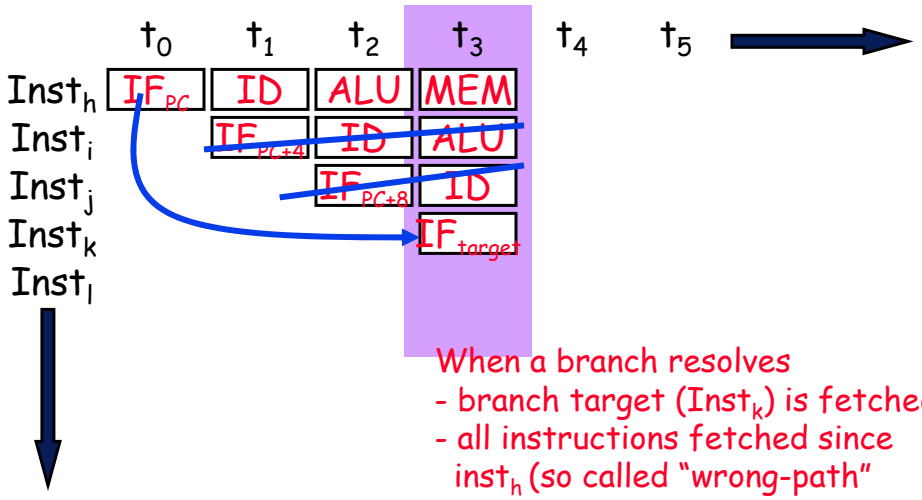
18-447 Lecture 13: Branch Prediction

James C. Hoe
Dept of ECE, CMU
March 4, 2009

Announcements: Spring break!!
Spring break next week!!
Project 2 due the week after spring break
HW3 due Monday after spring break
(no more homework until week 12)

Handouts:

Control Speculation: PC+4



When a branch resolves
- branch target ($Inst_k$) is fetched
- all instructions fetched since $inst_h$ (so called "wrong-path" instructions) must be flushed

$Inst_h$ is a branch

Performance Impact

- correct guess ⇒ no penalty ~86% of the time
- incorrect guess ⇒ 2 bubbles
- Assume
  - no data hazards
  - 20% control flow instructions
  - 70% of control flow instructions are taken
  - IPC = 1 / [1 + (0.20*0.7) * 2] =
    = 1 / [1 + 0.14 * 2] = 1 / 1.28 = 0.78

probability of a wrong guess

penalty for a wrong guess

Can we reduce either of the two penalty terms?

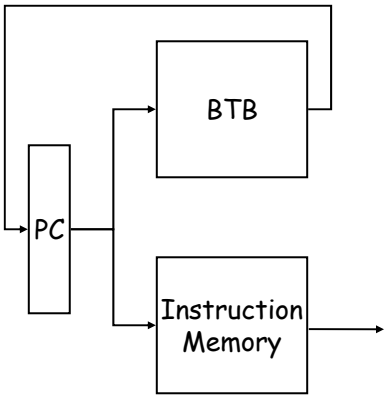
Making a Better Guess

- For ALU instructions
  - can't do better than guessing nextPC=PC+4
  - still tricky since must guess nextPC before the current instruction is fetched
- For Branch/Jump instructions
  - why not always guess in the taken direction since 70% correct
  - again, must guess nextPC before the branch instruction is fetched (but branch target is encoded in the instruction)

  ⇒ Must make a guess based only on the current fetch PC !!!

  ⇒ Fortunately,
    - PC-offset branch/jump target is static
    - We are allowed to be wrong some of the time

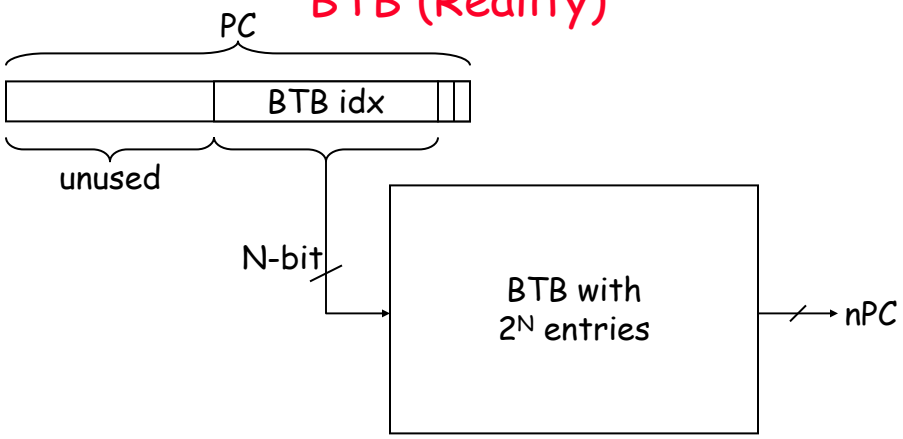
Branch Target Buffer (Oracle)



- BTB (Oracle)
  - a giant table indexed by PC
  - returns the guess for nextPC
- When encountering a PC for the first time, store in BTB
  - PC + 4 if ALU/LD/ST
  - PC+offset if Branch or Jump
  - ?? if JR
- Effectively guessing branches are always taken

  IPC = 1 / [1 + (0.20*0.3) * 2]
  = 0.89

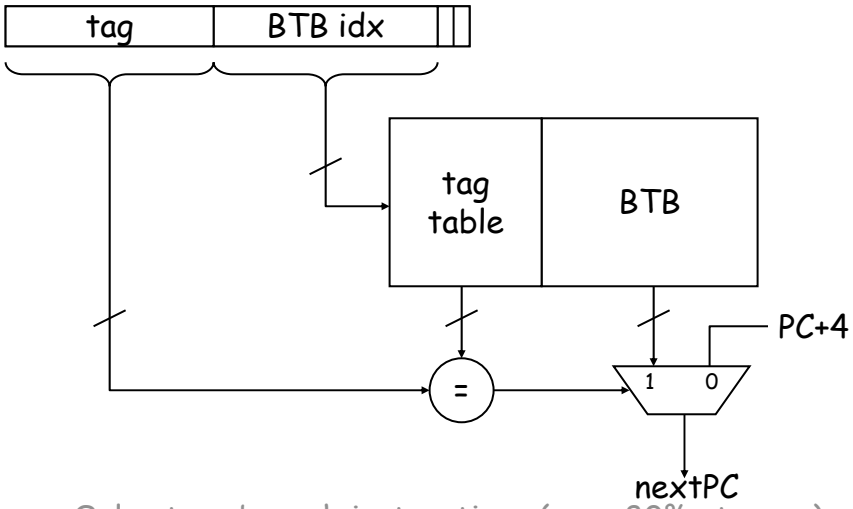
BTB (Reality)



- "Hash" PC into a 2^N entry table
- On collision, BTB returns something meaningless and possibly (since 80% of entries all hold PC+4) wrong

How big should this table be?

Tagged BTB



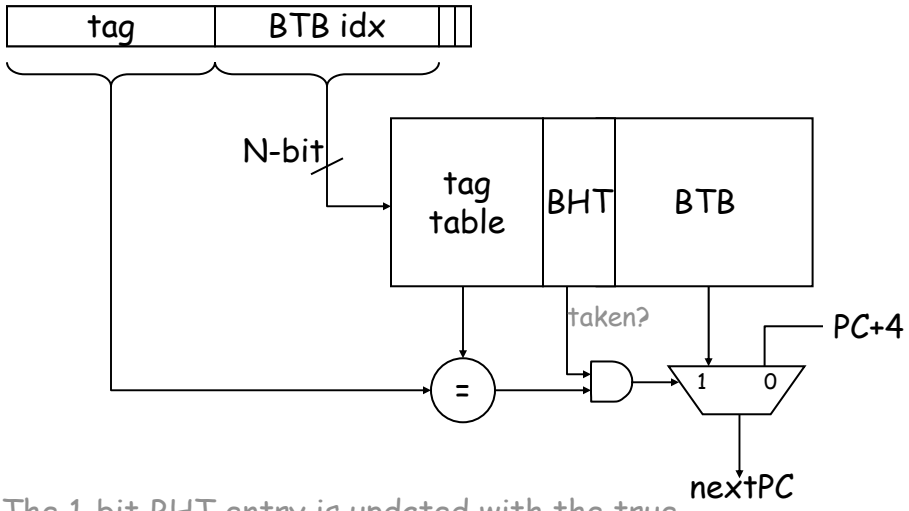
Only store branch instructions (save 80% storage)
Update tag and BTB for the new branch after each collision

Even Better Guess

- We can get 100% correct on non-branch instructions
- Can we do better than 70% on branch instructions?
  - We get 90% right on backward branch (dynamic)
  - We only get 50% right on forward branch (dynamic)

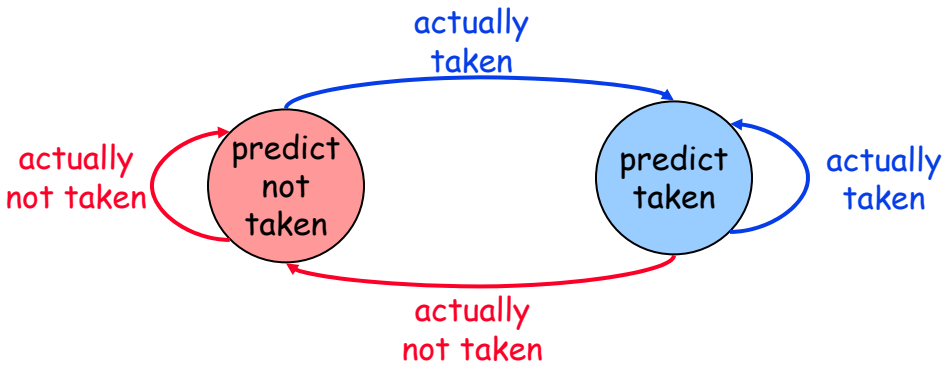
    What pattern can we leverage on forward branches?
- a given static branch instruction is likely to be highly biased in one direction (either taken or not taken
  - 80~90% correct if we always guessed the same outcome as the last time the same branch was executed
  - IPC = 1 / [1 + (0.20*0.15) * 2] = 0.94

Branch History Table and Target Buffer

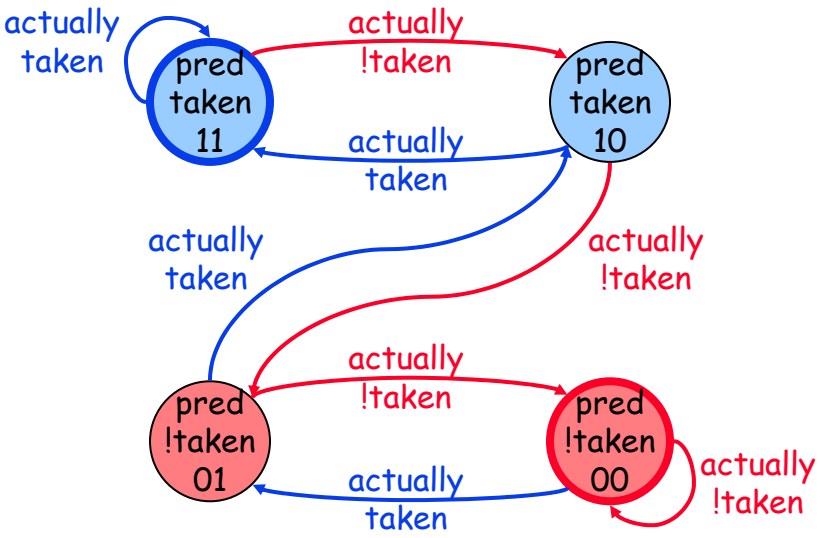


The 1-bit BHT entry is updated with the true outcome after each execution of a branch

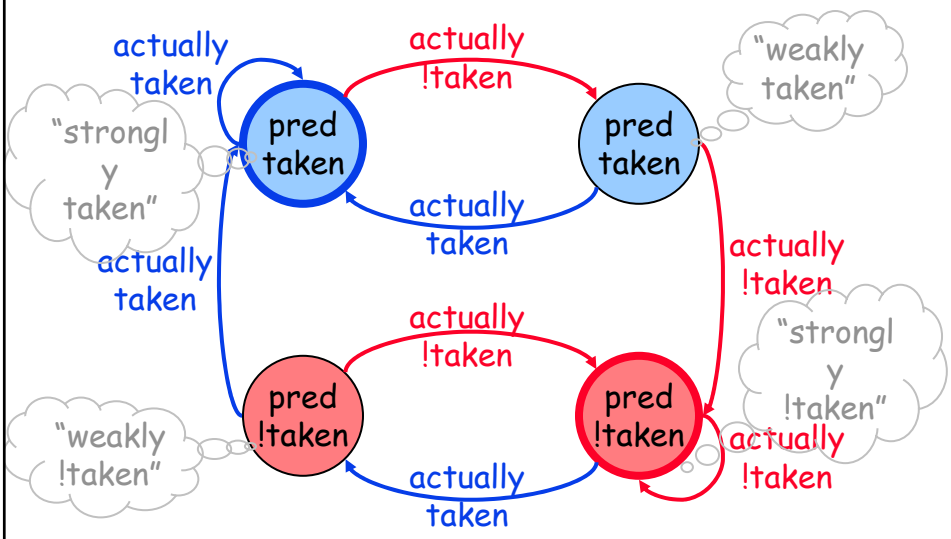
Branch Prediction State Machine



2-Bit Saturation Counter



2-Bit Hysteresis Counter



Change prediction after 2 consecutive mistakes

State-Machine-Based Predictors

- 2-bit predictor can get >90% correct
  - IPC = 1 / [1 + (0.20*0.10) * 2] = 0.96
  - any "reasonable" 2-bit predictor does about the same
- Adding more bits to counters does not help much more
- Major branch behaviors exploited
  - almost always do the same thing again and again (>80%)
    - 1-bit and 2-bit predictors equally effective
  - occasionally do the opposite once (5~10%)
    - 2 misprediction with a 1-bit predictor
    - 1 misprediction with a 2-bit predictor
  - miscellaneous (<10%)
    - some could be captured with more elaborate predictors
    - what does Amdahl's law say about this? (be careful!!)

Path History

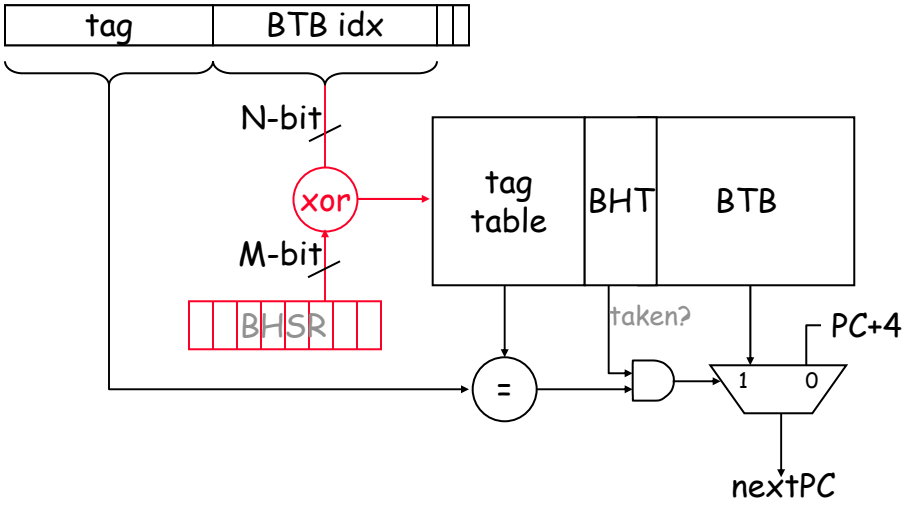
- Branch outcome can be correlated to other branches
- Equntott, SPEC92

```
if (aa==2)                  ;; B1
    aa=0;
if (bb==2)                  ;; B2
    bb=0;
if (aa!=bb) {               ;; B3
    ....
}
```

If B1 is not taken (i.e. aa==0@B3) and B2 is not taken (i.e. bb=0@B3) then B3 is certainly taken

How do you capture this information?

Gshare Branch Prediction [McFarling]



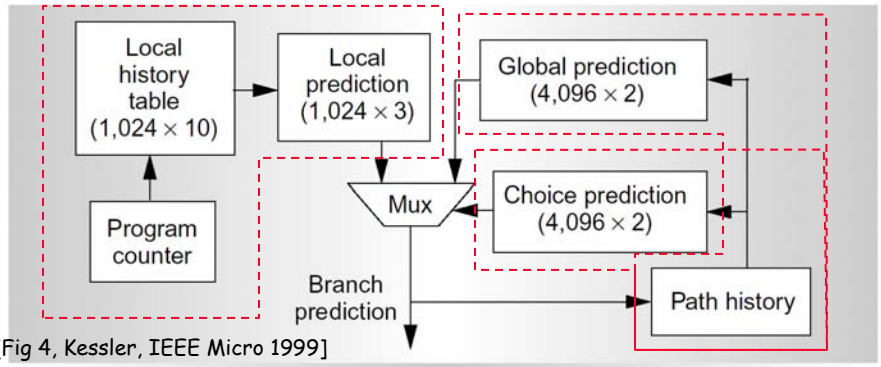
Global BHSR (Branch History Shift Register) tracks the outcomes of the last M branch instructions

Return Address Stack

- The targets of register-indirect jumps have little locality
  - history-based predictors don't work
  - but a simple "stack" captures the usage pattern of function call and return very well
- Return Address Stack (RAS)
  - the return address is pushed when a link instruction (e.g., JAL) is executed
  - when the PC of a return instruction (e.g., JR) is encountered predict nPC from the top of the stack and pop

What happens when the stack overflows?
How do you know when to follow RAS vs BTB?

Alpha 21264 Tournament Predictor

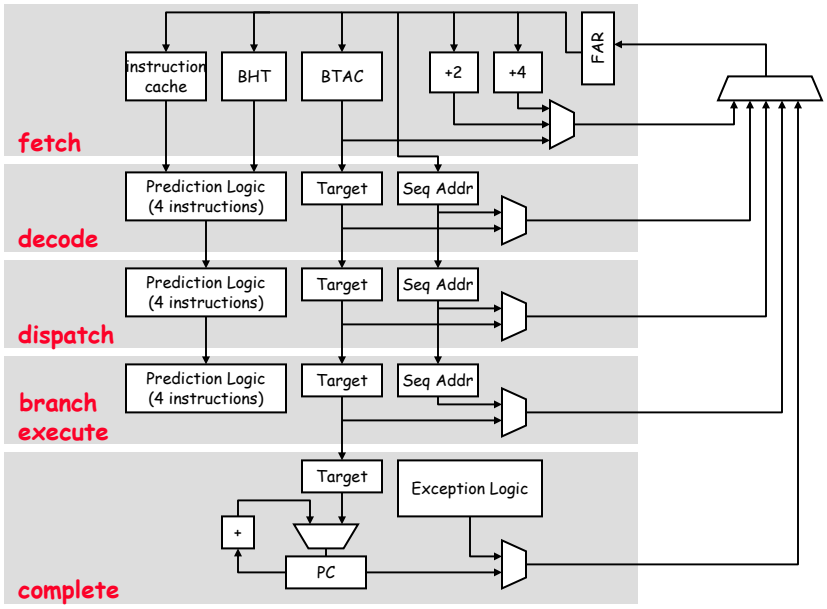


[Fig 4, Kessler, IEEE Micro 1999]

- Make separate predictions using local history (per branch) and global history (correlating all branches) to capture different branch behaviors
- A meta-predictor decides which predictor to believe

Better than 97% correct

Multiple Predictors: PPC 604



Speculative Execution Summary

- Each control flow instruction must carry the predicted nextPC down the pipeline
- When the control flow outcome of an instruction certain, the predicted nextPC is checked
- if nextPC was predicted correctly
  - update BHT (reinforce prediction)
  - do nothing more
- if nextPC was predicted incorrectly
  - update BHT and/or BTB
  - flush all younger instructions in the pipeline
  - restart fetching at the correct target

Involving SW in Branch Prediction

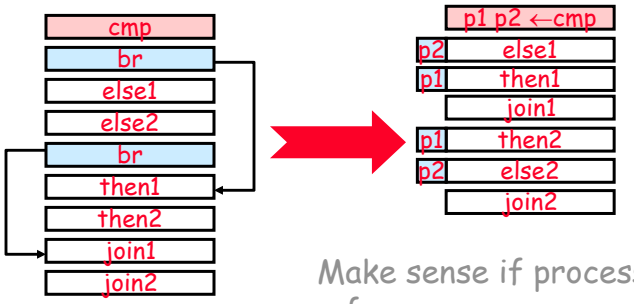
- Static branch hints can be encoded with every branch
  - taken vs. not-taken
  - whether to allocate an entry in the dynamic BP hardware
- SW and HW has joint control of BP hardware
  - Intel Itanium has a "brp" (branch prediction) instruction that can be issued ahead of the actual branch to preset the state of the BTB
- TAR (Target Address Register)
  - a small, fully-associative BTB
  - controlled entirely by "prepare-to-branch" instructions
  - a hit in TAR overrides all other predictions

Why wait until the last instruction in the basic block to calculate branch condition and target?

Predicated Execution: If-conversion

- Example: predication in Intel Itanium
  - each instruction can be separately predicated
  - 64 one-bit predicate registers

    each instruction carries a 6-bit predicate field
  - an instruction is effectively a NOP if its predicate is false
- Converts control flow into dataflow



Make sense if processors have lots of spare resources and BP is hard

Branch Prediction: the bottom-line

- Given current PC, how to determine the next PC

  waiting for anymore information would need stalls
- The easy part
  - the same PC always points to the same instruction (barring self-modifying code)
  - nextPC is always PC+4 for non-control-flow instructions,
  - the target of a PC-offset control-flow is always the same

    A memoization table can get these nearly100% right
- The not so easy part
  - taken versus not-taken decision is not static
    - 90% of backward branches are taken (loops)
    - 50% of forward branches are taken (if-then-else)
  - a given branch "almost" "always" repeats itself