计算机体系结构

Topic: Instruction level parallelism

?

- **动态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?**
  - 我们可以用硬件做循环展开必须可以解决分支指令问题

关于异常处理???

- **乱序完成加大了实现精确中断的难度**
  - 在前面指令还没有完成时，寄存器文件中可能会有后面指令的运行结果.
  - 如果这些前面的指令执行时有中断产生，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:**
  - 精确中断的含义是其返回地址为：
    - 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
- **实现精确中断的技术：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
```

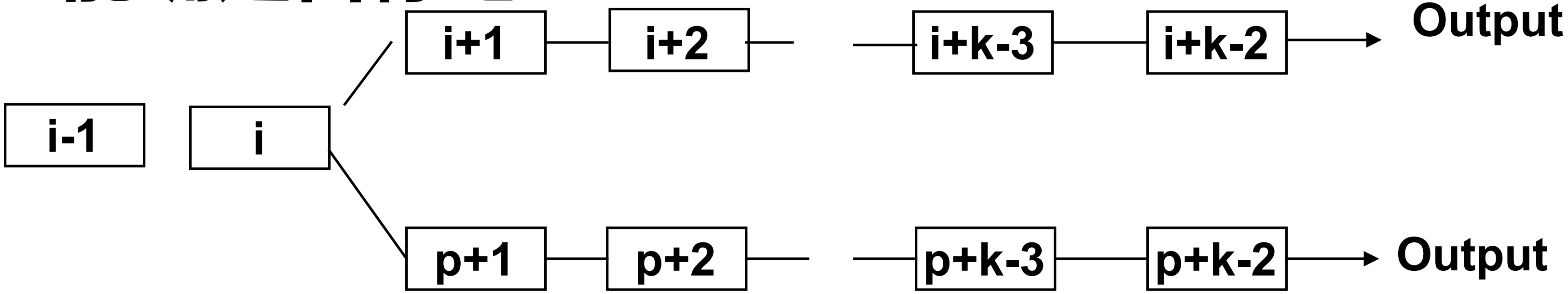
- **如果分支依赖于multd,怎么办??**
  - 需要能预测分支方向
  - 如果分支成功，我们就可以重叠执行循环
- **对于superscalar机器这一问题更加突出**

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
- **处理条件转移和中断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

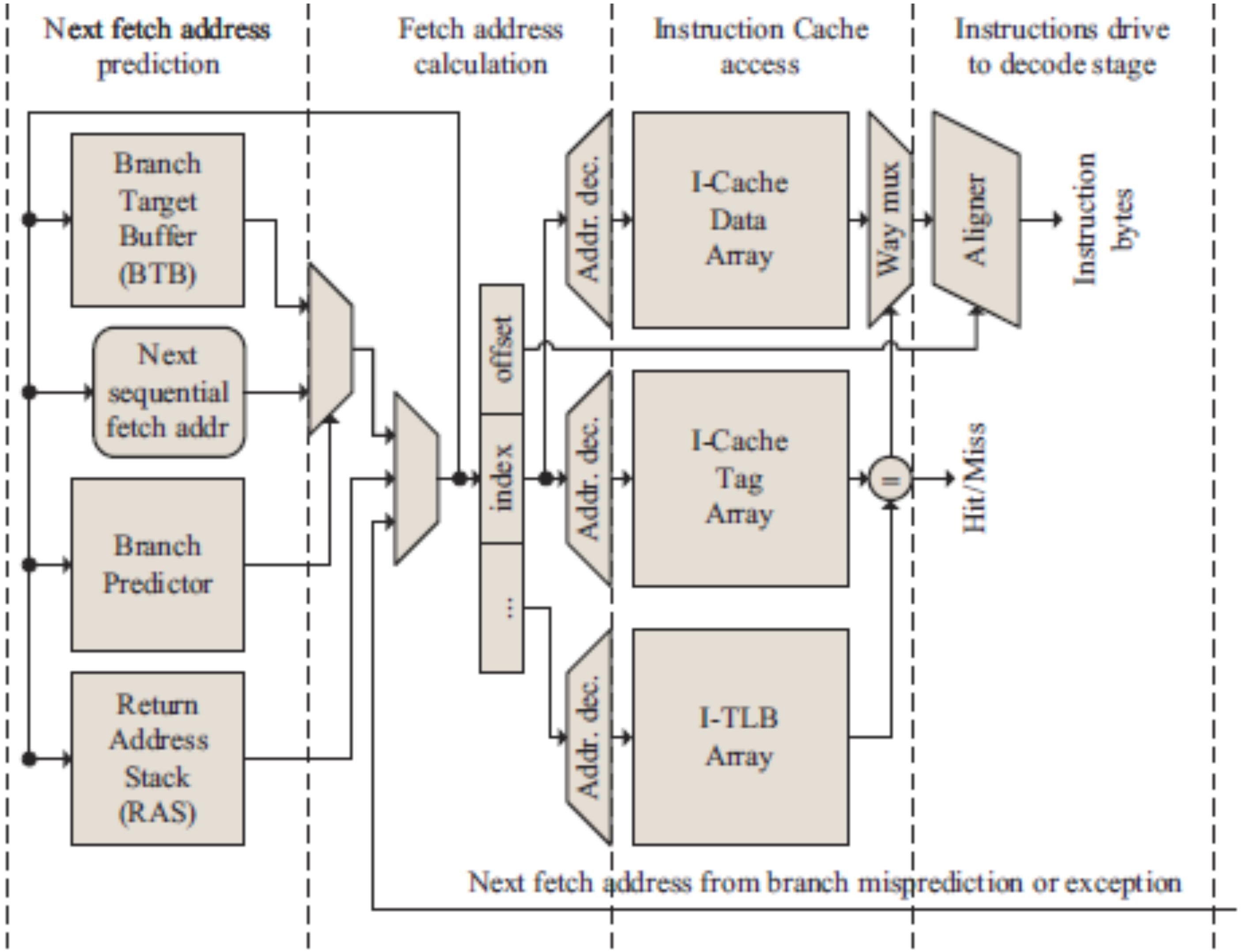
- **假设对于一条有K段的流水线，由于条件分支的影响，在最坏情况下，每次条件转移将造成k-1个时钟周期的断流。假设条件分支在一般程序中所占的比例为p, 采用分支预测失败策略，条件成功的概率为q。试分析分支对流水线的影响。**
- **结论：条件转移指令对流水线的影响很大，必须采取相关措施来减少这种影响。**
- **预测可以是静态预测“Static” (at compile time)或动态预测 “Dynamic” (at runtime)**
  - 例如：一个循环供循环10次，它将分支成功9次，1次不成功。
  - 动态分支预测 vs. 静态分支预测，哪个好?

分支预测

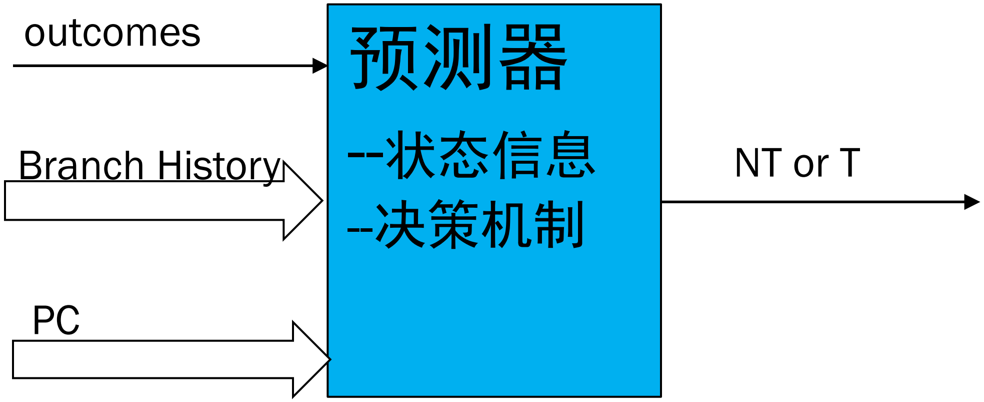
- **分支预测对提高性能是非常重要的**
  - 分支预测在哪个阶段完成?
  - 预测器设计的核心问题是什么?
  - 预测器的基本结构及输入输出?
- **预测器的分类**
  - 基于BHT表的预测器:
    - Basic 2-bit predictor:
    - Correlating predictor:
      - Multiple 2-bit predictors for each branch
      - One for each possible combination of outcomes of preceding n branches
    - Local predictor:
      - Multiple 2-bit predictors for each branch
      - One for each possible combination of outcomes for the last n occurrences of this branch
    - Tournament predictor: Combine correlating predictor with local predictor
  - 基于BTB的分支预测器

CH. 3.3

Instruction Fetch Unit


Next fetch address prediction
Fetch address calculation
Instruction Cache access
Instructions drive to decode stage
Branch Target Buffer (BTB)
Next sequential fetch addr
Branch Predictor
Return Address Stack (RAS)
offset
index
...
Addr. dec.
Addr. dec.
Addr. dec.
I-Cache Data Array
I-Cache Tag Array
I-TLB Array
Way mux
Aligner
Instruction bytes
=
Hit/Miss
Next fetch address from branch misprediction or exception

预测器的基本结构及输入输出

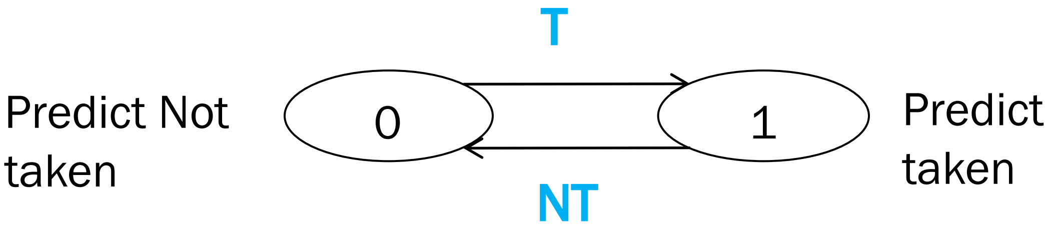


- **根据outcomes来更新状态信息**
- **使用FSM来完成决策**
  - 根据Branch History和PC来选择状态
  - 由状态决定输出

Dynamic Branch Prediction

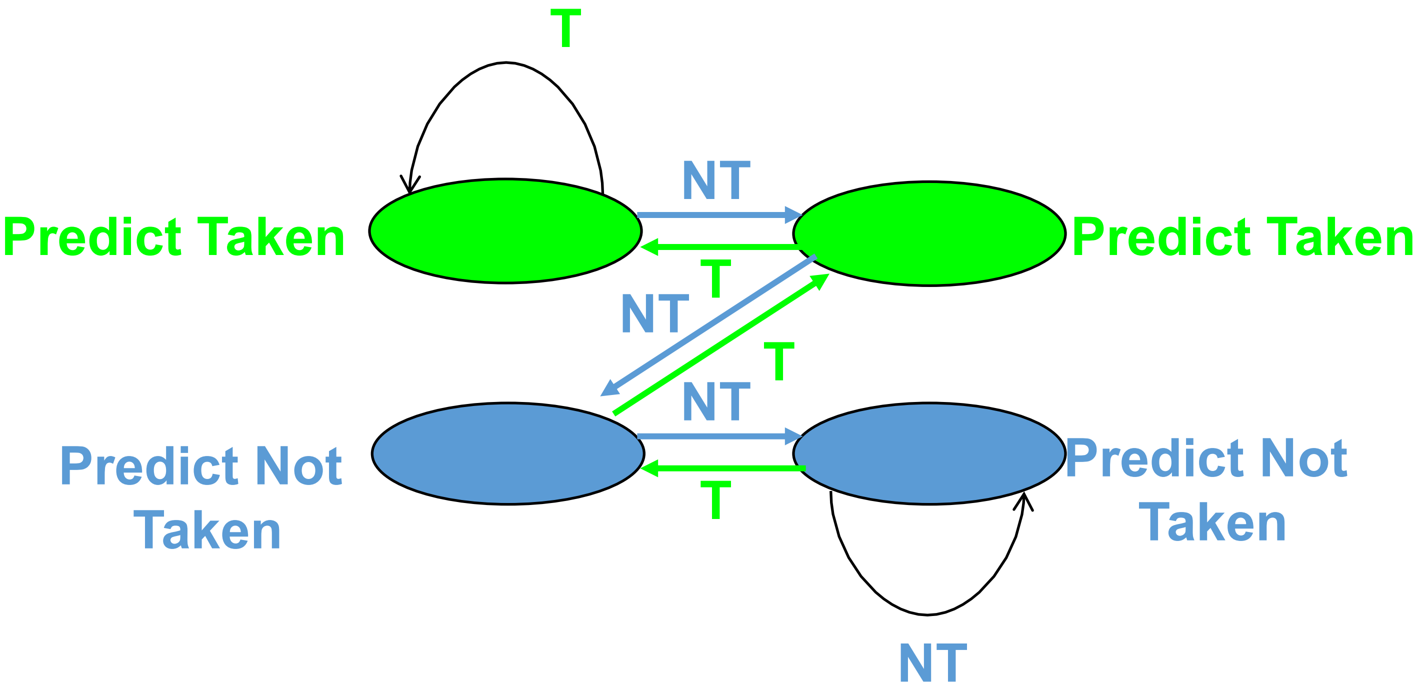
- **动态分支预测：预测分支的方向在程序运行时刻动态确定**
- **需解决的关键问题是：**
  - 如何记录转移历史信息
  - 如何根据所记录的转移历史信息，预测转移的方向
- **主要方法**
  - 基于BPB(Branch Prediction Buffer)或BHT(Branch History Table)
    - 1-bit BHT和2-bit BHT
    - Correlating Branch Predictors
    - Tournament Predictors: Adaptively Combining Local and Global Predictors
  - High Performance Instruction Delivery
    - BTB
    - Integrated Instruction Fetch Units
    - Return Address Predictors
- **Performance = *f*(accuracy, cost of misprediction)**
  - Misprediction □ Flush Reorder Buffer

1-bit BHT



- **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
    - 前面为预测成功，最后一次需要退出循环
  - 首次循环
    - 前面为预测为失败，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**
- **Green: go, taken**

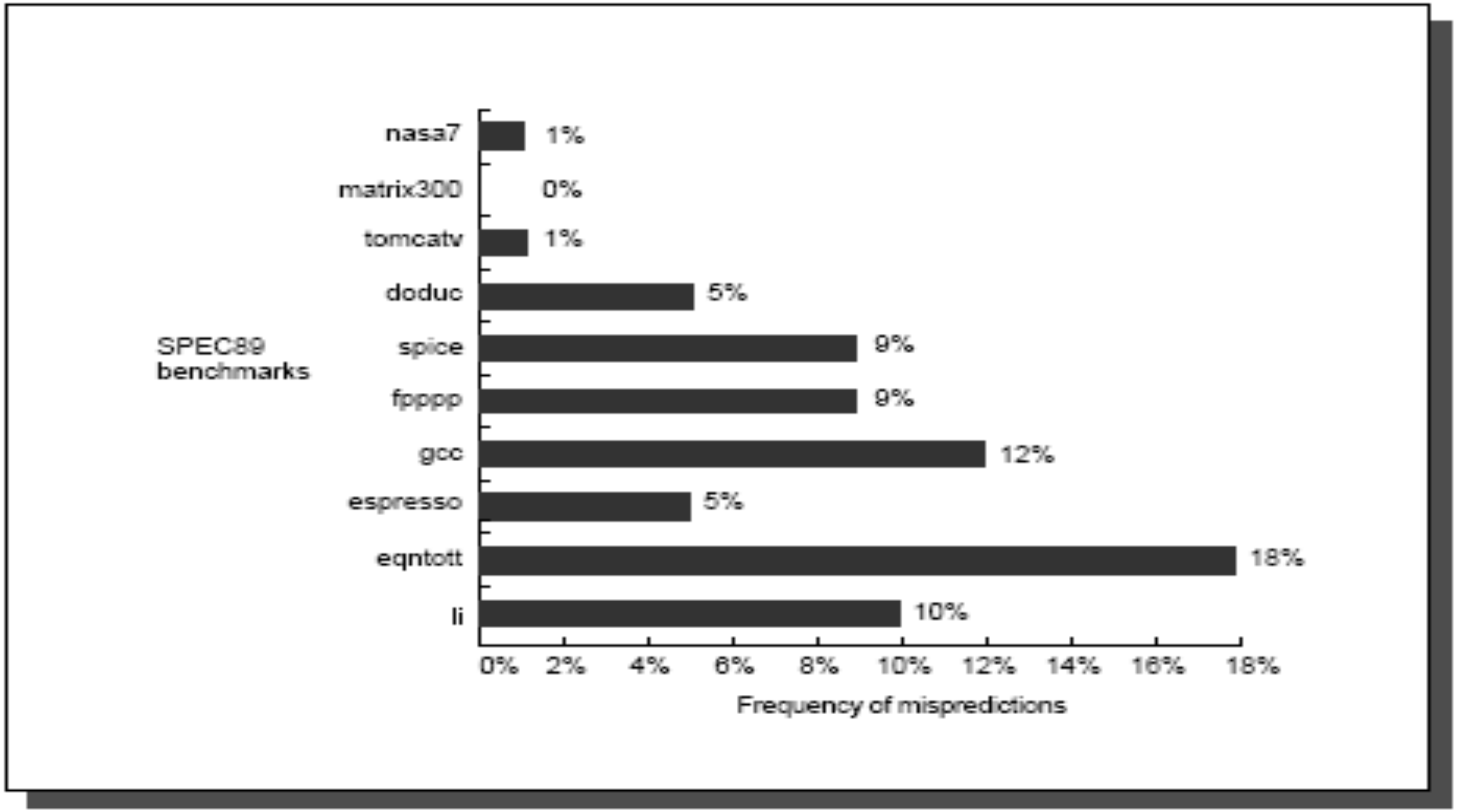


FIGURE 3.8 Prediction accuracy of a 4096-entry two-bit prediction buffer for the SPEC89 benchmarks. The misprediction rate for the integer benchmarks (gcc, espresso, eqntott, and li) is substantially higher (average of 11%) than that for the FP programs (average of 4%). Even omitting the FP kernels (nasa7, 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].

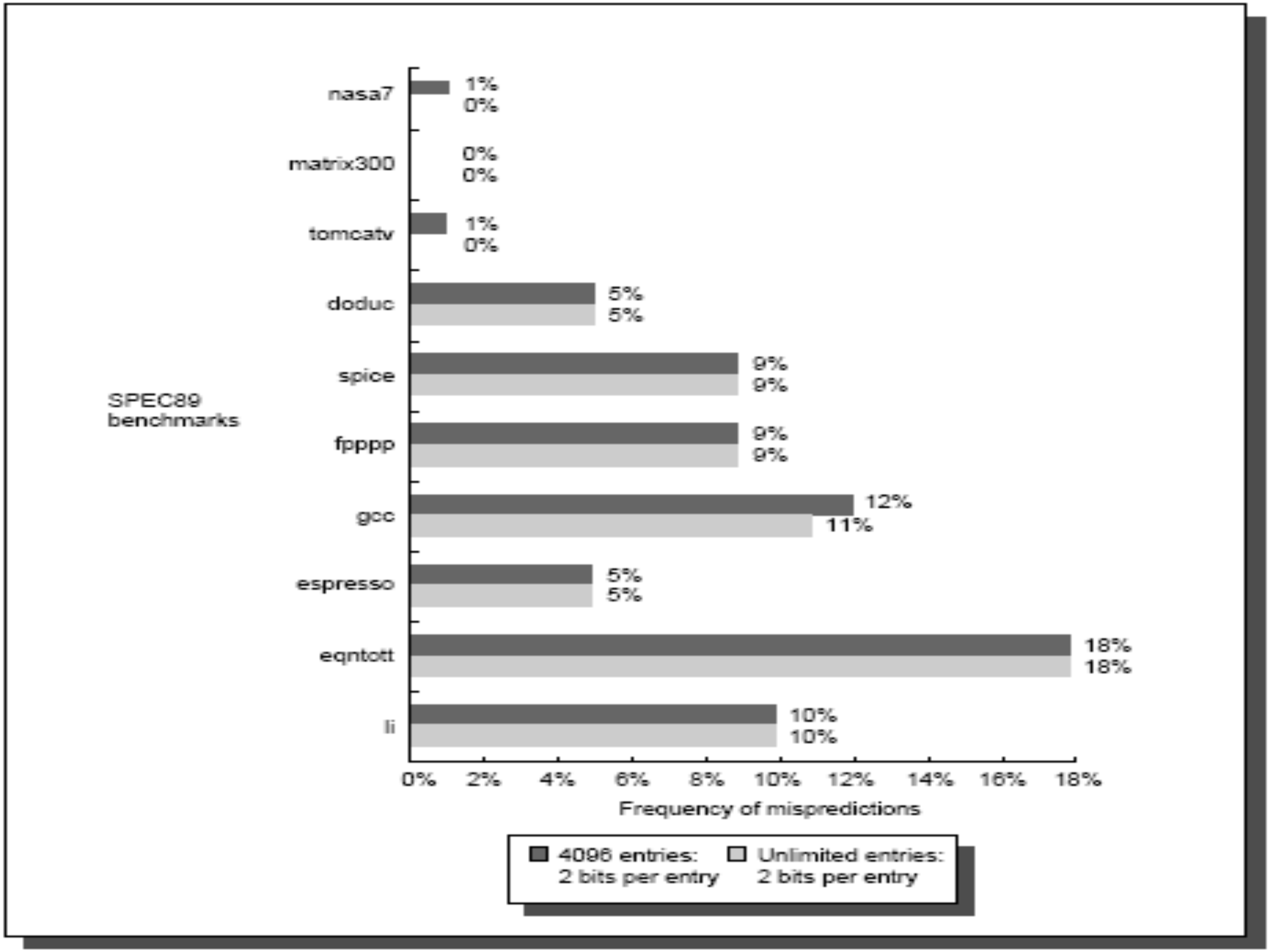


FIGURE 3.9 Prediction accuracy of a 4096-entry two-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks.

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MIPS

```
      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都分支失败，则b3一定成功。

Correlating Branches

- **Correlating predictors 或 两级预测器：**
  - 分支预测器根据其他分支的行为来进行预测
- **工作原理：**
  - 根据一个简单的例子来看其基本原理

翻译为MIPS

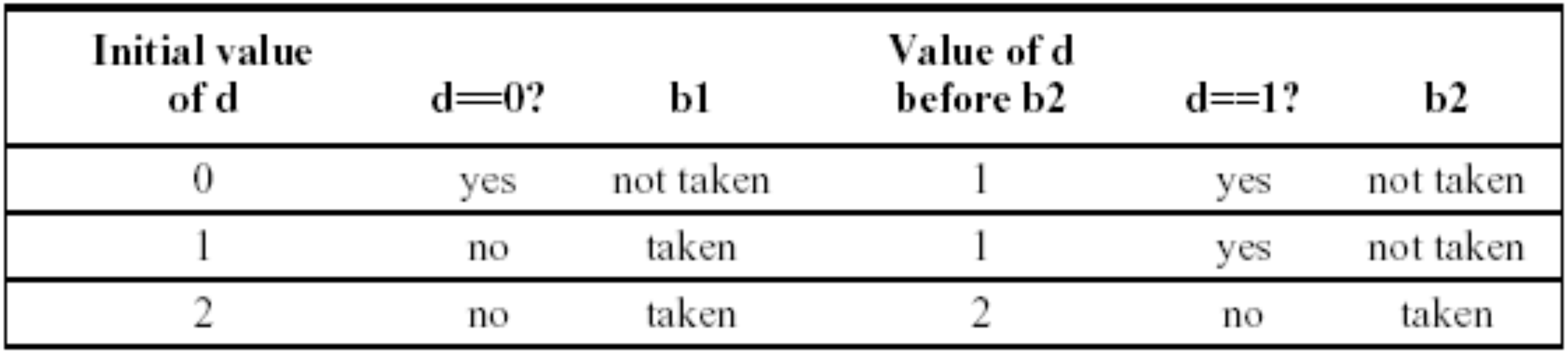
```
 if (d==0)d=1;
 if (d==1) d=0;

        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:
```

两级预测器基本工作原理

- 假设d的初始值序列为0，1，2
- b1 如果分支失败，b2一定也分支失败。
- 前面的两位标准的预测方案就没法利用这一点，而两级预测方案就可以。

```
 if (d==0)d=1;
  if (d==1) d=0;
翻译为DLX
        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)
```

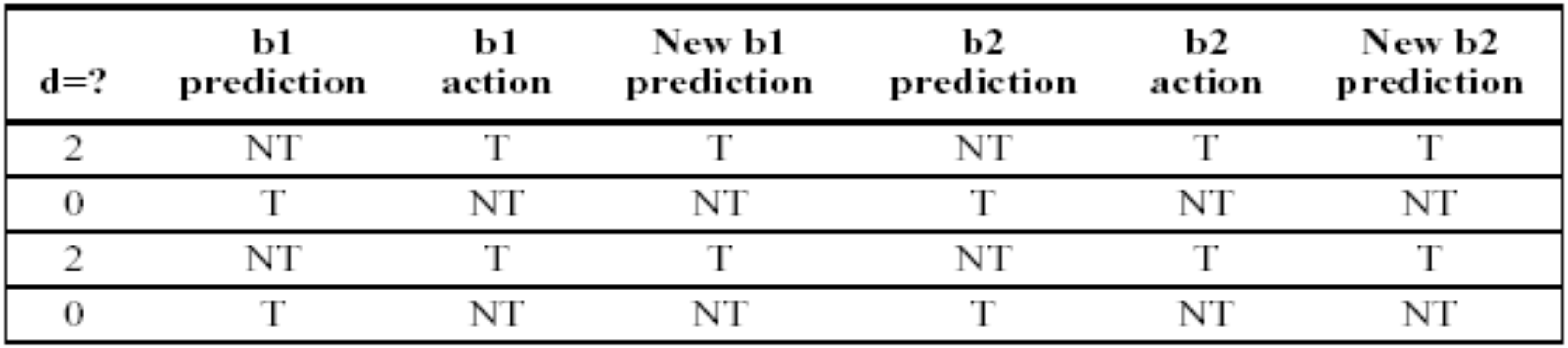


Initial value of d	d==0?	b1	Value of d before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
2	no	taken	2	no	taken

FIGURE 3.10 Possible execution sequences for a code fragment.

- 假设d的初始值在2和0之间切换。
- 用1-bit预测器，初始设置为预测失败，T表示预测成功，NT表示预测失败。
- 结论：这样的序列每次预测都错，预测错误率100%

```
        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)
```

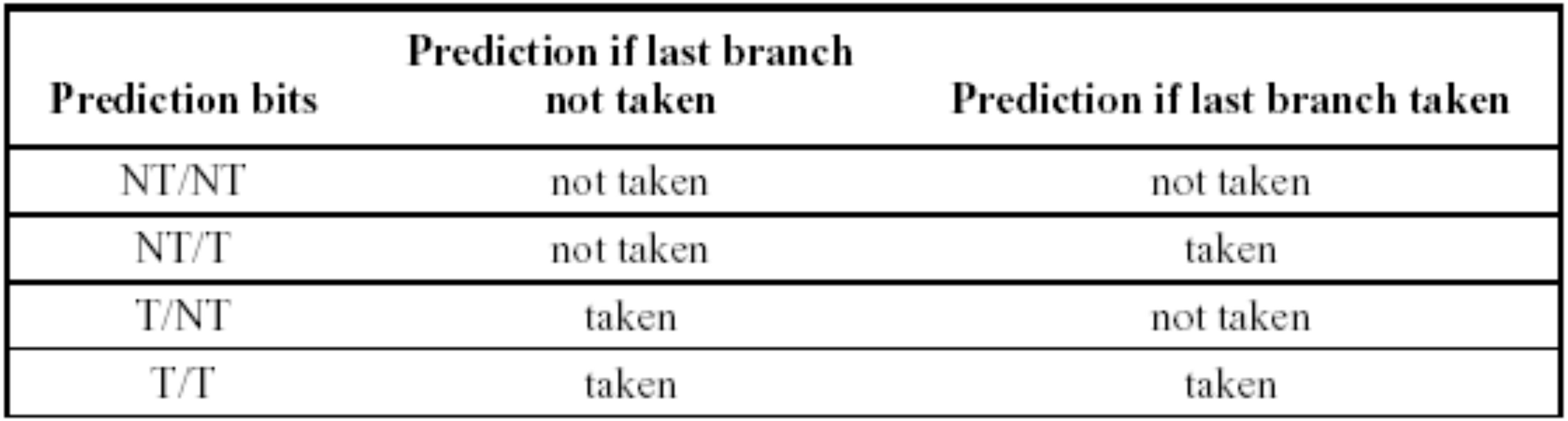


d=?	b1 prediction	b1 action	New b1 prediction	b2 prediction	b2 action	New b2 prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

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

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失败时的预测位，另一个预测位是假设最近一次执行的分支成功时的预测位。
- 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

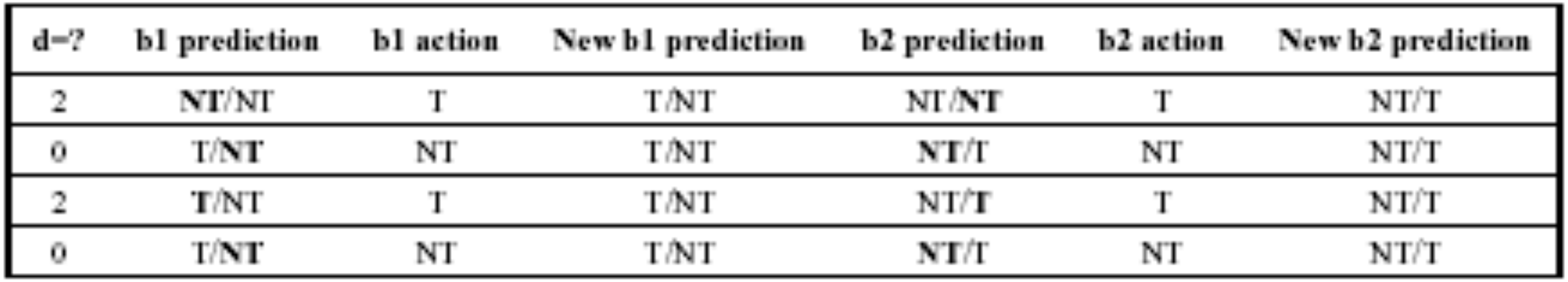


Prediction bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	not taken	not taken
NT/T	not taken	taken
T/NT	taken	not taken
T/T	taken	taken

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

- Correlating 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的行为来选择一对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

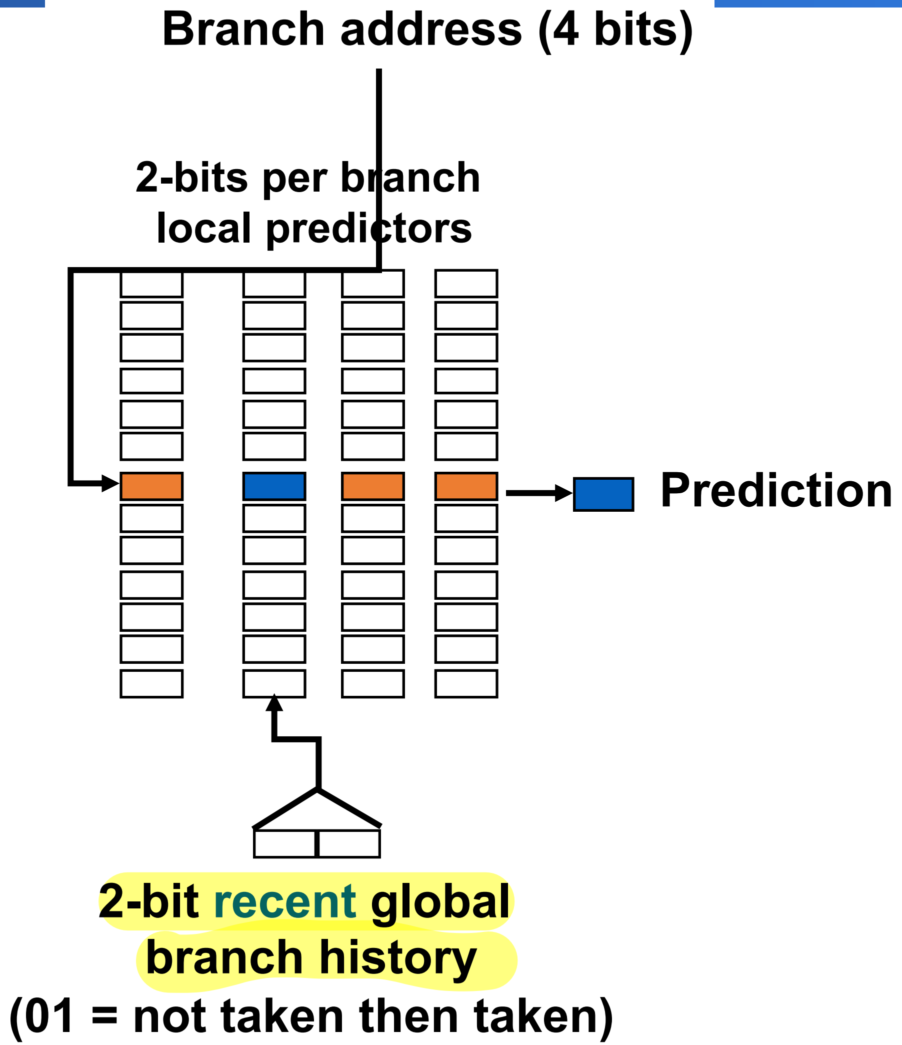
```
        BNEZ R1,L1              ;branch b1(d!=0)
        ADDI R1,R0,#1          ;d==0, so d=1
  L1:  ADDI R3,R1,#-1
        BNEZ R3,L2              ;branch b2(d!=1)
```



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.

Correlating Branches



Branch address (4 bits)

2-bits per branch
local predictors

Prediction

2-bit recent global
branch history
(01 = not taken then taken)

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

Gshare predictor

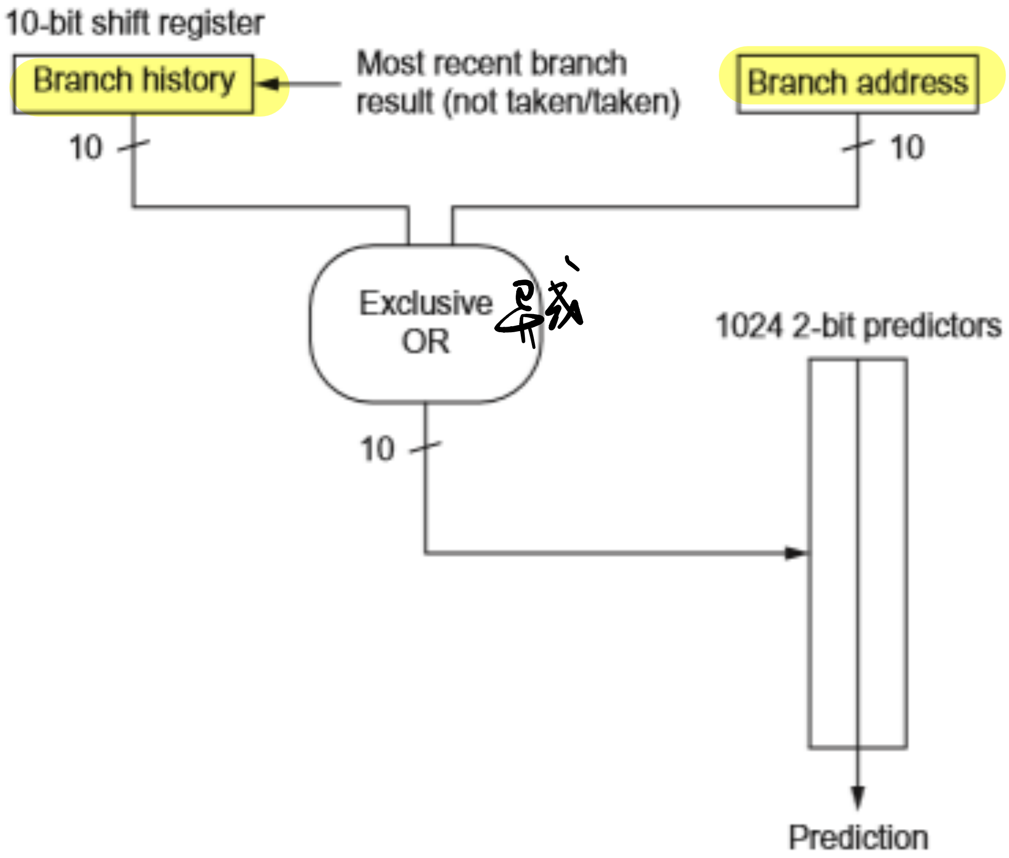


Figure 3.4 A gshare predictor with 1024 entries, each being a standard 2-bit predictor.

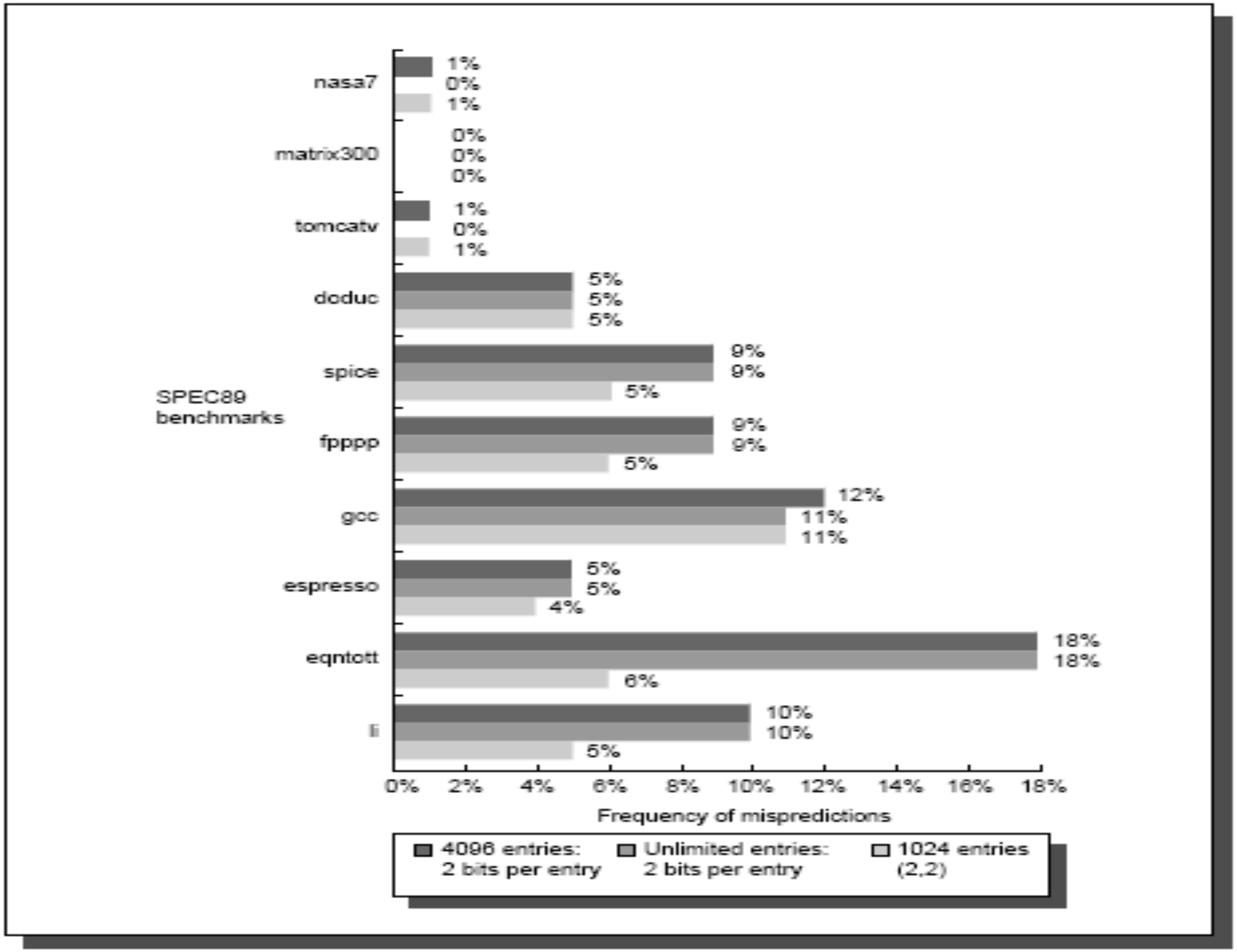
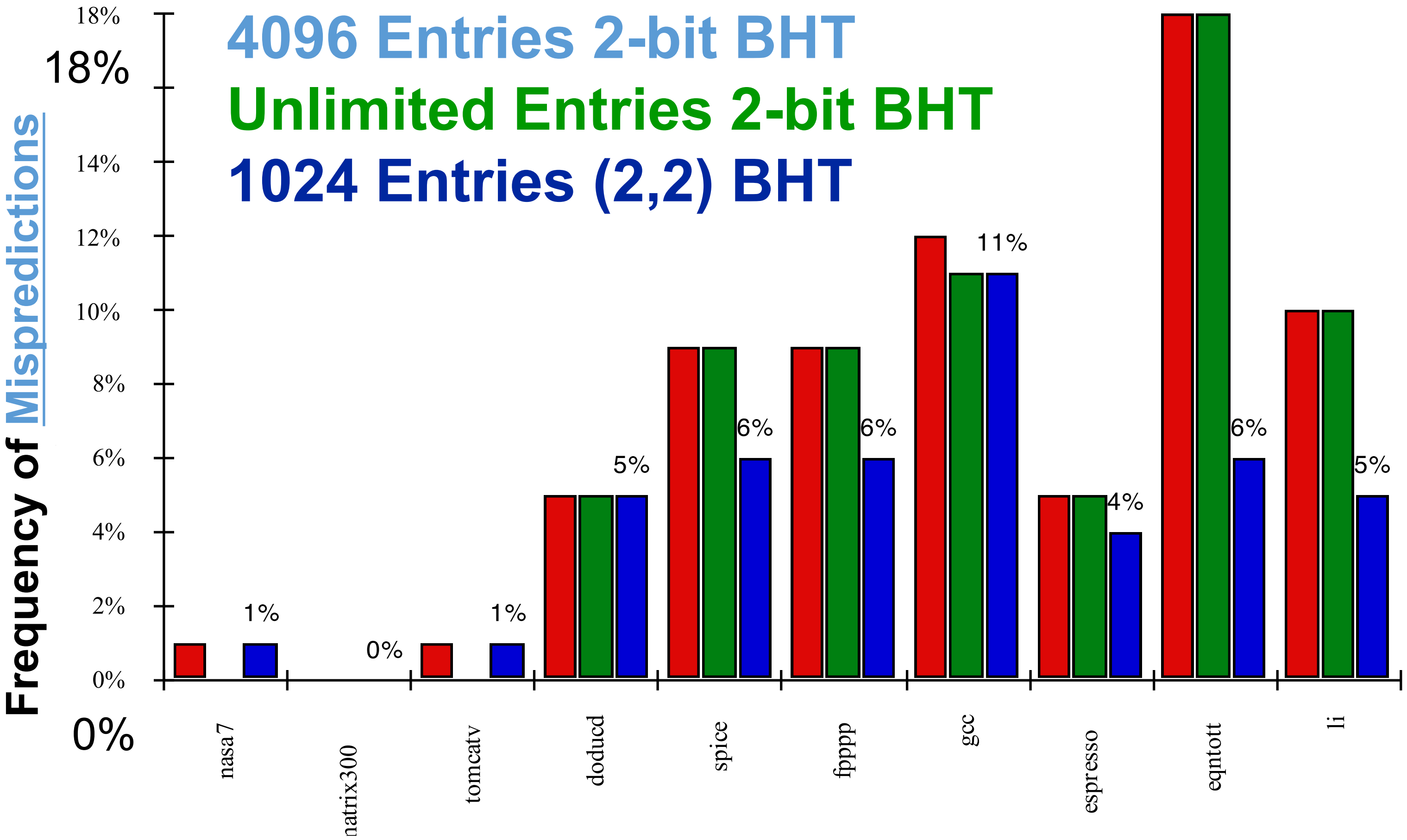


FIGURE 3.15 **Comparison of two-bit predictors.** A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating two-bit predictor with unlimited entries and a two-bit predictor with two bits of global history and a total of 1024 entries.

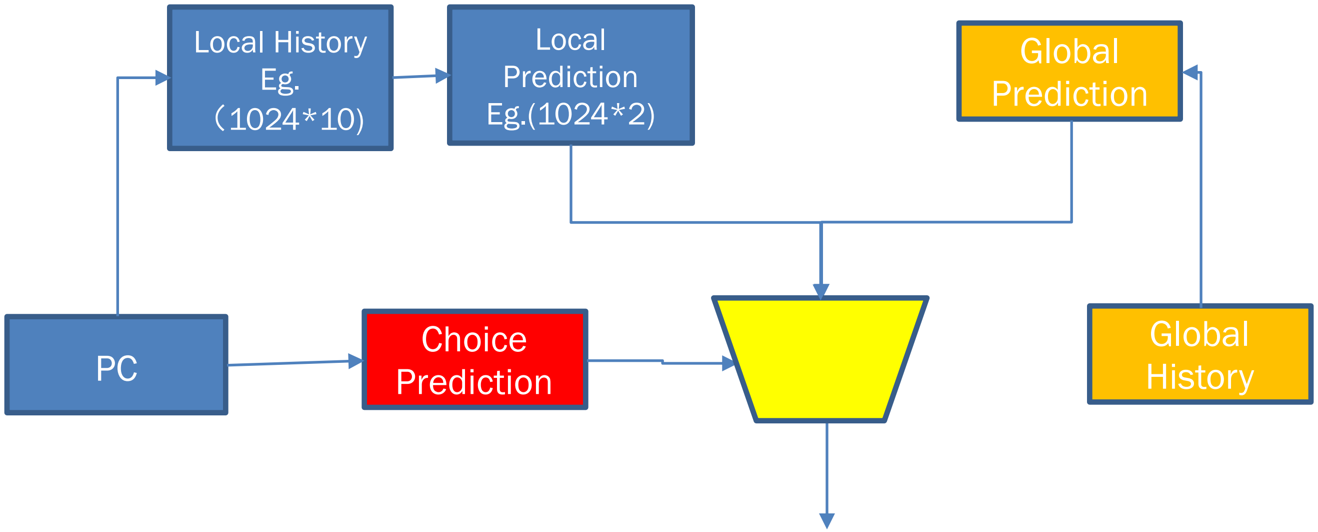
Accuracy of Different Schemes



Branch Prediction

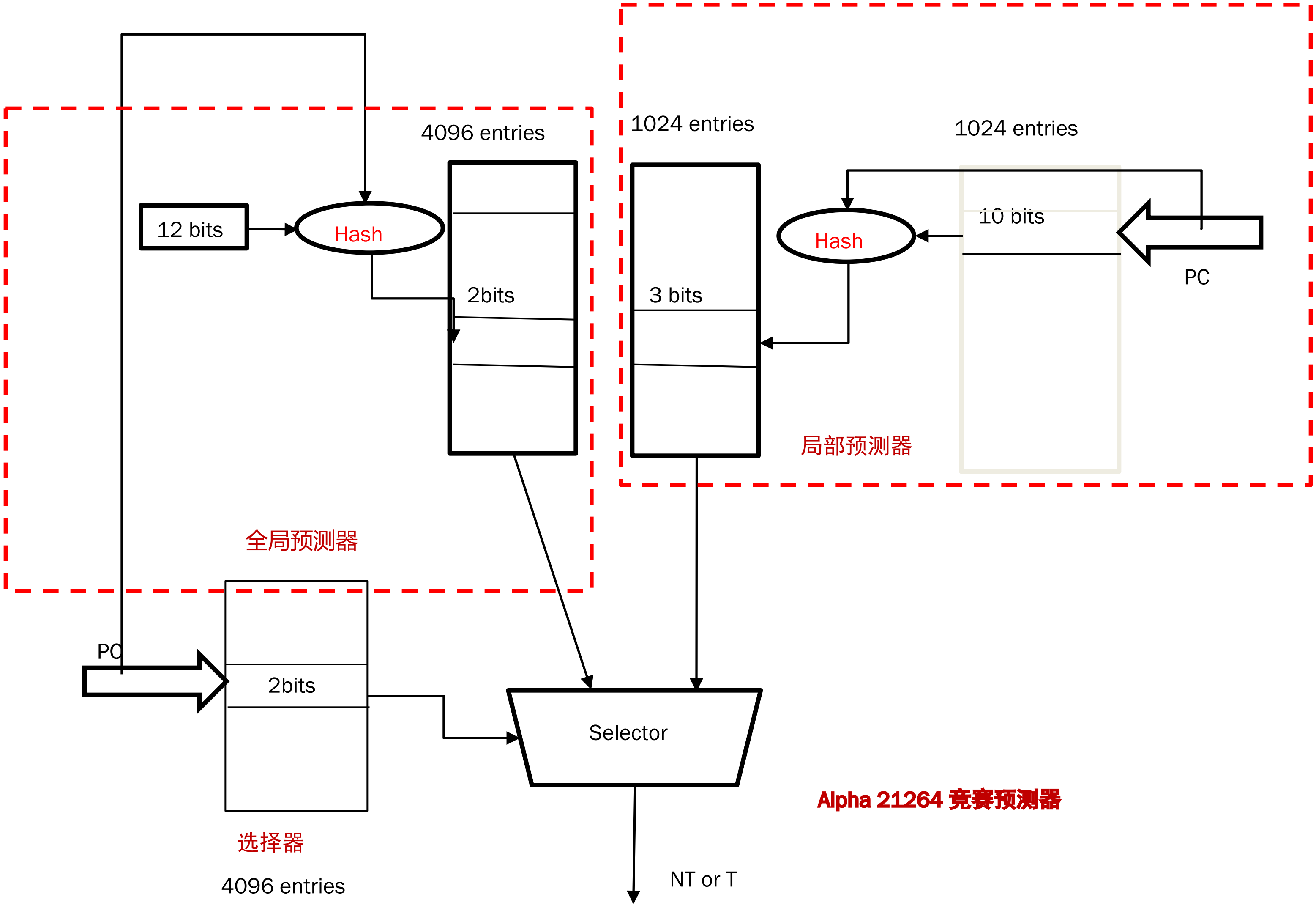
- **Basic 2-bit predictor:**
- **关联预测器(n,2):**
  - 每个分支有多个 2-bit 预测器
  - 根据最近n次分支的执行情况从2^n中选择预测器
- **两级局部预测器(Local predictor):**
  - 每个分支有多个2-bit 预测器
  - 根据该分支的最近n次分支的执行情况从2^n中选择预测器
- **竞赛预测器(Tournament predictor):**
  - 结合关联预测器和两级局部预测器

竞赛预测器



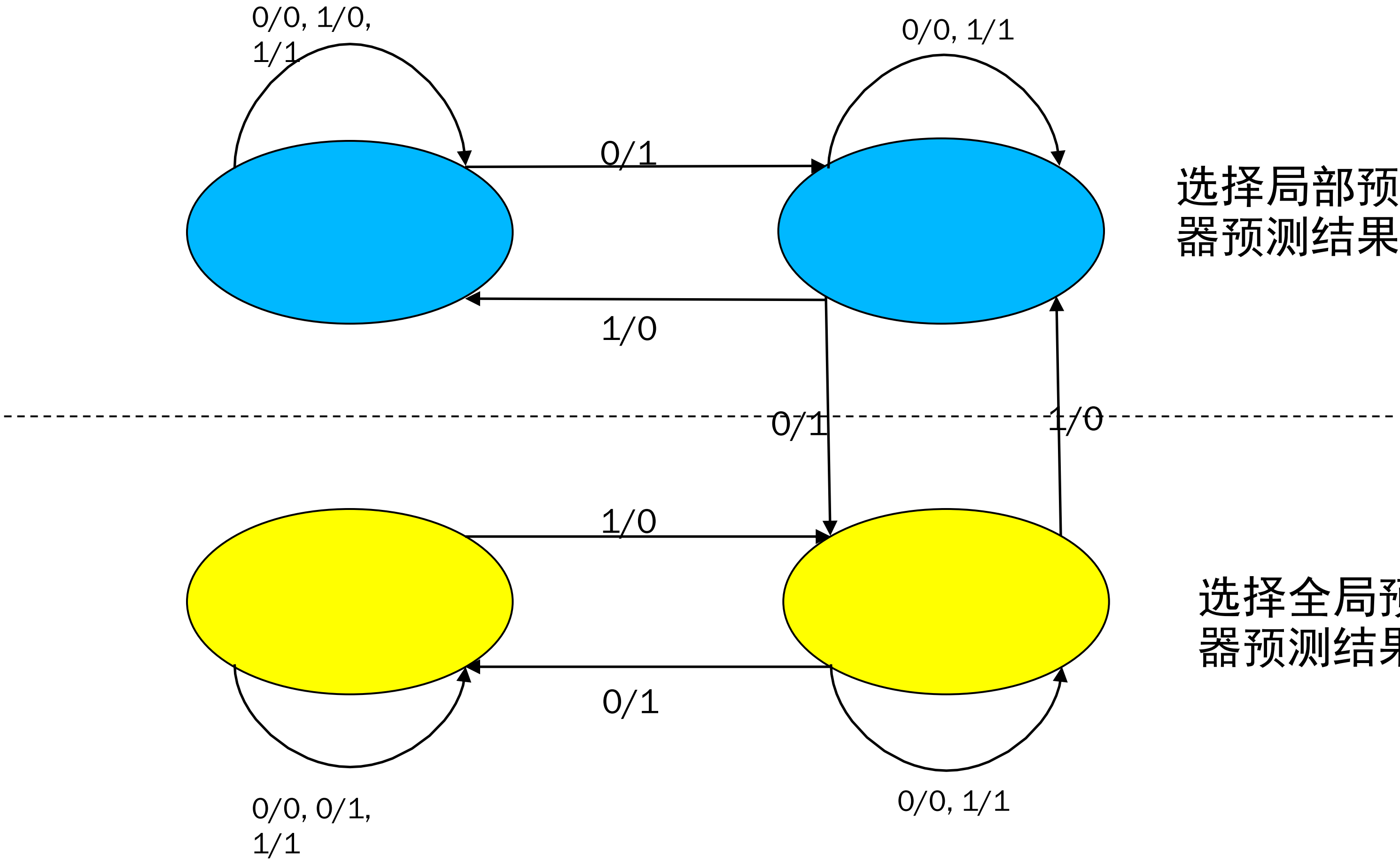
- **全局预测器**
  - 使用最近n次分支跳转情况来索引，即全局预测器入口数：2^n每个入口是一个标准的2位预测器
- **局部预测器：设计为两层。**
  - 一个局部历史记录,使用指令地址的低m位进行索引，每个入口k位，分别对应这个入口最近的k次分支，即最近k次分支的 跳转情况
  - 从局部历史记录选择出的入口对一个2^K的入口表进行索引，这些入口由2位计数器构成，以提供本地预测。
- **选择器：**
  - 使用分支局部地址的低m位分支局部地址索引，每个索引得到一个两位计数器，用来选择使用局部预测器还是使用全局预测器的预测结果。
  - 在设计时默认使用局部预测器，当两个预测器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



Alpha 21264 竞赛预测器

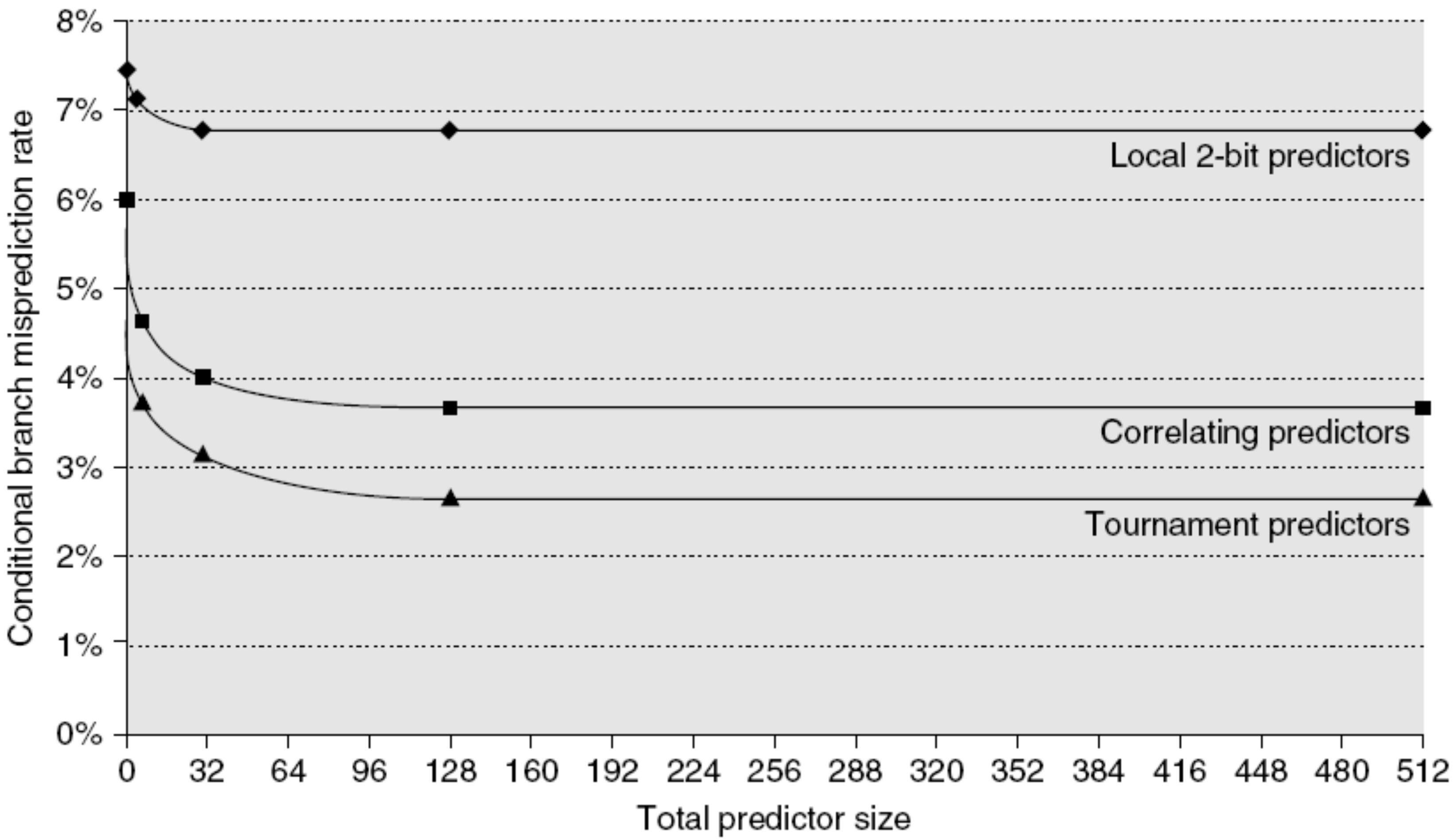
选择器状态转移图



选择局部预测器预测结果

选择全局预测器预测结果

Branch Prediction Performance



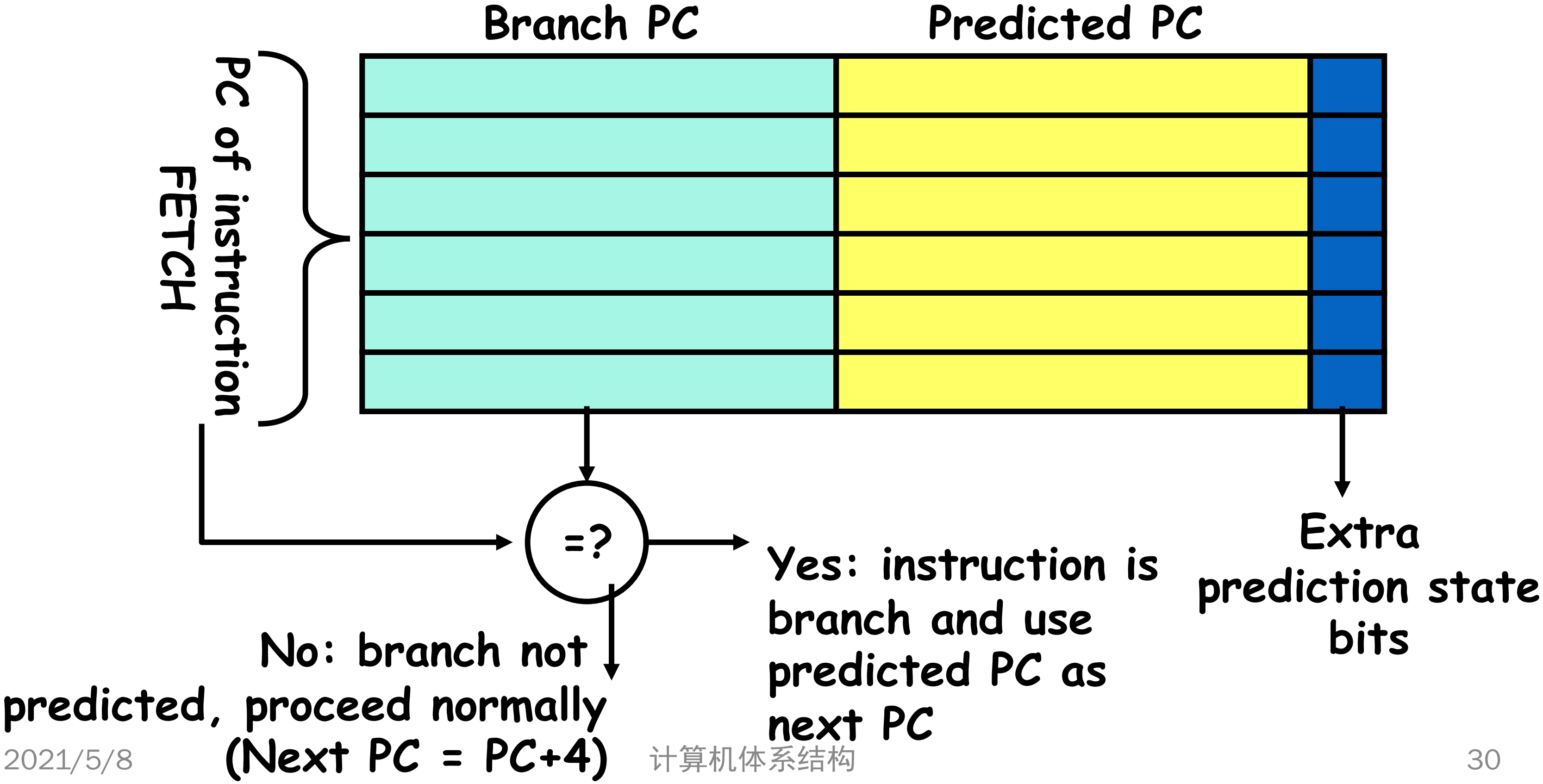
Branch predictor performance

review

- **基于BHT表的预测器:**
  - Basic 2-bit predictor:
  - Correlating 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：Combine correlating predictor with local predictor
- **基于BTB的分支预测器**

Simple dynamic prediction: 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


Branch PC
Predicted PC
PC of instruction FETCH
=?
Yes: instruction is branch and use predicted PC as next PC
No: branch not predicted, proceed normally (Next PC = PC+4)
Extra prediction state bits

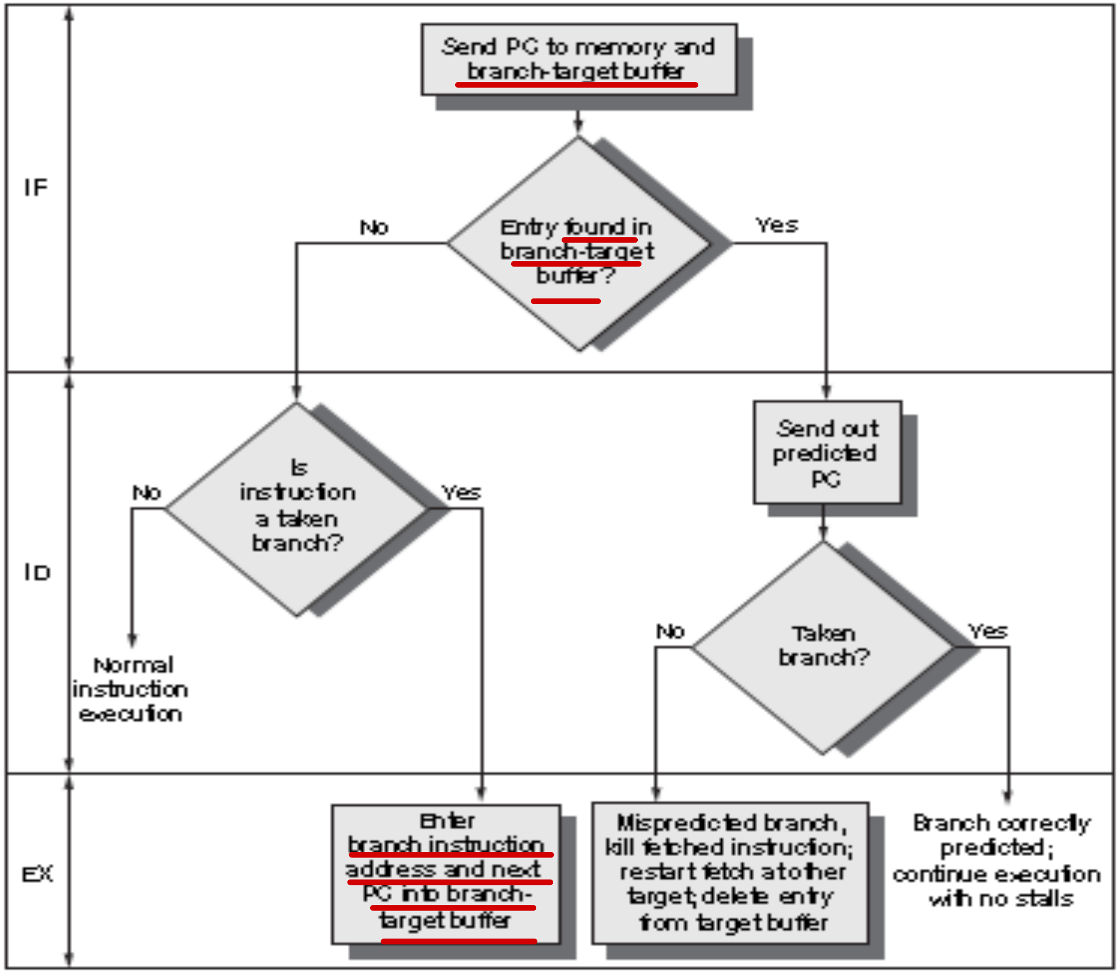
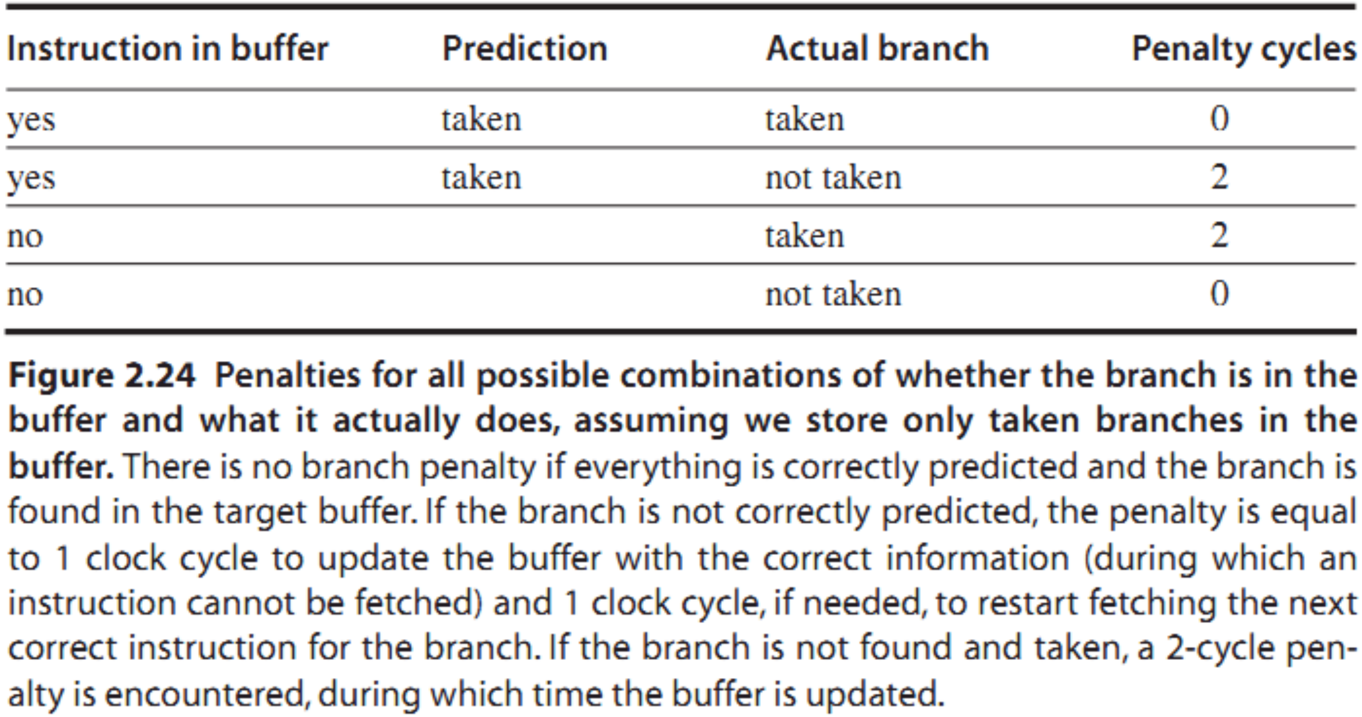

IF
ID
EX
Send PC to memory and branch-target buffer
Entry found in branch-target buffer?
No
Yes
Is instruction a taken branch?
No
Yes
Normal instruction execution
Send out predicted PC
Taken branch?
No
Yes
Enter branch instruction address and next PC into branch-target buffer
Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
Branch correctly predicted; continue execution with no stalls

Figure 2.23 The steps involved in handling an instruction with a branch-target buffer.



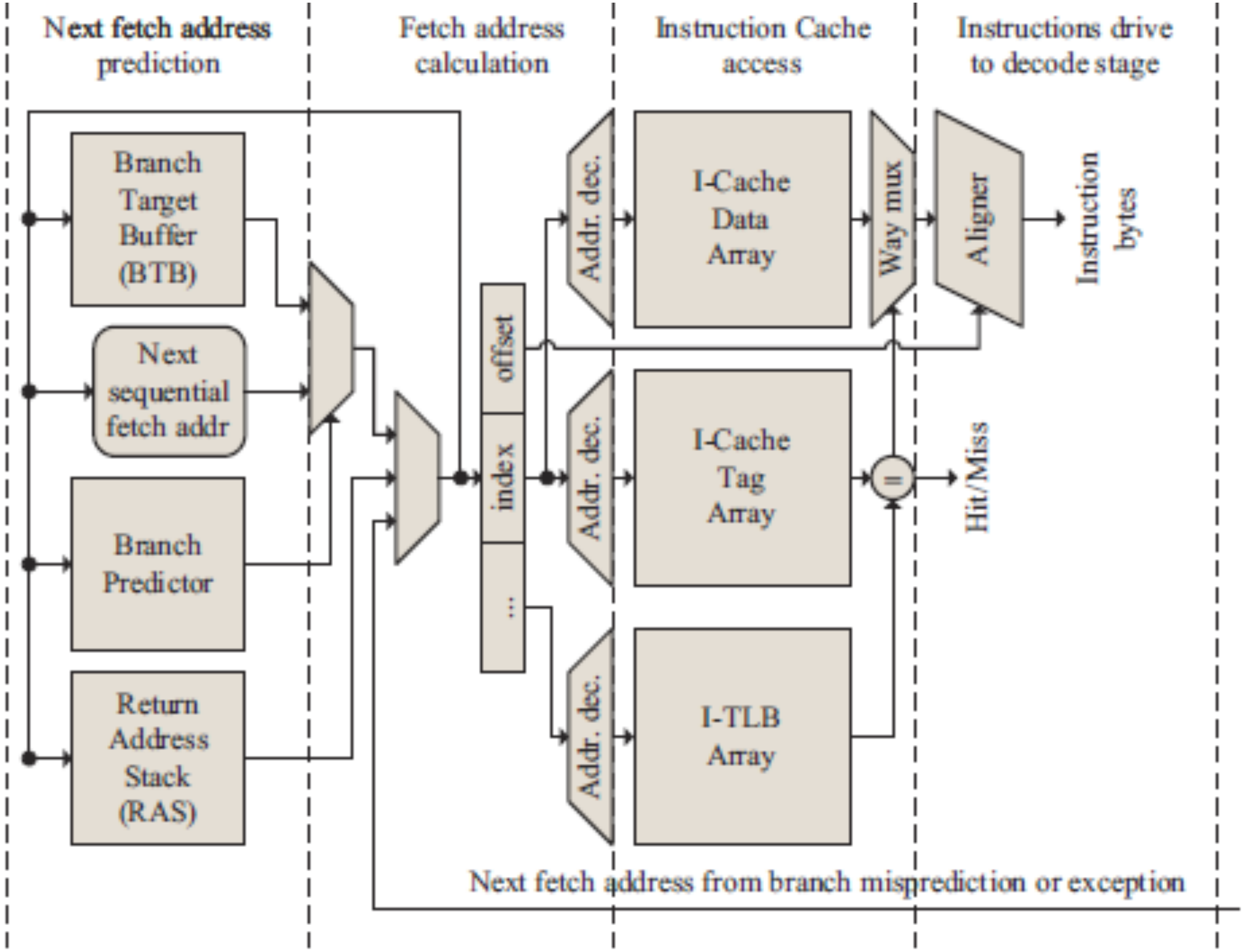
Instruction in buffer	Prediction	Actual branch	Penalty cycles
yes	taken	taken	0
yes	taken	not taken	2
no		taken	2
no		not taken	0

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.

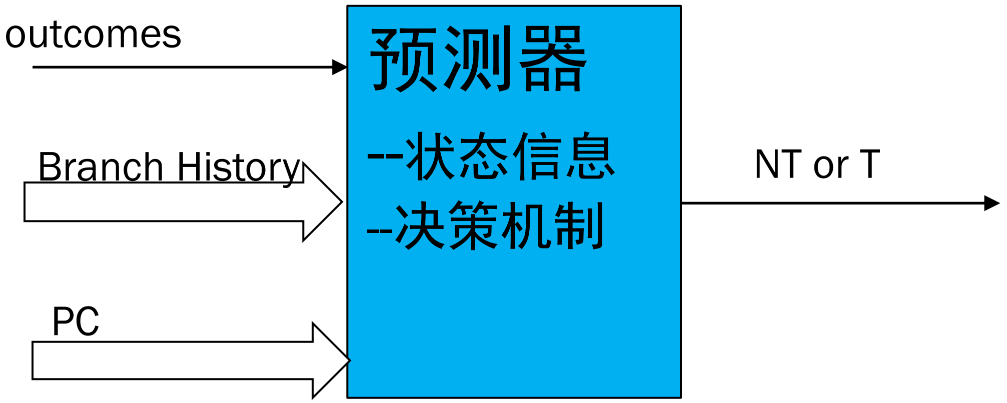
Determine the total branch penalty for a branch-target buffer assuming the penalty cycles for individual mispredictions from Figure 2.24. Make the following assumptions about the prediction accuracy and hit rate:

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

Instruction Fetch Unit


Next fetch address prediction
Fetch address calculation
Instruction Cache access
Instructions drive to decode stage
Branch Target Buffer (BTB)
Next sequential fetch addr
Branch Predictor
Return Address Stack (RAS)
offset
index
...
Addr. dec.
Addr. dec.
Addr. dec.
I-Cache Data Array
I-Cache Tag Array
I-TLB Array
Way mux
A ligner
=
Instruction bytes
Hit/Miss
Next fetch address from branch misprediction or exception

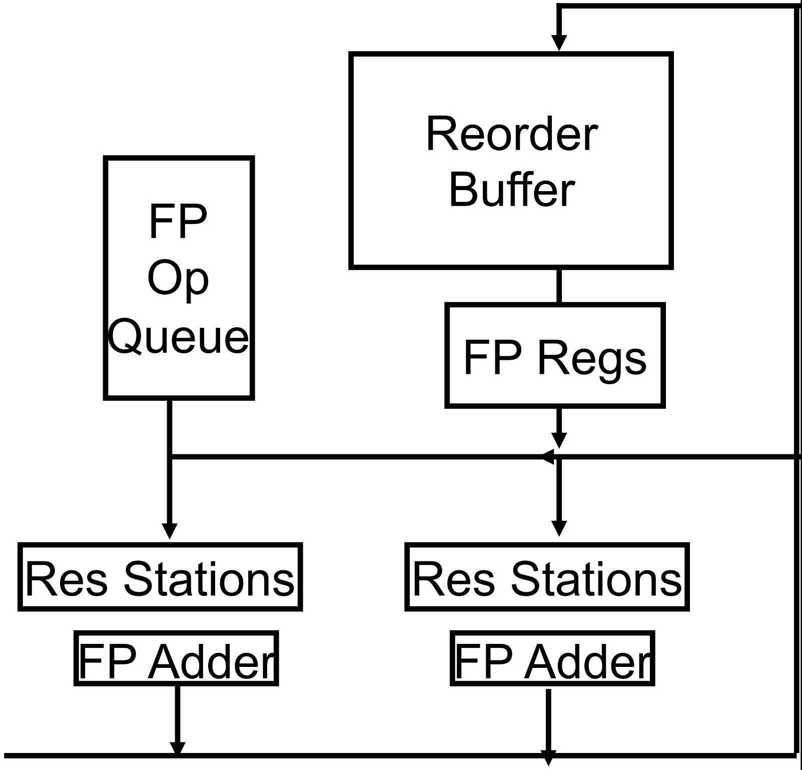
预测器的基本结构及输入输出



- **使用FSM来完成决策**
  - 根据Branch History和PC来选择状态
  - 由状态决定输出
- **根据实际结果修改状态**

硬件支持推断执行以及精确中断

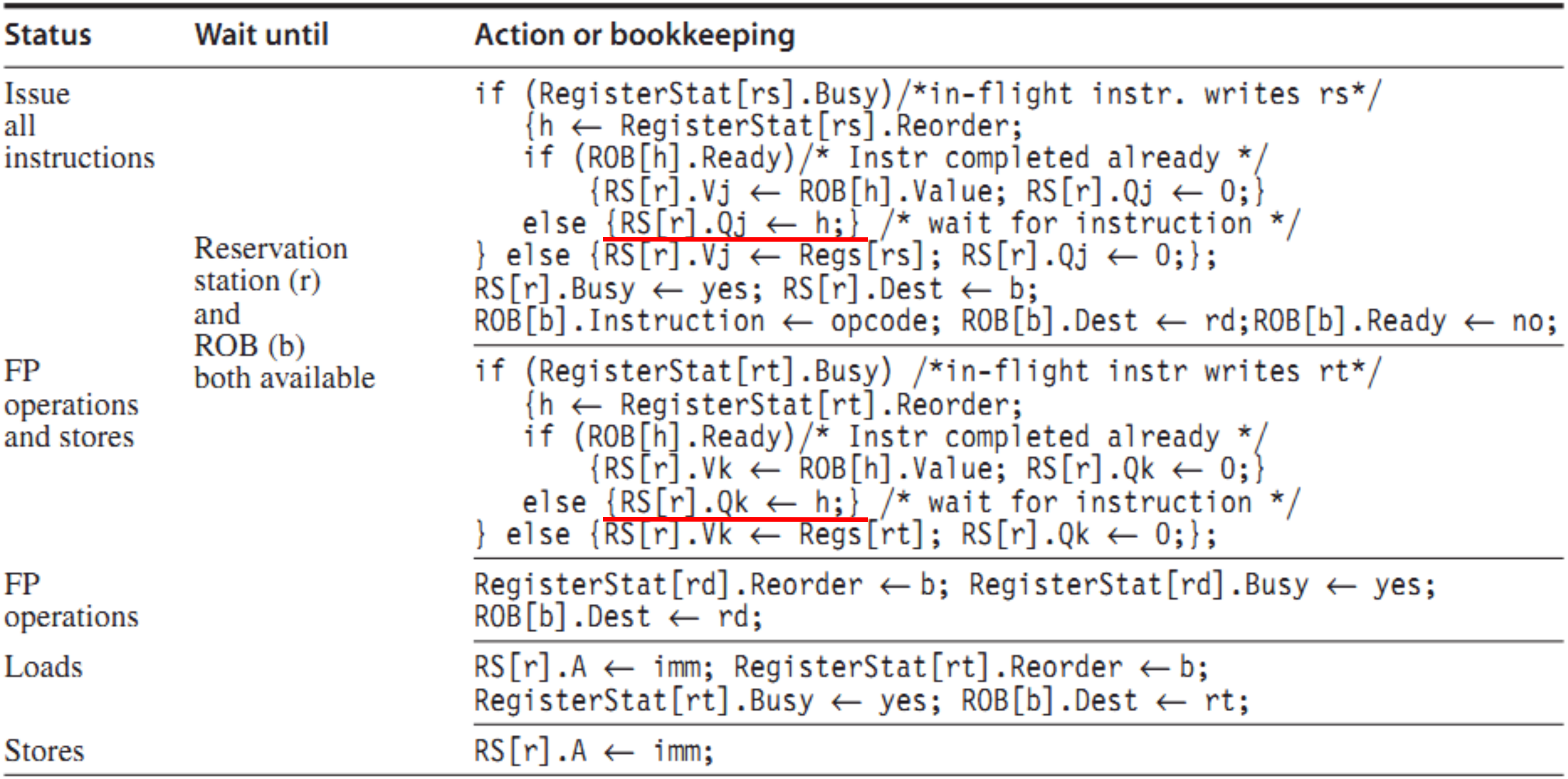
- **需要硬件缓存没有提交的指令结果: reorder buffer (ROB)**
  - 3 个域: 指令类型,目的地址, 值
  - Reorder buffer 可以作为操作数源 => 就像有更多的寄存器（与RS类似)
  - 当指令执行阶段完成后，用ROB的编号代替RS中的值
  - 增加指令提交阶段
  - ROB提供执行完成阶段和提交阶段的操作数
  - 一旦操作数提交，结果就写入寄存器
  - 在预测失败时，容易恢复推断执行的指令，或发生异常时，容易恢复状态



支持推断执行的 Tomasulo 算法的四阶段

- **1. Issue—get instruction from FP Op Queue**
  - 如果RS和ROB有空闲单元就发射指令。如果寄存器或ROB中源操作数可用，就将其发送到RS，目的地址的ROB编号也发送给RS
- **2. Execution—operate on operands (EX)**
  - 当操作数就绪后，开始执行。如果没有就绪，监测CDB，检查RAW相关
- **3. Write result—finish execution (WB)**
  - 将运算结果通过CDB传送给所有等待结果的FU以及ROB单元，标识RS可用
- **4. Commit—update register with reorder result**
  - 按ROB表中顺序，如果结果已有，就更新寄存器（或存储器），并将该指令从ROB表中删除
  - 预测失败或有中断时，刷新ROB
  - P191 Figure 3.14 (英文版)，P141 Figure 3-9 (中文版)

Issue



Status	Wait until	Action or bookkeeping

Issue
all
instructions
	Reservation station (r)
	and
	ROB (b)
	both available
		if (RegisterStat[rs].Busy)/*in-flight instr. writes rs*/
		   {h ← RegisterStat[rs].Reorder;
		   if (ROB[h].Ready)/* Instr completed already */
		      {RS[r].Vj ← ROB[h].Value; RS[r].Qj ← 0;}
		   else {RS[r].Qj ← h;} /* wait for instruction */
		} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0;};
		RS[r].Busy ← yes; RS[r].Dest ← b;
		ROB[b].Instruction ← opcode; ROB[b].Dest ← rd;ROB[b].Ready ← no;

FP
operations
and stores
		if (RegisterStat[rt].Busy) /*in-flight instr writes rt*/
		   {h ← RegisterStat[rt].Reorder;
		   if (ROB[h].Ready)/* Instr completed already */
		      {RS[r].Vk ← ROB[h].Value; RS[r].Qk ← 0;}
		   else {RS[r].Qk ← h;} /* wait for instruction */
		} else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0;};

FP
operations
		RegisterStat[rd].Reorder ← b; RegisterStat[rd].Busy ← yes;
		ROB[b].Dest ← rd;

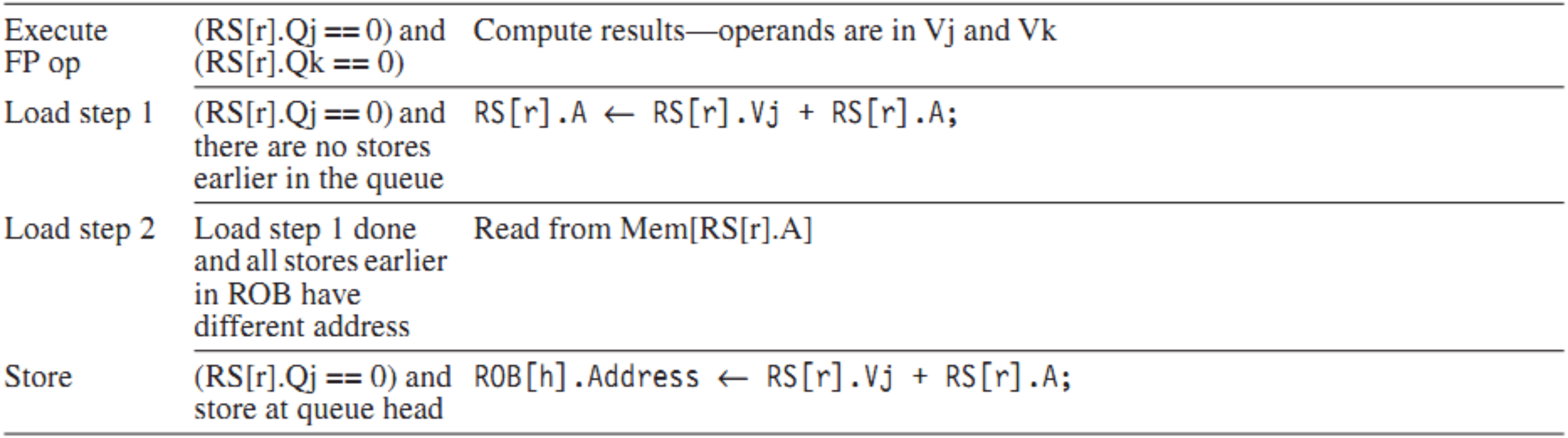
Loads		RS[r].A ← imm; RegisterStat[rt].Reorder ← b;
		RegisterStat[rt].Busy ← yes; ROB[b].Dest ← rt;

Stores		RS[r].A ← imm;

h：ROB中与当前指令相关的指令对应的ROB编号；

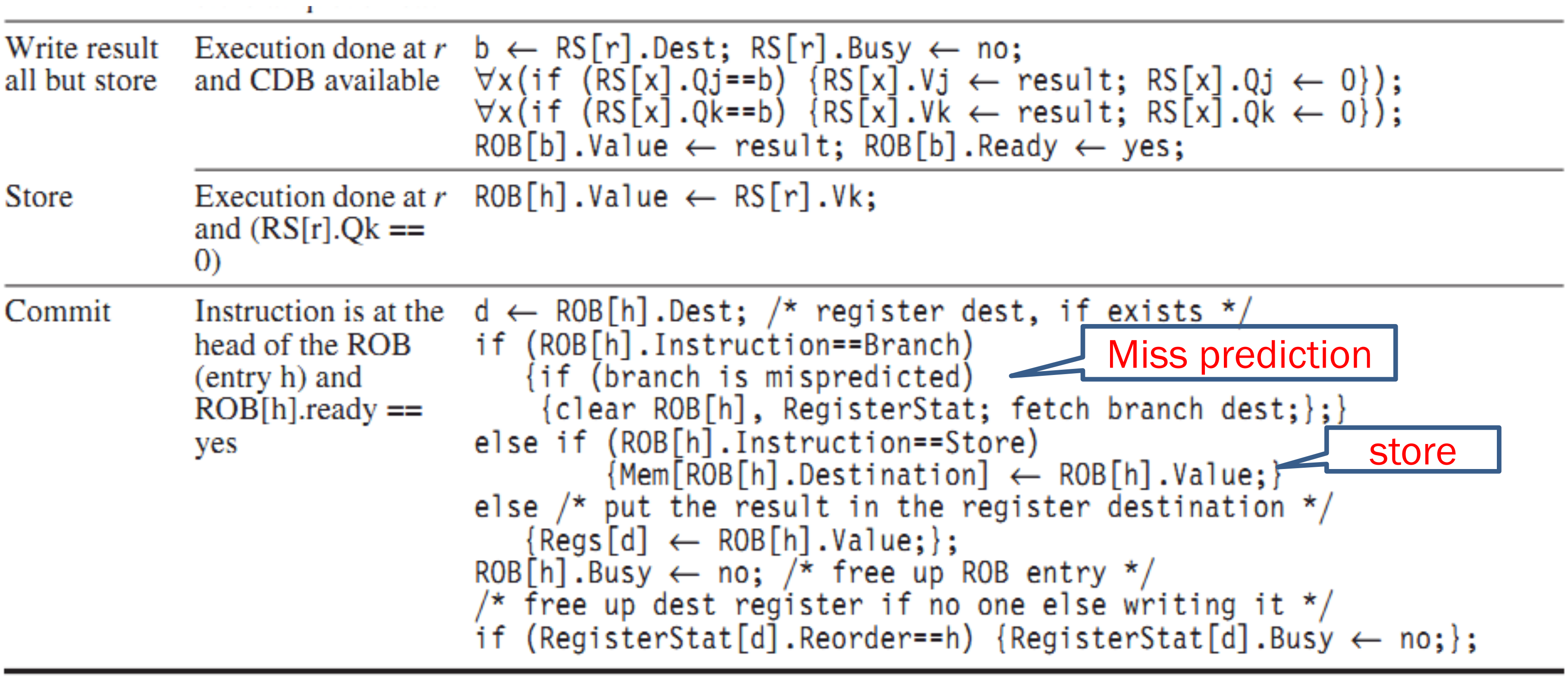
b：当前指令对应的ROB编号

Execute



Execute FP op	(RS[r].Qj == 0) and (RS[r].Qk == 0)	Compute results—operands are in Vj and Vk
Load step 1	(RS[r].Qj == 0) and there are no stores earlier in the queue	`RS[r].A ← RS[r].Vj + RS[r].A;`
Load step 2	Load step 1 done and all stores earlier in ROB have different address	Read from Mem[RS[r].A]
Store	(RS[r].Qj == 0) and store at queue head	`ROB[h].Address ← RS[r].Vj + RS[r].A;`

Write result & Commit



Write result all but store	Execution done at *r* and CDB available	b ← RS[r].Dest; RS[r].Busy ← no;
		∀x(if (RS[x].Qj==b) {RS[x].Vj ← result; RS[x].Qj ← 0});
		∀x(if (RS[x].Qk==b) {RS[x].Vk ← result; RS[x].Qk ← 0});
		ROB[b].Value ← result; ROB[b].Ready ← yes;

Store	Execution done at *r* and (RS[r].Qk == 0)	ROB[h].Value ← RS[r].Vk;

Commit	Instruction is at the head of the ROB (entry h) and ROB[h].ready == yes	d ← ROB[h].Dest; /* register dest, if exists */
		if (ROB[h].Instruction==Branch)
		   {if (branch is mispredicted)
		    {clear ROB[h], RegisterStat; fetch branch dest;};}
		else if (ROB[h].Instruction==Store)
		        {Mem[ROB[h].Destination] ← ROB[h].Value;}
		else /* put the result in the register destination */
		   {Regs[d] ← ROB[h].Value;};
		ROB[h].Busy ← no; /* free up ROB entry */
		/* free up dest register if no one else writing it */
		if (RegisterStat[d].Reorder==h) {RegisterStat[d].Busy ← no;};

Miss prediction

store

例如：

```
LD      F6, 34(R2)
LD      F2, 45(R3)
MULT    F0, F2, F4
SUBD    F8, F6, F2
DIVD    F10, F0, F6
ADDD    F6, F8, F2
```

假设：执行阶段的周期数

LD： 1 cycles MULT: 10 cycles

SUBD/ADDD: 2cycles   DIVD: 40 cycles

Tomasulo With Reorder Buffer-Cycle 0

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Station

	Busy	Address
Load1		
Load2		
Load3		

LD F6, 34(R2)
LD F2, 45(R3)
MULT F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

Entry	Busy	Instruction	State	Destination	Value
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	......	F30
0	Reorder#									
	Busy	No	No	No	No	No	No	No		No

Tomasulo With Reorder Buffer-Cycle 1

Reservation Station

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

LD F6, 34(R2)
LD F2, 45(R3)
MULT F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

Reorder Buffer

	Entry	Busy	Instruction	State	Destination	Value
Head	1	Yes	LD F6, 34(R2)	Issue	F6	
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					

	Busy	Address
Load1	Yes	34+Regs[R2]
Load2		
Load3		

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
1	Reorder#				#1					
	Busy	No	No	No	Yes	No	No	No		No

Tomasulo With Reorder Buffer-Cycle 2

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Station

	Busy	Address
Load1	Yes	34+Regs[R2]
Load2	Yes	45+Regs[R3]
Load3		

LD F6, 34(R2)
LD F2, 45(R3)
MULT F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

	Entry	Busy	Instruction	State	Destination	Value
Head	1	Yes	LD F6, 34(R2)	Ex1	F6	
tail	2	Yes	LD F2, 45(R3)	Issue	F2	
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
2	Reorder#		#2		#1					
	Busy	No	Yes	No	Yes	No	No	No		No

Tomasulo With Reorder Buffer-Cycle 3

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult		Regs[F4]	#2		#3
0	Mult2	No						

Reservation Station

	Busy	Address
Load1	No	
Load2	Yes	45+Regs[R3]
Load3		

	Entry	Busy	Instruction	State	Destination	Value
Head	1	Yes	LD F6,34(R2)	Write	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Ex1	F2	
tail	3	Yes	MULT F0,F2,F4	Issue	F0	
	4					
	5					
	6					
	7					
	8					
	9					
	10					

LD F6, 34(R2)
LD F2, 45(R3)
MULT F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	······	F30
3	Reorder#	#3	#2		#1					
	Busy	Yes	Yes	No	Yes	No	No	No		No

Tomasulo With Reorder Buffer-Cycle 4

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
2	Add1	Yes	SUB	Regs[F6]	Mem[45+regs[R3]]		#2	#4
0	Add2	No						
0	Add3	No						
10	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	No						

Reservation Station

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
Head	2	Yes	LD F2,45(R3)	Write	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Issue	F0	
tail	4	Yes	SUBD F8,F6,F2	Issue	F8	
	5					
	6					
	7					
	8					
	9					
	10					

	Busy	Address
Load1	No	
Load2	No	
Load3		

Reorder Buffer

LD F6, 34(R2)
LD F2, 45(R3)
MULT F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
4	Reorder#	#3	#2			#4				
	Busy	Yes	Yes	No	No	Yes	No	No		No

Tomasulo With Reorder Buffer-Cycle 5

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
1	Add1	Yes	SUB	Regs[F6]	Mem[45+regs[R3]]			#4
0	Add2	No						
0	Add3	No						
9	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

LD F6, 34(R2)
LD F2, 45(R3)
MULT F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex1	F0	
	4	Yes	SUBD F8,F6,F2	Ex1	F8	
Tail	5	Yes	DIVD F10,F0,F6	Issue	F10	
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
5	Reorder#	#3	#2 ?			#4	#5			
	Busy	Yes	No	No	No	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 6

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+regs[R3]]			#4
0	Add2	Yes	ADD		Regs[F2]	#4		#6
0	Add3	No						
8	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex2	F0	
	4	Yes	SUBD F8,F6,F2	Ex2	F8	
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Issue	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
6	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 7

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
2	Add2	Yes	ADD	#4	Regs[F2]			#6
0	Add3	No						
7	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex3	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Issue	F6	
	7					
	8					
	9					
	10					

	Busy	Address
Load1	No	
Load2	No	
Load3		

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
7	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 8

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
1	Add2	Yes	ADD	#4	Regs[F2]			#6
0	Add3	No						
6	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex4	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Ex1	F6	
	7					
	8					
	9					
	10					

仍不能提交(∵其前面的指令还未提交 ⇒ 顺序提交)

Reorder Buffer

	Busy	Address
Load1	No	
Load2	No	
Load3		

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
8	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 9

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	ADD	#4	Regs[F2]			#6
0	Add3	No						
5	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex5	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Ex2	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
9	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 10

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
4	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex6	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
10	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 11

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
3	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex7	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
11	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 12

Reservation Station

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
2	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

	Busy	Address
Load1	No	
Load2	No	
Load3		

Reorder Buffer

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex8	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
12	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 13

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
1	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex9	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
13	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 14

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Ex10	F0	
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	......	F30
14	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 15

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
40	Mult2	Yes	DIV	#2*Regs[F4]	Regs[F6]			#5

Reservation Station

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
Head	3	Yes	MULT F0,F2,F4	Write	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Issue	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

	Busy	Address
Load1	No	
Load2	No	
Load3		

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
15	Reorder#	#3			#6	#4	#5			
	Busy	Yes	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 16

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
39	Mult2	Yes	DIV	#2*Regs[F4]	Regs[F6]			#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
Head	4	Yes	SUBD F8,F6,F2	Write	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Ex1	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
16	Reorder#				#6	#4	#5			
	Busy	No	No	No	Yes	Yes	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 17

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
38	Mult2	Yes	DIV	#2*Regs[F4]	Regs[F6]			#5

Reservation Station

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Commit	F8	F6-#2
Head	5	Yes	DIVD F10,F0,F6	Ex2	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

	Busy	Address
Load1	No	
Load2	No	
Load3		

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
17	Reorder#				#6		#5			
	Busy	No	No	No	Yes	No	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 18

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
37	Mult2	Yes	DIV	#2*Regs[F4]	Regs[F6]			#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Commit	F8	F6-#2
Head	5	Yes	DIVD F10,F0,F6	Ex3	F10	
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
18	Reorder#				#6		#5			
	Busy	No	No	No	Yes	No	Yes	No		No

Continue……37 Cycles

Tomasulo With Reorder Buffer-Cycle 55

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2*Regs[F4]	Regs[F6]			#5

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Commit	F8	F6-#2
Head	5	Yes	DIVD F10,F0,F6	Ex40	F10	#3/F6
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	······	F30
54	Reorder#				#6		#5			
	Busy	No	No	No	Yes	No	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 56

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Commit	F8	F6-#2
Head	5	Yes	DIVD F10,F0,F6	Write	F10	#3/F6
Tail	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
56	Reorder#				#6		#5			
	Busy	No	No	No	Yes	No	Yes	No		No

Tomasulo With Reorder Buffer-Cycle 57

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Commit	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Commit	F10	#3/F6
Head	6	Yes	ADDD F6,F8,F2	Write	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
57	Reorder#				#6					
	Busy	No	No	No	Yes	No	No	No		No

Tomasulo With Reorder Buffer-Cycle 58

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Station

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Dest.	Value
	1	Yes	LD F6,34(R2)	Commit	F6	Mem[load1]
	2	Yes	LD F2,45(R3)	Commit	F2	Mem[load2]
	3	Yes	MULT F0,F2,F4	Commit	F0	#2*F4
	4	Yes	SUBD F8,F6,F2	Commit	F8	F6-#2
	5	Yes	DIVD F10,F0,F6	Commit	F10	#3/F6
Head	6	Yes	ADDD F6,F8,F2	Commit	F6	#4+F2
	7					
	8					
	9					
	10					

Reorder Buffer

Cycle		F0	F2	F4	F6	F8	F10	F12	……	F30
58	Reorder#									
	Busy	No	No	No	No	No	No	No		No

Tomasulo With Reorder Buffer-Summary

Issue、 commit 是按序

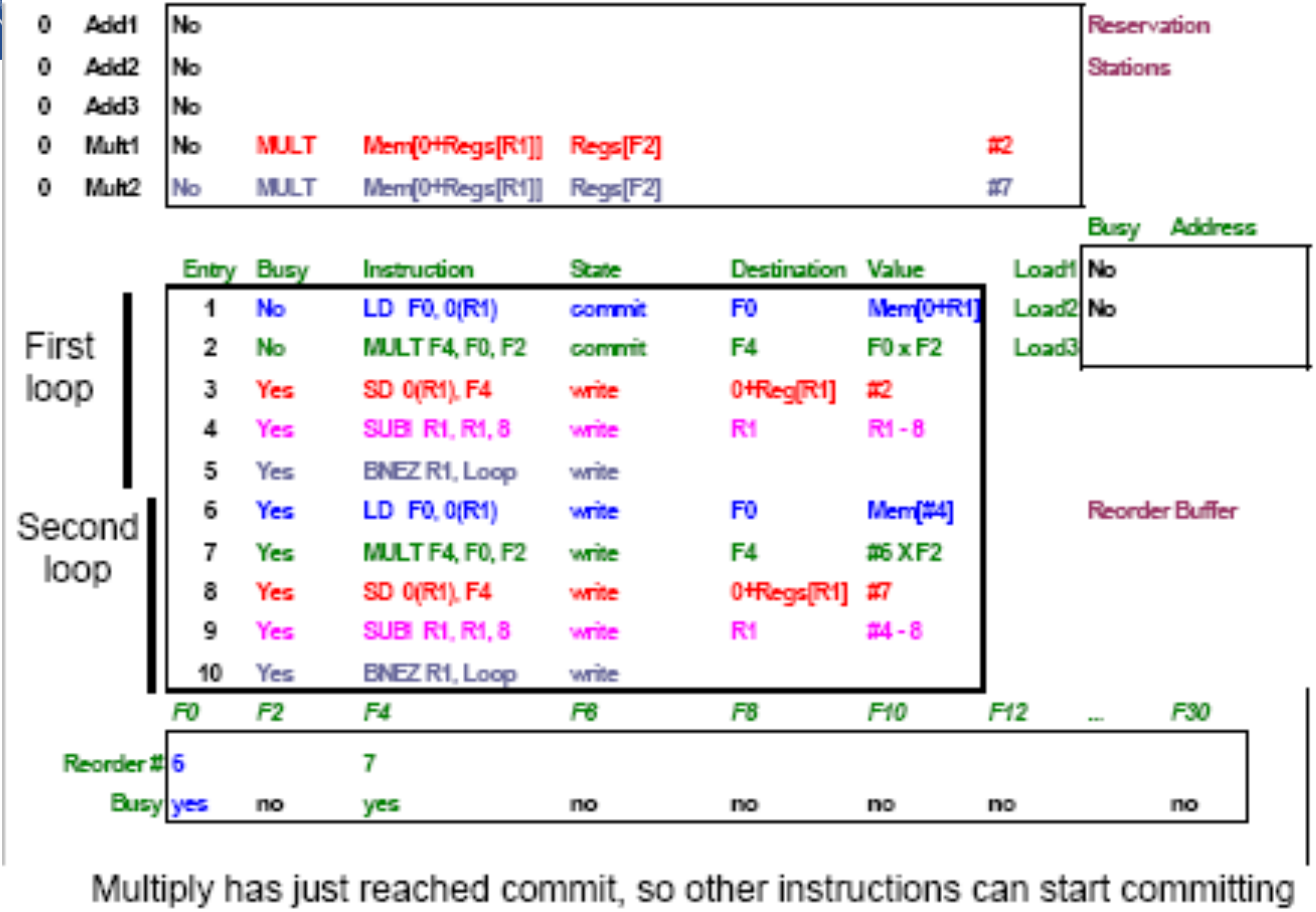
Exec comp 和 Writeback 是乱序执行

Instruction	Issue	Exec Comp	WriteBack	Commit
LD F6,34(R2)	1	2	3	4
LD F2,45(R3)	2	3	4	5
MULT F0,F2,F4	3	5~14	15	16
SUBD F8,F6,F2	4	5~6	7	17
DIVD F10,F0,F6	5	16~55	56	57
ADDD F6,F8,F2	6	8~9	10	58

In-order Issue/Commit Out-of-Order Execution/WriteBack

是不是可以进一步优化？

Example of Speculative State of Reorder Buffer



Reservation Stations

0	Add1	No				
0	Add2	No				
0	Add3	No				
0	Mult1	No	MULT	Mem[0+Regs[R1]]	Regs[F2]	#2
0	Mult2	No	MULT	Mem[0+Regs[R1]]	Regs[F2]	#7

	Busy	Address
Load1	No	
Load2	No	
Load3		

Reorder Buffer

	Entry	Busy	Instruction	State	Destination	Value
First loop	1	No	LD F0, 0(R1)	commit	F0	Mem[0+R1]
	2	No	MULT F4, F0, F2	commit	F4	F0 x F2
	3	Yes	SD 0(R1), F4	write	0+Reg[R1]	#2
	4	Yes	SUBI R1, R1, 8	write	R1	R1 - 8
	5	Yes	BNEZ R1, Loop	write		
Second loop	6	Yes	LD F0, 0(R1)	write	F0	Mem[#4]
	7	Yes	MULT F4, F0, F2	write	F4	#6 X F2
	8	Yes	SD 0(R1), F4	write	0+Regs[R1]	#7
	9	Yes	SUBI R1, R1, 8	write	R1	#4 - 8
	10	Yes	BNEZ R1, Loop	write		

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	6		7						
Busy	yes	no	yes	no	no	no	no		no

Multiply has just reached commit, so other instructions can start committing

使用ROB保持机器的精确状态

- **ROB维持了机器的精确状态，允许投机执行**
  - 直到确认无异常 然后进入提交阶段
  - 直到确定分支预测正确进入提交阶段
  - 如果有异常或预测错误
    - 刷新ROB、RS和寄存器结果状态表
- **存储器操作使用类似的方法**
  - Memory Ordering Buffer（MOB）
    - Store操作的结果先存放到MOB中，然后提交阶段按存储操作的程序序提交

Memory Disambiguation：处理对存储器引用的数据相关

- **Question: 给定一个指令序列，store，load 这两个操作是否有关？即下列代码是否有相关问题?**

```
Eg:     st  0(R2),R5
        ld  R6,0(R3)
```

- **我们是否可以较早启动ld?**
  - Store的地址可能会延迟很长时间才能得到.
  - 我们也许想在同一个周期开始这两个操作的执行.
- **两种方法:**
  - No Speculation: 不进行load操作，直到我们确信地址 0(R2) □ 0(R3)
  - Speculation: 我们可以假设他们相关还是不相关 (called "dependence speculation")，如果推测错误通过ROB来修正
- **参考书：Gonzalez, A., et al. (2011). "Processor Microarchitecture: An Implementation Perspective." Synthesis Lectures on Computer Architecture #12, Morgan & Claypool Publishers**

Memory Disambiguation

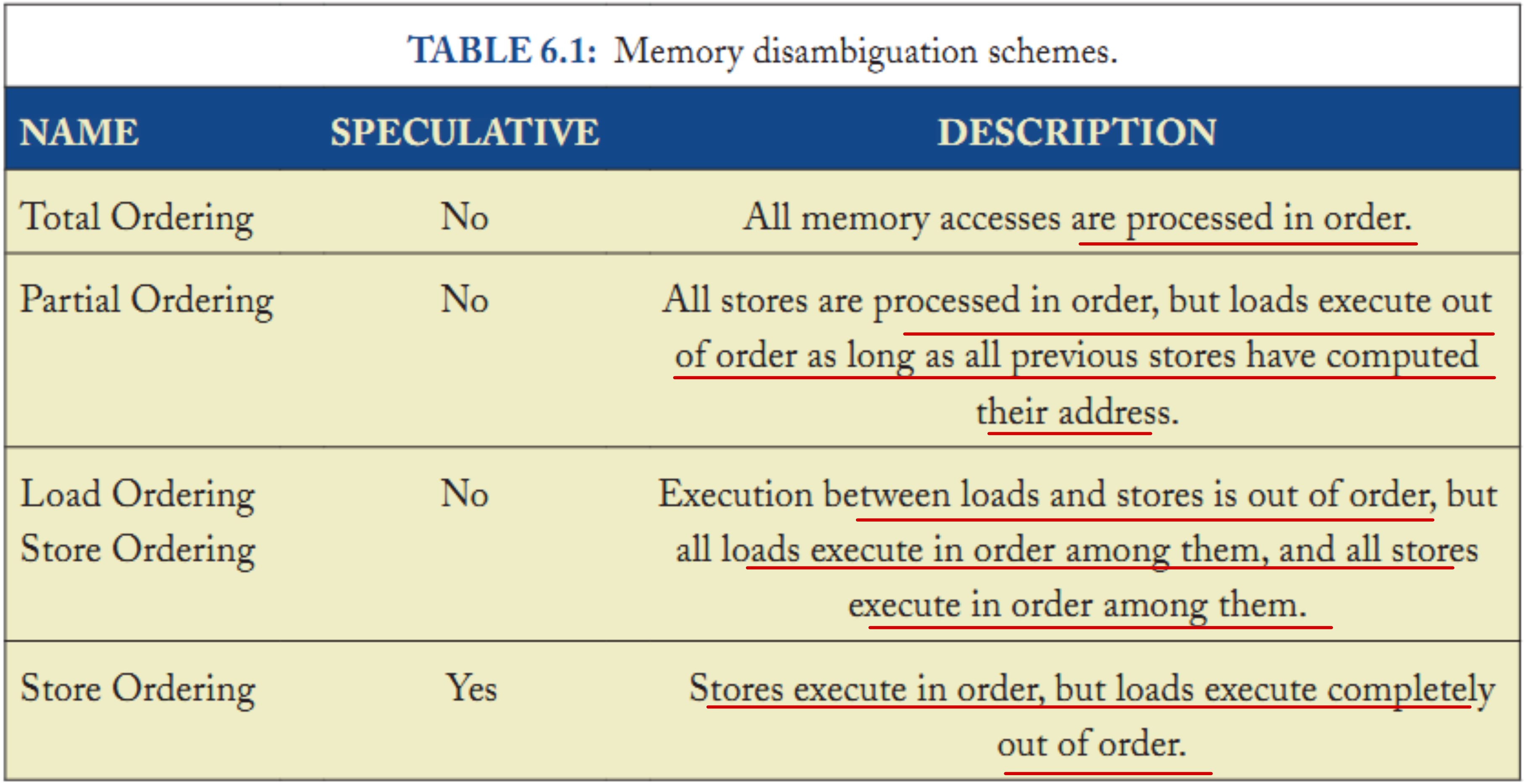


TABLE 6.1: Memory disambiguation schemes.

NAME	SPECULATIVE	DESCRIPTION
Total Ordering	No	All memory accesses are processed in order.
Partial Ordering	No	All stores are processed in order, but loads execute out of order as long as all previous stores have computed their address.
Load Ordering Store Ordering	No	Execution between loads and stores is out of order, but all loads execute in order among them, and all stores execute in order among them.
Store Ordering	Yes	Stores execute in order, but loads execute completely out of order.

- **非投机方式的基本原则：当前存储器指令之前的store指令计算存储器地址后，才能执行当前的存储器操作**