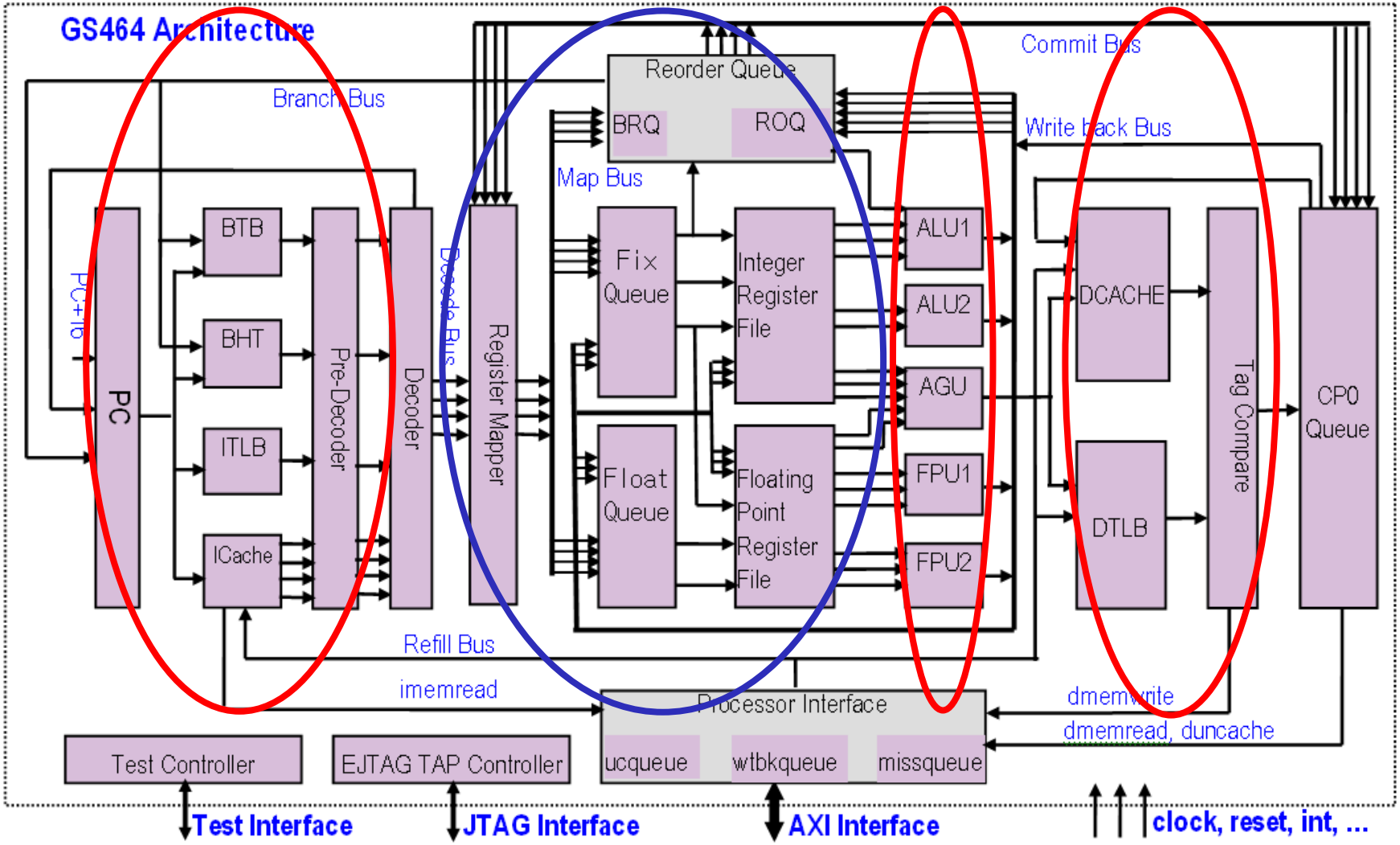
计算机体系结构

胡伟武、汪文祥

提高流水线效率的技术

- 指令流水线
  - 多发射：多车道
  - 动态调度：允许超车
- 喂饱“饥饿”的运算器
  - 转移猜测：提供足够的指令
  - 存储管理：提供足够的数据
  - 冯诺依曼结构：存储程序和顺序执行

龙芯2号处理器核结构图



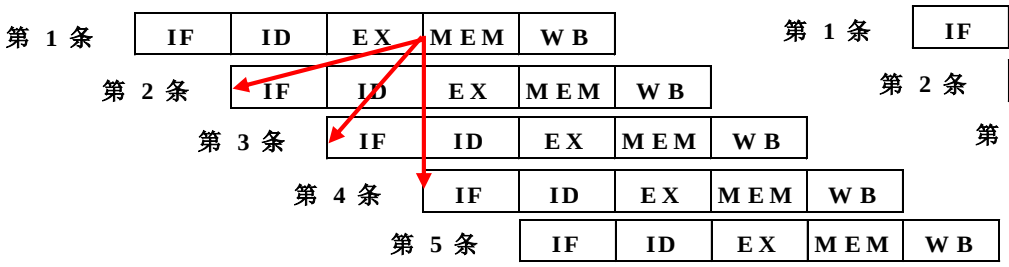
转移预测

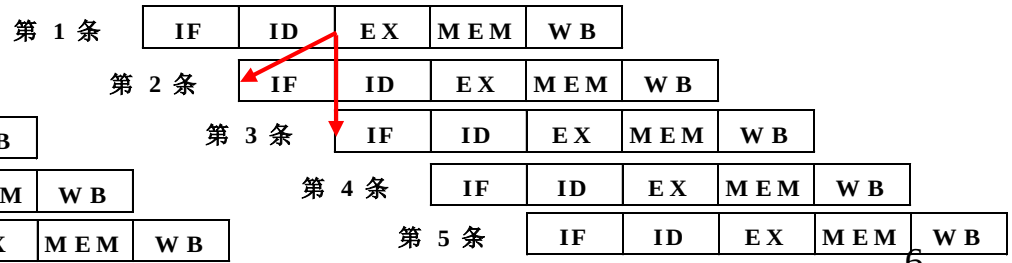
- 转移指令
- 程序的转移行为
- 软件方法解决控制相关
- 硬件动态转移预测
- 常见处理器的转移猜测

转移指令

控制相关

- 如果转移指令计算下一条指令地址在EX阶段计算，下一条指令等2拍
- 使用专门的地址运算部件把地址计算提前到译码阶段可以少等一拍
- 使用一个delay slot可以不用等待
  - 多发射情况下延迟槽成为需要专门照顾的负担





延迟槽指令的例子

```
1 Loop:   LD     F0,0(R1)     ;F0=vector element
2         stall
3         ADDD   F4,F0,F2     ;add scalar in F2
4         stall
5         stall
6         SD     0(R1),F4     ;store result
7         SUBI   R1,R1,8      ;decrement pointer 8B (DW)
8         BNEZ   R1,Loop      ;branch R1!=zero
9         stall                ;delayed branch slot
```

```
1 Loop:   LD     F0,0(R1)
2         stall
3         ADDD   F4,F0,F2
4         SUBI   R1,R1,8
5         BNEZ   R1,Loop      ;delayed branch
6         SD     8(R1),F4     ;altered when move past SUBI
```

转移指令对性能的影响

- 分支指令的影响是开发指令级并行性的重要障碍
  - 一条指令流中，平均每5-7条指令中就有一条是分支指令，也即基本块大小为5-7条指令
- 增大发射宽度
  - 在发射宽度为n的处理器中，遇到分支指令的速度也快了n倍
- 增加流水线深度
  - 流水线越深，处理分支指令所需要的时钟周期数就越多

一个例子

- 假设平均每8条指令中有一条转移指令，某处理器采用4发射结构，第10级流水解决转移地址相关（即第10级流水算出转移方向和目标）
  - 在A系统中不进行转移预测，遇到转移指令就等待，指令带宽浪费36/(36+8)=82%；
  - 在B系统中进行简单的转移预测，转移猜错率为50%，那么平均每16条指令预测错误一次，指令带宽浪费36/(36+16)=75%
  - 在C系统中，转移指令猜错率为10%，那么平均每80条指令预测错误一次，指令带宽浪费36/(36+80)=31%；
  - 在D系统中，转移指令猜错率为4%，平均每200条指令预测错误一次，取指令带宽浪费36/(36+200)=15%。

转移指令的属性

- 条件转移与无条件转移
  - 条件转移：需等待条件确定后才能取指，程序中多数转移指令是条件转移指令
  - 无条件转移：不用判断条件就转移，如call/return
- 直接转移与间接转移
  - 直接转移：转移目标根据指令内容直接得出
  - 间接转移：转移目标在寄存器中
- 相对转移与绝对转移
  - 相对转移：转移目标为当前PC值加上偏移量
  - 绝对转移：转移目标由指令或寄存器内容直接给出
- 理论上有8种组合：实际上不实现所有组合

MIPS指令系统的转移指令

- 条件转移
  - 都是直接、相对转移，如**BEQ，BGEZ**
- 无条件直接转移
  - 转移目标为**{PC[31:28],IR[25:0],2'b0}**
  - **J**与**JAL**
- 无条件间接转移
  - 转移目标为寄存器内容
  - **JR**与**JALR**
- **MIPS**转移指令特点
  - **likely**与非**likely**，如**BEQL，BGEZL**
  - 没有**Call/Return**，通过**link**类转移指令实现**call**功能，如**JAL**

程序的转移行为

SPEC CPU2000转移指令统计（1）

	程序名	语言	程序功能	动态无条件分支频度（%）	动态条件分支频度（%）
SPECint2000	**gzip**	C	压缩	3.05	6.73
	vpr	C	FPGA电路布局布线	2.66	8.41
	gcc	C	C程序语言编译器	0.77	4.29
	crafty	C	游戏：象棋	2.79	8.34
	parser	C	字处理	4.78	10.64
	perlbmk	C	perl编程语言	4.36	9.64
	gap	C	群论，翻译	1.41	5.41
	vortex	C	面向对象数据库	5.73	10.22
	bzip2	C	压缩	1.69	11.41
	twolf	C	布局布线模拟器	1.95	10.23
	平均			2.919	8.532

龙芯2号统计结果

SPEC CPU2000转移指令统计（2）

	程序名	语言	程序功能	动态无条件分支频度(%)	动态条件分支频度(%)
SPECfp2000	**wupwise**	F77	物理：量子色动学	2.02	7.87
	swim	F77	浅水模拟	0.00	1.29
	mgrid	F77	多网格求解器：3 维势场	0.00	0.28
	applu	F77	差分方程	0.01	0.42
	mesa	C	3 维图形库	2.91	5.83
	art	C	图像识别/神经网络	0.39	10.91
	equake	C	地震波传导模拟	6.51	10.66
	facerec	F90	图像处理：脸部识别	1.03	2.45
	ammp	C	计算化学	2.69	19.51
	lucas	F90	数论/素数测试	0.00	0.74
	fma3d	F90	有限元碰撞模拟	4.25	13.09
	apsi	F77	气象学：污染分布	0.51	2.12
	平均			1.69	6.26

龙芯2号统计结果

转移指令间距离统计（1）

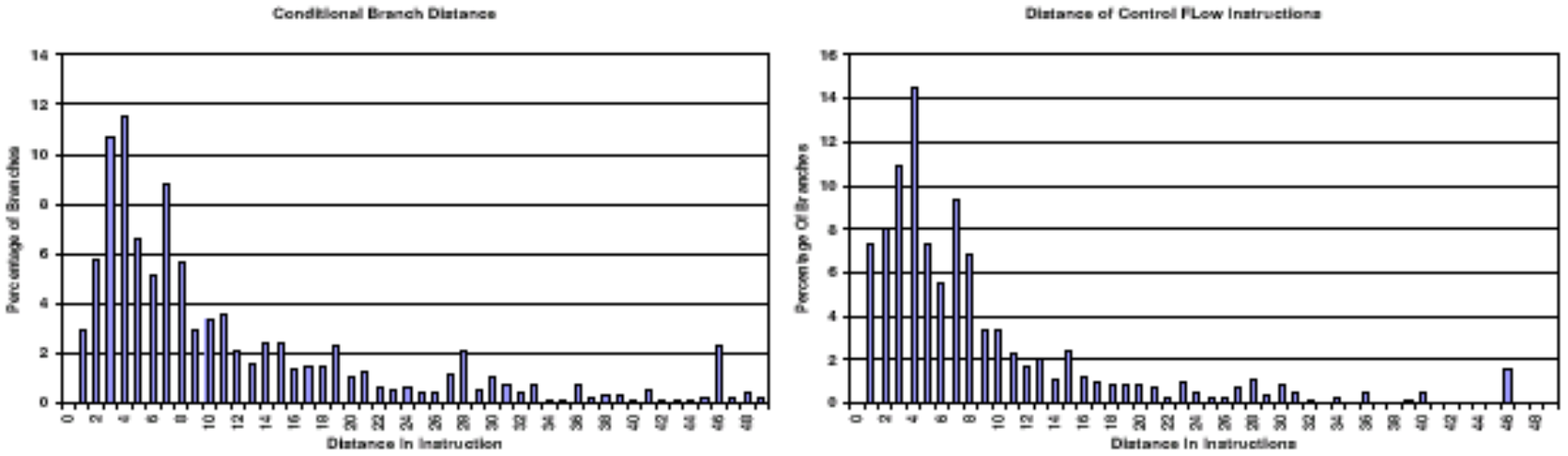
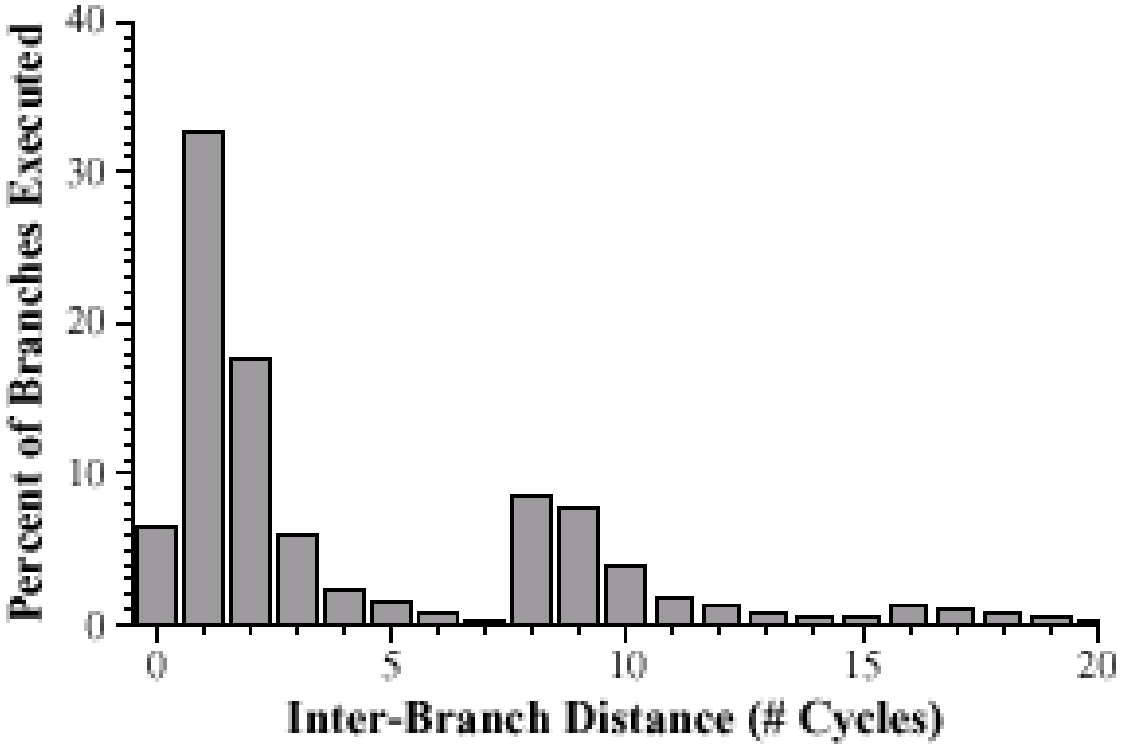


Figure 8. (a) Average distance (in terms of instructions) between conditional branches. (b) Average distance between control-flow instructions (conditional branches plus unconditional jumps).

发射宽度增加，分支间的动态距离就会越小，分支预测的延迟对性能的影响就越大，当要求更高的发射宽度时，需要每个周期做不止一个预测

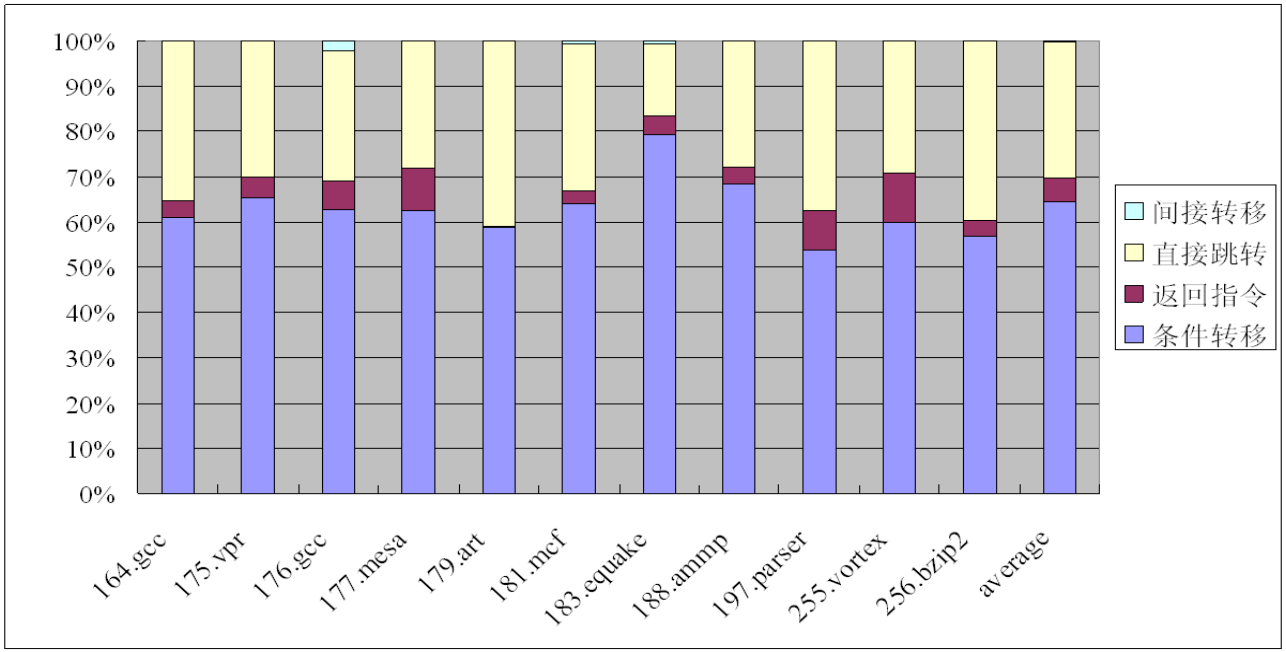
转移指令间距离统计（2）

- **Inter-branch distance(cycles)**
  模拟环境：4发射乱序机器 @SPECint2000



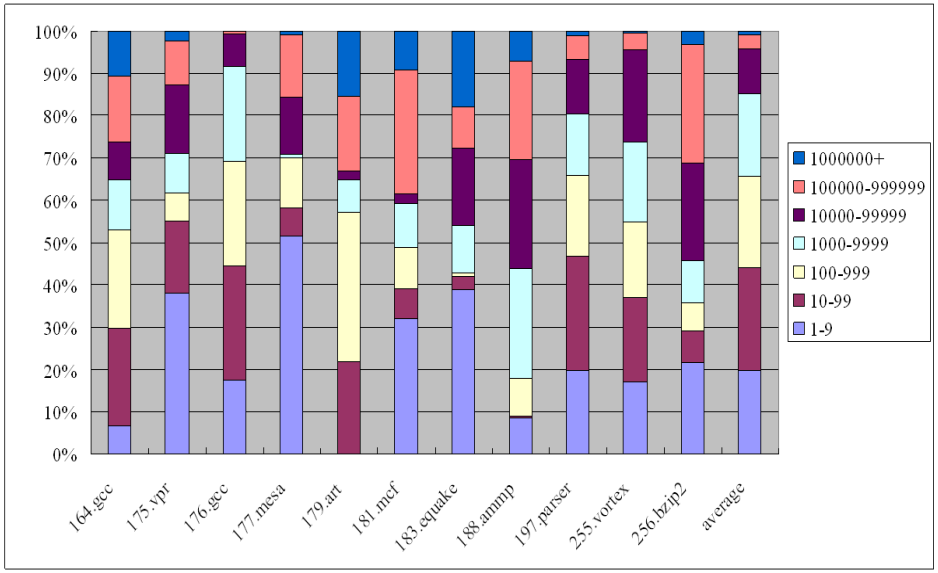
不同类型转移指令分布

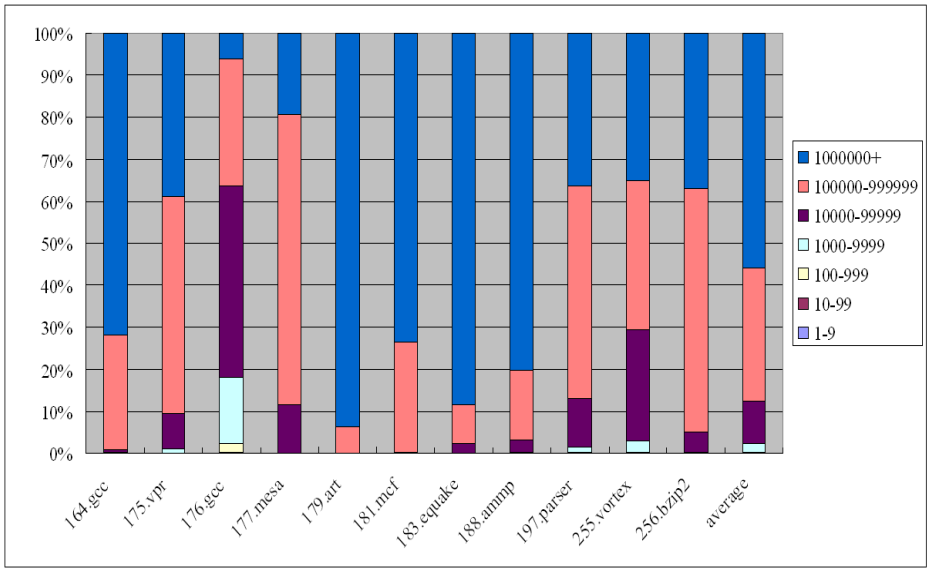
- 龙芯2号上部分SPEC CPU2000程序中转移指令分布
  - 1%是间接分支指令，采用BTB预测
  - 5%是返回指令，返回地址栈RAS预测
  - 29%是无条件立即跳转指令，跳转地址在指令中
  - 65%是条件分支，大量的工作是进行条件分支预测



转移指令的执行频率分布

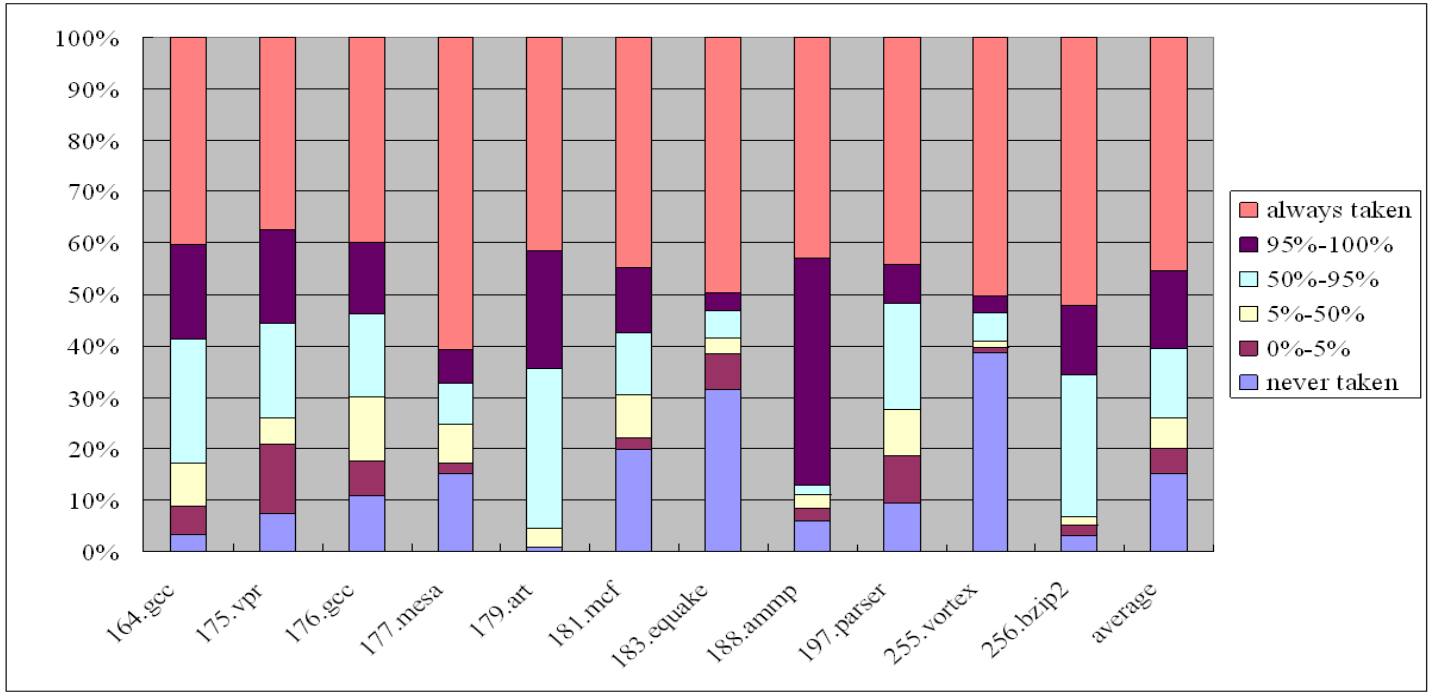
- 龙芯2号上部分SPEC CPU2000程序转移指令执行频率分布
  - 左图表示执行一定次数的转移指令在所有转移指令中的静态分布，右图表示执行一定次数的转移指令在所有转移指令中的动态分布
  - 所有转移指令的44%仅仅执行99次或者更少，这44%的转移指令占所有转移指令总执行次数的0.03%
  - 只有4.2%的转移指令执行了超过100000次或者更多，占所有转移指令总执行次数的87%（或者说，只有14.7%的转移指令超过了10000次，占所有转移指令总执行次数的97%）。





转移成功率分布

- 龙芯2号上部分SPEC CPU2000程序转移指令的跳转成功率
  - 45%的指令总是跳转，15%的指令总是不跳转
  - 20%的指令跳转的几率小于5%或者大于95%，也就是说，80%的转移指令具有强烈的一个跳转方向，具有挑战性的工作是预测其余20%转移指令的跳转方向。



分支的可预测性

- 利用单个转移指令的重复性
  - 基于模式的预测方法
- 利用不同转移指令之间的关系
  - 基于相关的预测方法
  - 方向相关、路径相关

利用单个转移指令重复性

- 循环型分支
  - for型循环：TT...TN（成功n次后跟一次不成功 ）
  - while型循环：NN...NT（不成功n次后跟一次成功）
- 周期重复模式型分支
  - 定长重复模式类分支：$\{pat\}^q$ ， $|pat|=k$ （每隔k个分支模式就重复一次）
  - 块模式类分支：$\{T^n\ N^m\}^q$ 成功n次，不成功m次，如此循环

利用转移指令之间的方向相关

- 两个分支的条件（完全或部分）基于相同或相关的信息
- 第二个分支的结果基于第一个分支的结果产生

Example:

Y：if (cond1)
⋮
X：if (cond1 AND cond2)

Z：if (cond1)
⋮
Y：if (cond2)
⋮
X：if (cond1 AND cond2)

Y：if (cond1) a = 2
⋮
X：if (a = = 0)

利用转移指令之间的路径相关

- 如果一个分支是通向当前分支的前n条分支之一，则称该分支处在当前分支的路径之上（Y和Z在通往X的路径上）
- 处在当前分支路径上的分支与当前分支结果之间的相关称为路径相关

Example:

Z： if (NOT (cond1))
Y： else if (NOT(cond2))
V： else if (cond3)
⋮
X： if (cond1 AND cond2)

分支指令行为小结

- 分支指令是很频繁的
- 分支指令有较好的局部性
- 分支指令具有可预测性

软件方法解决控制相关

解决转移条件相关的方法

- 阻塞
  - 等待直到转移条件确定
- 用延迟槽容忍延迟
  - 延迟槽指令来源
- 编译器优化
  - 循环：循环展开减少转移指令、软流水减少阻塞
  - 分支：全局代码调度（越过分支调度指令）
  - 函数调用：**inline**
- 转换为数据相关
  - 条件指令、谓词
- 硬件转移预测
  - 转移条件未确定时预测转移是否成功
  - 静态与动态预测

利用延迟槽

- 延迟槽指令的来源
  - 来自转移指令前：肯定执行
  - 来自转移目标地址：转移成功才执行
  - 来自转移不成功地址：转移不成功才执行
- 单延迟槽的编译效果
  - 能为 60% 左右的转移延迟槽找到有效操作
  - 大约80%的延迟槽指令用于有效计算
  - 因此大约50% (60% x 80%) 的延迟槽操作用于有效计算
- 延迟槽的限制
  - 超流水情况下一条延迟槽不够
  - 多发射情况下延迟槽反而成为需要特殊照顾的兼容负担

软件循环展开消除控制相关

```
LD      F0    0    R1
MULTD   F4    F0   F2
SD      F4    0    R1
LD      F0    0    R1
MULTD   F4    F0   F2
SD      F4    0    R1
```

```
LD      F0    0    R1
MULTD   F4    F0   F2
SD      F4    0    R1
LD      F6    0    R1
MULTD   F8    F6   F2
SD      F8    0    R1
```

```
LD      F0    0    R1
LD      F6    0    R1
MULTD   F4    F0   F2
MULTD   F8    F6   F2
SD      F4    0    R1
SD      F8    0    R1
```

- 软件展开两个循环
  - 循环展开
  - 寄存器重命名
  - 变换次序
- 软件循环展开的不足
  - 有些循环不好展开（如循环次数不定的循环）
  - 增加指令CACHE的负担

循环的数据相关

- 循环内相关：S2使用同一次循环中S1计算的A[i+1].
- 循环间（loop-carried）相关：本次循环计算的A[i+1]/B[i+1]将被下一次循环使用。
- 循环内相关导致一个循环体内的多条指令不能并行执行，循环间相关导致多个循环体不能并行执行

```
For (i=0;i<100;i++){
        A[i+1] = A[i] + C[i];  //S1
        B[i+1] = B[i] + A[i+1];  //S2
}
```

```
For (i=0;i<100;i++){
        A[i] = A[i] + B[i];
        B[i+1] = C[i] + D[i];
}
```

```
A[0] = A[0] + B[0];
For (i=0;i<99;i++){
        B[i+1] = C[i] + D[i];
        A[i+1] = A[i+1] + B[i+1];
}
B[100] = C[100] + D[100];
```

循环展开的条件

- 数组元素相关的判断
  - 仿射（Affine）和非仿射（nonaffine）
  - X[a*i+b]
  - X[Y[i]]
- 最大公约数法
  - X[a*j+b] = X[c*k+d]
  - GCD(a,c)整除(d-b)则有相关
- 名字相关的消除
  - 重命名技术
- 指针相关的判断
  - 只有一些经验的方法

```
For (i=0;i<100;i++){
        A[2*i+3] = A[2*i] * 5.0;
}
```

```
For (i=0;i<100;i++){
        Y[i] = X[i] / c;
        X[i] = X[i] + c;
        Z[i] = Y[i] + c;
        Y[i] = c – Y[i];
}
```

```
For (i=0;i<100;i++){
        T[i] = X[i] / c;
        X1[i] = X[i] + c;
        Z[i] = T[i] + c;
        Y[i] = c – T[i];
}
```

软流水

- 新循环体的每个操作来自不同的循环体，以分开数据相关的指令，相当于软件的Tomasulo算法
- 符号级循环展开，比真正循环展开代码开销小

```
Loop:   L.D     F0,0(R1)
        ADD.D   F4,F0,F2
        S.D     F4,0(R1)
        DADDUI  R1,R1,#-8
        BNE     R1,R2,Loop
```

```
Iteration i:      L.D     F0,0(R1)
                  ADD.D   F4,F0,F2
                  S.D     F4,0(R1)
Iteration i+1:    L.D     F0,0(R1)
                  ADD.D   F4,F0,F2
                  S.D     0(R1),F4
Iteration i+2:    L.D     F0,0(R1)
                  ADD.D   F4,F0,F2
                  S.D     F4,0(R1)
```

```
Loop:   S.D     F4,16(R1)       ;stores into M[i]
        ADD.D   F4,F0,F2        ;adds to M[i-1]
        L.D     F0,0(R1)        ;loads M[i-2]
        DADDUI  R1,R1,#-8
        BNE     R1,R2,Loop
```

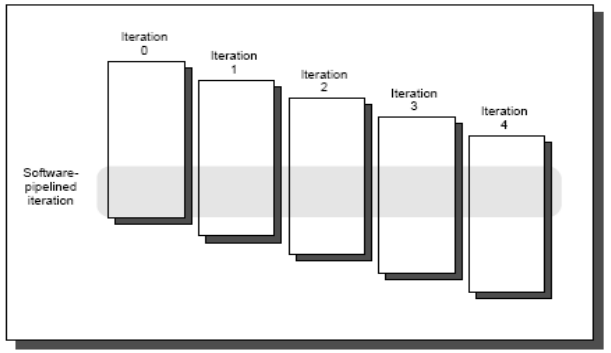


FIGURE 4.6 **A software-pipelined loop chooses instructions from different loop iterations, thus separating the dependent instructions within one iteration of the original loop.** The start-up and finish-up code will correspond to the portions above and below the software-pipelined iteration.

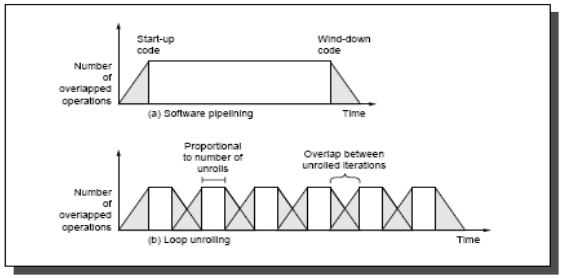
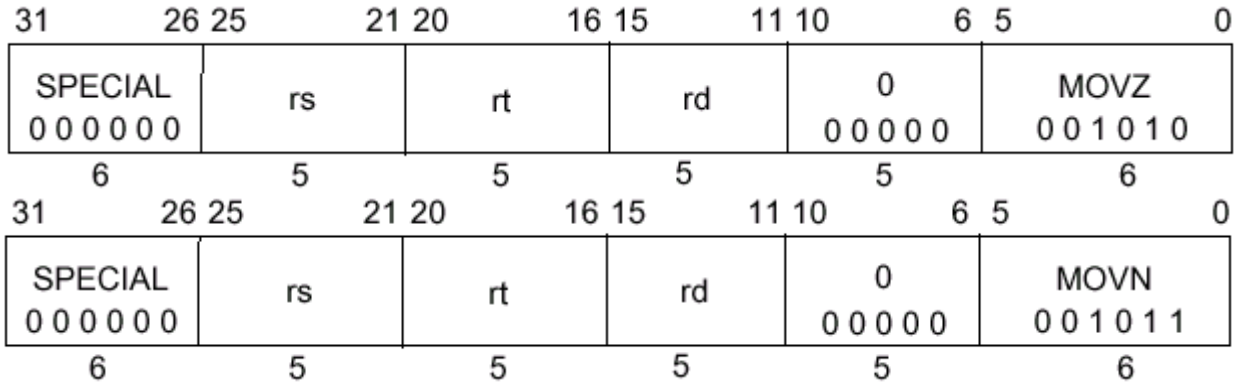


FIGURE 4.7 **The execution pattern for (a) a software-pipelined loop and (b) an unrolled loop.** The shaded areas are the times when the loop is not running with maximum overlap or parallelism among instructions. This occurs once at the beginning and once at the end for the software-pipelined loop. For the unrolled loop it occurs *m*/*n* times if the loop has a total of *m* iterations and is unrolled *n* times. Each block represents an unroll of *n* iterations. Increasing the number of unrollings will reduce the start-up and clean-up overhead. The overhead of one iteration overlaps with the overhead of the next, thereby reducing the impact. The total area under the polygonal region in each case will be the same, since the total number of operations is just the execution rate multiplied by the time.

把控制相关转换成数据相关

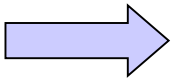
- 把条件转移指令转换为条件执行
  - **if (x) then A = B op C else NOP**
  - 只有条件为真时才写结果，为假时不写结果也不发生例外
  - **RISC系统如Alpha, MIPS, PowerPC, SPARC都增加了条件MOVE指令; PA-RISC的nullification**
  - **EPIC: 使用64个1位的谓词寄存器来选择是否写执行结果**
- 条件指令的缺点
  - 条件为假时仍需要1拍，占用发射槽和功能部件
  - 条件未确定仍需要在执行前等待，转移猜测反而在执行后
  - 条件复杂时会降低效率，因为条件在执行时才确定



条件指令举例

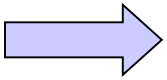
- 假设转移指令没有延迟槽
- 条件指令可消除简单的条件转移，对取绝对值等操作有用
- 条件指令仍要在执行前等待条件，注意例外的处理

```
//if (A=0) {S=T;}
        BNEZ R1,L
        ADDU R2,R3,R0
L:
```



```
CMOVZ R2,R3,R1
```

```
LW      R1,40(R2)       ADD R3,R4,R5
                        ADD R6,R3,R7
BEQZ    R10,L
LW      R8,0(R10)
LW      R9,0(R8)
```



```
LW      R1,40(R2)       ADD R3,R4,R5
LWC     R8,0(R10),R10   ADD R6,R3,R7
BEQZ    R10,L
LW      R9,0(R8)
```

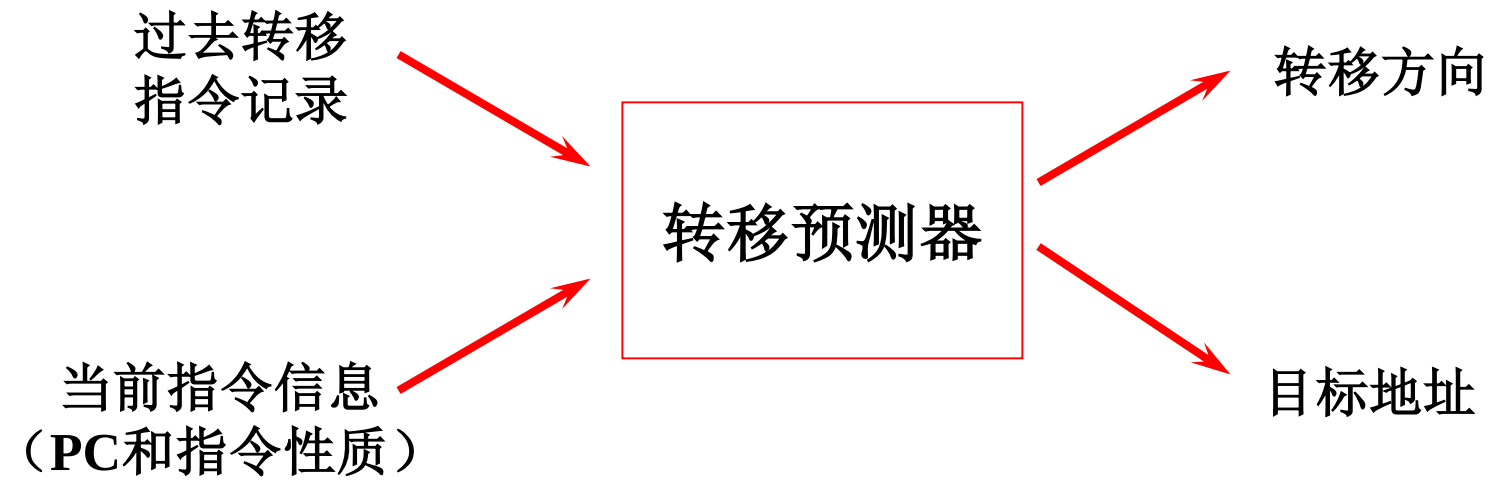
硬件转移预测

硬件转移预测基本思路

- 在取指或译码阶段预测转移是否成功以及转移目标进行后续指令的取指
  - 以减少指令流水线由于控制相关而堵塞
- 在执行阶段判断转移预测是否正确
  - 如果猜测正确，则正常提交
  - 如果猜测错误，则取消该转移指令及其后续指令

硬件转移预测基本原理

- 猜测依据
  - 当前指令的地址（PC）和性质（是否转移指令）
  - 过去转移指令历史记录
- 猜测内容
  - 转移方向、转移目标地址



分支处理机制的性能取决于

- 预测精度（BPA）=> 设计好的预测器
  - 预测精度越高，能抽取的并行性就越多
- 预测正确所付的代价：转到目标地址处执行所需的延迟
  - 译码时根据IR内容预测：有一拍的延迟槽，在4发射情况下有4条指令的延迟槽
  - 取指时根据PC预测：没有延迟槽，需要BTB/Trace Cache等机制
  - MIPS R10000无BTAC，MIPS R12000有32项BTAC
- 预测错所付的代价
  - 尽量提前执行转移操作
  - Pentium II/III和Alpha 21264重新刷新流水线需要11周期以上

转移预测关键技术

- 如何保证准确的预测：根据记录的历史进行预测
  - 如何记录转移历史，记录哪些转移历史
  - 记录多少转移历史
  - 何时更新：更新太早，转移指令也可能被取消；更新太晚，导致转移历史不准确
- 如何在取消猜测执行的操作时保证现场精确性
  - 增加提交流水级，在提交时修改寄存器和内存
  - I/O指令的猜测执行难以取消
- 如何识别流水线中的指令哪些需要取消，哪些不要取消
  - 例外取消一般在提交时，取消所有后续指令
  - 转移取消一般在执行后，只取消部分指令
- 延迟槽指令的处理
- 每个周期多个分支预测
  - 每周期1个预测，基本可满足4-6 发射需要的取指带宽

静态/动态转移预测

- 静态预测：总是预测转移成功或总是预测转移不成功
  - 预测转移成功：较精确，计算转移地址需要**delay slot**
  - 预测转移不成功：直接用**PC+4**
- 动态预测：根据转移指令执行历史进行预测
  - 复杂预测技术：精确、控制复杂
- 混合预测：利用编译器的提示，结合动态和静态预测

局部转移预测

- 独立考虑单个循环的历史记录，寻找其中的重复性规律，并根据该规律预测未来的转移行为
- 对于重复性特征明显的转移指令（如循环）效果好
- 例子
  - **for (I=0, I<10; I++){ }**
  - 转移模式为$(1111111110)^n$

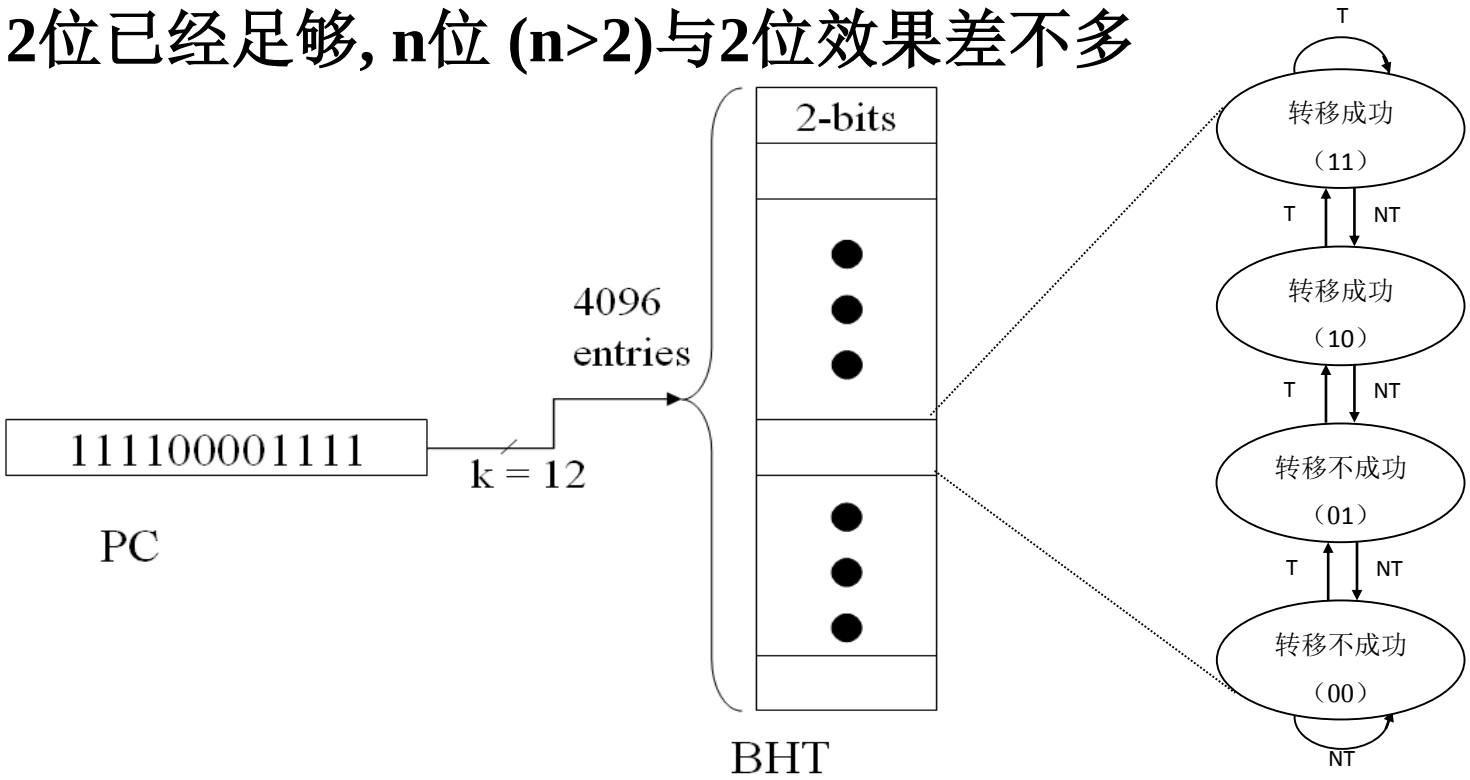
利用单个分支的重复性---BHT

- 转移历史表BHT（Branch History Table）
  - 用PC的低位索引，每项1位
  - 记录同一项上次转移是否成功，表示是否转移成功
  - 不进行地址比较检查（cache tag用于地址比较检查）
- 问题
  - 对循环进行猜测时，1位 BHT引起两次猜错
  - 循环退出时，转移方向不一致
  - 进入循环时，和上次退出时的转移方向不一致

for (i=0;i<10;i++) for (j=0; j<10; j++) { …… }

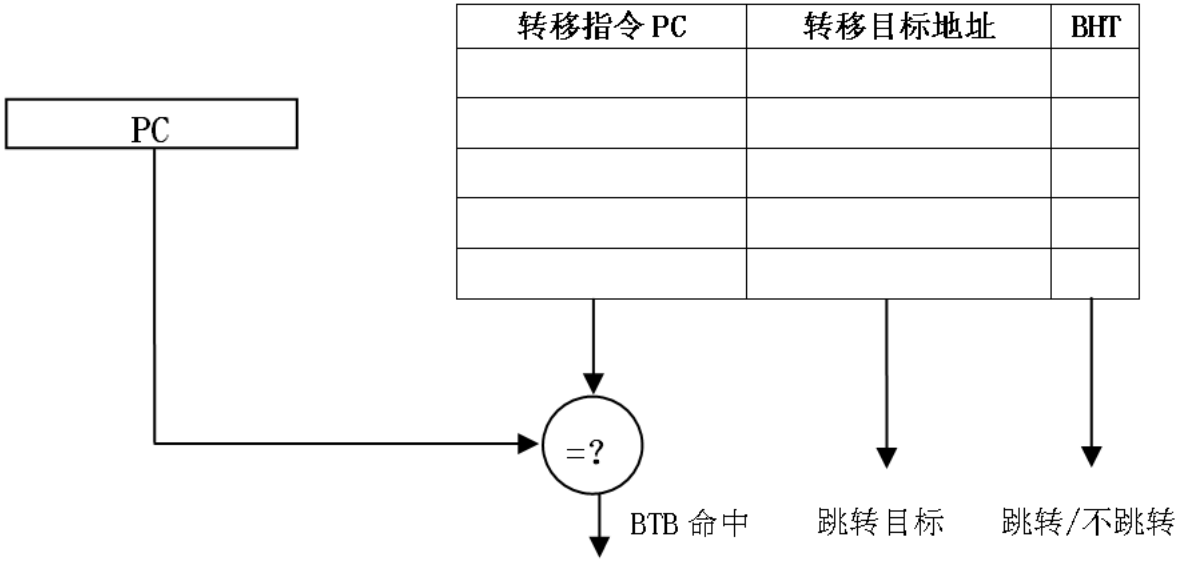
两位BHT表（课本有错）

- 只有连续两次猜错，才会改变猜测方向
  - 在前述两重循环的例子中，循环预测准确率从（8+80）/（10+100）= 80%提高到（7+88）/（10+100）= 87.2%
- 4096项已经足够，和无穷项效果差不多
- 2位已经足够, n位 (n>2)与2位效果差不多



减少猜测延迟---BTB

- 在取指阶段根据当前PC值预测转移方向和转移地址
  - 需要进行地址全相等比较
  - 直接预测PC值而不是根据指令内容计算
  - 失效时进行替换



返回地址栈

- 返回地址栈（**Return Addresses Stack**）预测返回地址
- 函数调用时压栈，返回时从栈顶弹出作为返回地址
- 针对函数调用有很高的预测准确率

转移指令的相关性

- 2位分支预测之后，预测正确率难以提高
- 主要原因是分支指令的相关性

```
If (d >=10)      //分支1
   d = 12;
If (d ==12)     //分支2依靠分支1
   d = 2
```

Yeh和Patt分类

- 当前的转移依赖于两种情况：
  - 该指令的过去m次转移记录：PHT（Pattern History Table）
  - 程序中所有转移指令过去m次的转移记录：BHR（Branch History Register）
- BHR的组织
  - “PA”表示per address BHR
  - “GA”表示global address BHR
  - “SA”表示set address BHR
- PHT的组织
  - 只用历史记录索引PHT表，用“g”表示
  - 用全地址和历史记录一起索引PHT表，用“p”表示
  - 使用部分地址和历史记录一起索引PHT表，用“s”表示

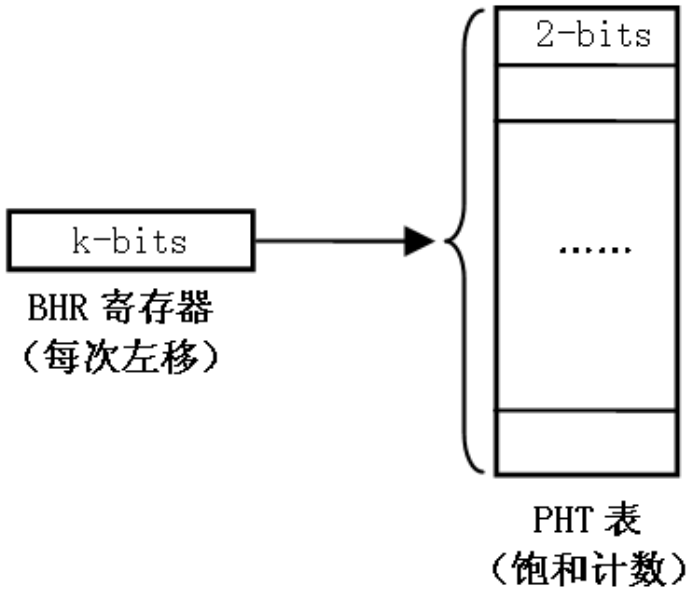
两层自适应预测器组合情况

	Global PHT	per-address PHTs	per-set PHTs
Global BHR	GAg	GAp	GAs
per-address BHR	PAg	PAp	PAs
per-set BHR	SAg	SAp	SAs

- **BHR: Branch History Register**
- **PHT: Pattern History Table**

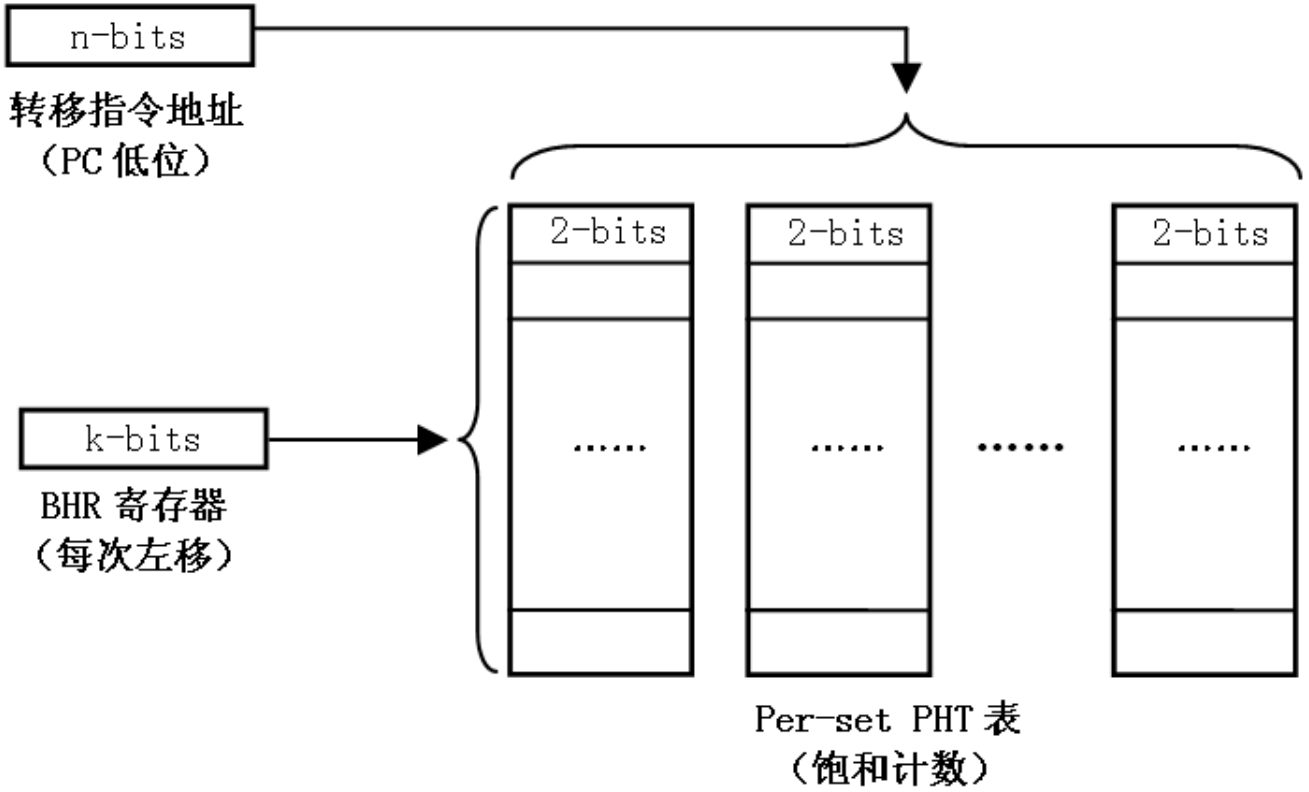
GAg结构

- **BHR和PHT都是全局的，全局的BHR又称为GHR（Global History Rigster）。其中GHR存储过去k次转移历史，并用GHR的k位值去索引2^k个入口的PHT，PHT每项利用2位饱和计数器进行预测。**



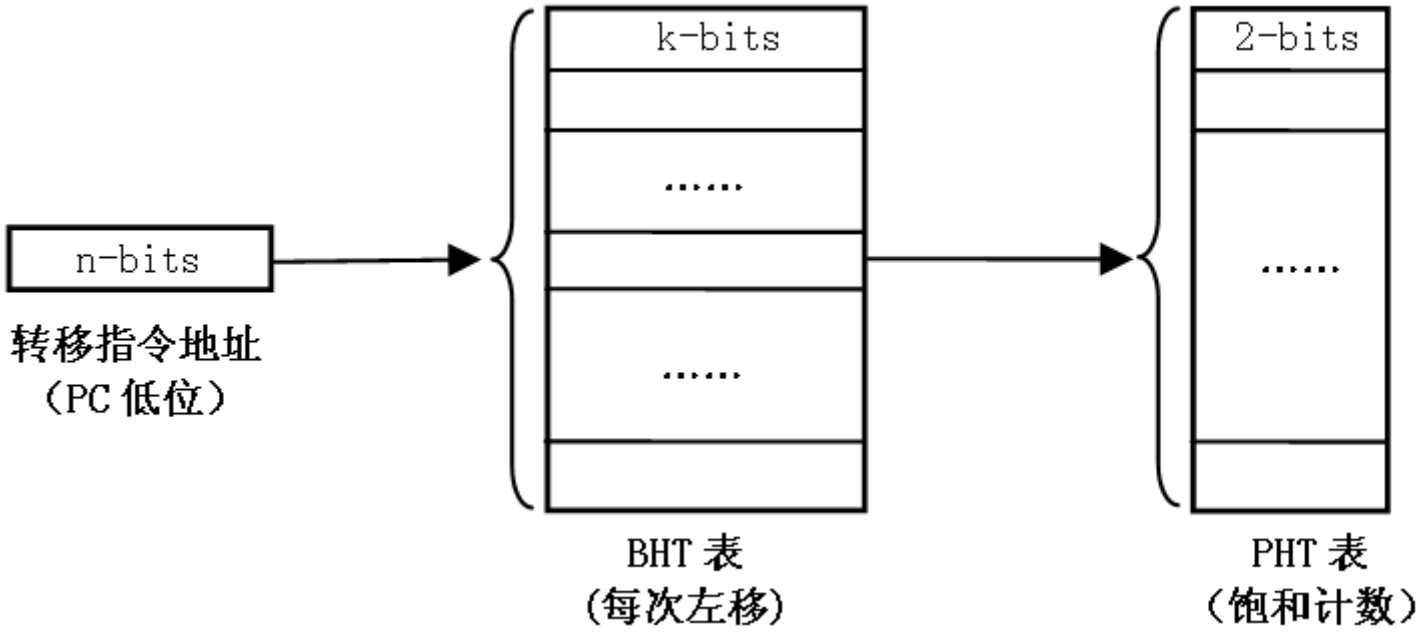
GAs结构

- 其中BHR表还是全局的，只有k位；PHT表用k位的GHR和PC的低n位进行索引，因此一共有2^{k+n}项。



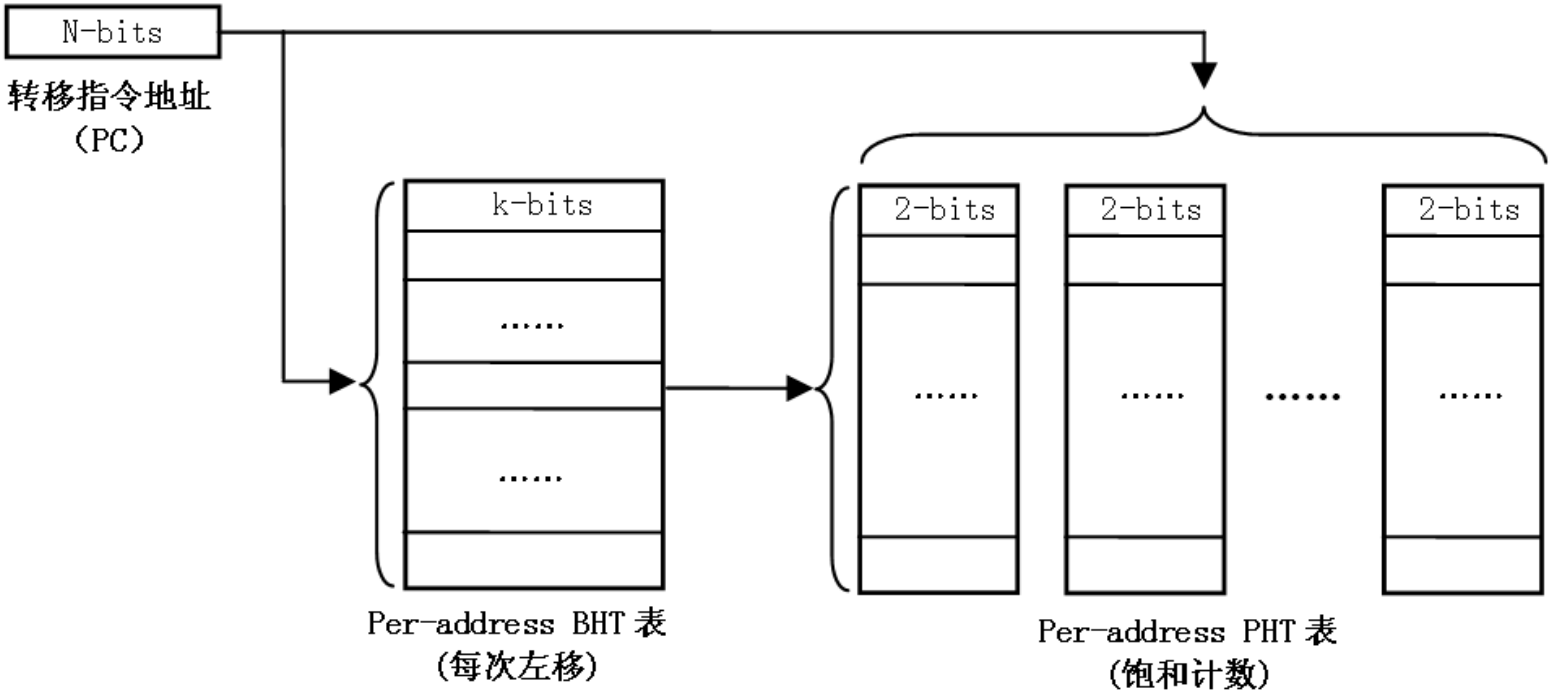
SAg(k)的结构

- **PHT是全局的，BHR寄存器一共有2^n个，每个BHR为k位。先用PC的低n位索引GHR，然后再用GHR的值索引PHT表。**

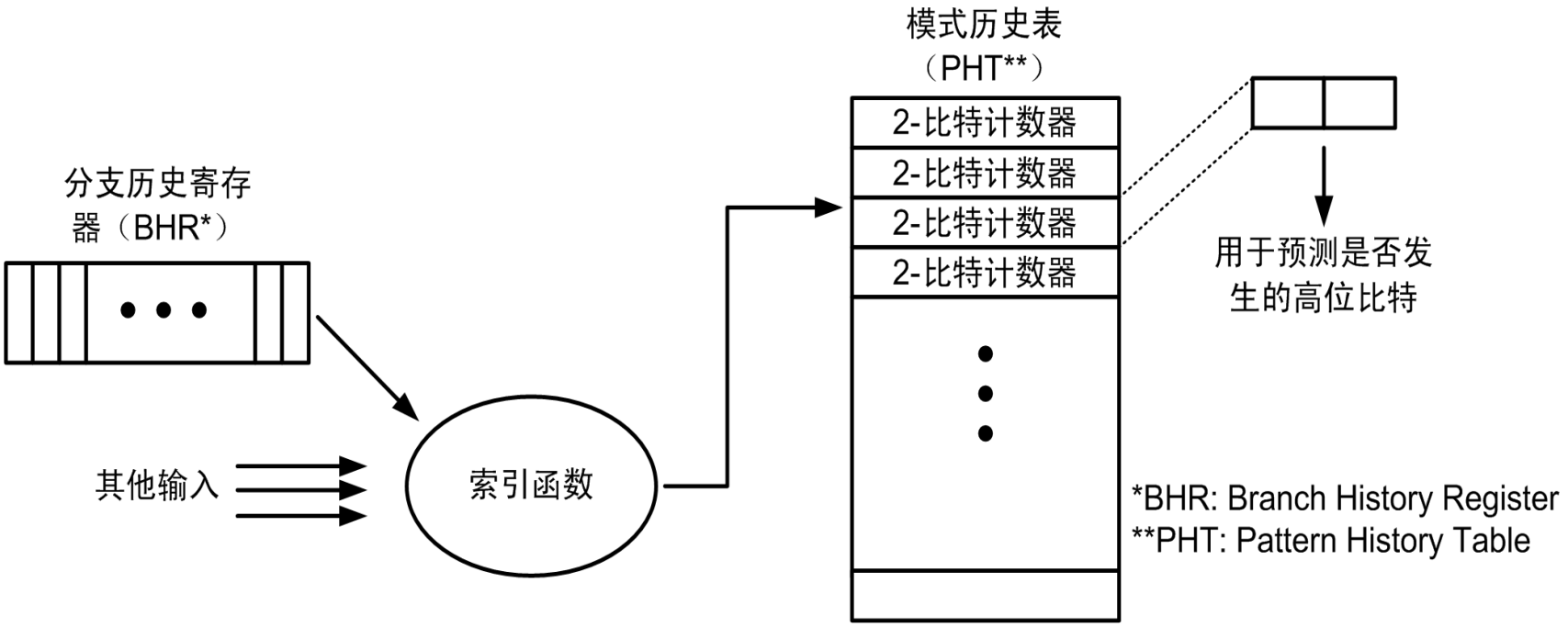


PAp(4)结构

- 每个PC值一个BHR寄存器，每个BHR为k位。先用PC索引BHR，然后再用BHR的值和PC一起索引PHT表。



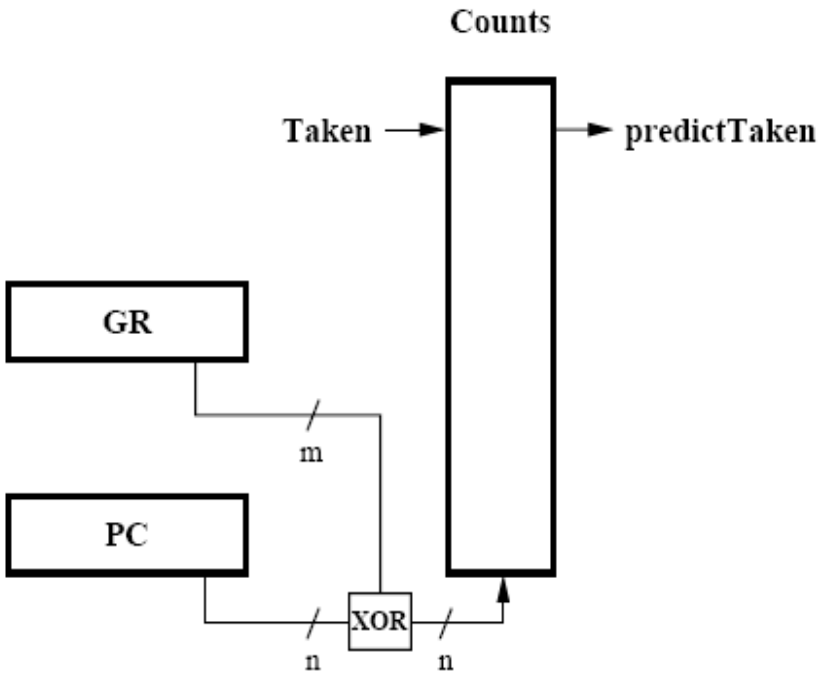
分支别名干扰问题

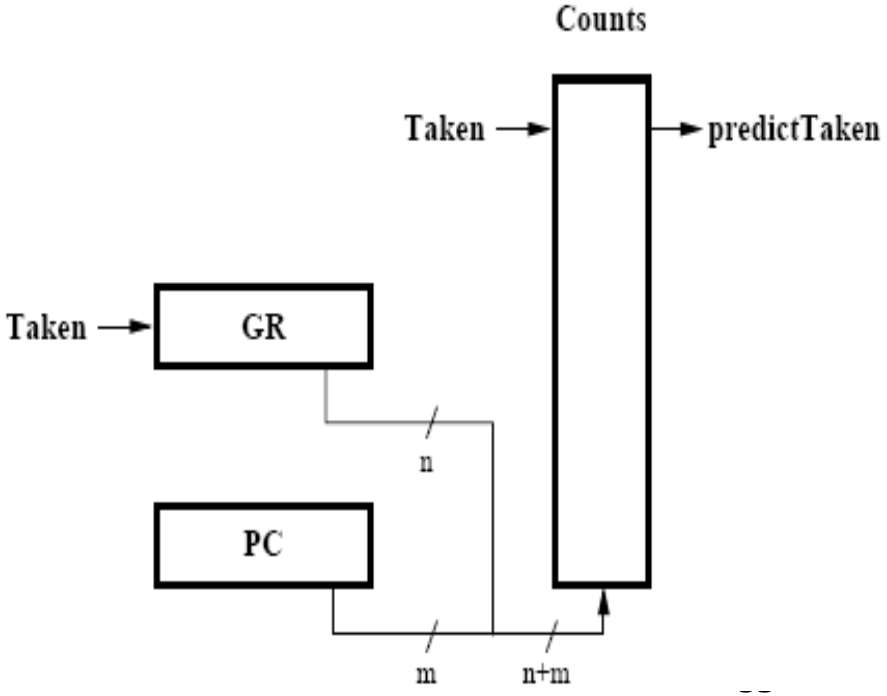


- **无论BHR和PHT表如何增大，效果也不是很明显**
- **主要原因是不同分支地址访问同一个PHT，造成分支干扰**

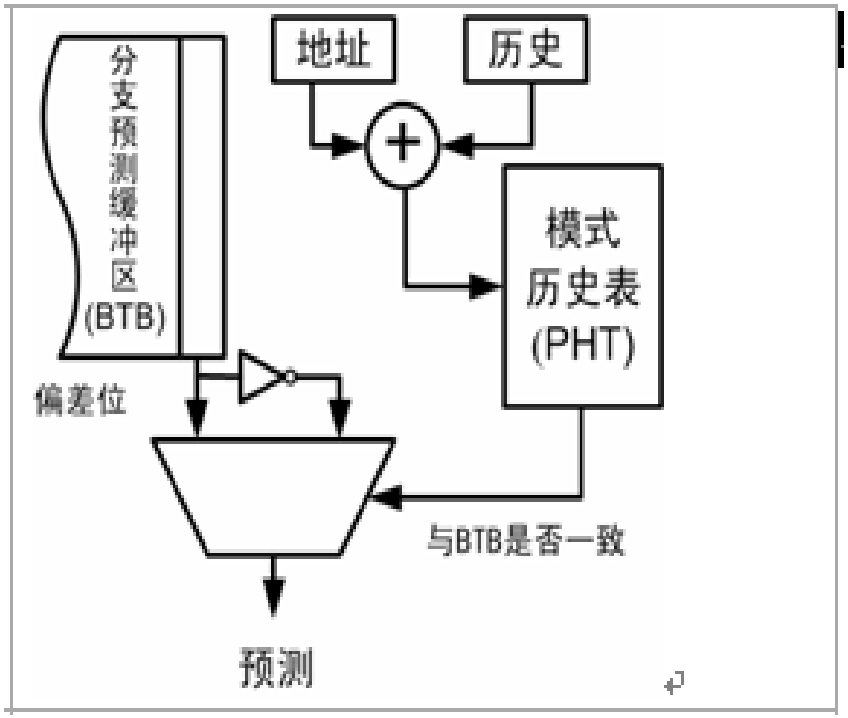
分支别名干扰的消除

- **Gselect: 全局历史m位和地址n位组合寻址**
- **Gshare: 部分地址和全局历史异或寻址**
- **性能分析结果表明：gshare稍微好于gselect**





Agree分支预测



- 在指令Cache或转移目的地址缓存（BTB）中为每一个转移都加上一个偏向位，偏向位中保存的是这条指令最常见的转移方向。
- 2位计数器不是用来预测转移方向的，而是用来决定是否按照偏向位来转移的。
- 当转移的实际结果与偏向位一致时，计数器加一，否则减一

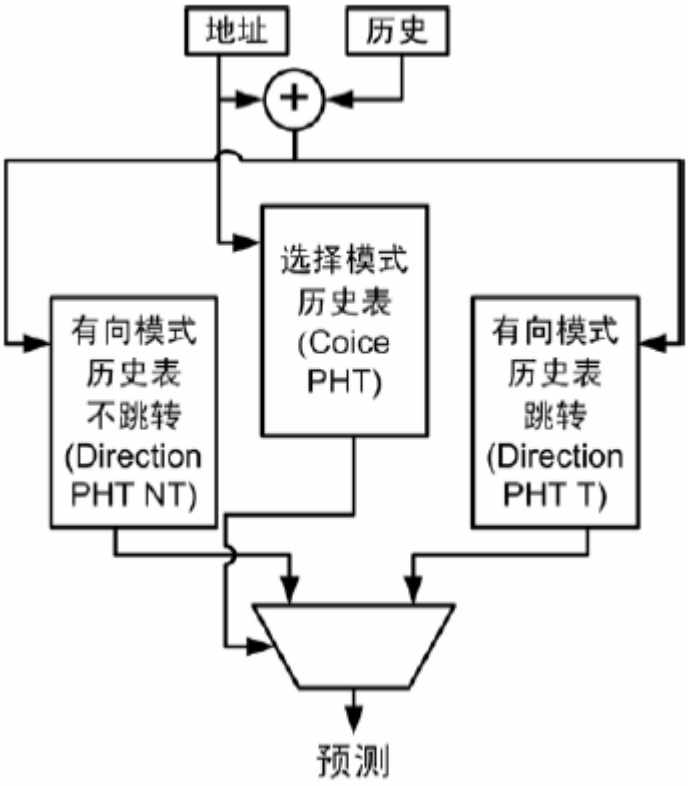
Agree分支预测

- **2条分支预测正确率分别为 85%和15%，使用同一项PHT**
  - **传统方法 - 两条分支结果相反（分支冲突）的概率:**
  - **(br1taken, br2nottaken) + (br1nottaken, br2taken)**

    = (85% * 85%) + (15% * 15%) = 74.5%
  - **Agree 方法–两条分支结果相反（分支冲突）的概率:**
  - **(br1agree, br2disagree) + (br1disagree, br2agree)**

    = (85% * 15%) + (15% * 85%) = 25.5%
- **优点**
  - **2条不同方向的分支可以映射到同一表项**
  - **偏向位不变，只改变 PHT**
  - **gcc误预测在64k的PHT下减少8.6%, 1K的PHT误预测减少33.3%**
- **HP的PA-8700处理器中得到应用**

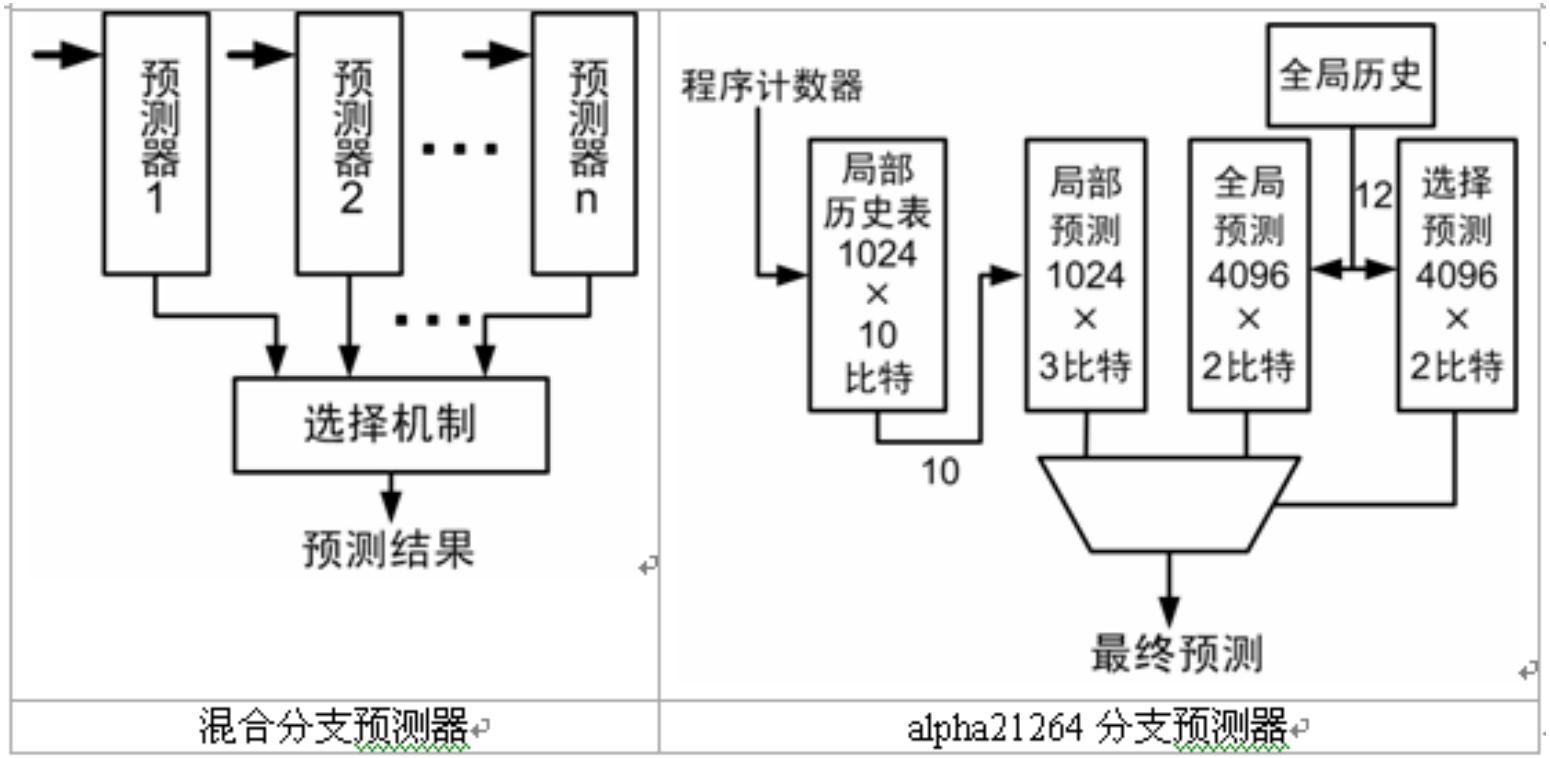
Bi-Mode预测器



- **Bi-mode 和Agree 分支预测的思想一致，不过它是把容易发生跳转和不跳转的分支放入不同的PHT。它由3 部分组成，一部分用来选择PHT，另外两部分表示PHT 的方向，分别为跳转和不跳转，PHT 的方向被全局历史索引。**
- **由于对预测器的选择，达到了针对每条转移的程度，因此命中率又有所提高。**

组合分支预测器

- 不同的分支预测只能对某类的分支行为有效
- 不同分支预测组合起来，根据分支行为选不同分支预测器



混合分支预测器　　alpha21264 分支预测器

动态转移猜测小结

- 转移的重复性和偏向性
  - **BHT**
- 转移指令的相关性问题
  - 两层转移预测
- 分支别名干扰问题
  - **Gshare等**
- 混合预测器
  - 不同的分支预测只能对某类的分支行为有效
- 不要执着于具体办法，关键是抓住应用程序的特点

常见处理器的转移预测

一些典型商品处理器的分支预测机制

	处理器	分支预测器
静态分支预测	Intel 8086	无分支预测
	Intel 486	总是跳转
	Sun superSparc	总是不跳转
	HP 7x00	BTFN
	早期的 PowerPC	Profile
动态分支预测	Alpha21064 、 AMD K5	1 位分支预测
	PowerPC604 MIPS R10000	2 位分支预测
	Pentium Pro Pentium II	2 级分支预测
	Alpha 21264	组合分支预测

Alpha 21264的分支预测器

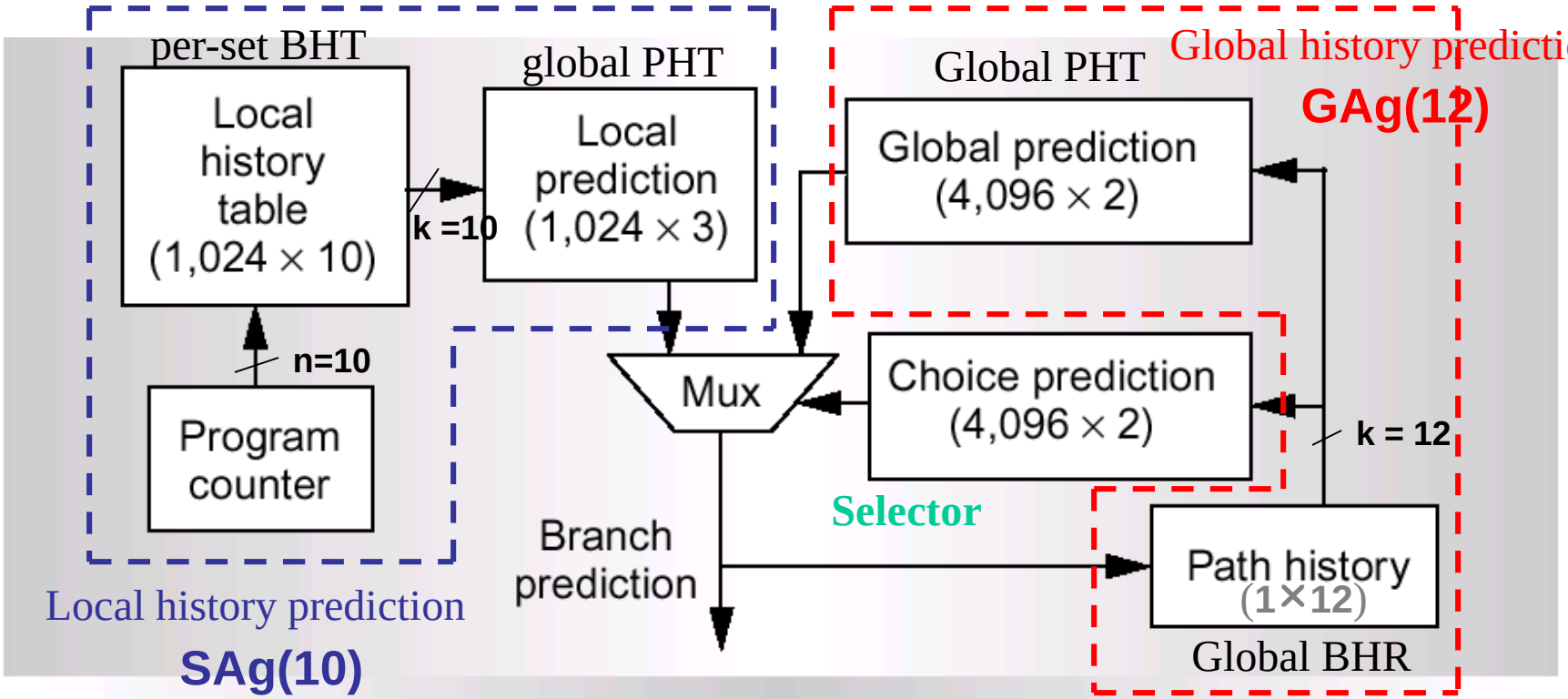
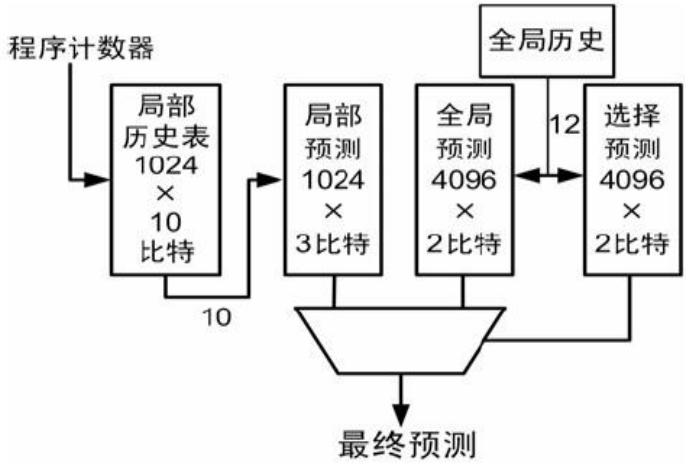
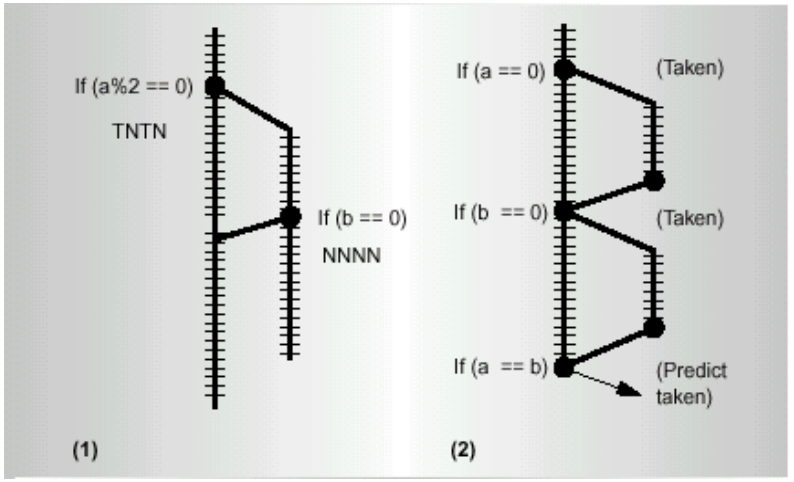


Figure 4. Block diagram of the 21264 tournament branch predictor. The local history prediction path is on the left; the global history prediction path and the chooser (choice prediction) are on the right.

Alpha 21264分支预测器举例

- 程序（a）中，假设第一个条件语句的转移模式为“TNTNTN…”，第二个条件语句的转移模式为“NNNNNN…”。
  - 简单的PHT表就可以准确猜测第二个条件语句的跳转方向，但对第一个条件语句的猜测就无能为力。Alpha 21264的两级局部预测器中“局部历史表”相应的表项有两个可能的值“1010101010”和“0101010101”，这两个值分别指向“局部预测表”中的两个表项，前者预测跳转，后者预测不跳转。
- 程序段（b）中，第一个和第二个条件都成立时第三个条件也成立。
  - 当全局转移历史为“xxxxxxxxx11”时，全局预测表将预测“if (a==b)”指令跳转。经过训练后的选择预测表也会根据在全局转移历史最后两位为“11”时选择全局预测表而不是局部预测表的预测结果。





Pentium IV的转移预测器

- 两个动态预测器（BTB）和静态预测混合预测
  - 静态预测：跳转方向是backward时则预测taken，反之nottaken
  - BTB：记录历史信息和目标地址
  - 选择：当BTB中没有该跳转指令时采用静态预测

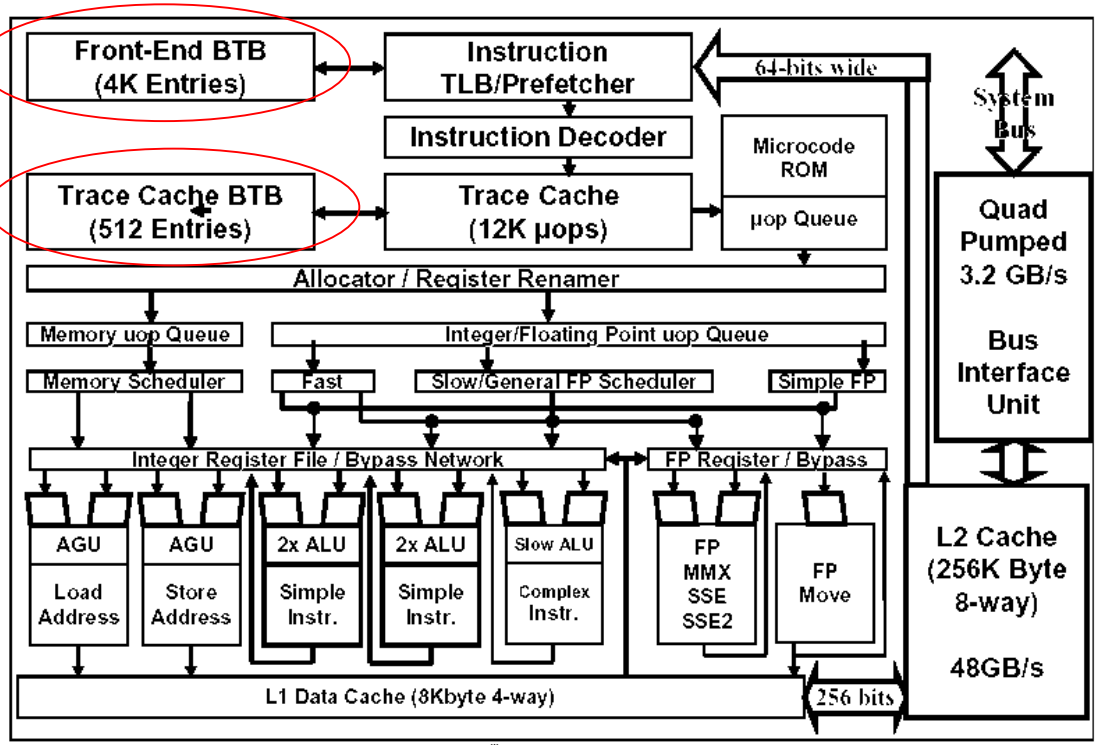


Figure 4: Pentium™ 4 processor microarchitecture

Power4转移预测器

- 静态预测
  - 每条转移指令保留两位，一位表示是否用静态预测，若用则动态预测失效，另一位表示是否跳转
- 动态预测器由3个16K*1的表组成
  - 局部预测器由指令地址来索引
  - 全局预测器由一个11位的历史向量与地址异或来索引，历史向量记录前11组指令是否连续的信息（Power4将指令分组，1组指令为5条）
  - 选择器的索引同全局预测器
- 返回地址栈Link stack
  - 遇到*branch and link*指令预测taken，同时将link地址压栈，由编译器对*branch to link*指令作上标记（hint bit），从栈中弹出地址
- 计数缓存（Count cache）
  - 32项，直接相联，软件标志该指令会多次重复跳转，则将其目标地址写入计数缓存，计数缓存的每一项保存62位地址

MIPS R10000的转移预测器

- **R10000**
  - **512-entry, 2-bit BHT**
  - **Branch stack**
- **R12000**
  - **2048-entry, 2-bit BHT**
  - **32-entry BTB**

龙芯2号分支预测器

- 静态预测和动态预测结合的转移预测方法。
  - “likely”类的转移直接预测跳转
  - 其它转移指令采用动态预测方法。
- 采用Gshare结构进行条件指令的转移预测
  - BHR共9位，PHT表有4096项
- 16项BTB预测普通JR指令
  - 在译码阶段访问
- 4项RAS预测函数返回指令
  - 目标寄存器为31号寄存器的JR指令

常见处理器的分支预测（最近）

	Intel Sandybridge	AMD BullDozer	IBM Power7	GS464E
前端				
取指宽度	16 字节/时钟周期	32 字节/时钟周期	8 指令/时钟周期	8 指令/时钟周期
一级指令缓存	32KB, 8 路, 64B/行	64KB, 2路, 64B/行	32KB, 4 路, 128B/行	64KB, 4 路, 64B/行
指令TLB	128 项, 4 路, L1 ITLB	72 项, 全相联, L1 ITLB 512 项, 4 路, L2 ITLB	64 项, 2 路, L1 ITLB	64 项, 全相联, L1 ITLB
分支预测	BTB (8K-16K?项) 间接目标队列 (? 项) RAS (? 项) 循环检测	512 项, 4 路 L1 BTB 5120 项, 5 路 L2 BTB 512 项 间接目标队列 24 项 RAS 循环检测	8K 项本地 BHT 队列, 16K 项全局BHT 队列, 8K 项 全局 sel 队列 128 项间接目标队列 16 项 RAS	8K项本地 BHT 队列 8K项全局BHT 队列 8K项 全局 sel 队列 128项间接目标队列 16项 RAS 循环检测
乱序执行				
Reorder缓存	168 项	128 项	120 项	128 项
发射队列	54-项 统一	60-项 浮点(共享) 40-项 定点, 访存	48-项 标准	16-项 浮点; 16-项 定点; 32-项 访存
寄存器重命名	160 定点; 144 浮点	96 定点; 160 浮点(双核共享)	80 定点,浮点; 56 CR; 40 XER, 24 Link&Count	128 定点; 128 浮点/向量; 16 Acc; 32 DSPCtrl; 32 FCR
执行单元				
执行单元个数	ALU/LEA/Shift/128位 MUL/128位 Shift/256位FMUL/256位Blend + ALU/LEA/Shift/128位 ALU /128bit Shuffle/256位 FADD + ALU/Shift/Branch/ 128位 ALU/128bit Shuffle/256位 Shuffle/256位 Blend	ALU/IMUL/Branch + ALU/IDIV/Count + 128位FMAC/128位 IMAC + 128位FMAC/128位 XBAR + 128位 MMX + 128位 MMX/128位 FSTO	2 定点 + 2 浮点/向量 + 1 转移 + 1 CR	2 定点/转移/DSP + 2 浮点/向量
向量部件宽度	256位	128位	128位	256位

分支预测未来

- 进一步提高很难
  - **95%－98%**左右
  - 工业界分支预测越做越大，**Alpha21464**有**48KB**
- 进一步提高很有意义
  - 预测精度提高**0.5%**，**10000**条分支指令就能减少**50**次流水线刷新
- 新应用及新结构需要新的预测机制
  - **Power-aware**分支预测
  - **SMT**结构分支预测
  - 神经网络分支预测器
- 分支预测大赛
  - 工业界支持，**2**年一次

作 业