ADSP-TS101分支目标缓冲的解释

最后更新：2/02

C.L.提供

以下的EE工程文件将讨论TigerSHARC DSP ADSP-TS101中的分支目标缓冲（BTB），将解释分支目标缓冲区如何工作，什么时候使用它有优势，以及使用它可能引起的错误。

简介

TigerSHARC之所以能够到达那么高的执行速度，部分原因是它有8级深度的流水线。然而，这种深指令流水的缺陷就是当执行分支指令时，必须刷新指令流水。将被分支的新指令在被执行之前，必须遍历这个流水线，这会导致3到6个周期的时延。使用BTB，就可以减少这种延迟到零周期。

BTB是针对分支指令的一个4路相关高速缓存。分支指令包括：中断返回，调用返回和计算类跳转指令。它有128个入口深度，并且采用替换近段时间最不常用的指令的法则进行更新。

BTB中的每个入口都有一个TAG和TAEGET段。TAG 段存储包含分支指令的指令行的四字地址。如果发生分支跳转，TARGET段保存程序控制器将要跳转到的地址。

BTB如何减少时钟周期

如果在程序中分支指令的条件满足，分支指令第一次执行，BTB将不为它分配入口。因此，将用TAG放入了到一个新的入口，TAG将是PC寄存器中的值。在这一点上，将花费两个周期作为开销。

然而，如果每次都运行到了相同的分支指令，则PC寄存器的值将与上次保存在TAG段中的值相匹配。程序控制器首先看是否有BTB命中，然后抓取保存在TARGET段中地址单元中的值，而不是抓取接下来的一条指令。因此，如果分支指令在结束时条件为真，并发生了分支跳转，则适当的指令已经通过流水线发送出去，此时发生分支执行的时钟周期开销就降低为零个时钟周期。

什么时候使用BTB有优势?

当不断的执行相同的分支指令时，而且每次控制分支指令的条件都为真，此时使用BTB将节省大部分时钟周期。例如，如果在循环中包含分支指令，并且总是执行分支指令，则在第一次执行循环时会有两个周期的延迟，随后的每次循环就是零周期延迟。与此相比，同样的程序代码，没有使用BTB，取决于分支判断条件是否依赖IALU或者计算块，在每次执行循环时将有3或者6个周期的延迟开销。

和前面一样，一遍一遍的不断循环运行程序，而分支指令执行的条件只是在第一次运行是满足，此时，使用BTB时情况最糟糕。如果第一次发生了分支，则随后的每次都会 对分支指令进行预测。被预测而又没有发生分支的指令仍然有3或者6个时钟周期的延迟开销。与没有使用分支预测的情况相比，第一次运行循环将有3或者6个时钟周期的延迟开销，但是其他的每次循环就是零开销的了。

总之，如果分支指令将被多次执行，就应当预测分支指令，如果大部分时间分支指令都不会

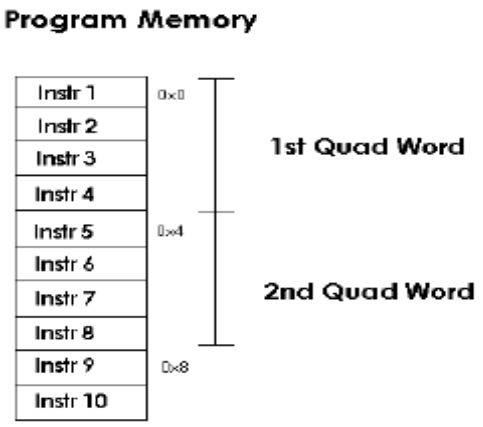
被执行，就可以不预测它。可以通过象如下所示指令形式所示在指令的后边加上后缀NP，强制的不预测分支指令。

If jeq, jump 0x0000 (NP);;

一个四组字中放入两个分支指令

使用分支目标缓冲时，为了防止误解，需要说明一点，应当注意在相同的一个四组字存储器空间中，不能放两条分支指令。否则，编译器如果认为出现了这种情况，将发出警告消息，“Detected two instruction lines with predicted jumps ending within 4 words”（在四字中检测到两个有预测跳转指令结尾的指令行）。

要理解如何避免这种情况，首先应理解指令是如何从内部存储器移到处理器核的。在每个周期，DSP都充分利用其128位总线，从内部存储器取四条指令。四条指令中的第一条总是一个四组字对齐的地址（即该地址是四的倍数）。



记住TigerSHARC可以从每个指令行中的四条指令的任意一条开始执行。因此，当四条指令在同一个周期载入处理器核时，该四条指令不必在同一个指令行中执行。要处理器这种情况，这些指令保存在一个名为指令分配缓冲区（IAB）的5个字的FIFO中。这里，他们按照其在源程序中的定义，对齐到适当的指令行中。比如，有如下的指令：

Instr1;;
Instr2; Instr3; Instr4; Instr5;;
Instr6; Instr7;;
Instr8;;

在第一个周期，从存储器中取前四组指令字，包括指令1，2，3和4。由于第一个指令行只包括一条指令Instr1，可以被执行。由于第二个指令行的执行需要Instr5，它必须等待下一个四组指令字从内部存储器中载入，才能执行。指令2，3，4和5对齐到IAB中的一个指令行中。接下来的两个指令行的指令已经从内部存储器载入，在下一个指令周期就可以执行。

要理解这一点，我们必要考察它如何影响BTB的操作。保存在BTB TAG段中的地址只能是四组字地址（0x0，0x4，0x8…），所以，如果分支指令出现在指令1，2，3，或者4中，则将被保存的TAG段就是四组字地址0x0。如果分支指令出现在指令5，6，7，或者8中，则将被保存的TAG段就是四组字地址0x4。行容易看到，在存储器的一个相同的四组字地址中，不能放置两个分支指令。如果指令Instr3和Instr4都是分支指令，它们将有相同的TAG值0x0，此时BTB将不能区分这两条指令。

另外需要理解的重要一点是，BTB TAG段存储了包含分支指令的指令行中的最后一条指令的四组字地址。比如，以下指令：

Instr1;;
Instr2; Instr3; Branch(Instr4); Instr5;;
Instr6;;

此处的分支指令（存储器图中的Instr4）在程序代码的第二个指令行中。但是即使Instr4在

地址为0x0的四组字中，这也不是保存在BTB TAG段中的地址。由于程序控制器必须载入在同一个指令行中执行的指令，因此Instr5也必须被载入。因为其四组字地址是0x4，它就是载入BTB TAG段中的地址。

现在，当编译器发出警告信息“Detected two instruction lines with predicted jumps ending within 4 words”，我们就可以充分的理解如何解决这个问题。指令行中包含分支指令的最后一条指令从含有分支指令的另外一个指令行开始，必须大于四个存储器地址。比如，有以下程序代码：

```
Branch(Instr2); Instr3; Instr4; Instr5;;
Instr6; Branch(Instr7);
```

这将引起编译器警告和BTB中的错误，因为这两个指令行均会试图载入0x4四组地址作为其TAG。以下代码结构的改变能解决这个问题。

```
Branch(Instr2); Instr3; Instr4; Instr5;;
Instr6; Branch(Instr7); Instr8; Instr9;;
```

现在，第一个指令行的TAG将是四组字地址0x4，第二个指令行的TAG将是四组地址0x8。不但解决了问题，而且也没有增加额外的指令行，但是增加了额外的指令周期开销。