窗口和 Statistics 窗口运行结果如下：



这三种调度方法中，拷贝目标更适合于这个程序，因为尽管来自以前的这种调度方法是最好的选择，但是由于此程序语句之间的数据依赖关系，在考虑一个延迟槽的情况下，且保证运行结果正确，可选择调整到延迟槽的语句唯一，但由于该语句是 **store** 型，所以调度后的程序会产生新的 **RAW** 型数据冒险，对程序运行性能而言，得不偿失；而且由于此程序是 **loop** 循环语句的转移指令，且循环次数较大，符合拷贝目标方法的适用条件，所以拷贝目标更适合这个程序。

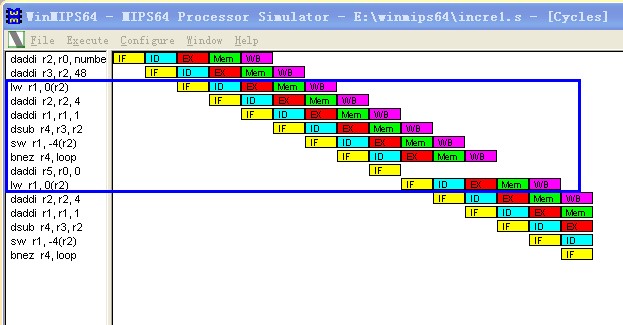
**3.1.3** 比较 **3.1.1** 和 **3.1.2** 的运行结果

观察 3.1.2 使用不同调度方法，运行修改后的程序后，由 Data 窗口可知，程序运行的结果均是正确的。在上一小节中分析得，拷贝目标更适合于此程序。

对比源程序和采用拷贝目标调度后的程序

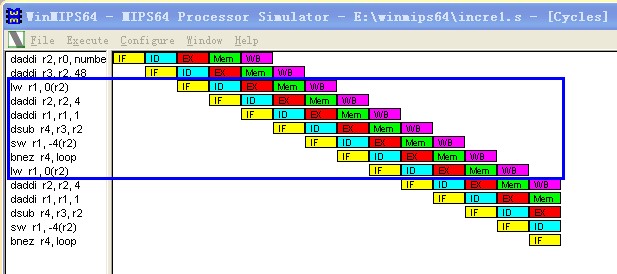
1. 单个周期内运行对比：
   * + - 不允许使用延迟槽时，单个周期内，由于流水线的原理，无法在取值 IF

阶段就知道转移指令跳转的目标指令，因此，当寄存器 r4 的值不为 0 时， bnez r4, loop 这条语句执行后应该跳转到 loop 标志的循环处，尽管有前送设置，但是由于需要计算转移地址交给 PC，仍然会有一个周期的 stall。

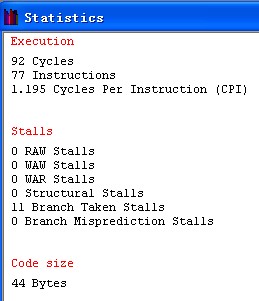


* + - * 考虑一个延迟槽时，将循环执行的第一条语句 lw r1, 0(r2)放入延迟槽中，

并将循环标志 loop 移到循环部分的第二条语句 daddi r2, r2, 4 处，这样除了寄存器 r4 的值为 0 的情况下，会多执行一次 lw r1, 0(r2)语句，其余情况下，转移指令语句的下一条语句直接是循环部分的第一条语句，则每次转移语句后取到的指令均为正确的目标语句，因此在每次循环时都避免了一个周期的 branch taken stall，因而显著地减少了程序的运行周期。

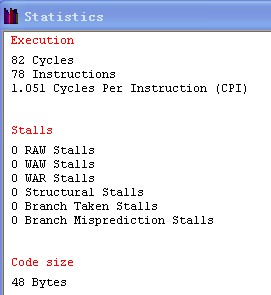


1. 程序运行完后性能结果对比：
   * + - 不允许延迟槽时，由于每次转移条件成立时都要进行跳转，流水线都需要重新取值，因此整个程序运行结束后，共产生 11 个 branch taken stall，运行了 92 个周期。此条件下程序的 CPI = 1.195.



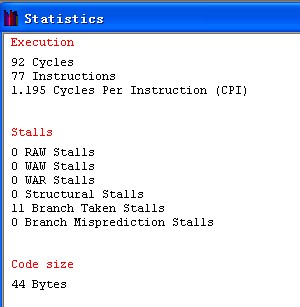
* + - * 考虑一个延迟槽时，经过拷贝目标的调度下，之前每次转移进行跳转需要重新取值的情况发生改变，每次转移语句的下一句取到的指令即为条件成立时需要跳转到的目标指令，因此，除了唯一的 r4 = 0 的情况下，每一次转移指令执行的下一句都是正确得到目标指令，消除了所有的 branch taken

stall. 当然程序为此付出的代价是，当转移条件不成立时执行了一条多余的指令，增加了 1 个周期。但是相比不使用延迟槽，已经显著地减少了周期数，整个程序运行结束后，没有产生任何类型的 stall，运行了 82 个周期。这种情况下程序的 CPI = 1.051. 与标准流水线的 CPI = 1 已经很接近了。

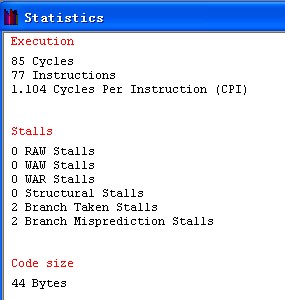


**3.2** 理解分支目标缓冲器**(BTB)**的作用

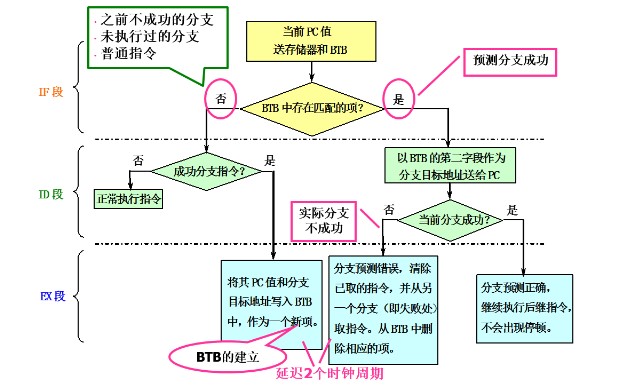
* + 1. 允许 **Enable forwarding** 和不允许 **Enable Branch Target buffer** 运行结果-Statistics 窗口：



* + 1. 允许 **Enable Forwarding** 和允许 **Enable Branch Delay Slot** 运行结果-Statistics 窗口：



* + 1. 比较 **3.2.1** 和 **3.2.2** 的运行结果



允许 BTB 时，首先在第一次执行条件转移语句 bnez r4, loop 时，因为 BTB 表为空，

|  |  |
| --- | --- |
| 执行过的成功分支指令的地址 | 预测的分支目标地址 |
| - | - |

所以在 BTB 表中并未找到该 PC 值对应的预测分支目标地址，但是接下来发现该指令是一个转移成功的指令，就将该指令连同目标地址送入缓冲器。此时，BTB 的内容如下表：

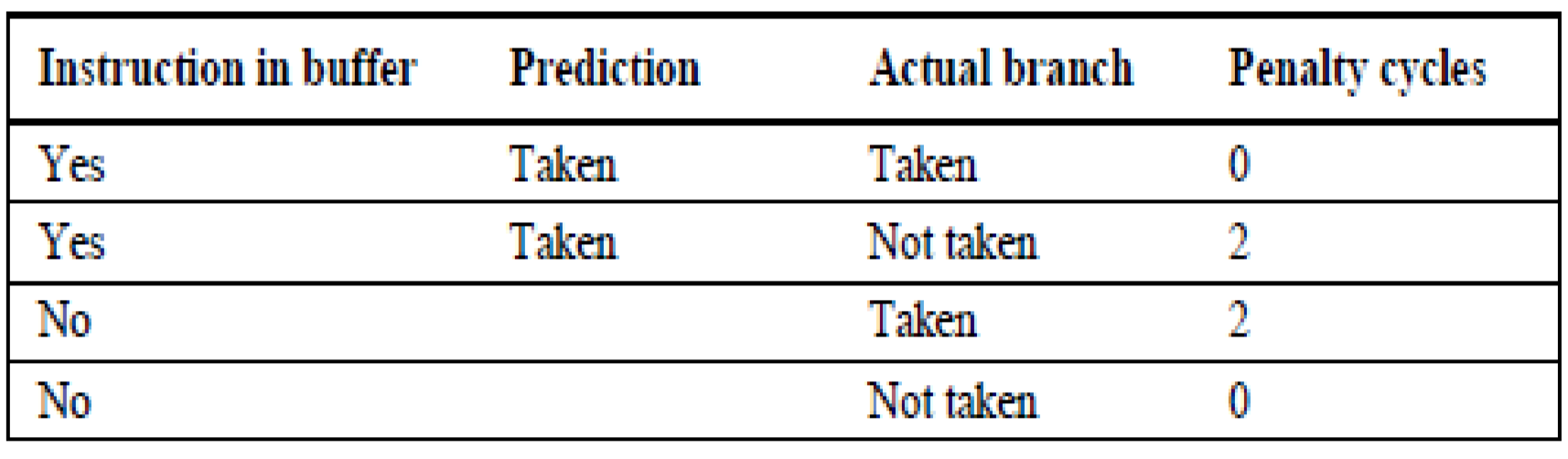
|  |  |
| --- | --- |
| 执行过的成功分支指令的地址 | 预测的分支目标地址 |
| bnez r4, loop 语句的地址值 | lw r1, 0(r2)语句的地址值 |

接下来，每次循环 PC = bnez r4, loop 语句的地址值时，均能从 BTB 表中找到该 PC，同时由于寄存器 r4 的值不为 0，所以该指令是一个成功的转移，因此，BTB 表中的该项不会被移除。直到执行到循环的最后一次，此时 r4 = 0，需要跳出循环，但由于取到 bnez r4, loop 指令时，仍然可以在 BTB 表中找到预测项，因此会在 ID 段就立即从预测的 PC 处取指，然而，会发现这是一个不成功的转移，需要将其从缓冲器中移除。BTB 表被清空。

|  |  |
| --- | --- |
| 执行过的成功分支指令的地址 | 预测的分支目标地址 |
| - | - |

不允许 BTB 和允许 BTB 的运行性能结果对比和分析：

* 不允许 BTB 时，运行周期数为 92，CPI = 1.195，共产生 11 个 Branch Taken stall；
* 允许 BTB 时，运行周期数位 85，CPI = 1.104，共产生 2 个 Branch Taken stall 和 2 个 Branch Misprediction stall；



在第一次转移指令执行时，因为在BTB 中并未找到对应的预测目标地址，惩罚取决于实际是否转移，因为实际情况是转移，所以按照预测不转移的策略处理，由于需要重新取值和更新缓冲器，更新缓冲器时需要将该 PC 和与其对应的预测目标地址作为一组记录放入缓冲器中，所以会有两个周期的 stall.

在最后一个转移指令执行时，因为在 BTB 中找到了对应于该 PC 的预测目标地址，在 ID 阶段会直接按照预测目标地址取值，但是由于实际情况并不需要转移，因此转移不成功，需要重新取值，并且更新缓冲器，将该 PC 从 BTB 中移除。同样会产生 2 个周期的 stall.。