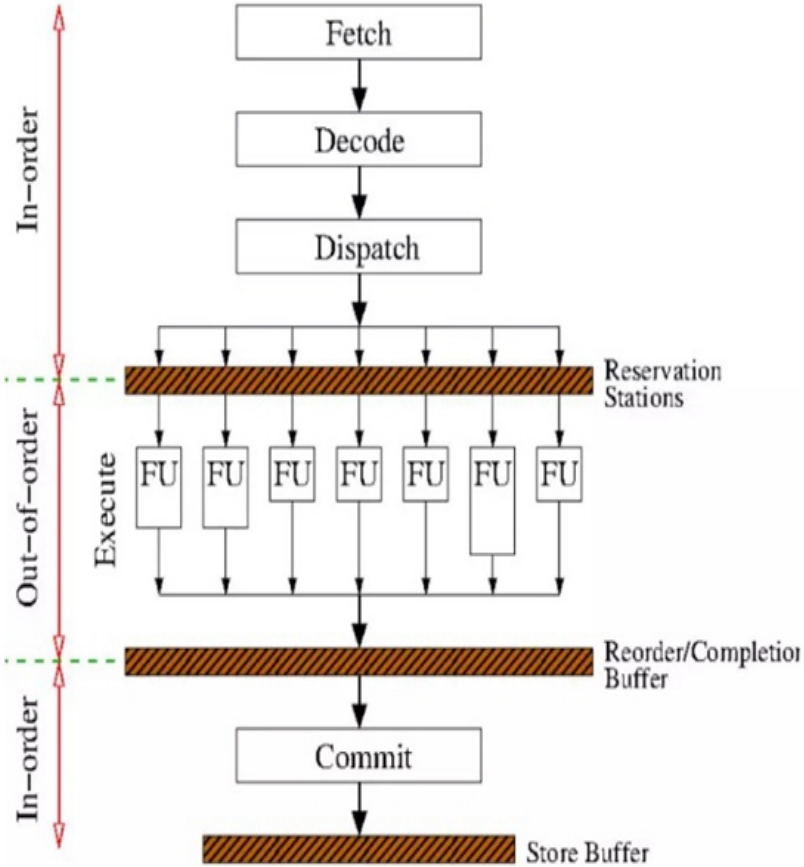
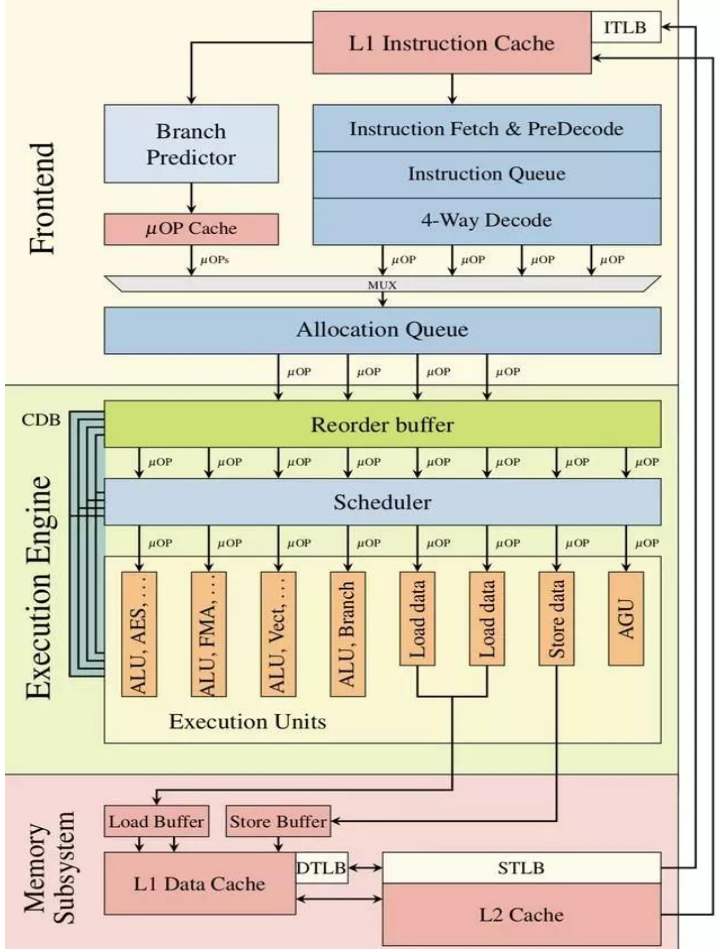
清華大學
Tsinghua University

Branch Prediction





熔毁利用了图中execution engine的部分，幽灵利用了图中branch predictor的部分

Branches

- Instructions which can alter the flow of instruction execution in a program

Branch Prediction is More Important Today

Conditional branches still comprise about 20% of instructions

pipelines deeper branch not resolved until more cycles from fetching therefore the misprediction penalty greater

cycle times smaller: more emphasis on throughput (performance), more functionality between fetch & execute

multiple instruction issue (superscalars & VLIW) branch occurs almost every cycle flushing & refetching more instructions

object-oriented programming more indirect branches which harder to predict

dual of Amdahl's Law other forms of pipeline stalling are being addressed so the portion of CPI due to branch delays is relatively larger

On the other hand,

chips are denser so we can consider sophisticated HW solutions

hardware cost is small compared to the performance gain

All this means that the potential stalling due to branches is greater

Types of Branches

	Conditional	Unconditional
Direct	if - then- else for loops (bez, bnez, etc)	procedure calls (jal) goto (j)
Indirect		return (jr) virtual function lookup function pointers (jalr)

问题1：采用循环程序计算 **S=1+2+3+…+99+100**

采用1位预测器，预测准确率是多少？

采用2位预测器，预测准确率是多少？

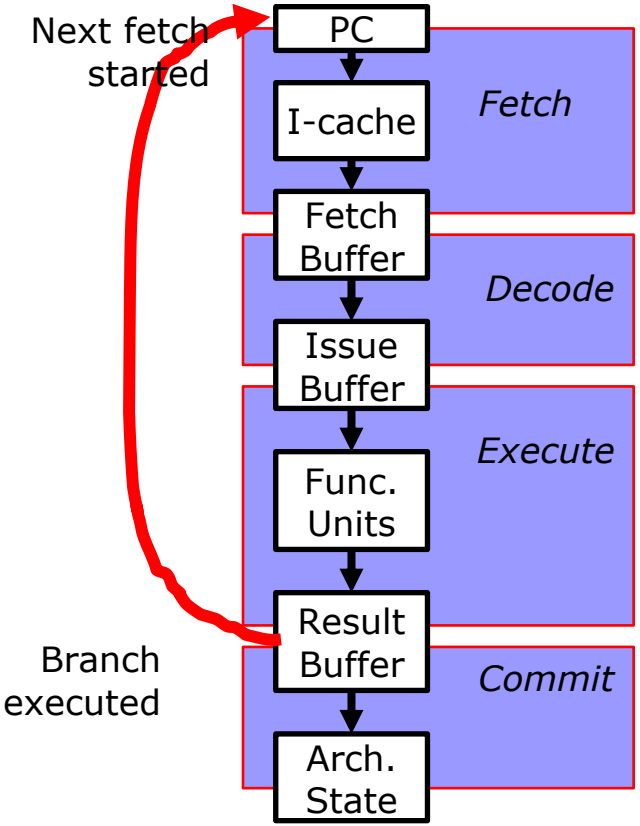
■问题2：饱和计数器（ **saturating counter** ）？

Control Flow Penalty

Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution !

How much work is lost if pipeline doesn't follow correct instruction flow?

~ Loop length x pipeline width



Average Run-Length between Branches

Average dynamic instruction mix from SPEC92:

	SPECint92	SPECfp92
ALU	39 %	13 %
FPU Add		20 %
FPU Mult		13%
load	26 %	23 %
store	9 %	9 %
branch	16 %	8 %
other	10 %	12 %

SPECint92: *compress, eqntott, espresso, gcc , li*
SPECfp92: *doduc, ear, hydro2d, mdijdp2, su2cor*

MIPS Branches and Jumps

Each instruction fetch depends on one or two pieces of information from the preceding instruction:

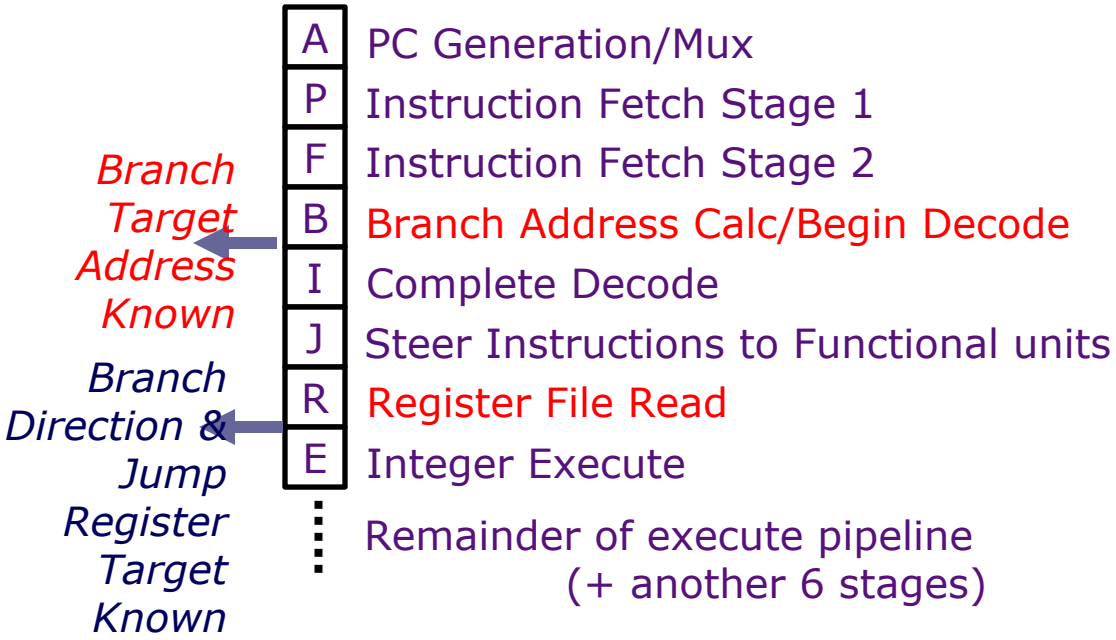
1) Is the preceding instruction a taken branch?

2) If so, what is the target address?

J 指令?

Branch Penalties in Modern Pipelines

UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)



Reducing Control Flow Penalty

Software solutions

- *Eliminate branches* - *loop unrolling*
  Increases the run length
- *Reduce resolution time* - *instruction scheduling*
  Compute the branch condition as early as possible (of limited value)

Hardware solutions

- Find something else to do - *delay slots*
  Replaces pipeline bubbles with useful work (requires software cooperation)
- *Speculate* - *branch prediction*
  Speculative execution of instructions beyond the branch

Branch Prediction

Motivation:

Branch penalties limit performance of deeply pipelined processors

Modern branch predictors have high accuracy (>95%) and can reduce branch penalties significantly

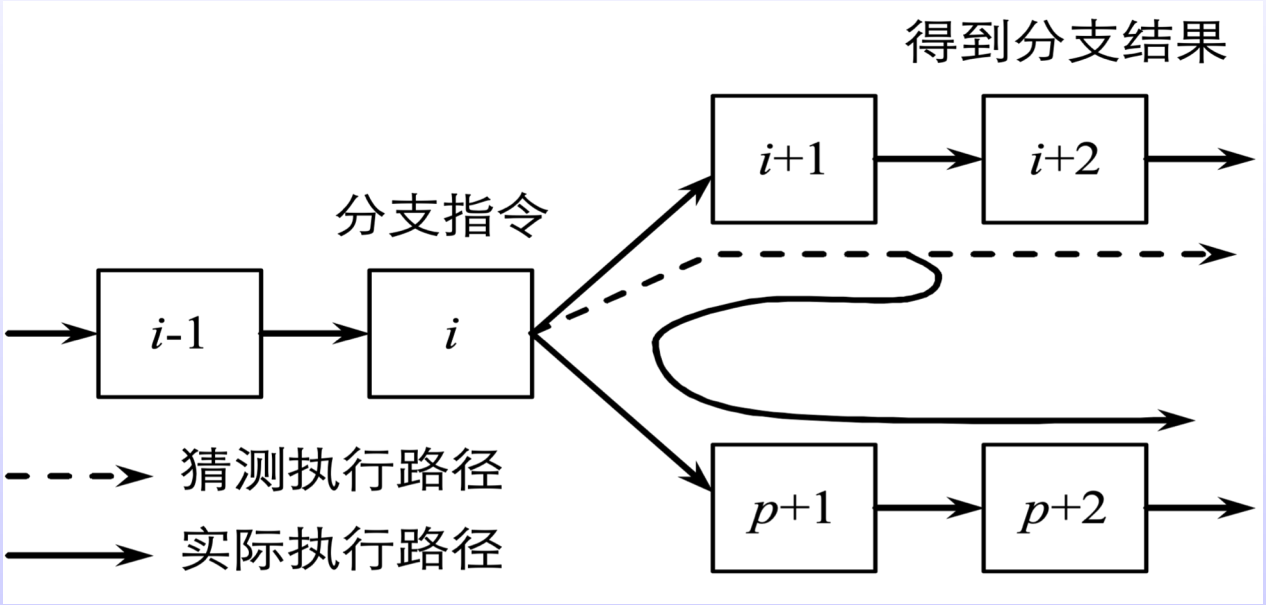
Required hardware support:

Prediction structures:

- Branch history tables, branch target buffers, etc.

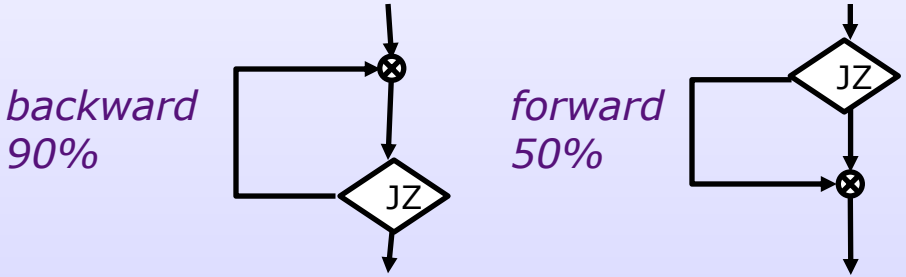
Mispredict recovery mechanisms:

- *Keep result computation separate from commit*
- Kill instructions following branch in pipeline
- Restore state to state following branch


得到分支结果
分支指令
i-1
i
i+1
i+2
p+1
p+2
猜测执行路径
实际执行路径

Static Branch Prediction

Overall probability a branch is taken is ~60-70% but:



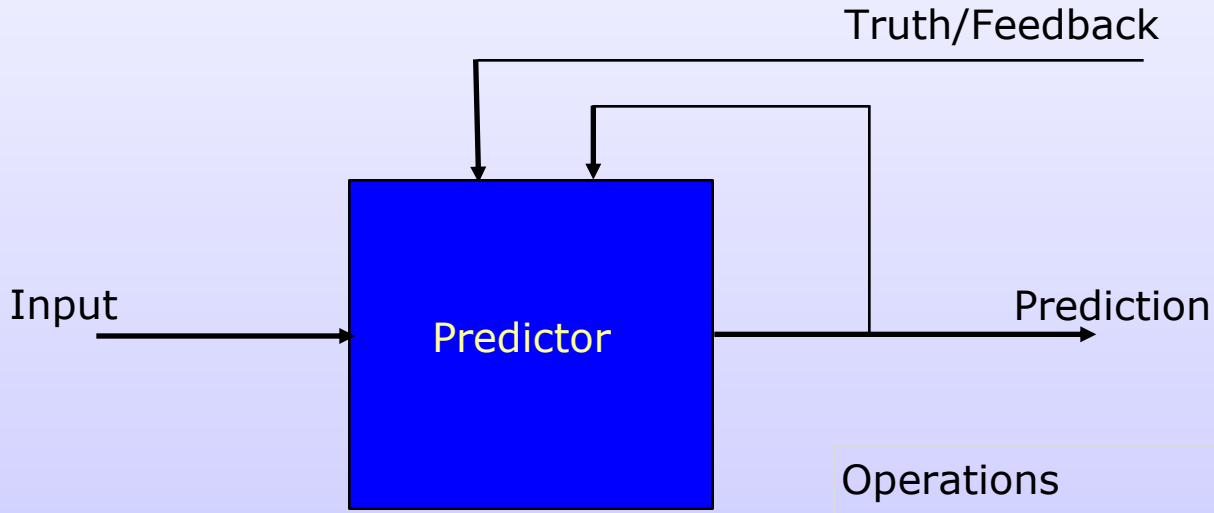
ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110

bne0 *(preferred taken)* beq0 *(not taken)*

ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64

typically reported as ~80% accurate

Dynamic Prediction



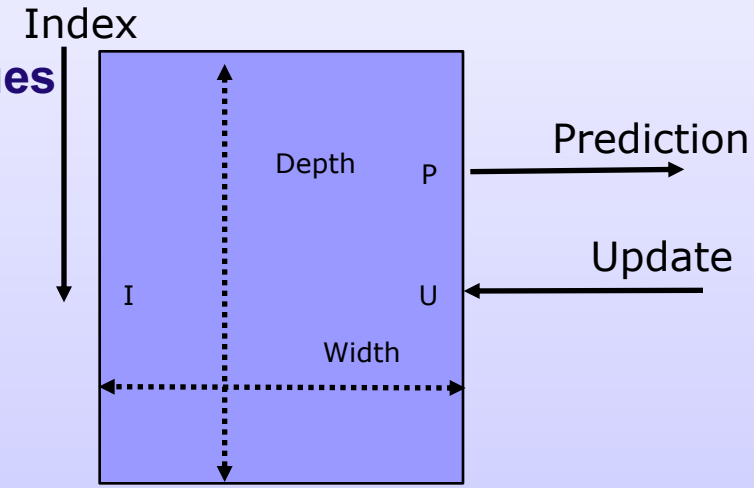
Prediction as a feedback control process

Operations

- Predict
- Update

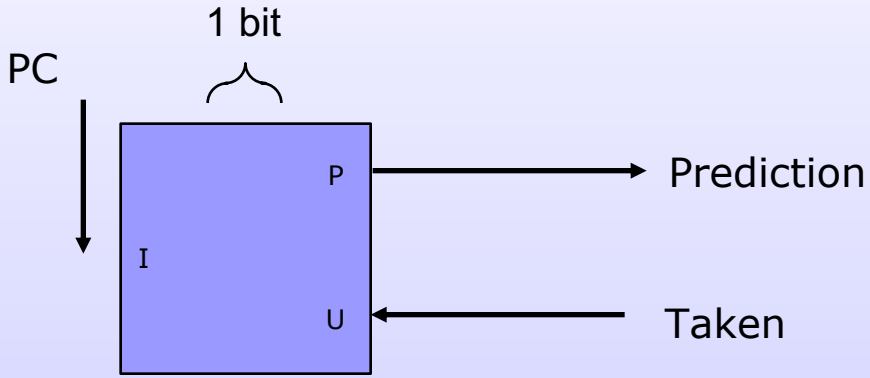
Predictor Primitive

- Indexed table holding values
- Operations
  - Predict
  - Update



Prediction = P[Width, Depth](Index; Update)

One-bit Predictor



A21064(PC; T) = P[1, 2K](PC; T)

What happens on loop branches?

At best, mispredicts twice for every use of loop

Dynamic Branch Prediction

learning based on past behavior

Temporal correlation

The way a branch resolves may be a good predictor of the way it will resolve at the next execution

Spatial correlation

Several branches may resolve in a highly correlated manner *(a preferred path of execution)*

```
if (x[i] < 7) then
      y += 1;
if (x[i] < 5) then
      c -= 4;
```

Branch Prediction

- **Easiest (static prediction)**
  - **Always taken, always not taken**
  - **Opcode based**
  - **Displacement based (forward not taken, backward taken)**
  - **Compiler directed (branch likely, branch not likely)**
- **Next easiest**
  - **1 bit predictor – remember last taken/not taken per branch**
    - Use a ***branch-prediction buffer*** **or** ***branch-history*** *table*
    - Use part of the PC (low-order bits) to index buffer/table
      - Multiple branches may share the same bit！ ！ ！ ！ ！
    - Invert the bit if the prediction is wrong
    - Backward branches for loops will be mispredicted twice(?)

Branch Prediction

Q: Assume a loop branch is taken nine times in a row, then not taken once. What is the prediction accuracy using 1-bit predictor?

A: After first loop, the predictor will say not to take because the last time the execution came out of loop, it set a "0" in the predictor. So, it's a misprediction. The bit will now be set to "1". Works fine until the last loop when it is predicted as taken. So, 2 mispredictions in in 10 loop executions => 80% accuracy.

Actual branches	Predictions
ttt ttt ttt n	011 111 111 1

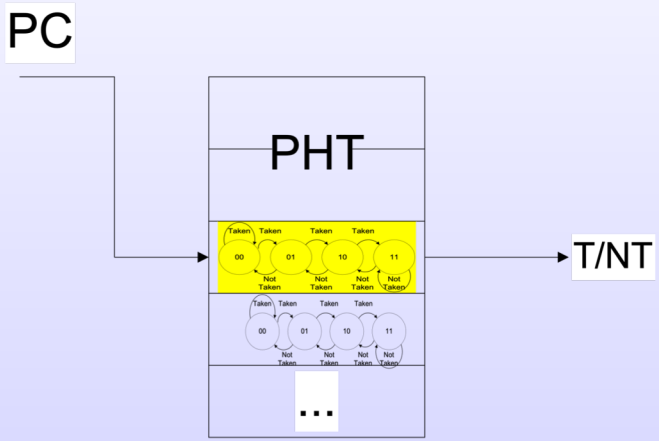
How about a 2-bit predictor? Let the prediction be changed only after it misses twice in a row.

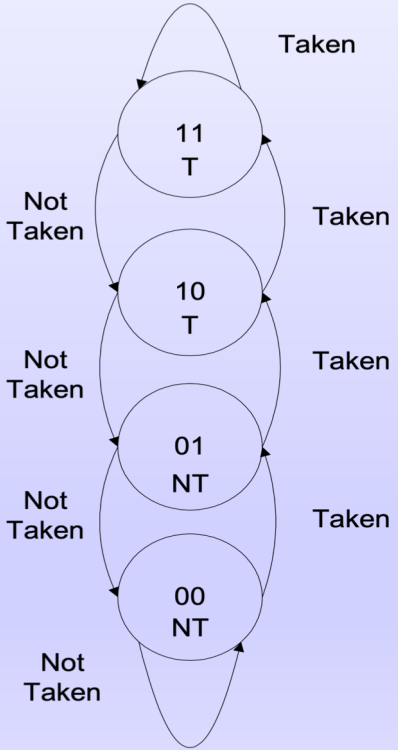
Branch Prediction Bits

- Assume 2 BP bits per instruction
- Use saturating counter or others

On ¬taken →	↑ On taken			
		1	1	Strongly taken
		1	0	Weakly taken
		0	1	Weakly ¬taken
		0	0	Strongly ¬taken

Bimodal Prediction

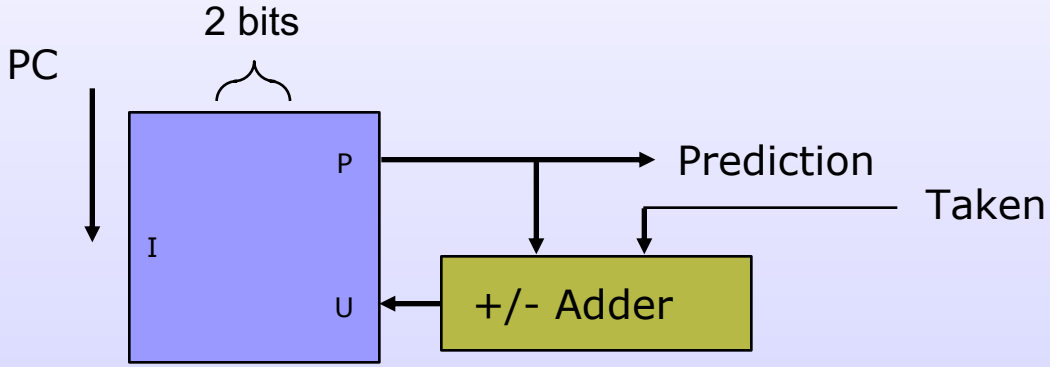




- **Table of 2-bit saturating counters**
  Predict the most common direction

Advantages: simple, cheap, "good" accuracy

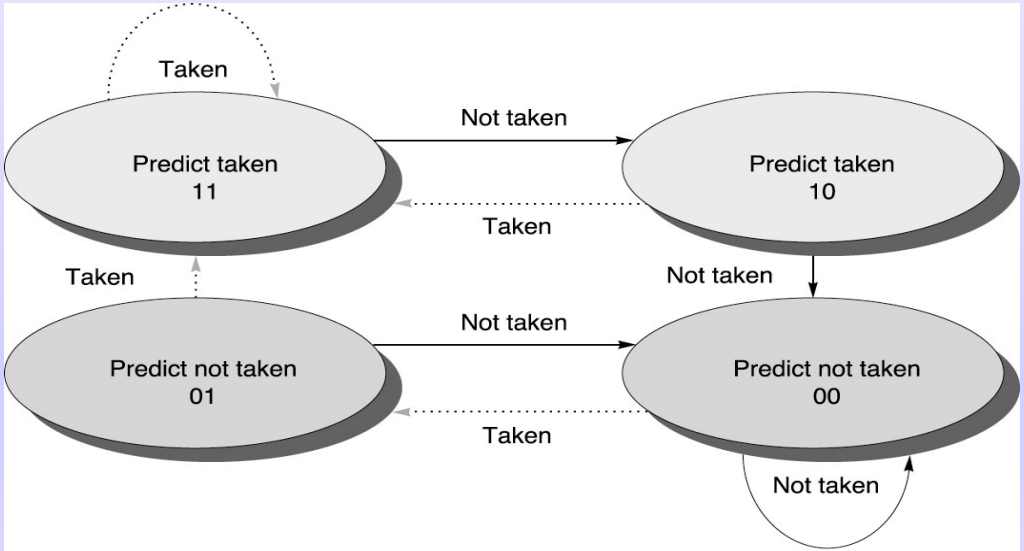
Two-bit Predictor（adder）



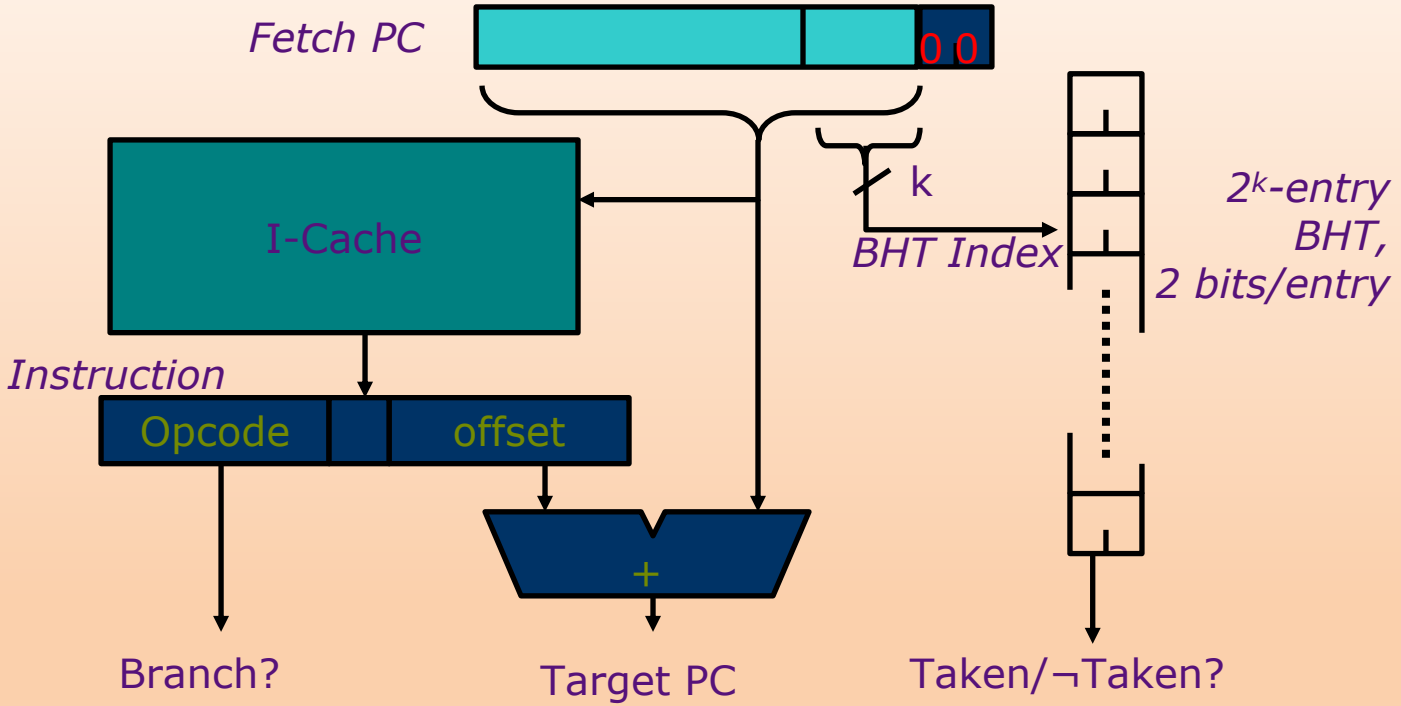
Counter[W,D](I; T) = P[W, D](I; if T then P+1 else P-1)

2-bit Branch Prediction(实现？)

- Has 4 states instead of 2, allowing for more information about tendencies
- A prediction must miss twice before it is changed



Branch History Table



4K-entry BHT, 2 bits/entry, ~80-90% correct predictions

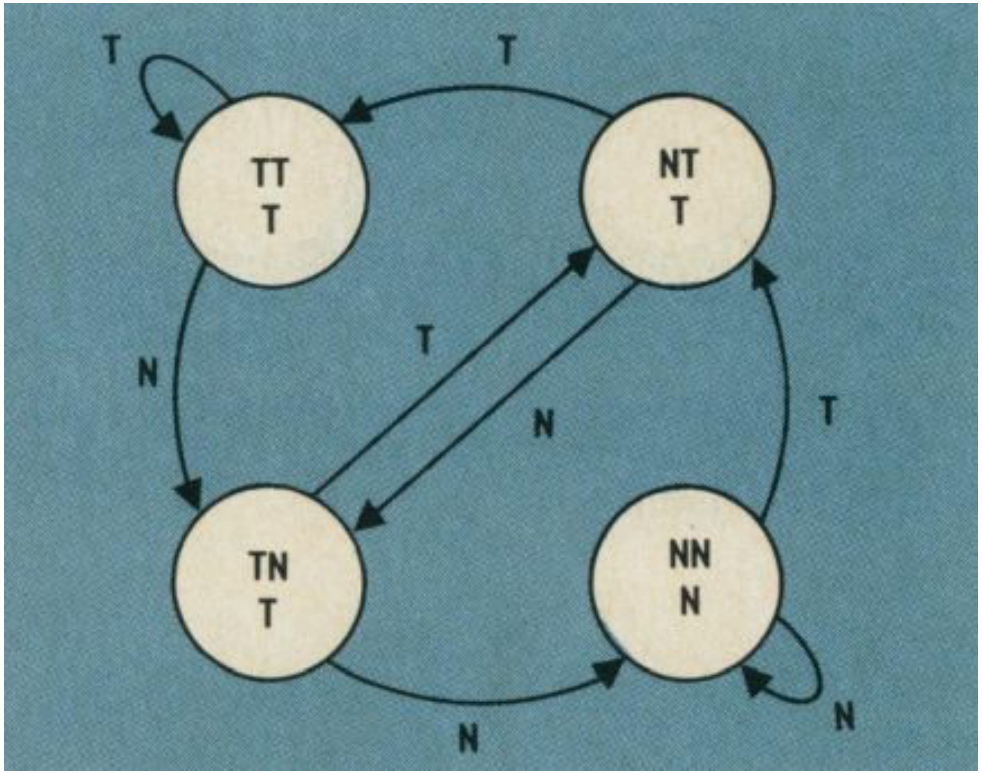
2-bit Branch Prediction

- **Good for backward branches of loops**

Actual branches
ttt ttt ttt n

Predictions
t t t t t t t t t t

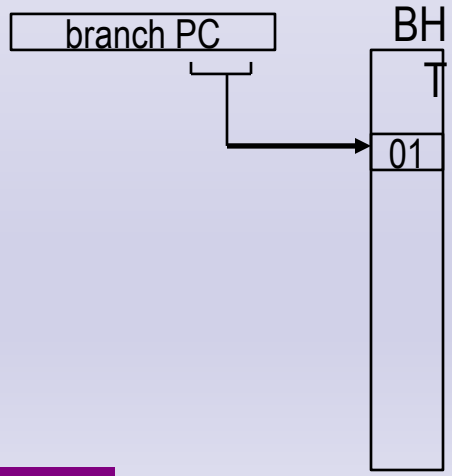
Prediction status bits
10 11 11 11 11 11 11 11 11 11

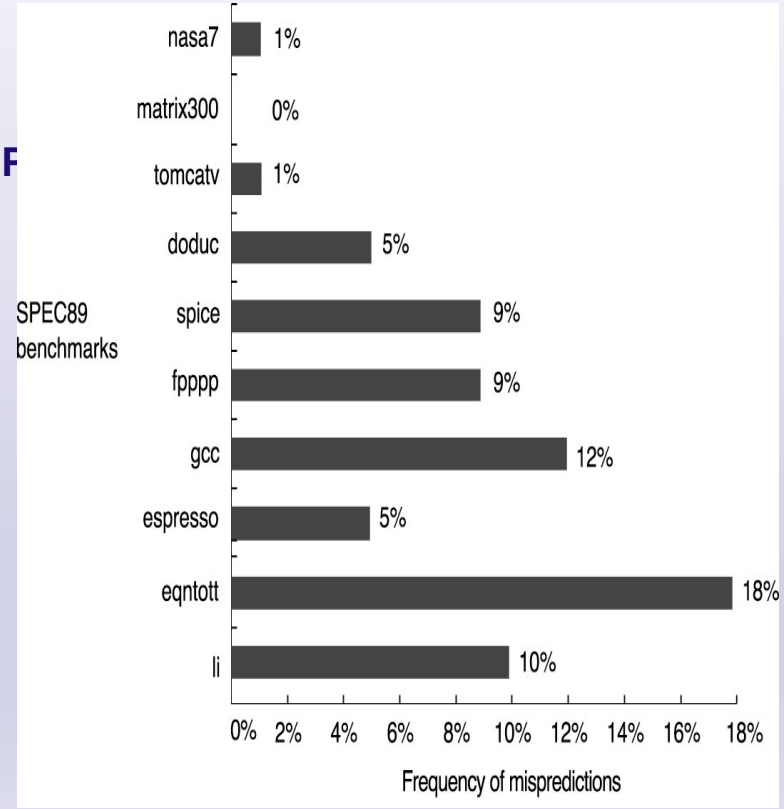


实现？

Branch History Table

- Has limited size
- 2 bits by N (e.g. 4K)
- Uses low-order bits of branch P
  choose entry





Branch Target Buffer (BTB)

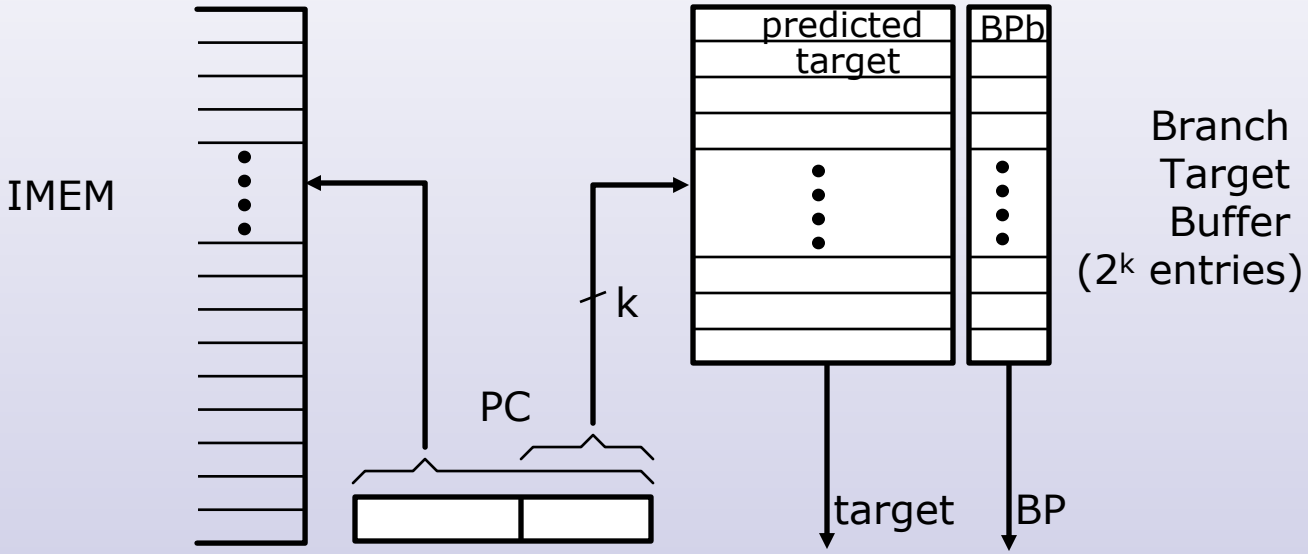
Is the current instruction a branch ?

- BTB provides the answer before the current instruction is decoded and therefore enables fetching to begin after IF-stage .

What is the branch target ?

- BTB provides the branch target if the prediction is a taken direct branch (for not taken branches the target is simply PC+4) .

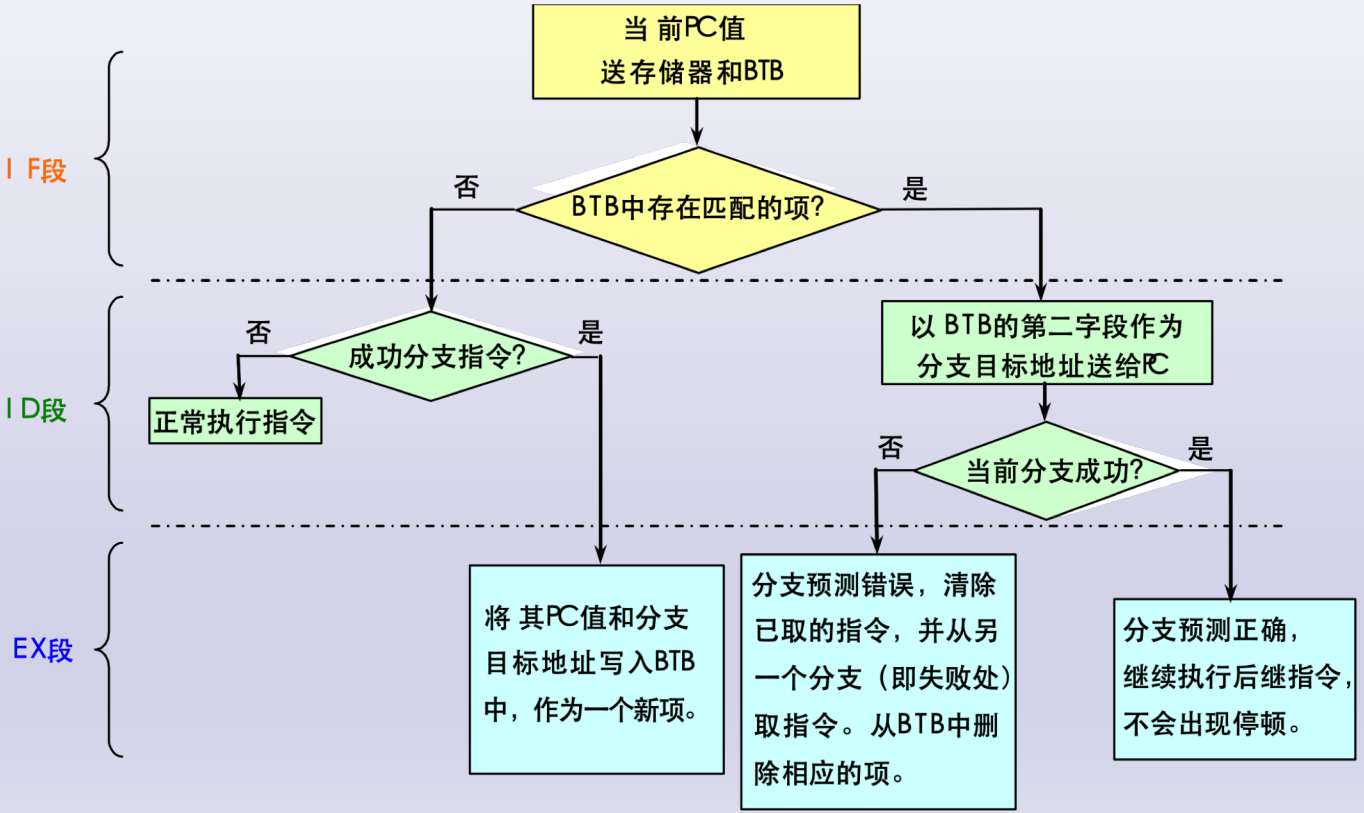
Branch Target Buffer

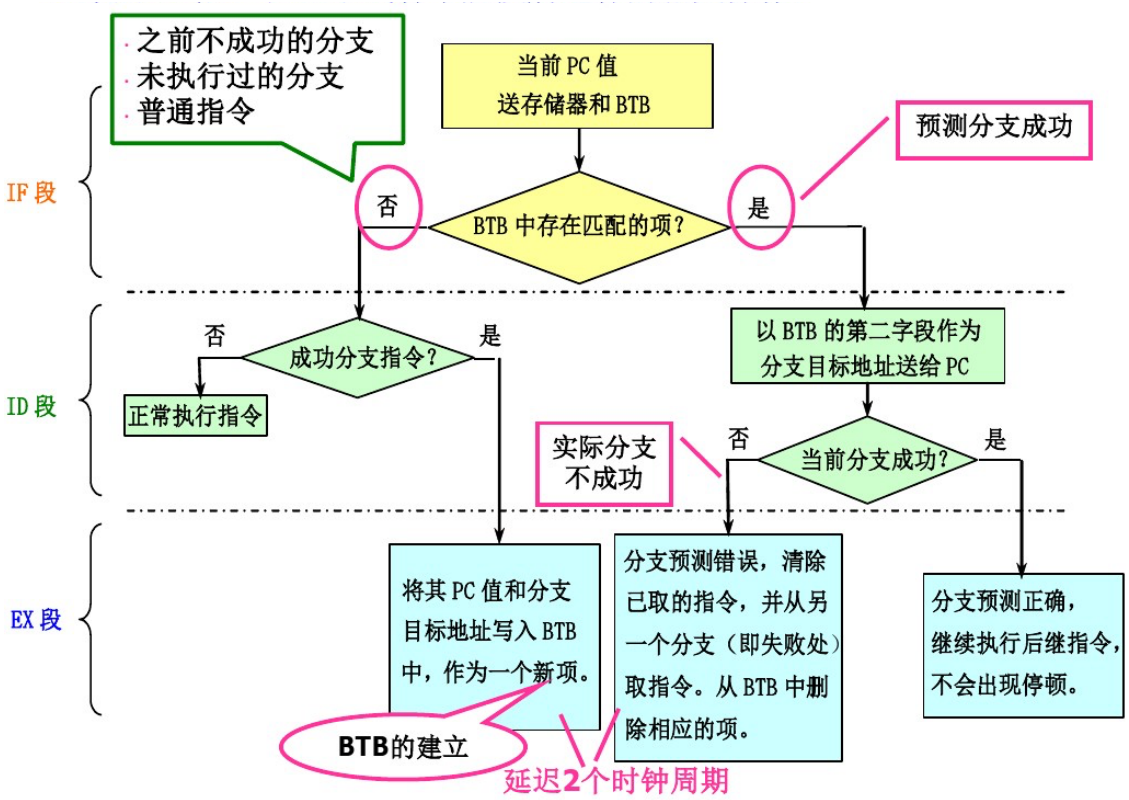


BP bits are stored with the predicted target address.

IF stage: *If (BP=taken) then nPC=target else nPC=PC+4*
later: *check prediction, if wrong then kill the instruction and update BTB & BPb else update BPb*

Case Study

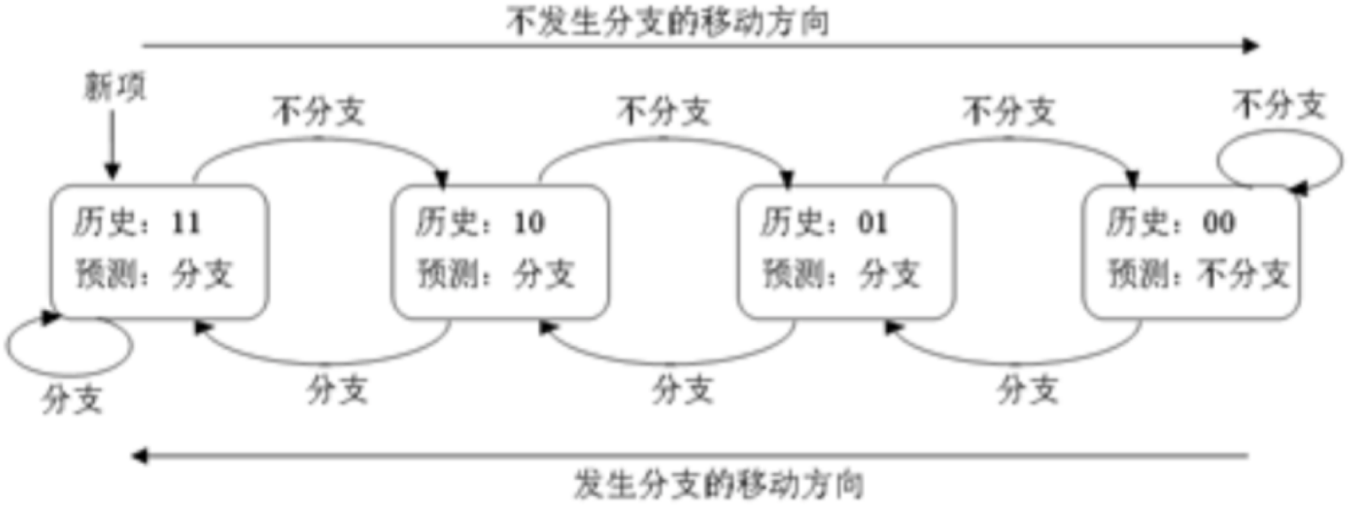

当 前PC值
送存储器和BTB
IF段
否
BTB中存在匹配的项?
是
ID段
否
成功分支指令?
是
正常执行指令
以 BTB的第二字段作为
分支目标地址送给PC
否
当前分支成功?
是
EX段
将 其PC值和分支
目标地址写入BTB
中，作为一个新项。
分支预测错误，清除
已取的指令，并从另
一个分支（即失败处）
取指令。从BTB中删
除相应的项。
分支预测正确，
继续执行后继指令，
不会出现停顿。


之前不成功的分支
未执行过的分支
普通指令
当前 PC 值
送存储器和 BTB
预测分支成功
IF 段
否
BTB 中存在匹配的项？
是
否
成功分支指令？
是
以 BTB 的第二字段作为
分支目标地址送给 PC
ID 段
正常执行指令
实际分支
不成功
否
当前分支成功？
是
将其 PC 值和分支
目标地址写入 BTB
中，作为一个新项。
分支预测错误，清除
已取的指令，并从另
一个分支（即失败处）
取指令。从 BTB 中删
除相应的项。
分支预测正确，
继续执行后继指令，
不会出现停顿。
EX 段
BTB的建立
延迟2个时钟周期

指令在BTB中？	预测	实际情况	延迟周期
是	成功	成功	0
是	成功	不成功	2
不是		成功	2
不是		不成功	0

BTB-Pentium

（a） 分支目标缓冲器 BTB 项



（b） 分支目标缓冲器 BTB 的历史状态流程