计算机体系结构

周学海
xhzhou@ustc.edu.cn
0551-63492149
中国科学技术大学

Review

- **Tomasulo Algorithm 三阶段**
  1. **Issue—从FP操作队列中取指令**
     - 如果RS空闲(no structural hazard), 则控制发射指令和操作数(renames registers).
  2. **Execution—operate on operands (EX)**
     - 当两操作数就绪后，就可以执行
       如果没有准备好，则监测Common Data Bus 以获取结果
  3. **Write result—finish execution (WB)**
     - 将结果通过Common Data Bus传给所有等待该结果的部件;
       表示RS可用
- **基本数据结构**
  1. **Instruction Status**
  2. **Reservation Station**
  3. **Register Result Status**

Review

- **Reservations stations: 寄存器重命名，缓冲源操作数**
  - 避免寄存器成为瓶颈
  - 避免了Scoreboard中无法解决的 WAR, WAW hazards
  - 允许硬件做循环展开
- **不限于基本块(快速解决控制相关)**
- **贡献**
  - Dynamic scheduling （动态指令流调度）
  - Register renaming （寄存器重命名）
  - Load/store disambiguation **（存储器访问歧义消解）**
- **360/91 后 Pentium II; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264使用这种技术**

Review：Tomasulo算法实现循环覆盖执行?

- **寄存器重命名技术**
  - 不同的循环使用不同的物理寄存器 (dynamic loop unrolling).
  - 将代码中的静态寄存器名修改为动态寄存器指针“pointers”
  - 有效地增加了寄存器文件的大小
- **关键: 快速解决控制相关，以便能发射多个循环中的操作**

第5章 指令级并行

5.1 指令级并行的基本概念及静态指令流调度

ILP及挑战性问题

软件方法挖掘指令集并行

基本块内的指令集并行

5.2硬件方法挖掘指令级并行

5.2-1 指令流动态调度方法之一：Scoreboard

5.2-2 指令流动态调度方法之二：Tomasulo

5.3 分支预测方法：（教材C-27，3.3，3.9）

5.4 基于硬件的推测执行

5.5 存储器访问冲突消解及多发射技术

5.6 多线程技术

5.3 分支预测方法

控制相关对性能的影响

基于BHT的分支预测

1、基本2-bit预测器 （c-27)
2、关联预测器（两级预测器）(3.3)
3、组合预测器 (3.3)

基于BTB的分支预测

1、分支目标缓冲区(3.9)
2、Return Address预测器(3.9)

?

- **动态指令流调度硬件方案可以用硬件进行循环展开**
- **如何处理精确中断?**
  - Out-of-order execution -> out-of-order completion!
- **如何处理分支?**
  - **用硬件做循环展开必须快速解决控制相关问题**

关于异常处理???

- **乱序完成加大了实现精确异常的难度**
  - 在前面指令还没有完成时，寄存器文件中可能有后面指令的运行结果.
  - 如果这些前面的指令执行时有异常产生，怎么办?
  - 例如：DIVD F10, F0, F2
    SUBD F4, F6, F8
    ADDD F12, F14, F16
- **需要“rollback” 寄存器文件到原来的状态:**
  - 精确异常的含义:
    - 该地址之前的所有指令都已完成
    - 其后的指令还都没有完成
- **实现精确异常的技术：顺序完成（或提交）**
  - 即提交指令完成的顺序必须与指令发射的顺序相同

进行循环重叠执行需要尽快解决分支问题!

- **在循环展开的例子中，我们假设整数部件可以快速解决分支问题，以便进行循环重叠执行!**

```
Loop:   LD      F0      0       R1
        MULTD   F4      F0      F2
        SD      F4      0       R1
        SUBI    R1      R1      #8
        BNEZ    R1      Loop
```

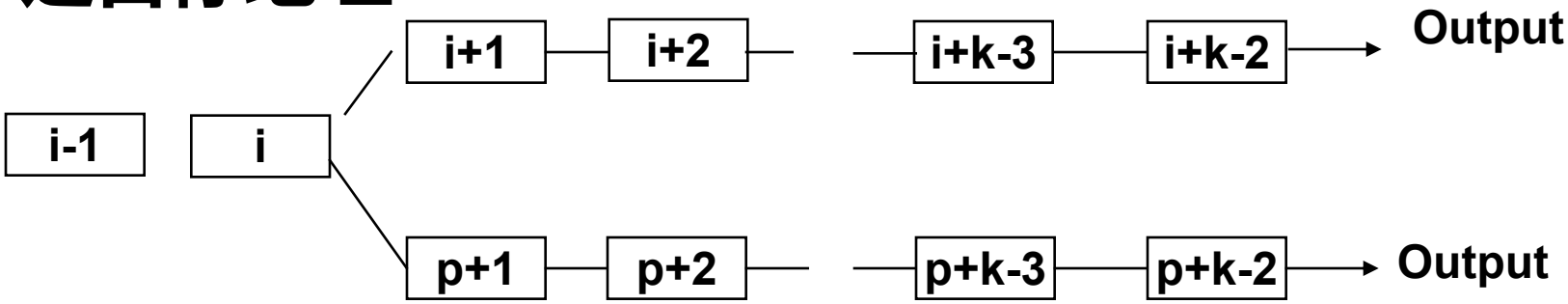
- **如果分支与其他指令有依赖关系,怎么办??**
  - 需要能预测分支方向
  - 如果分支成功，我们就可以重叠执行循环
- **对于superscalar机器这一问题更加突出**

控制相关的动态解决技术

- **控制相关：**
  - 由条件转移或程序中断引起的相关，也称全局相关。
  - 控制相关对流水线的吞吐率和效率影响相对于数据相关要大得多
    - 条件指令在一般程序中所占的比例相当大
    - 中断虽然在程序中所占的比例不大，但中断发生在程序中的哪条指令，发生在一条指令执行过程中的哪个功能段都是不确定的
- **处理条件转移和异常引起的控制相关的关键问题：**
  - 要确保流水线能够正常工作
  - 减少因断流引起的吞吐率和效率的下降

分支对性能的影响

- **假设在一条有K段的流水线中，在最后一段才能确定目标地址**



- **当分支方向预测错误时**
  - 流水线中有多个功能段要浪费
  - 可能造成程序执行结果发生错误
  - 因此当程序沿着错误方向运行后，作废这些程序时，一定不能破坏通用寄存器和主存储器的内容。

条件转移指令对流水线性能的影响

- **假设对于一条有K段的流水线，由于条件分支的影响，在最坏情况下，每次分支“跳转”将造成k-1个时钟周期的断流。假设条件分支在一般程序中所占的比例为p，采用静态分支预测“不跳转”策略，条件“跳转”的概率为q。试分析分支对流水线的影响。**
- **结论：条件转移指令对流水线的影响很大，必须采取相关措施来减少这种影响。**

- **预测可以是静态预测“Static” (at compile time) 或动态预测 “Dynamic” (at runtime)**
  - 例如：一个循环供循环10次，它将分支成功9次，1次不成功。
  - 动态分支预测 vs. 静态分支预测，哪个好?

5.3 分支预测方法

控制相关对性能的影响

基于BHT的分支预测

1、基本2-bit预测器
2、关联预测器（两级预测器）
3、组合预测器

基于BTB的分支预测

1、分支目标缓冲区
2、Return Address预测器

分支预测概览

- **分支预测对提高性能是非常重要的**
  - 分支预测在哪个阶段完成?
  - 预测器设计的核心问题是什么?
  - 预测器的基本结构及输入输出?
- **预测器的分类：基于BHT表的预测器 和 优化取指令的带宽**
- **基于BHT表的预测器:**
  - 基本的2-bit预测器 （饱和预测器）
  - 关联预测器（Correlating predictor）or 2级预测器:
    - GAp (Global History table and per-address predictor table)
      - 每条分支有多个2-bit预测器
      - 由最近的n次分支的结果来选择对应的2-bits预测器
    - PAp (Per-address history table and per-address predictor table)
      - 每条分支有多个2-bit预测器
      - 由**该分支**最近的n次分支结果来选择对应的2-bits预测器
  - Tournament predictor（竞赛预测器）：组合GAp和PAp
- **优化取指令的带宽**
  - Branch Target Buffer
  - Return Address Predicator

Instruction Fetch Unit

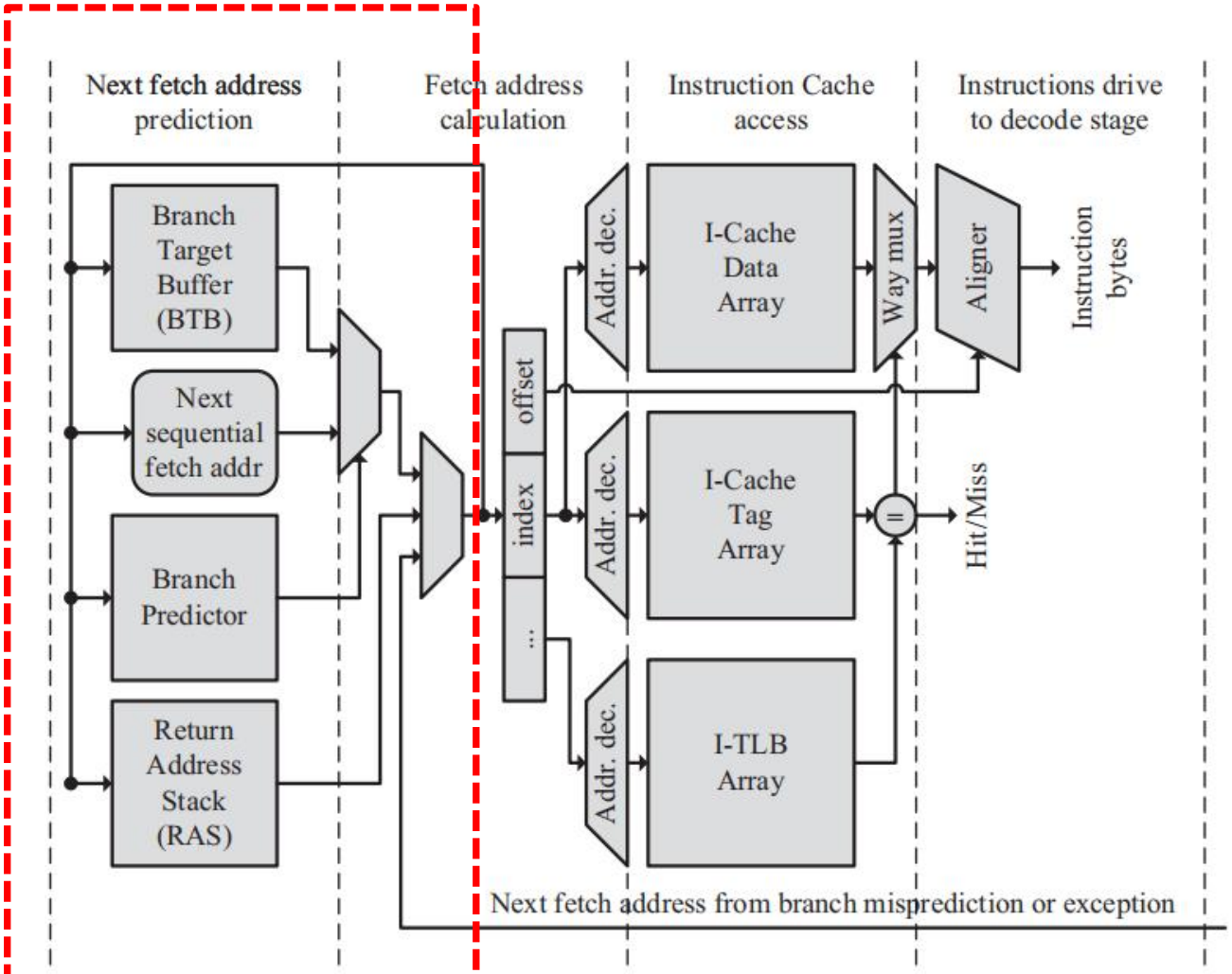
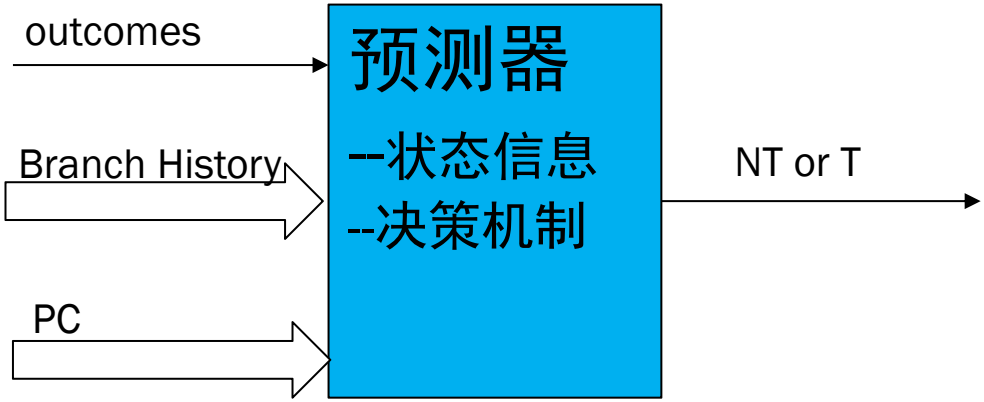


FIGURE 3.1: Example fetch pipeline.

分支预测在哪个阶段完成?

预测器的基本结构及输入输出

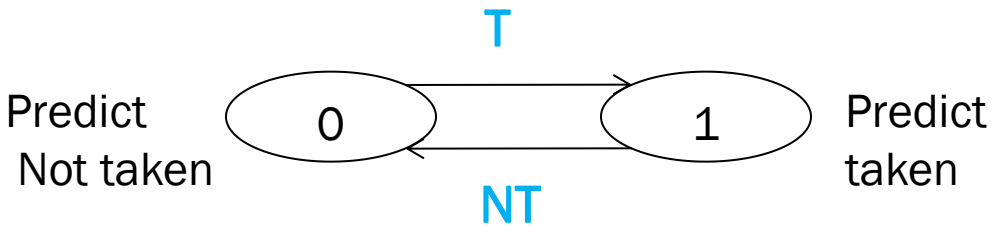


- **根据转移历史(和PC)来选择预测器**
- **由预测器的状态决定预测值(输出)**
- **根据实际结果(outcomes)更新预测器的状态信息**

Dynamic Branch Prediction

- **动态分支预测：预测分支的方向在程序运行时刻动态确定**
- **需解决的关键问题是：**
  - 如何记录转移历史信息
  - 如何根据所记录的转移历史信息，预测转移的方向（跳转或不跳转）
- **主要方法**
  - 基于BPB(Branch Prediction Buffer)或BHT(Branch History Table)
    - 1-bit BHT和2-bit BHT
    - Correlating Branch Predictors （GAp or PAp）
    - Tournament Predictors: Adaptively Combining Local and Global Predictors
  - High Performance Instruction Delivery（优化取指令带宽）
    - BTB
    - Return Address Predictors
    - Integrated Instruction Fetch Units （单独的取指部件连接到流水线的其他部分，其中集成了分支预测器、指令预取、指令Cache的存取和缓存等）
- **Performance = *ƒ*(accuracy, cost of misprediction)**
  - Misprediction　　Flush Reorder Buffer

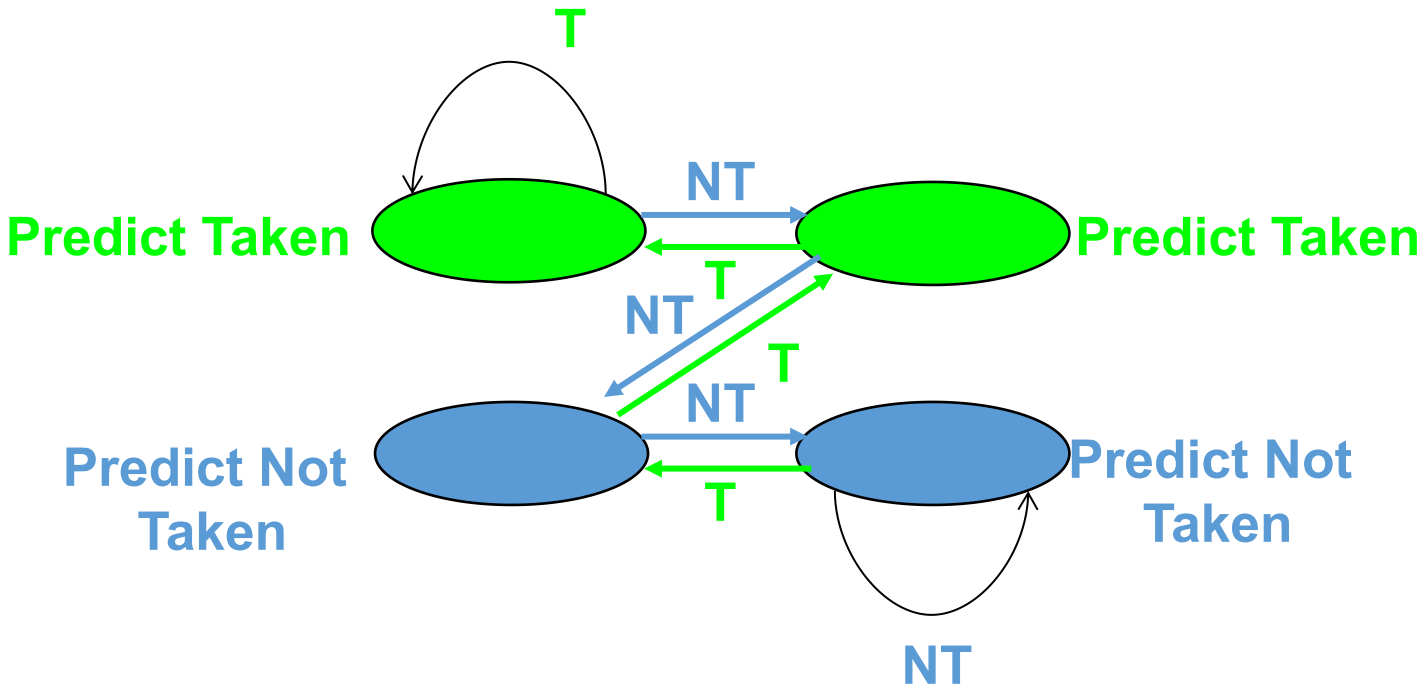
1-bit BHT



- **术语:**
  - Not taken | taken 跳转|不跳转 （成功|失败）
  - 预测准确率（Accuracy），预测错误率(Misprediction)
- **Branch History Table:**
  - 分支指令的PC的低位索引
  - 该表记录上一次转移是否成功
  - 不做地址检查
  - 1-bit BHT

- **问题: 在一个循环中, 1-bit BHT 将导致2次分支预测错误**
  - 假设一循环次数为10次的简单程序段
  - 最后一次循环： 前面预测“跳转”，最后一次需要退出循环
  - 首次循环：前面预测为“不跳转”，这次实际上为成功

2-bit BHT



- **解决办法: 2位记录分支历史**
- **Blue: stop, not taken (不跳转)**
- **Green: go, taken （跳转）**

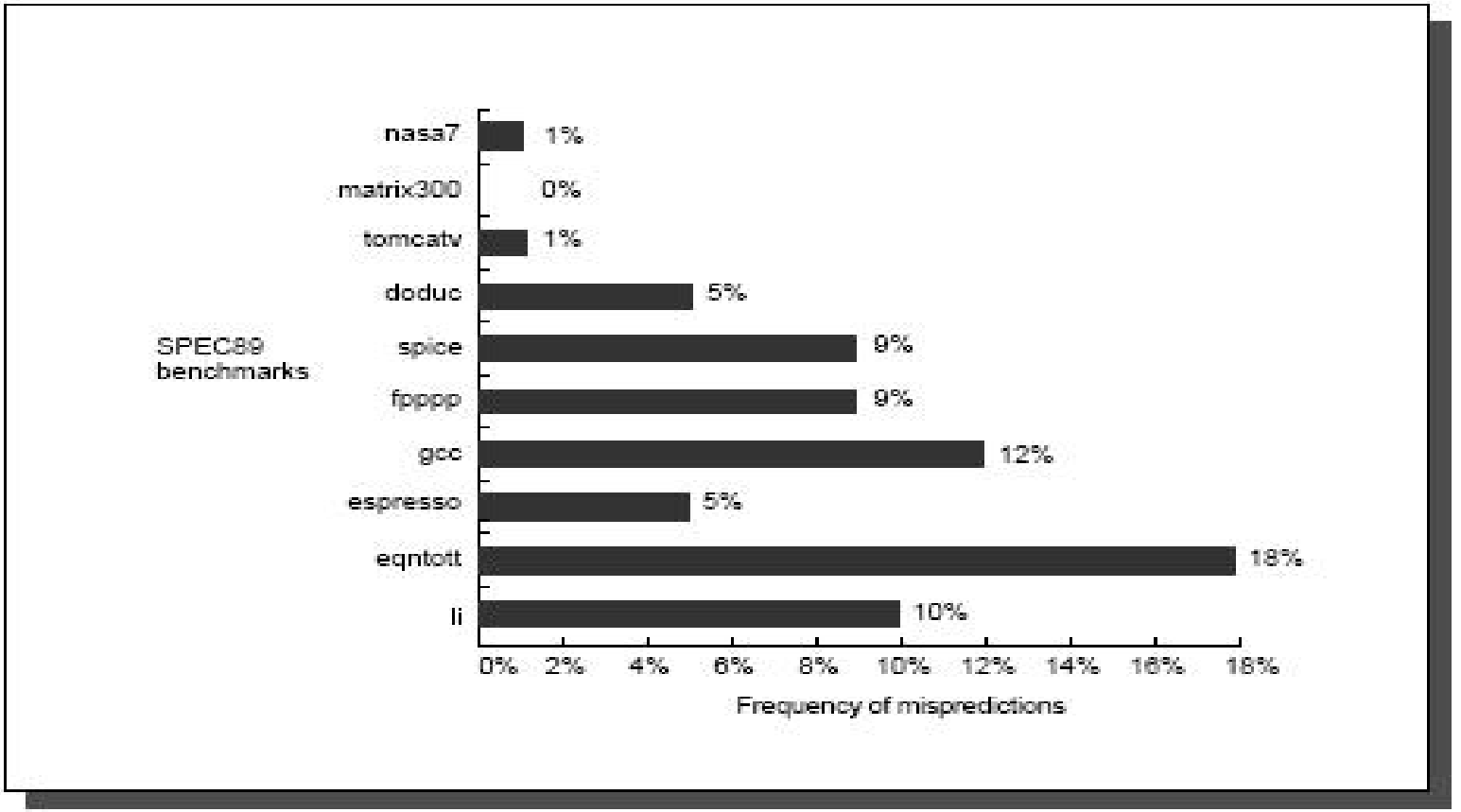


FIGURE 3.8 Prediction accuracy of a 4096-entry two-bit prediction buffer for the SPEC89 benchmarks. The misprediction rate for the integer benchmarks (gcc, espresso, eqntott, and li) is substantially higher (average of 11%) than that for the FP programs (average of 4%). Even omitting the FP kernels (nasa7, matrix300, and tomcatv) still yields a higher accuracy for the FP benchmarks than for the integer benchmarks. These data, as well as the rest of the data in this section, are taken from a branch prediction study done using the IBM Power architecture and optimized code for that system. See Pan et al. [1992].

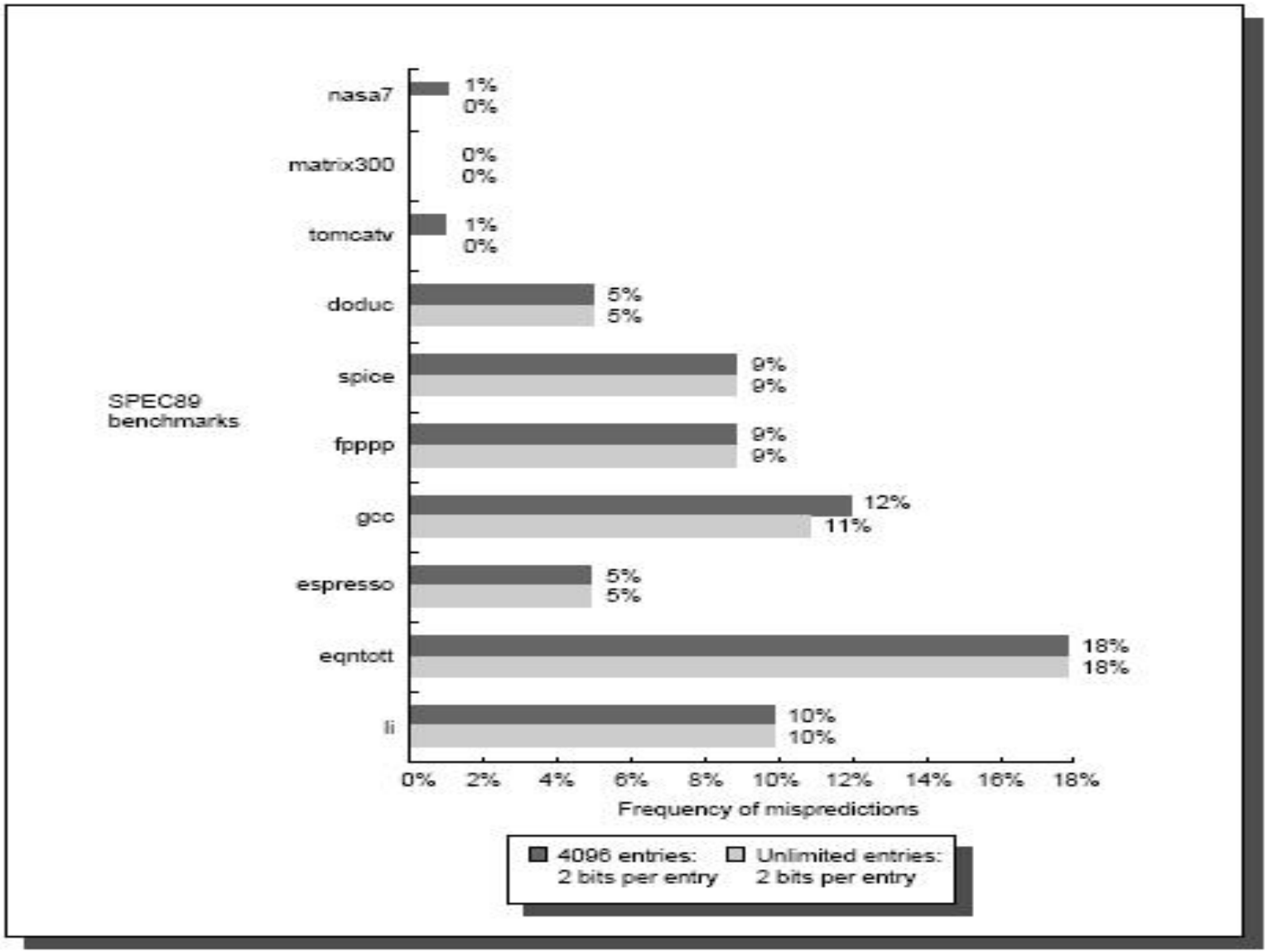


FIGURE 3.9 Prediction accuracy of a 4096-entry two-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks.

BHT Accuracy

- **分支预测错误的原因:**
  - 预测错误
  - 由于使用PC的低位查找BHT表，可能得到错误的分支历史记录
- **BHT表的大小问题**
  - 4096 项的表分支预测错误的比例为1%（nasa7, tomcatv) to 18% (eqntott)，spice at 9% and gcc at 12%
  - 再增加项数，对提高预测准确率几乎没有效果 (in Alpha 21164)

Correlating Branch Predicator

例如：

```
if (aa==2)   aa=0;
if (bb==2)  bb=0;
if (aa!=bb) {
```

翻译为汇编指令

```
      SUBI R3,R1,#2
      BNEZ R3,L1          ; branch b1 (aa!=2)
      ADDI  R1,R0,R0      ;aa=0
L1:   SUBI R3,R2,#2
      BNEZ R3,L2          ;branch b2(bb!=2)
      ADDI R2,R0,R0       ; bb=0
L2:   SUBI R3,R1,R2       ;R3=aa-bb
      BEQZ R3,L3          ;branch b3 (aa==bb)
```

观察结果：

- b3 与分支b2 和b1相关。
- 如果b1和b2都分支“不跳转”，则b3一定成功。即：

(aa ==2; bb == 2，则 aa == bb)

分支间存在关联

- Correlating predictors 或 两级预测器：
  - 设计两级预测器，考虑“其他”分支行为
  - “其他”：全局或局部
- 工作原理：
  - 根据一个简单的例子来看其基本原理

存在关联的分支

```
if (d==0)  d=1;
if (d==1)  d=0;

BNEZ R1,L1          ;branch b1(d!=0)
ADDI R1,R0,#1       ;d==0, so d=1
L1:  ADDI R3,R1,#-1
     BNEZ R3,L2     ;branch  b2(d!=1)
          …
L2:
```

分支间存在关联的情况举例(1/2)

- 假设d的初始值序列为0，1，2
- b1 如果分支“不跳转”，b2一定也分支“不跳转”。
- 前面基本的1-bit 2-bit预测器都没法利用这一点

→ 两级预测器

```
 if (d==0)d=1;

  if (d==1) d=0;
翻译为汇编指令
        BNEZ R1,L1      ;branch b1(d!=0)
        ADDI R1,R0,#1     ;d==0, so d=1
L1:  ADDI R3,R1,#-1
        BNEZ R3,L2      ;branch b2(d!=1)
```

Initial value of d	d==0?	b1	Value of d before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
2	no	taken	2	no	taken

FIGURE 3.10 Possible execution sequences for a code fragment.

分支间存在关联的情况举例（2/2）

- **假设d的初始值在2和0之间切换。T:“跳转”，NT:“不跳转”**
- **用1-bit预测器，b1和b2的初始设置为预测NT**

```
        BNEZ R1,L1          ;branch b1(d!=0)
        ADDI R1,R0,#1       ;d==0, so d=1
L1:     ADDI R3,R1,#-1
        BNEZ R3,L2          ;branch b2(d!=1)
```

d=?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

FIGURE 3.11 Behavior of a one-bit predictor initialized to not taken. T stands for taken, NT for not taken.

- **结论：这样的序列每次预测都错，预测错误率100%。**
- **问题出在哪里？有办法改善吗？**

Correlating Branches

- **基本思想：记为 （1，1）**
  - 用1位作为correlation位。记录最近一次执行的分支
  - 每个分支都有两个相互独立的预测位：一个预测位假设最近一次执行的分支”不跳转“时的预测位，另一个预测位是假设最近一次执行的分支”跳转“时的预测位。
- **最近一次执行的分支与要预测的分支可能不是同一条指令**

Prediction bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	not taken	not taken
NT/T	not taken	taken
T/NT	taken	not taken
T/T	taken	taken

FIGURE 3.12 Combinations and meaning of the taken/not taken prediction bits. T stands for taken, NT for not taken.

Correlating Branches

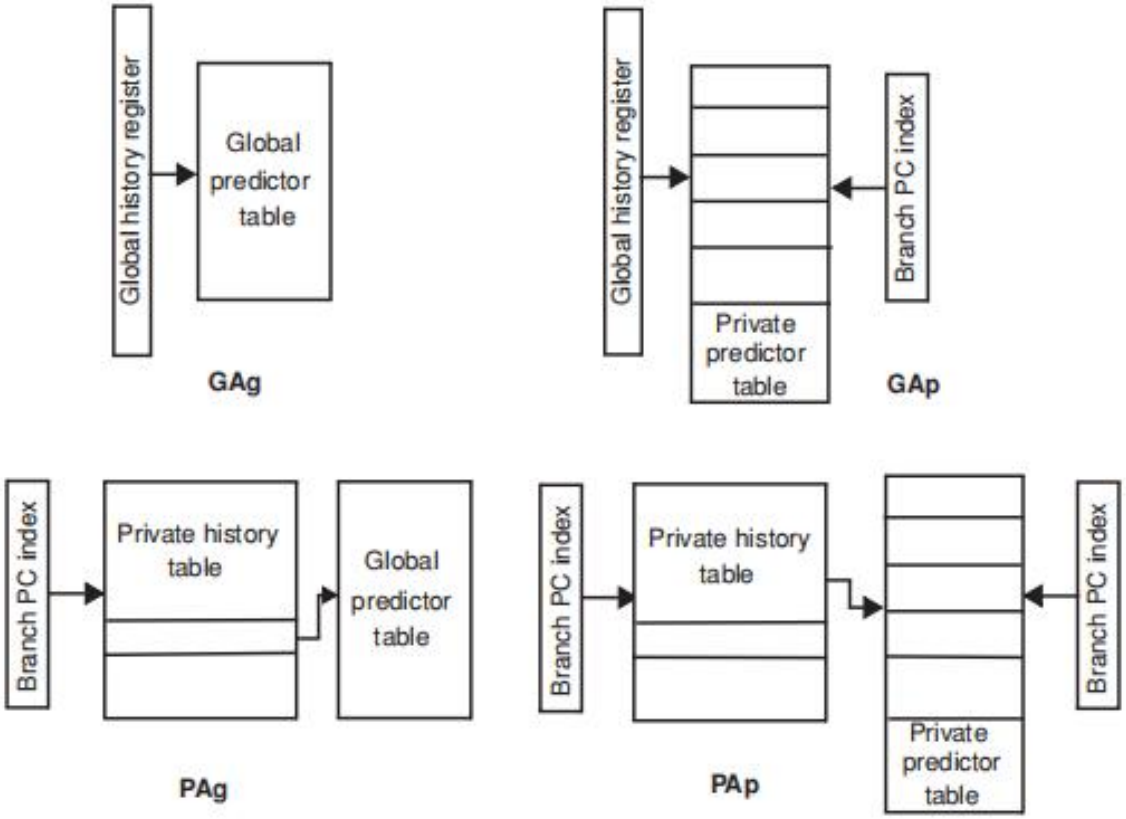
- **Correlating 预测器的预测和执行情况**
- **显然只有在第一次d=2时，预测错误，其他都预测正确**
- **记为（1，1）预测器，即根据最近一次分支行为来选择一对1-bit预测器中的一个。**
- **更一般的表示为（m, n)，即根据最近的m个分支，从2^m个分支预测器中选择预测器，每个预测器的位数为n**

```
        BNEZ R1,L1          ;branch b1(d!=0)
        ADDI R1,R0,#1       ;d==0, so d=1
L1:     ADDI R3,R1,#-1
        BNEZ R3,L2          ;branch b2(d!=1)
```

d=?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	**NT**/NT	T	T/NT	**NT**/NT	T	NT/T
0	**T**/NT	NT	T/NT	**NT**/T	NT	NT/T
2	**T**/NT	T	T/NT	**NT**/T	T	NT/T
0	**T**/NT	NT	T/NT	**NT**/T	NT	NT/T

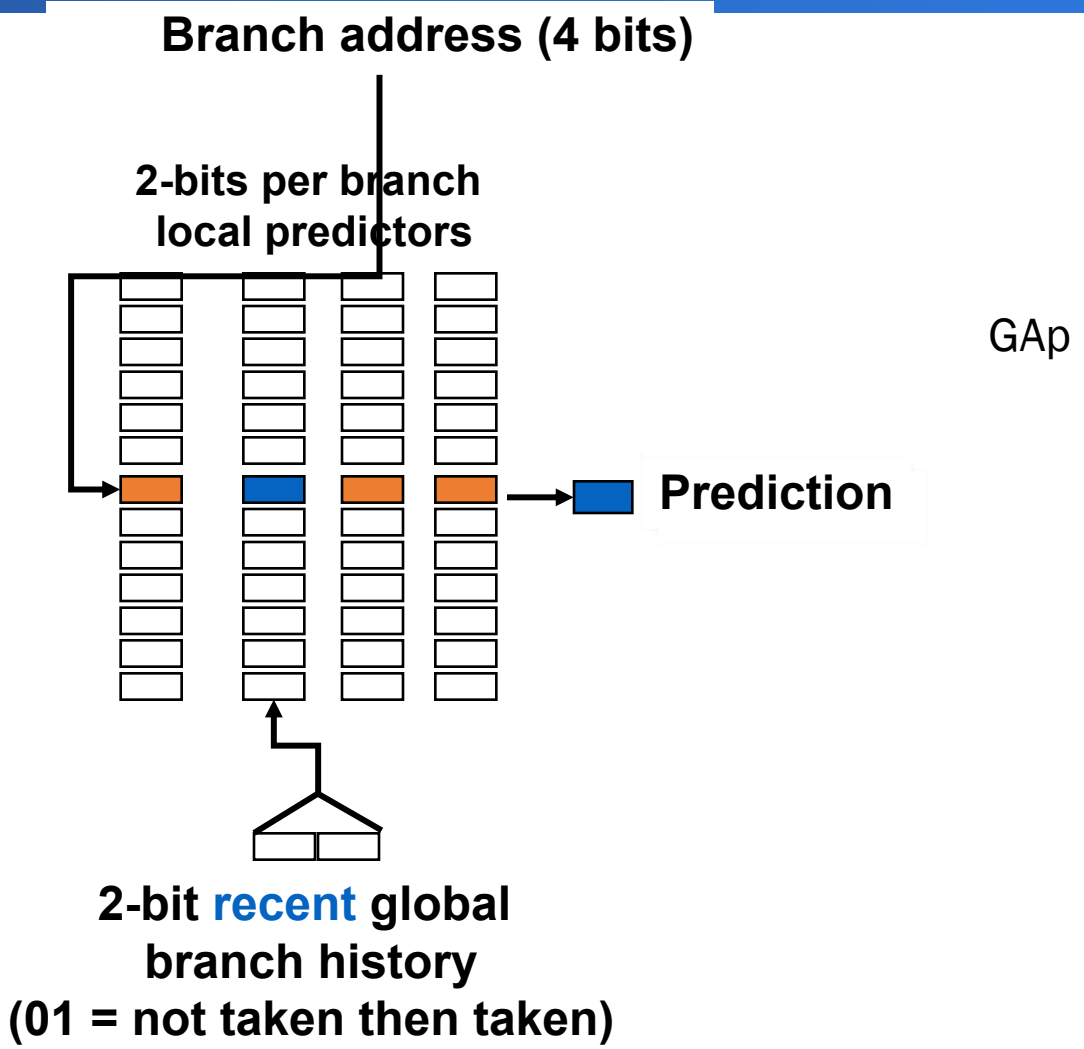
FIGURE 3.13 The action of the one-bit predictor with one bit of correlation, initialized to not taken/not taken. T stands for taken, NT for not taken. The prediction used is shown in bold.

两级预测器（枚举）



- **关联预测器 也称为 两级预测器**
- **两级预测器的四种组合：** 依据两级的全局或局部属性
  - ①GAg：全局历史表和全局预测表；②GAp：全局历史表和单地址预测表
  - ③PAg：单地址历史表和全局预测器表；④PAp: 单地址历史表和单地址预测器表

两级全局预测器（GAp）



- (2,2) predictor: 2-bit global, 2-bit local

Gshare predictor

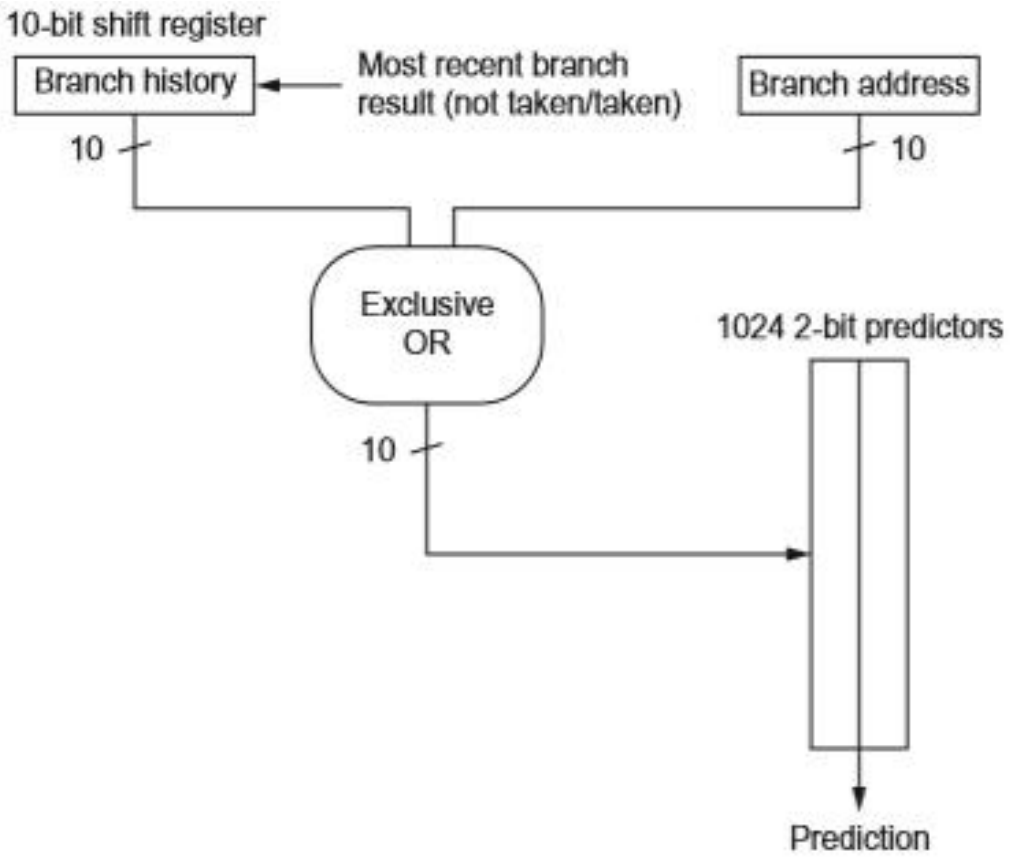
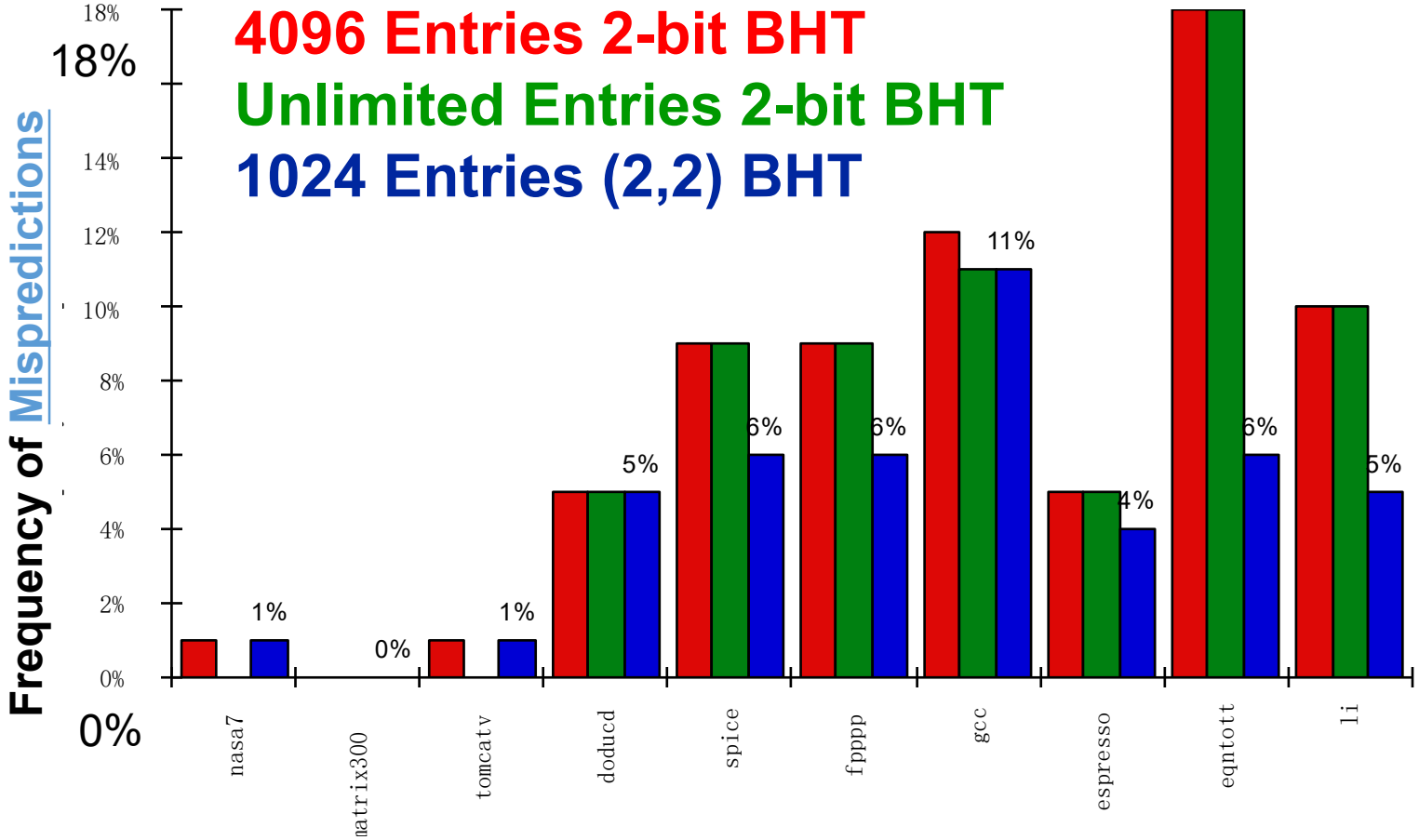


Figure 3.4 A gshare predictor with 1024 entries, each being a standard 2-bit predictor.

带有全局分支历史的分支预测器Gshare（一种GAp的实现）

Accuracy of Different Schemes



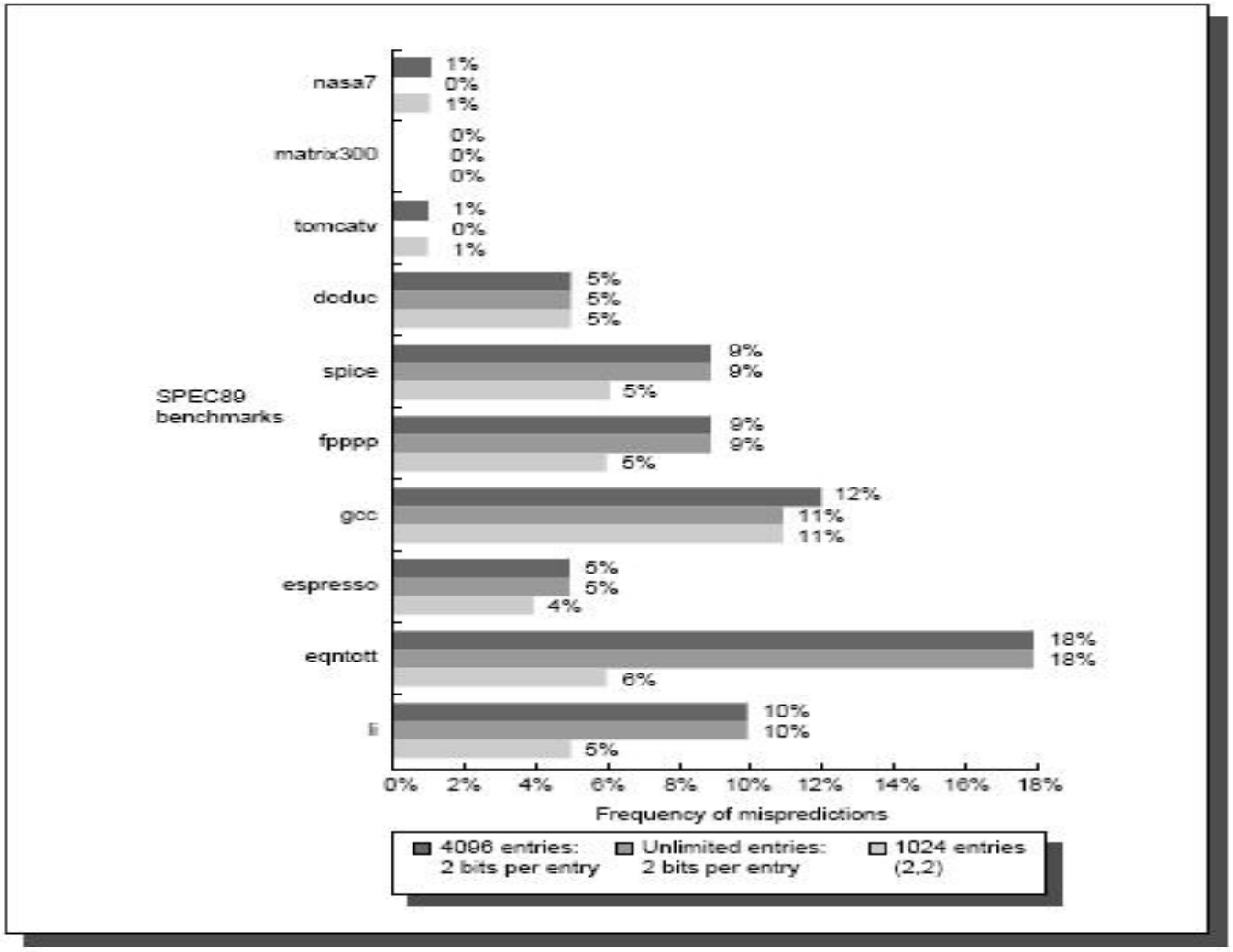


FIGURE 3.15 Comparison of two-bit predictors. A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating two-bit predictor with unlimited entries and a two-bit predictor with two bits of global history and a total of 1024 entries.

Branch Prediction

- **Basic 2-bit predictor:**
- **关联预测器(n,2):**
  - 两级全局预测器（GAp）
    - 每个分支有多个 2-bit 预测器
    - 根据**最近n次分支**的执行情况从2^n中选择预测器
  - 两级局部预测器(Local predictor) PAp
    - 每个分支有多个2-bit 预测器
    - 根据**该分支的最近n次分支**的执行情况从2^n中选择预测器
- **竞赛（组合）预测器(Tournament predictor):**
  - 例如：结合两级全局预测器和两级局部预测器

竞赛（组合）预测器

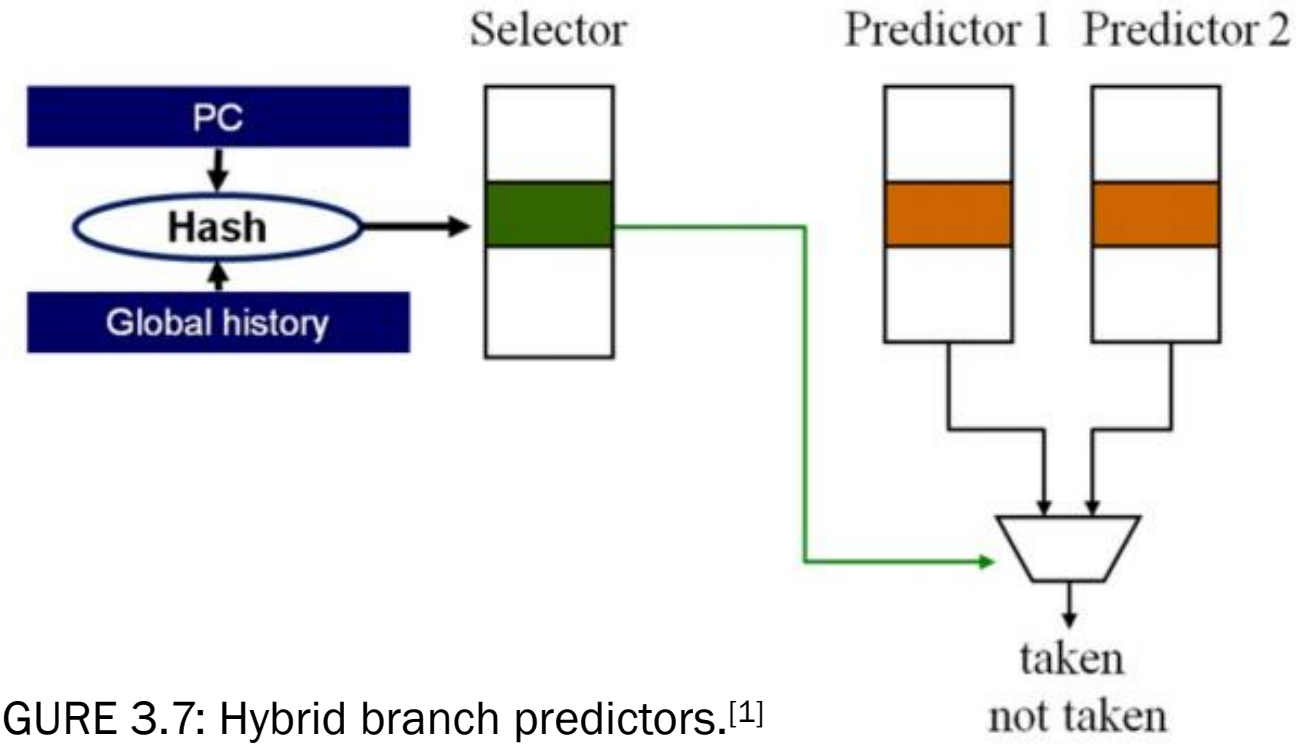
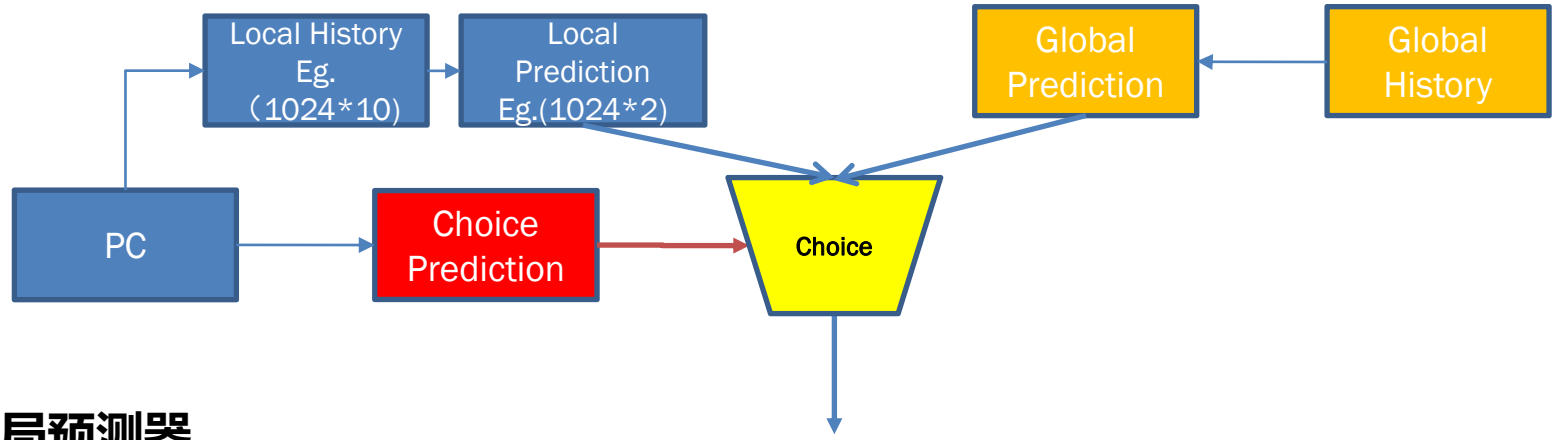


FIGURE 3.7: Hybrid branch predictors.[1]

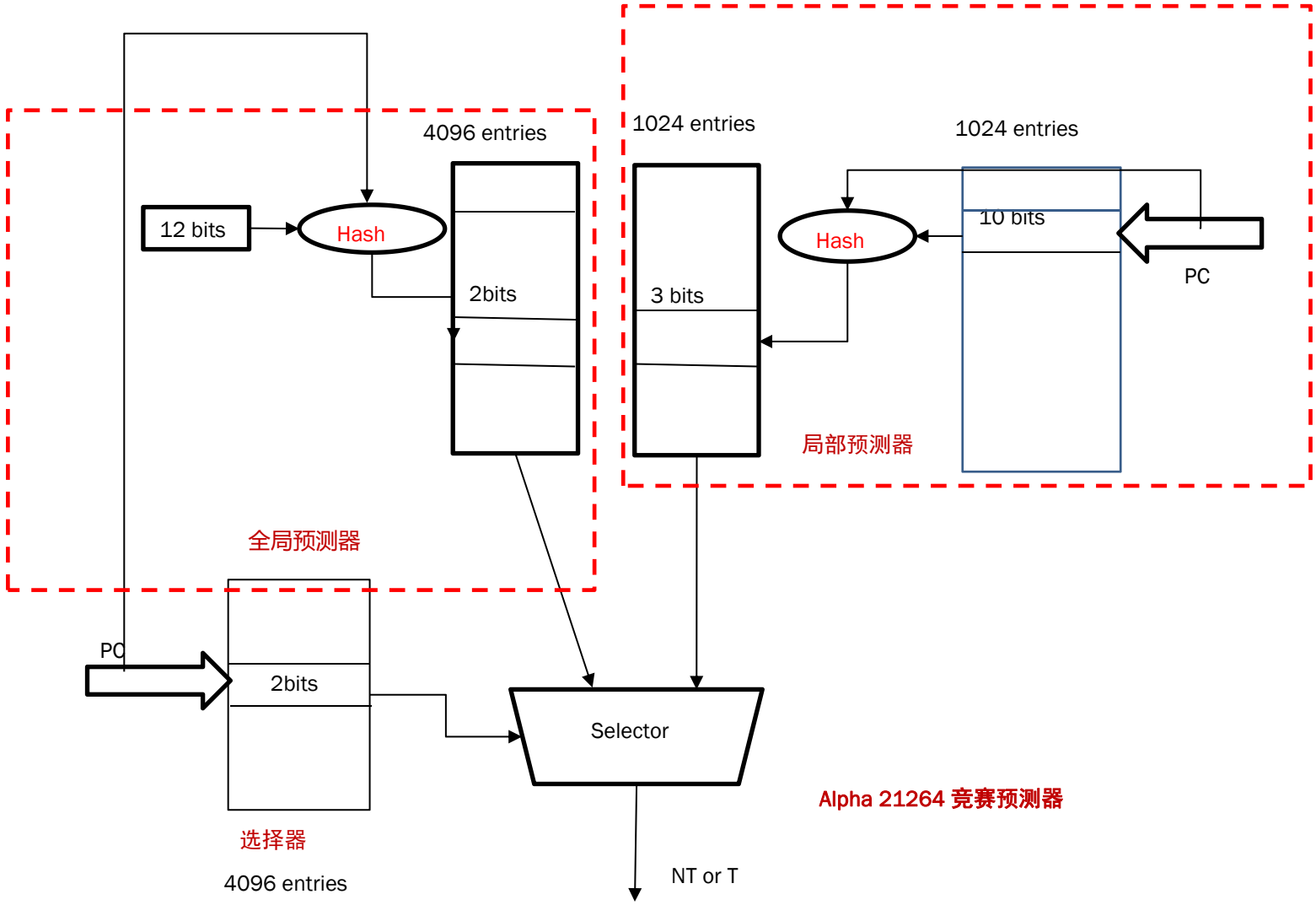
- **[1] A González，Latorre F，Magklis G . Processor Microarchitecture: An Implementation Perspective[J]. Synthesis Lectures on Computer Architecture, 2010, 5(1).**
- **Tournament predictor 也称 Hybrid branch predictors**
- **例如：两级局部预测器（预热时间较短）与全局预测器（预热时间较长）的组合**

竞赛预测器（举例）

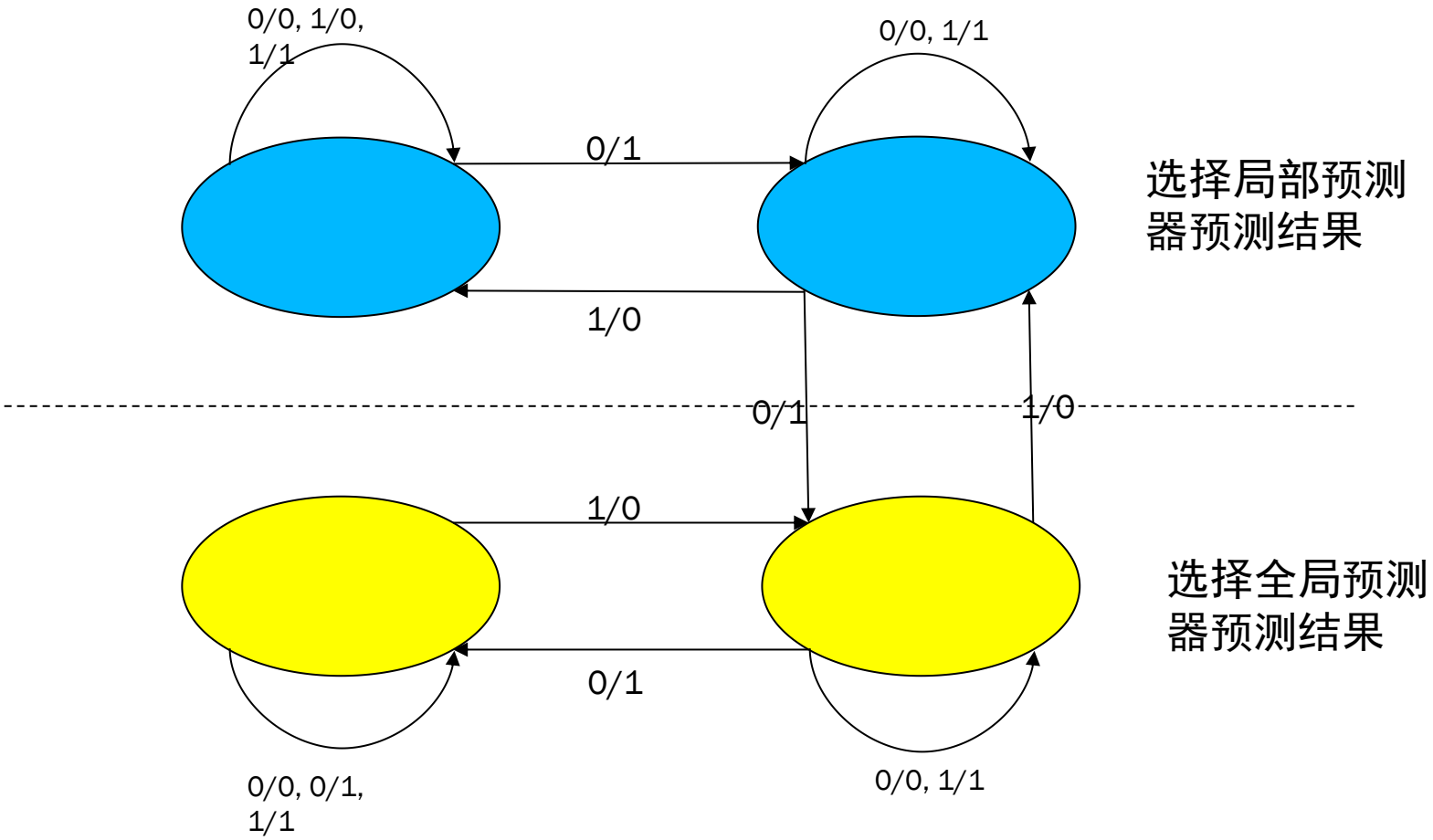


- **全局预测器**
  - 使用最近n次分支跳转情况来索引（2^n个entries），每个Entry是一个标准的2位预测器
- **两级局部预测器**
  - 一个局部历史记录表（Local History）：使用PC的低m位索引（2^m个entries），每个entry有k位，记录该指令最近的k次分支跳转情况
  - 根据Local History选择的entry的k位，索引选择下一级（Local Prediction）的entries，这些entries由2位计数器构成，以提供本地预测。
- **选择器：**
  - 使用PC低m位索引，每个索引得到一个两位计数器，用来选择使用局部预测器还是使用全局预测器的预测结果。
  - **在设计时默认使用局部预测器，当两个预测器都正确或都不正确时，不改变计数器；当全局预测器正确而局部预测器预测错误时，计数器加1，否则减1。**

Alpha 21264

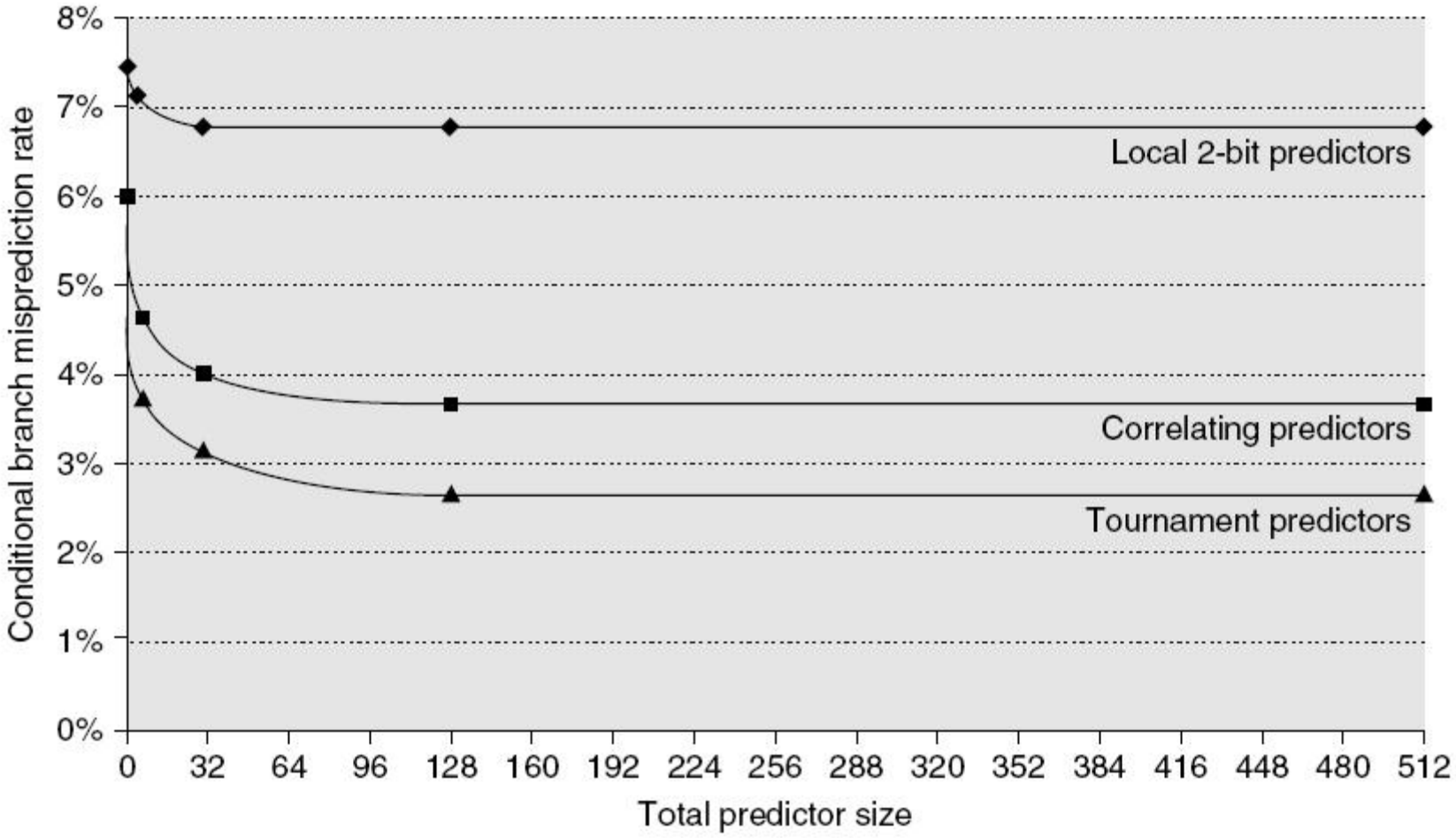

4096 entries
12 bits
Hash
2bits
全局预测器
1024 entries
3 bits
1024 entries
Hash
10 bits
PC
局部预测器
PC
2bits
Selector
选择器
4096 entries
NT or T
Alpha 21264 竞赛预测器

选择器状态转移图



- **局部预测器/全局预测器：**
  - 0/1：局部预测器预测错误，全局预测器预测正确
  - 1/0：局部预测器预测正确，全局预测器预测错误

Branch Prediction Performance



Branch predictor performance

5.4 分支预测技术

控制相关对性能的影响

基于BHT的分支预测

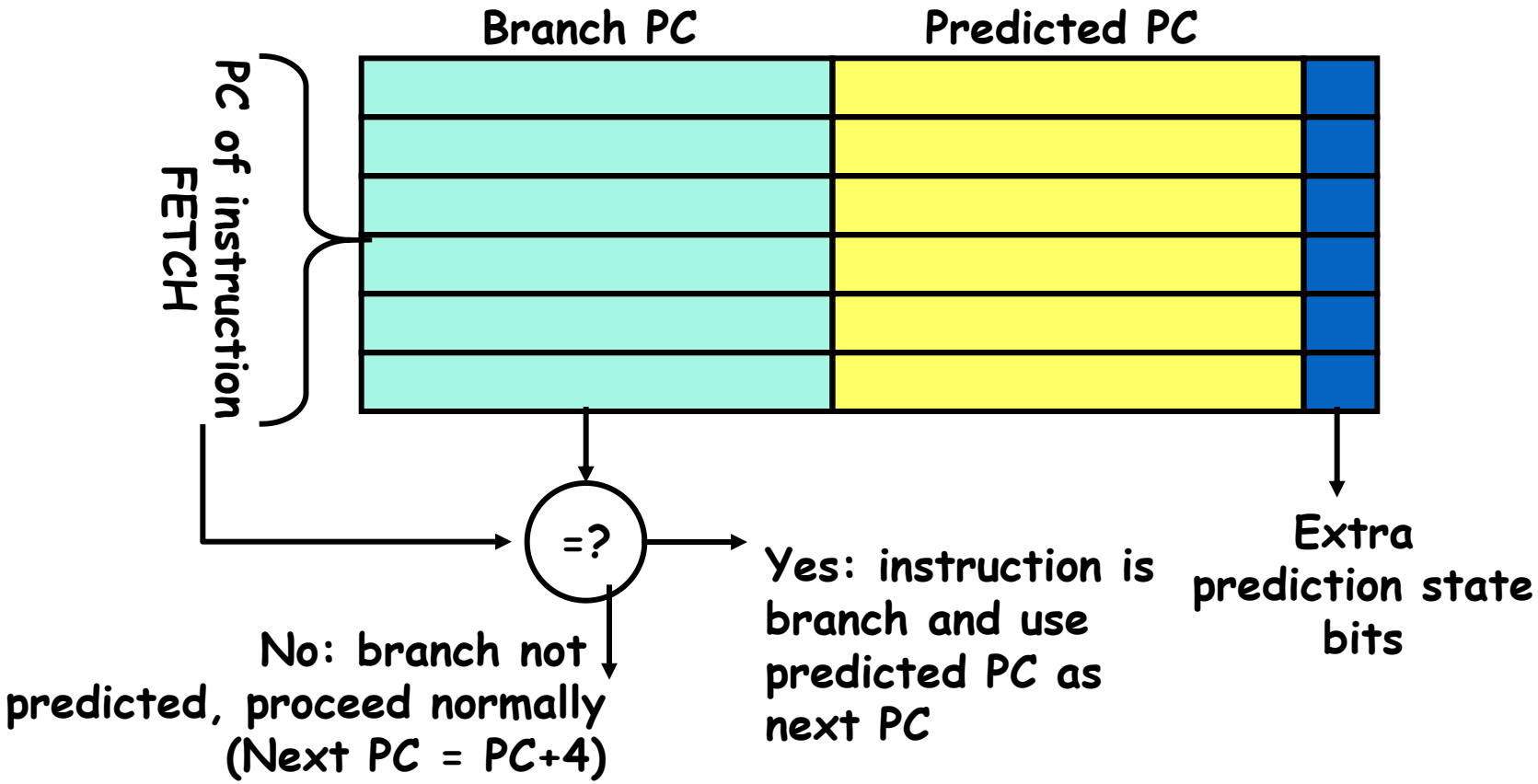
基于BTB的分支预测

1、基本2-bit预测器
2、关联预测器（两级预测器）
3、组合预测器

1、分支目标缓冲区
2、Return Address预测器

Branch Target Buffer (BTB)

- **分支指令的地址作为BTB的索引，以得到分支预测地址**
  - 必须检测分支指令的地址是否匹配，以免用错误的分支地址
  - 从表中得到预测地址
  - 分支方向确定后，更新预测的PC



BTB的换入换出

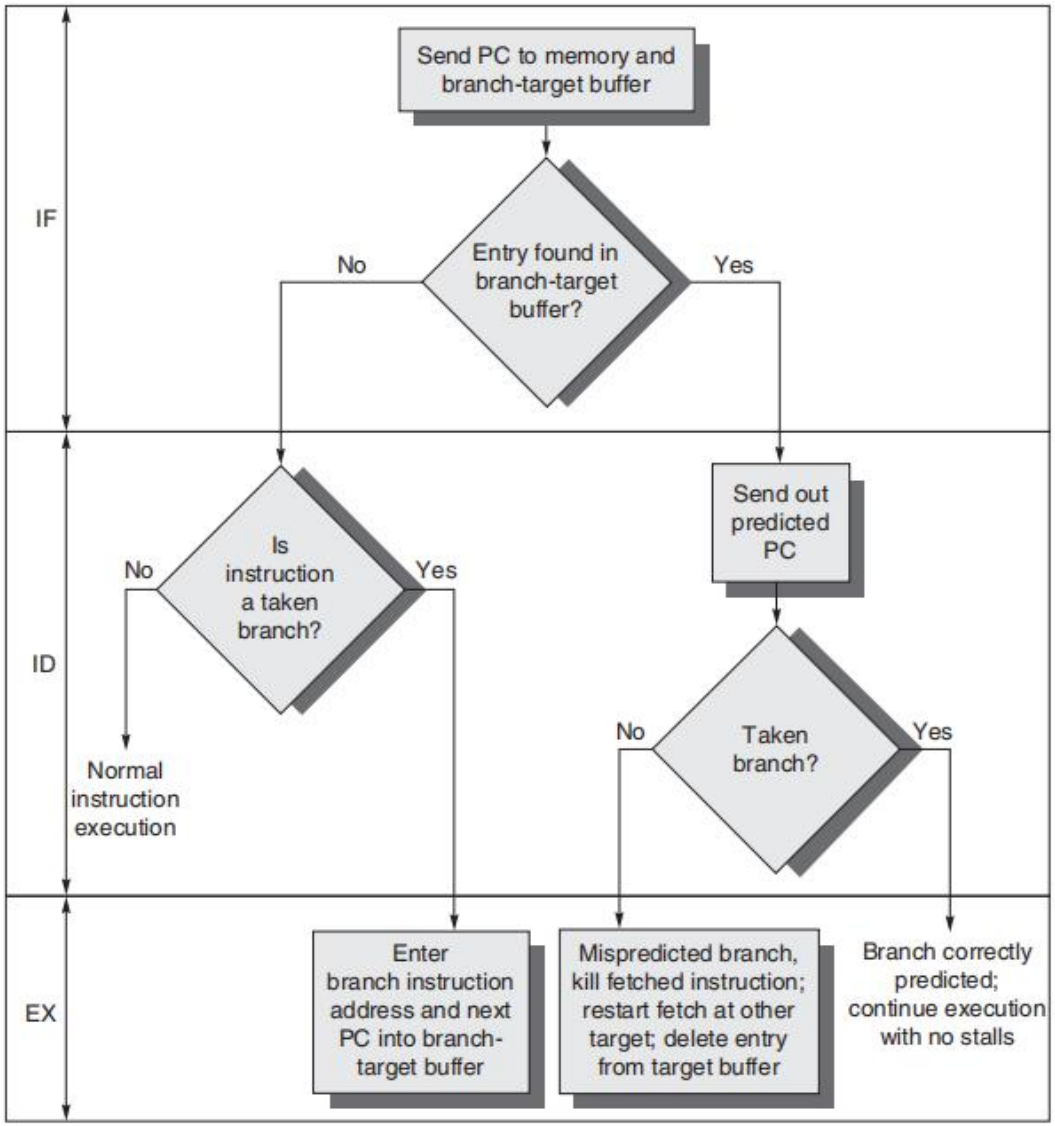


Figure 3.22 The steps involved in handling an instruction with a branch-target buffer.

例如：基本模型

- 简单的五段流水
- ID段 确认 是否 可以 跳转

- BTB预测器 分支目标缓存的换入换出

举例：性能分析

Instruction in buffer	Prediction	Actual branch	Penalty cycles
yes	taken	taken	0
yes	taken	not taken	2
no		taken	2
no		not taken	0

Figure 2.24 Penalties for all possible combinations of whether the branch is in the buffer and what it actually does, assuming we store only taken branches in the buffer. There is no branch penalty if everything is correctly predicted and the branch is found in the target buffer. If the branch is not correctly predicted, the penalty is equal to 1 clock cycle to update the buffer with the correct information (during which an instruction cannot be fetched) and 1 clock cycle, if needed, to restart fetching the next correct instruction for the branch. If the branch is not found and taken, a 2-cycle penalty is encountered, during which time the buffer is updated.

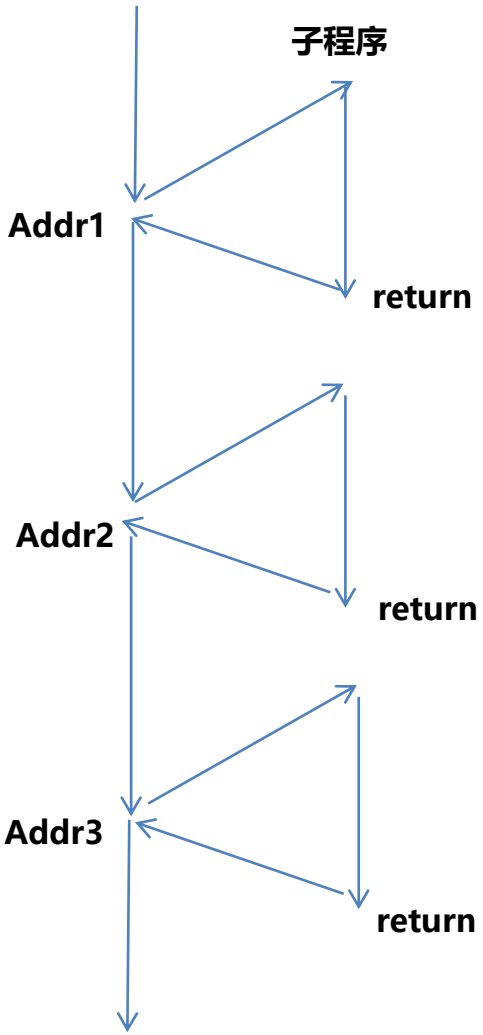
Branch Penalty：如果在BTB中命中，并且预测正确，则Penalty 为0，其他情况则Penalty为2

Determine the total branch penalty for a branch-target buffer assuming the penalty cycles for individual mispredictions from Figure 2.24. Make the following assumptions about the prediction accuracy and hit rate:

- Prediction accuracy is 90% (for instructions in the buffer).
- Hit rate in the buffer is 90% (for branches predicted taken).

Return Address Predictors

- **投机执行面临的挑战：预测间接跳转**
  - 运行时才能确定分支目标地址
- **多数间接跳转来源于Procedure Return**
  - 采用BTB时，过程返回的预测精度较低
    - 如果采用BTB，BTB中存放的信息包括：return指令本身的地址（PC）## 返回地址
    - 函数在不同位置调用，return指令本身的地址不变，但返回地址不同
  - SPEC CPU95测试，这类分支预测的准确性不到60%
- **使用一个小的缓存(栈）存放 Return Address**
  - 过程调用时将返回地址压入该栈
  - 过程返回时通过弹栈操作获得转移地址



Return Address Buffer entries

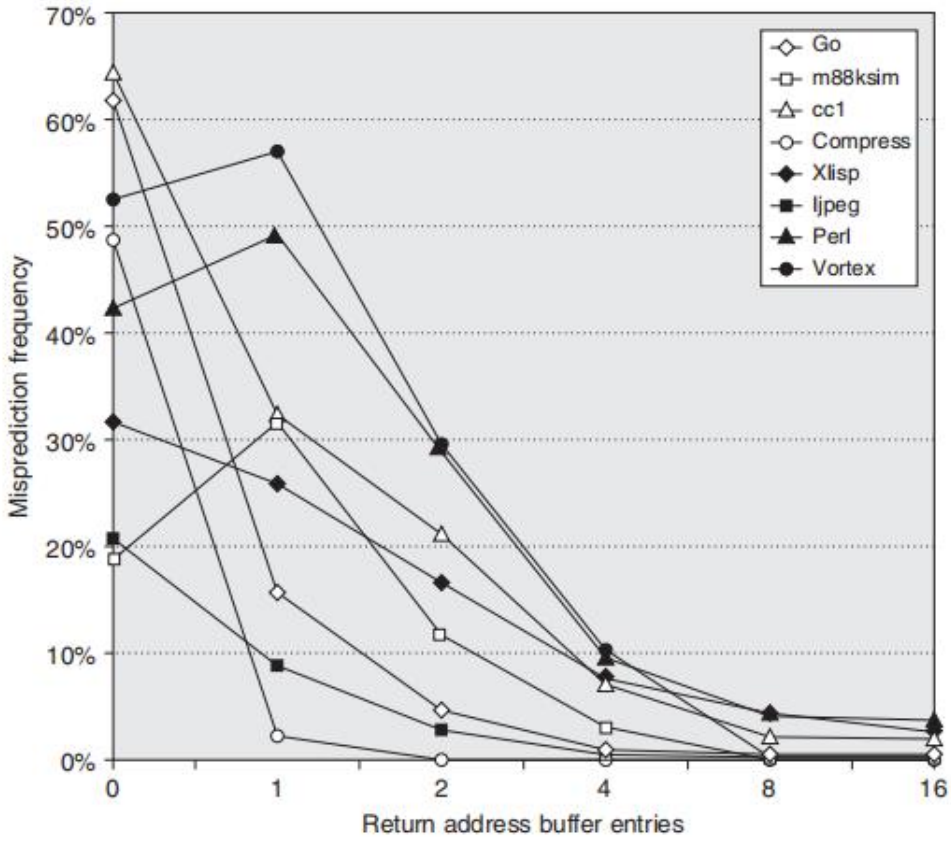


Figure 3.24 Prediction accuracy for a return address buffer operated as a stack on a number of SPEC CPU95 benchmarks. The accuracy is the fraction of return addresses predicted correctly. A buffer of 0 entries implies that the standard branch prediction is used. Since call depths are typically not large, with some exceptions, a modest buffer works well. These data come from Skadron et al. [1999] and use a fix-up mechanism to prevent corruption of the cached return addresses.

- **返回栈（Return Address Buffer)中表项数（entries)与预测精度的关系**

Instruction Fetch Unit

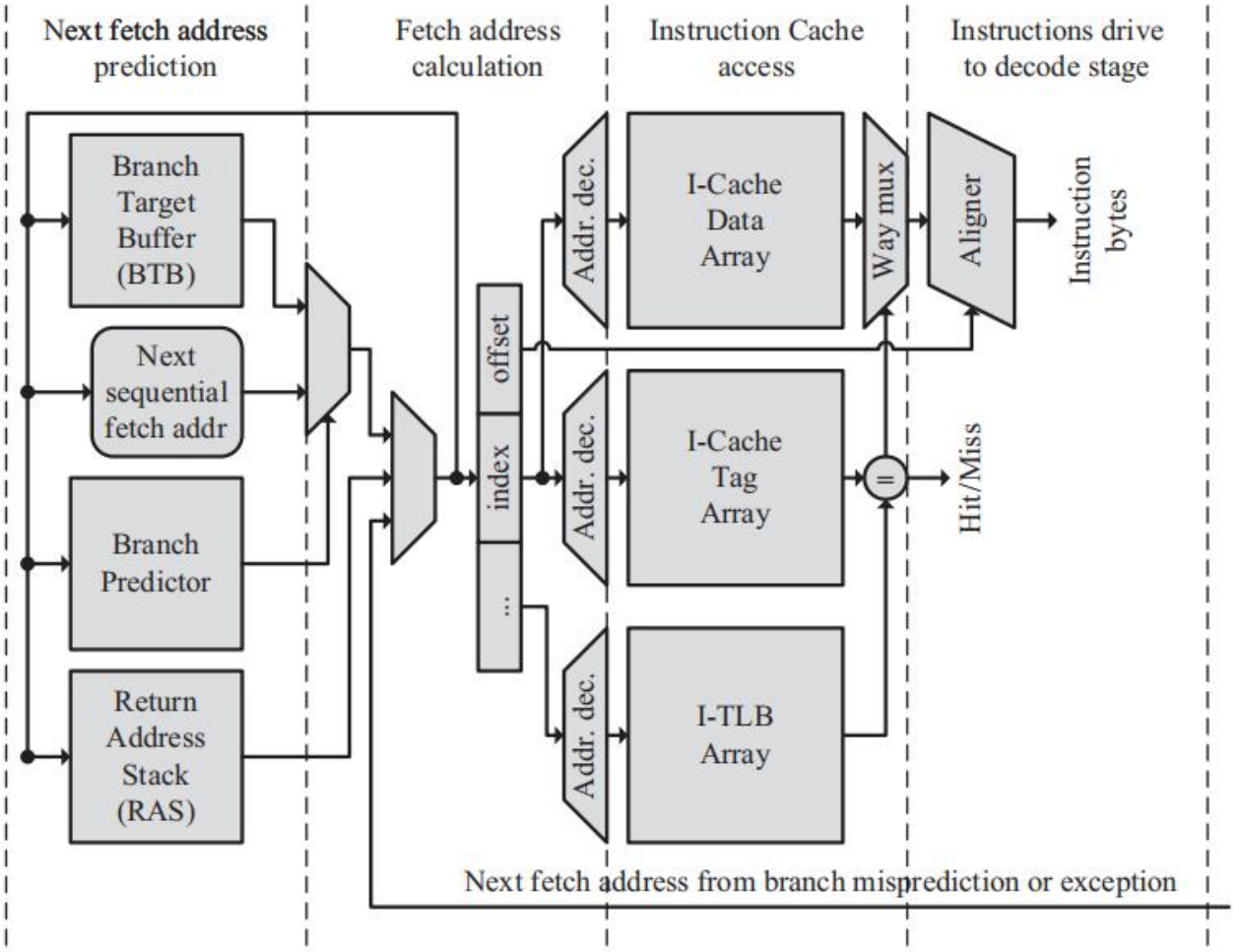


FIGURE 3.1: Example fetch pipeline.

Summary

- **基于BHT表的预测器:**
  - Basic 2-bit predictor:
  - Global predictor:
    - 每个分支对应多个m-bit预测器
    - 最近n次的分支转移的每一种情况分别对应其中一个预测器
  - Local predictor:
    - 每个分支对应多个m-bit预测器
    - 该分支最近n次分支转移的每一种情况分别对应其中一个预测器
  - Tournament predictor:
    - 从多种预测器的预测结果中选择合适的预测结果。
    - 例如：两级全局预测器与两级局部预测器
- **优化取指令的带宽**
  - 基于BTB的分支预测器
  - Return Address Stack
  - 集成的独立的取指部件

Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - John Hennessy (Standford)and David Patterson (UCB)
  - Chenxi Zhang (Tongji)
  - Muhamed Mudawar (KFUPM)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**