

**《计算机体系结构》课程**

**实验报告**

|  |  |
| --- | --- |
| **实验名称：** | 实验五 Branch-Target Buffers |
| **姓 名：** | 江家玮 |
| **学 号：** | 22281188 |
| **日 期：** | 2024.12.12 |

**目录**

**[1 实验目的 1](#_Toc30785)**

**[2 实验内容 1](#_Toc23711)**

**[3 实验流程 2](#_Toc20537)**

[3.1不采用BTB技术 2](#_Toc24764)

[3.2采用BTB技术 4](#_Toc17645)

[3.3 BTB技术与BPB技术的区别 6](#_Toc4125)

[3.4 在DLX架构中的性能提升 6](#_Toc23666)

**[4 实验结果分析 7](#_Toc28211)**

**[5 总结 8](#_Toc4960)**

# 1 实验目的

本次实验的主要目的是加深对Branch-Target Buffers的理解。掌握使用Branch-Target Buffers减少或增加分支带来的延迟的情况。

# 2 实验内容

将以下程序段修改为可利用WinMIPS64模拟器运行的程序。假设R3的初始值为R2+40

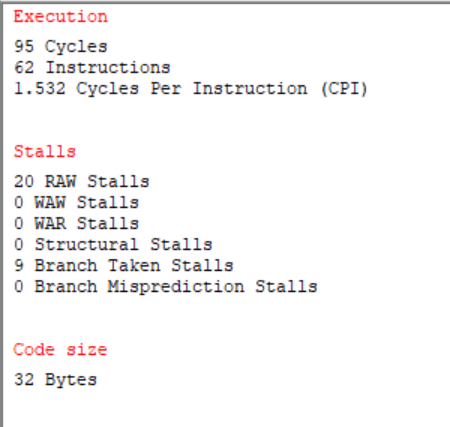
|  |
| --- |
| 程序段3 |
| Loop:    LW              R1,0(R2);    load R1 from address 0+R2                ADDI         R1,R1,#1;            R1=R1+1                SW           0(R2),R1;            store R1 at address 0+R2                ADDI        R2,R2,#4;            R2=R2+4                SUB           R4,R3,R2;           R4=R3-R2                BNEZ        R4,Loop;             Branch to loop if R4!=0 |

在使用forwarding的情况下，对比采用BTB与不采用BTB技术时流水线变化。**重点分析两种情况下每次循环的stall周期数，都是由什么原因造成的**？重点分析与分支指令相关的stall。**采用BTB技术何时能够减少分支指令带来的暂停？何时会增加暂停？为什么？**

BTB技术与BPB技术的区别是什么？对于DLX各自在什么情况下会带来性能提升？

# 3 实验流程

## 3.1不采用BTB技术



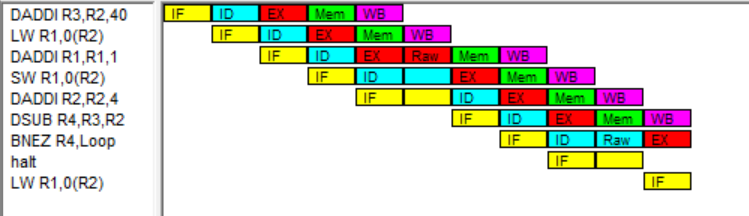
**图3-1-1 不采用BTB技术的执行周期**

这里的延迟分别对应以下两种情况：

* RAW相关带来的stall：每个循环中包含一次load指令，由于RAW相关，导致需要等待上一条指令的结果，这引发了stall。
* 分支指令引发的stall：分支指令在ID阶段需完成目标地址和转移条件的计算，而这些计算依赖于上一条指令的结果，但结果尚未生成，因而产生stall。

程序总共循环了十次，因此累计发生了20次RAW相关的停顿。

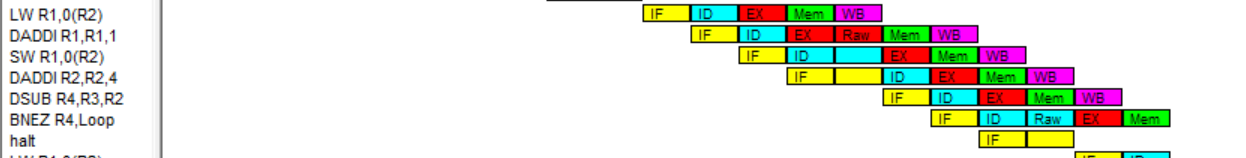
此外，由于分支指令采用了预测转移失败的方式处理，实际运行中，前九次分支转移均成功，因此后续预测错误的指令被取消，导致了九次控制相关的stall。



**图3-1-2 不采用BTB技术的第一周期执行效果展示**

在不采用BTB的情况下，程序在第一个循环周期中出现了以下停顿情况：

* 两个RAW相关的停顿：由于load指令的互锁和分支指令在ID阶段需要计算目标地址及转移条件所依赖的数据尚未计算完成，因此引发了两次RAW相关的停顿。
* 一个控制相关的停顿：由于分支指令采用了预测转移失败的方式，实际运行中发现分支成功，这导致需要废弃已经取出的下一条指令，从而产生一个额外的周期浪费，对应于branch taken stalls。



**图3-1-3 不采用BTB技术的第二周期执行效果展示**

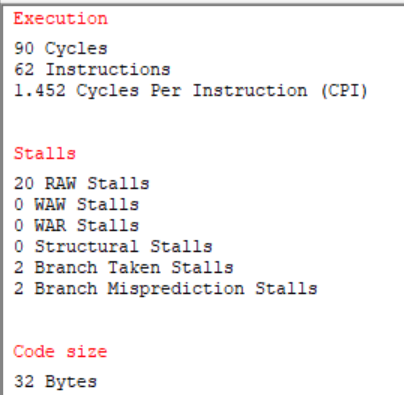
不采用BTB的第二个循环的情况和之前没有区别。



**图3-1-4 不采用BTB技术的第十周期执行效果展示**

没有采用BTB的情况，两次RAW相关都一样，然后是转移指令这里，因为这里循环结束转移失败，刚好[编译器](https://marketing.csdn.net/p/3127db09a98e0723b83b2914d9256174?pId=2782?utm_source=glcblog&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_51529433/article/details/_blank)用的是预测转移失败，所以相当于一个周期没有浪费直接就可以直接之后最后的halt指令。

# 3.2采用BTB技术



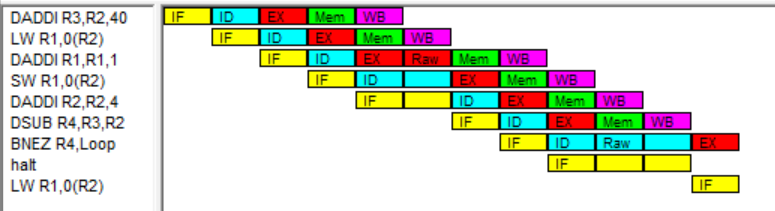
**图3-2-1 采用BTB技术的执行周期**

20次由于RAW相关引发的暂停，与未使用BTB（分支目标缓冲器）时的情况一致。此外，整个程序中总共出现了4个额外的周期停顿，具体分析如下：

* 前两个周期的停顿：在第一次遇到分支指令时，由于BTB表尚为空，默认按顺序执行PC+4地址的指令。然而，后续检测到分支转移成功，此时需要废弃之前取出的下一条指令（IF周期白干），并花费一个额外的周期重新取指。与此同时，分支指令及其目标地址会被存入BTB表。这种情况带来了两个周期的浪费，对应图中的branch taken stalls。
* 后续循环的处理：在接下来的多个循环中，由于BTB正确预测了分支转移成功，因此没有额外的周期浪费。
* 最后一个循环的停顿：在最后一次循环中，BTB预测分支转移成功，但实际并未转移，因此浪费了两个周期。

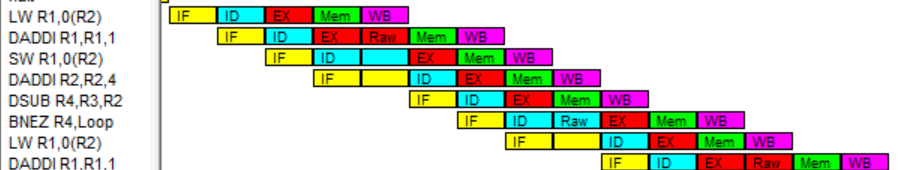
这种预测机制的优化，使得大多数循环中没有额外的停顿，但依然存在一定的预测误差开销。

采用了BTB的情况下，除了两个RAW相关和之前一样，这里一开始预测不转移，所以这里出现了halt指令，也就是出循环后的指令，而在前面的转移指令转移成功之后，需要取消指令，并且把这条分支指令和对应的目标地址存到BTB表中，所以红箭头所指的就是BTB导致的两个branch taken stalls（中间那个只是RAW暂停的顺延）：



**图3-2-2 采用BTB技术的第一周期执行效果展示**

在第二个执行周期中，采用BTB的情况下，除了两个RAW相关不变之外，因为此时的BTB表中已经有了对应分支指令地址的选择了，所以当同样的分支指令再次执行的时候，这次默认的就是转移成功，提前获得对应目标地址，所以可以看到这个周期的时候没有控制相关导致的暂停：



**图3-2-3 采用BTB技术的第二周期执行效果展示**

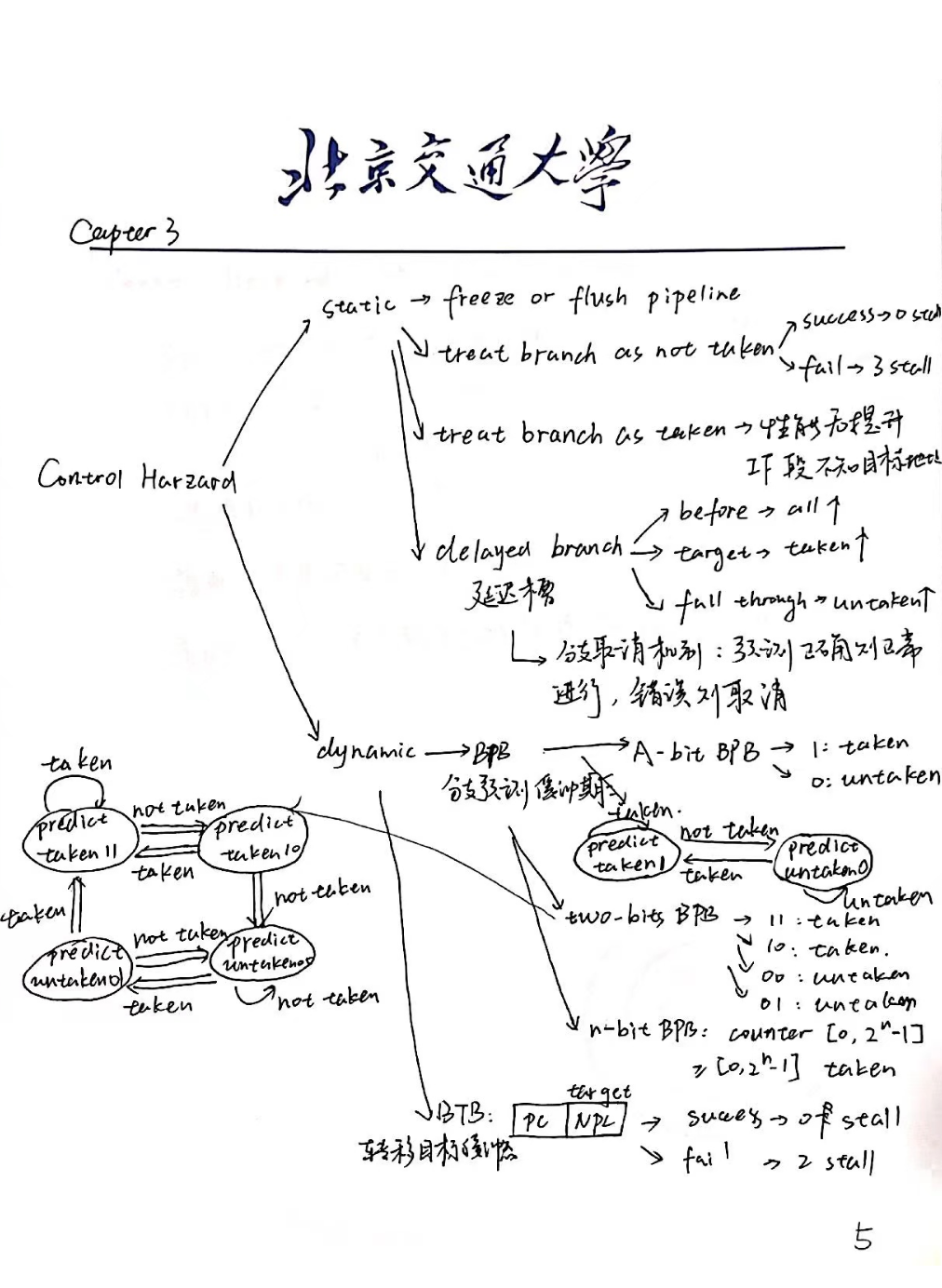
因为预测正确了，所以BTB表不会有变化。之后从第三个循环开始到第九个循环，都是和第二个循环一样的情况。



**图3-2-4 采用BTB技术的第十周期执行效果展示**

采用了BTB的情况，除了两次RAW相关之外，还有两个周期的branch misprediction stalls，因为之前的BTB表中是有这个分支指令和对应的目标地址的，所以这里依旧是预测转移成功，但是因为这里转移实际是失败了的，所以需要取消指令，重新取指，并且把BTB表中对应的指令项删除，浪费了两个周期。

## 3.3 BTB技术与BPB技术的区别



**BTB：**

* BTB是分支目标缓冲器，主要存储分支指令的目标地址。当处理器遇到分支指令时，如果该指令的目标地址已经在BTB中，则可以直接使用该地址进行跳转，而不需要等待分支指令的计算结果。
* BTB通过缓存已知的分支指令及其目标地址来减少分支指令引发的停顿，特别是在循环结构中，常常可以避免多次计算相同的目标地址，从而提高性能。

**BPB：**

* BPB是分支预测缓冲器，主要用于存储分支指令的历史执行信息（如：是否跳转）。BPB的作用是根据历史分支行为来预测分支是否会发生跳转。
* BPB通过记录每个分支指令的历史行为来提高分支预测的准确性。根据这些信息，处理器可以更好地预测分支是否会发生跳转，从而减少因错误预测导致的停顿。

**区别：**

* BTB的核心是**缓存分支指令的目标地址**，减少了因分支指令需要计算目标地址而导致的停顿；
* BPB的核心是**预测分支的跳转行为（是否跳转）**，减少因错误预测导致的停顿。

## 3.4 在DLX架构中的性能提升

* **BTB技术带来的性能提升：**
  + 当分支行为**稳定**时（如循环中的分支），BTB技术可以大幅减少由于重复计算目标地址而引发的停顿。因为分支的目标地址一旦被缓存到BTB，处理器可以直接跳转到目标地址，而不需要等待目标地址的计算。尤其是在循环中，如果分支的目标地址是固定的，BTB技术能够显著减少分支相关的停顿周期。
  + 当分支预测**正确**时，BTB能够直接提供正确的跳转目标，无需额外的控制相关延迟。
* **BPB技术带来的性能提升：**
  + 当分支指令具有较强的**规律性**时，BPB技术能够通过预测分支的跳转情况来减少因等待分支结果而造成的停顿。例如，在多次分支的情况下，BPB可以根据历史记录预测分支是否跳转，提前加载下一条指令，从而减少因错误预测导致的停顿。
  + 当分支指令的预测**准确性较高**时，BPB技术能够有效减少分支相关的控制停顿，提高程序执行效率。

**性能提升的情况**：

* **BTB**：适用于分支目标地址**固定**的场景，特别是循环中的分支。BTB可以提前获得目标地址，避免了每次执行分支指令时都需要计算目标地址。
* **BPB**：适用于分支跳转行为具有**规律性**的场景。BPB通过历史数据预测分支的跳转行为，减少了预测错误时的控制停顿。

然而，由于BPB其实在DLX中无太大效果，因为我们无法提前知道目标地址，而BTB改进了这个点，通过存储所预测的转移后下一条指令的地址，在IF段取来指令地址访问缓冲器，若命中，在IF就知道了预测地址，比BPB早了一个周期。

# 4 实验结果分析

实验表明，在预测正确的情况下，BTB技术显著减少了分支指令的停顿。这是因为在取指阶段就能提前确定分支指令是否转移及其目标地址，无需等待ID阶段的转移条件和目标地址计算即可继续取指，从而节省了分支指令本应引发的一个停顿周期。

但在预测失败时，停顿周期反而增加。失败后系统需重新取指，并将新计算出的转移地址和下一个PC值写入BTB表。不仅无法节省分支指令的原本停顿周期，还需要额外周期进行BTB表更新，导致额外开销。

# 5 总结

BTB（分支目标缓冲器）技术在循环执行过程中能够有效减少分支指令引起的停顿。当分支指令持续发生转移且BTB中已存有对应的目标地址时，系统会默认预测转移成功。这种情况下，无需浪费任何周期即可直接开始下一条指令的取指，从而大幅降低分支指令带来的停顿。

然而，BTB技术在循环开始和结束时可能会导致额外的停顿周期：

循环开始时的停顿：由于BTB中尚未记录当前分支指令相关信息，预测的转移状态可能错误，需要废弃错误指令、重新取指并修改BTB表，从而引发两个周期的控制相关停顿。

循环结束时的停顿：此时分支指令的转移状态发生变化（如预测转移但实际未转移），同样需要废弃错误指令、重新取指并更新BTB表，也会导致两个周期的控制相关停顿。