18-640 Foundations of Computer Architecture

Lecture 5:
"Branch Prediction Techniques"

John Paul Shen
September 9, 2014

- Required Reading Assignment:
  - **Chapters 5 and 9 of Shen and Lipasti (SnL).**
- Recommended References:
  - T. Yeh and Y. Patt, "Two-Level Adaptive Branch Prediction," Intl. Symposium on Microarchitecture, November 1991.
  - Kessler, R. E., "The Alpha 21264 Microprocessor," IEEE Micro, March/April 1999, pp. 24-36 (available at ieeexplore.ieee.org).

18-640 Foundations of Computer Architecture

Lecture 5:
"Branch Prediction Techniques"

A. Branch Prediction Algorithms
   a. Branch Target Speculation
   b. Branch Direction Speculation
B. Advanced Branch Prediction
C. Wide Instruction Fetch
D. Still More on Branch Predictors

A. Branch Prediction Algorithms

Review from Lecture 4

Dynamic Branch Prediction Tasks

- Target Address Generation
  - Access register
    - PC, GP register, Link register
  - Perform calculation
    - +/- offset, auto incrementing/decrementing

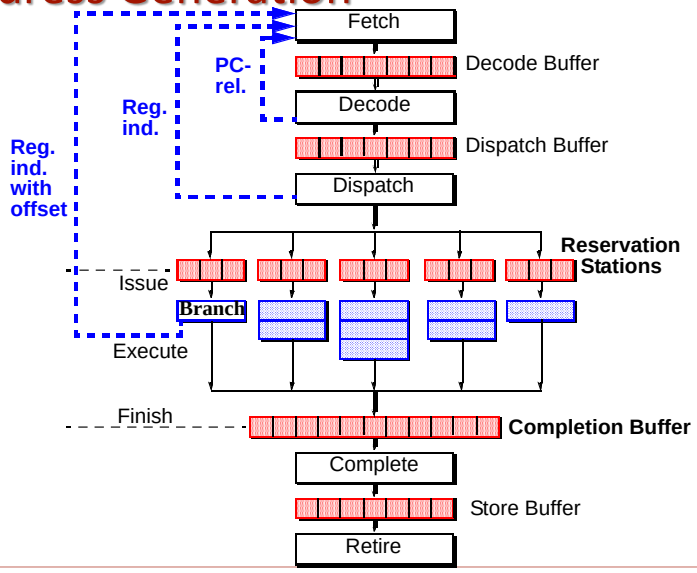
  ⇒ Target Speculation

- Condition Resolution
  - Access register
    - Condition code register, data register, count register
  - Perform calculation
    - Comparison of data register(s)

  ⇒ Condition Speculation

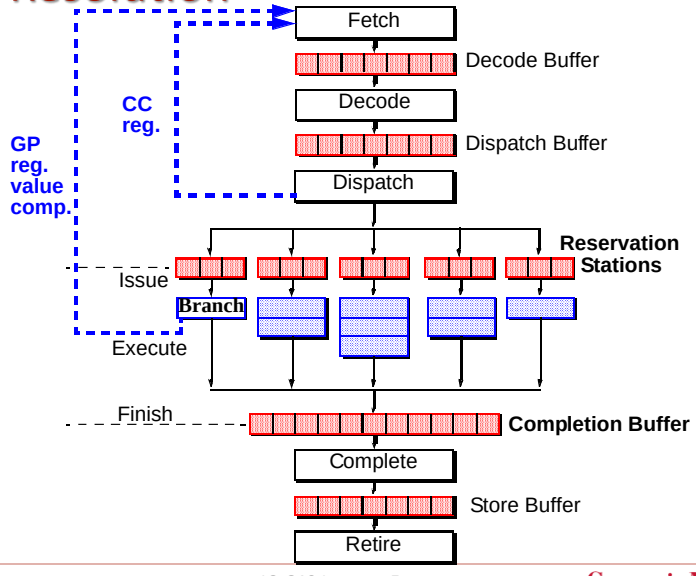
Review from Lecture 4

Target Address Generation



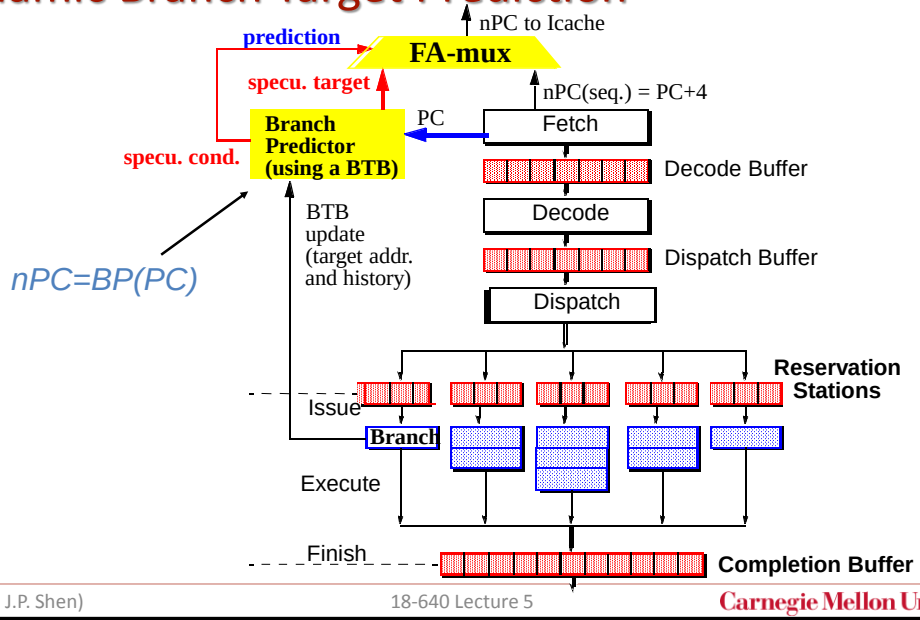
Review from Lecture 4

Condition Resolution



Review from Lecture 4

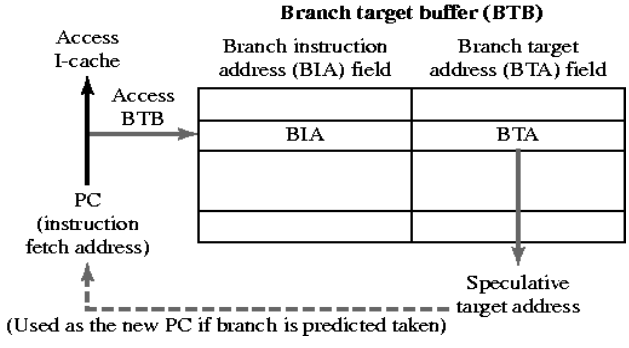
Dynamic Branch Target Prediction



Review from Lecture 4

Branch Target Prediction: Branch Target Buffer (BTB)

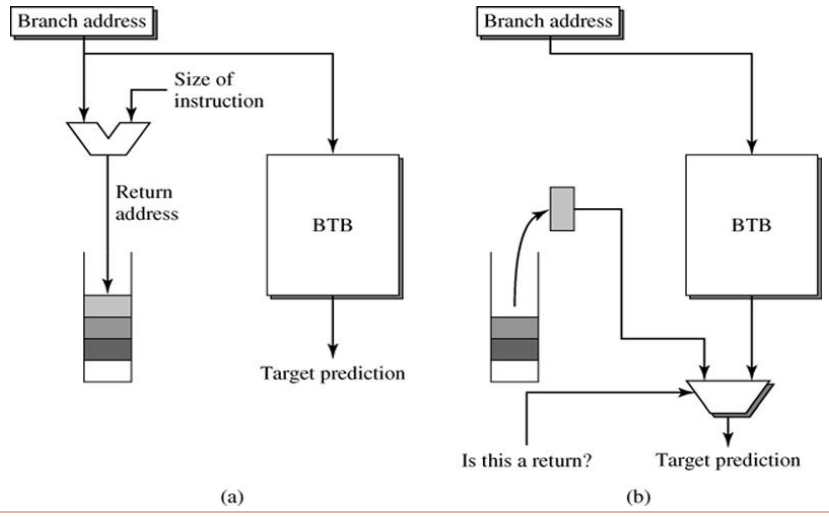
- A small "cache-like" memory in the instruction fetch stage
- Remembers previously executed branches, their addresses (PC), information to aid target prediction, and most recent target addresses
- I-fetch stage compares current PC against those in BTB to "guess" nPC
  - If matched then prediction is made else nPC=PC+4
  - If predict taken then nPC=target address in BTB else nPC=PC+4
- When branch is actually resolved, BTB is updated



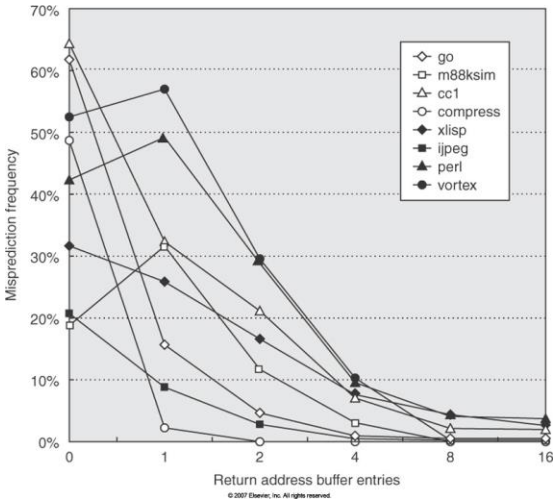
Branch Target Prediction for Function Returns

- In most languages, function calls are fully nested
  - If you call A() ⇒ B() ⇒ C() ⇒ D()
  - Your return targets are PCc ⇒ PCb ⇒ PCa ⇒ PCmain
- Return address stack (RAS)
  - A FILO structure for capturing function return addresses
  - Operation
    - On a function call retirement, push call PC into the stack
    - On a function return, use the top value in the stack & pop
  - A 16-entry RAS can predict returns almost perfectly
    - Most programs do not have such a deep call tree
  - Sources of RAS inaccuracies
    - Deep call statements (circular buffer overflow – will lose older calls)
    - Setjmp and longjmp C functions (irregular call semantics)

RAS Operation



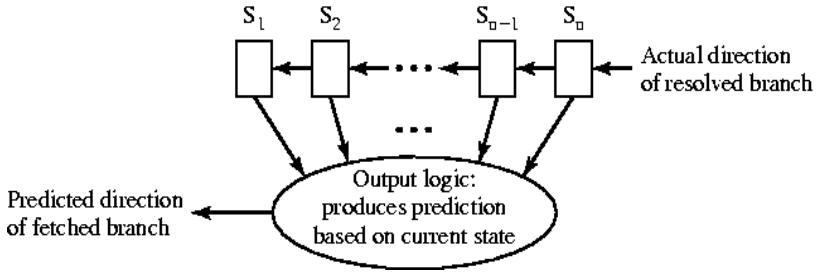
RAS Effectiveness & Size (SPEC CPU'95)



Branch Condition Prediction

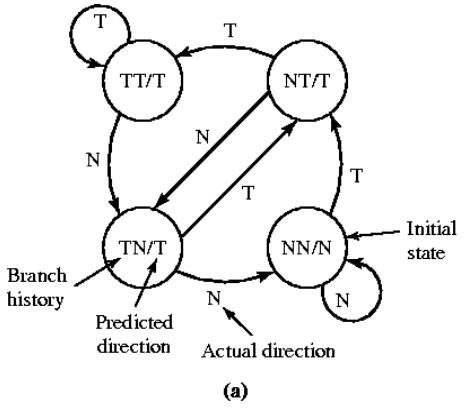
- Biased For Not Taken
  - Does not affect the instruction set architecture; Not effective in loops
- Software Prediction
  - Encode an extra bit in the branch instruction
    - Predict not taken: set bit to 0
    - Predict taken: set bit to 1
  - Bit set by compiler or user; can use profiling
  - Static prediction, same behavior every time
- Prediction Based on Branch Offsets
  - Positive offset: predict not taken
  - Negative offset: predict taken
- **Prediction Based on History**

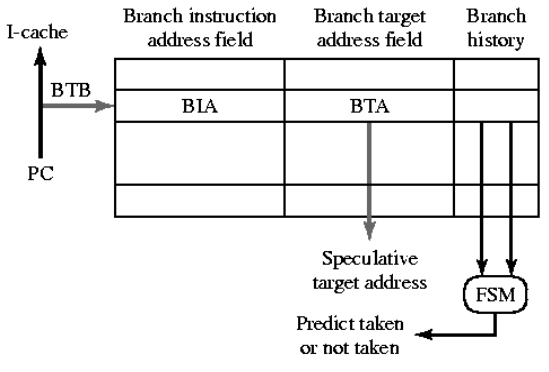
History-Based Branch Direction Prediction



- Track history of previous directions of branches (T or NT)
- History can be local (per static branch) or global (all branches)
- Based on observed history bits (T or NT), a FSM makes a prediction of Taken or Not Taken
- Assumes that future branching behavior is predictable based on historical branching behavior

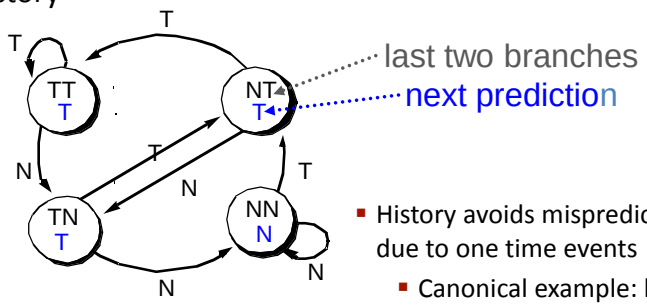
History-Based Branch Prediction





Example Prediction Algorithm

- Prediction accuracy approaches maximum with as few as 2 preceding branch occurrences used as history

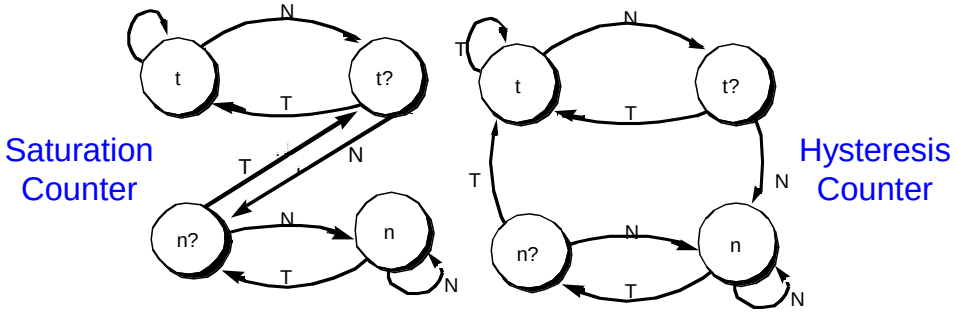


Results (%)

IBM1	IBM2	IBM3	IBM4	DEC	CDC
93.3	96.5	90.8	83.4	97.5	90.6

- History avoids mispredictions due to one time events
  - Canonical example: loop exit
- 2-bit FSM as good as n-bit FSM
- Saturating counter as good as any FSM

Other Prediction Algorithms

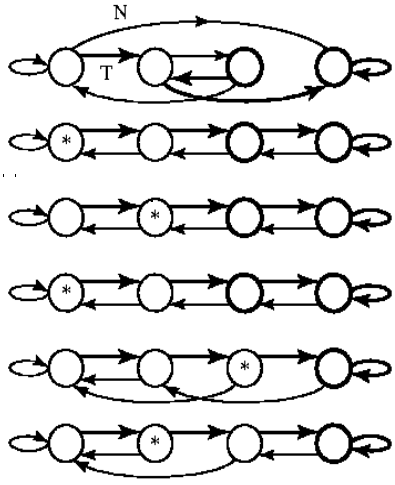


- Combining prediction accuracy with BTB hit rate (86.5% for 128 sets of 4 entries each), branch prediction can provide the net prediction accuracy of approximately 80%. This implies a 5-20% performance enhancement.

Optimal Predictor Exhaustive Search [Ravi Nair, IBM, 1992]

- There are 2^{20} possible state machines of 2-bit predictors
- Pruning uninteresting and redundant machines leaves 5248
- It is possible to exhaustively search and find the *optimal* predictor for a benchmark [Study by Ravi Nair, at IBM in 1992]

Benchmark	Optimal	"Counter"
spice2g6	97.2	97.0
doduc	94.3	94.3
gcc	89.1	89.1
espresso	89.1	89.1
li	87.1	86.8
eqntott	87.9	87.2



Saturation Counter is near optimal in all cases!

Number of Counter Bits Needed

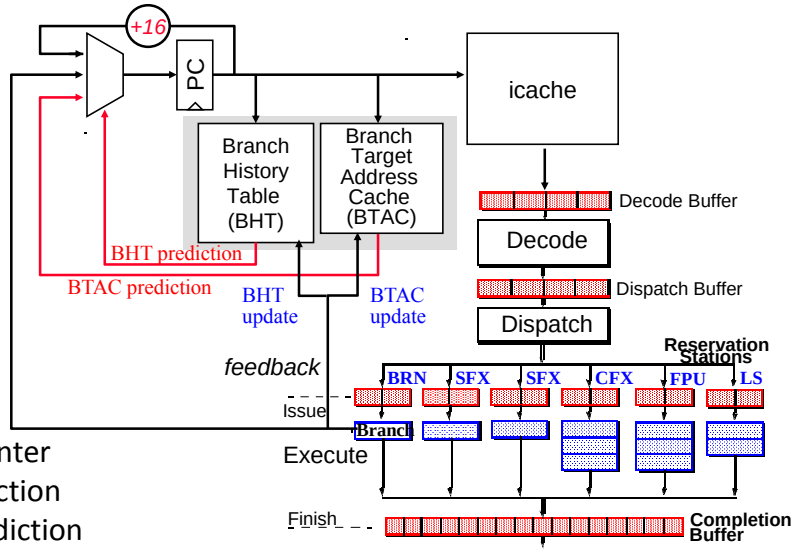
Benchmark	Prediction Accuracy (Overall CPI Overhead)			
	3-bit	2-bit	1-bit	0-bit
spice2g6	97.0 (0.009)	97.0 (0.009)	96.2 (0.013)	76.6 (0.031)
doduc	94.2 (0.003)	94.3 (0.003)	90.2 (0.004)	69.2 (0.022)
gcc	89.7 (0.025)	89.1 (0.026)	86.0 (0.033)	50.0 (0.128)
espresso	89.5 (0.045)	89.1 (0.047)	87.2 (0.054)	58.5 (0.176)
li	88.3 (0.042)	86.8 (0.048)	82.5 (0.063)	62.4 (0.142)
eqntott	89.3 (0.028)	87.2 (0.033)	82.9 (0.046)	78.4 (0.049)

- Branch history table size: Direct-mapped array of 2k entries
- Programs, like gcc, can have over 7000 conditional branches
- In collisions, multiple branches share the same predictor
- Variation of branch penalty with branch history table size level out at 1024

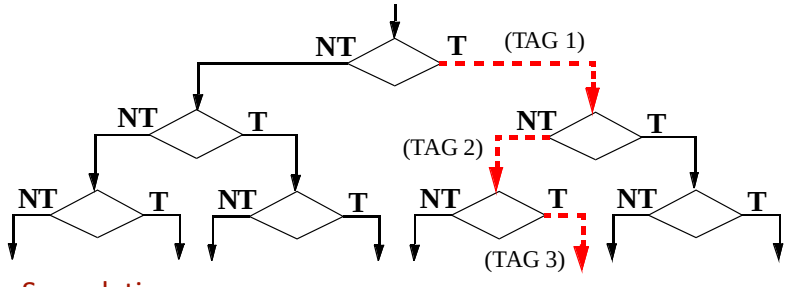
PowerPC 604 Branch Predictor: BHT & BTAC

BTAC:
- 64 entries
- Fully associative
- Hit → predict taken

BHT:
- 512 entries
- Direct mapped
- 2-bit saturating counter
- History based prediction
- Overrides BTAC prediction

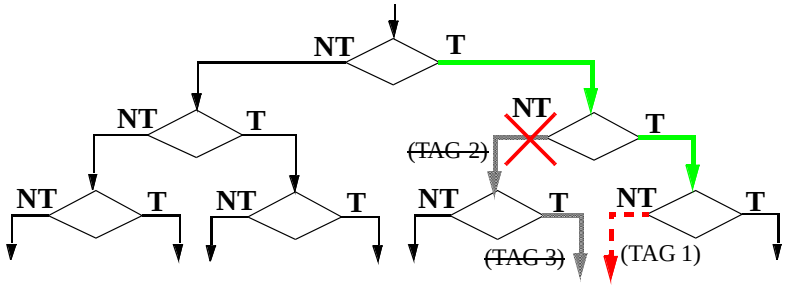


Control Flow Speculation



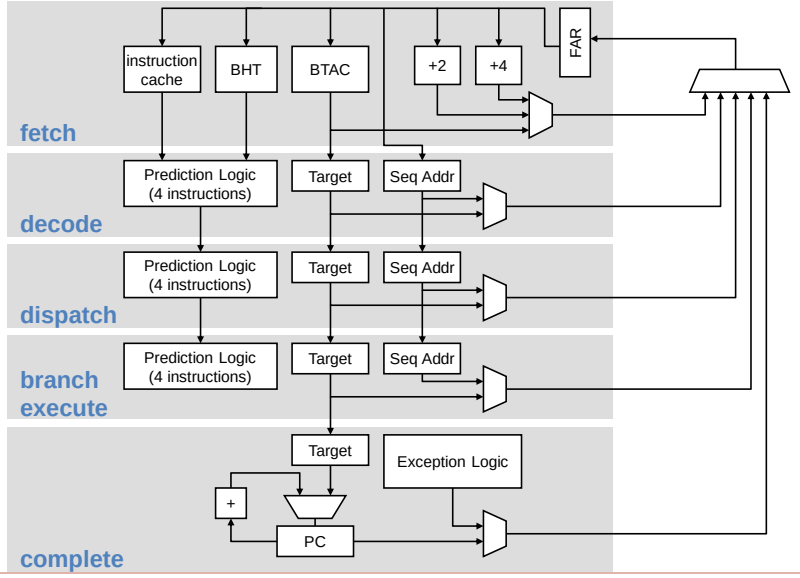
- Leading Speculation
  - Typically done during the Fetch stage
  - Based on potential branch instruction(s) in the current fetch group
- Trailing Confirmation
  - Typically done during the Branch Execute stage
  - Based on the next Branch instruction to finish execution

Mis-speculation Recovery



- Start New Correct Path
  - Must remember the alternate (non-predicted) path
- Eliminate Incorrect Path
  - Must ensure that the mis-speculated instructions produce no side effects

PowerPC 604 Fetch Address Generation



B. Advanced Branch Prediction

Electrical & Computer ENGINEERING

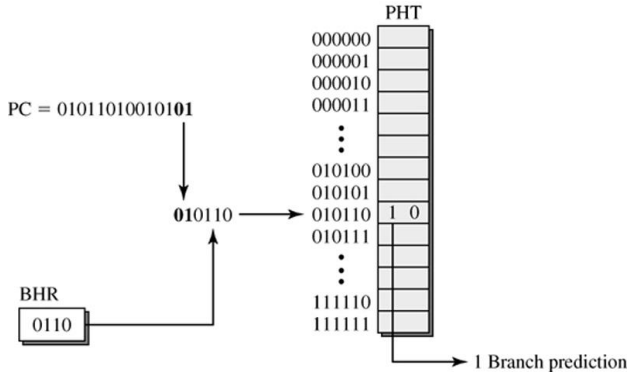
Branch Correlation

- So far, the prediction of each static branch instruction is based solely on its own past behavior and not the behaviors of other neighboring static branch instructions
- How about this one?

```
x=0;
If (someCondition) x=3;              /* Branch A*/
If (someOtherCondition) y+=19;  /* Branch B*/
If (x<=0) dosomething();             /* Branch C*?
```

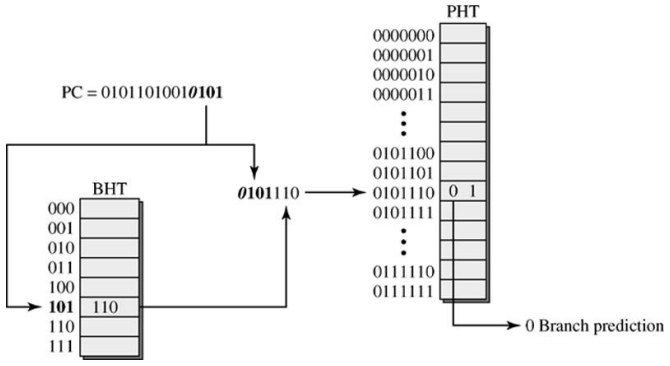
- Other correlation examples?

Global History Branch Predictor



- BHR: a shift register for global history
  - Shift in latest result in each cycle
- Advantages & shortcomings?

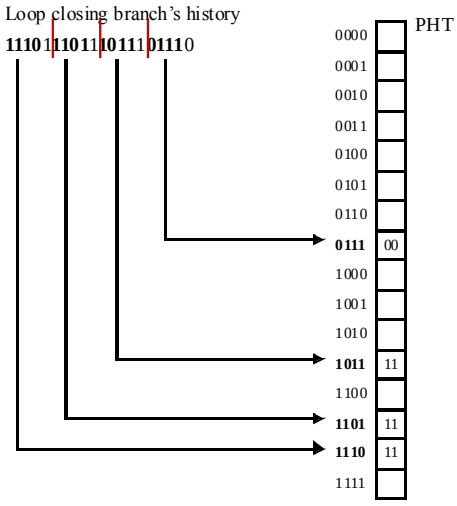
Local History Branch Predictor



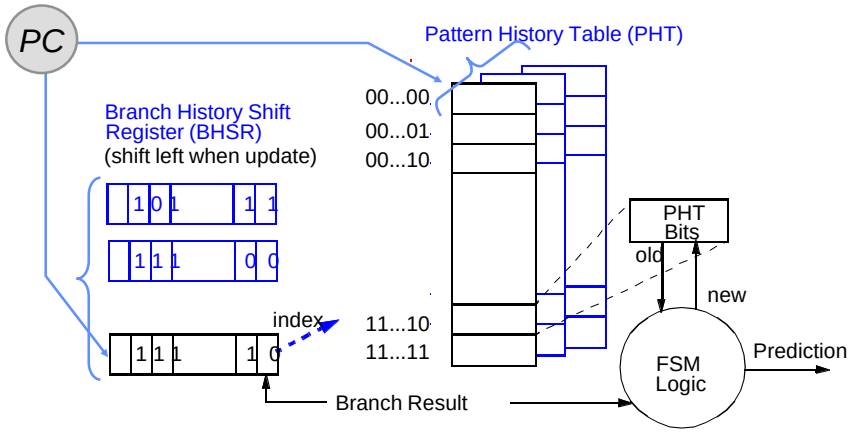
- What could be the patterns detected by such a predictor?
- Advantages & shortcomings?

Local History Predictor Example: Short Loops

- Must identify the last iteration of short loop
  - Predict its branch not-taken
- BHT allows us to use a different PHT entry for each iteration of the loop
  - In this case, the loop has 4 iterations
  - '0111' entry predicts not taken



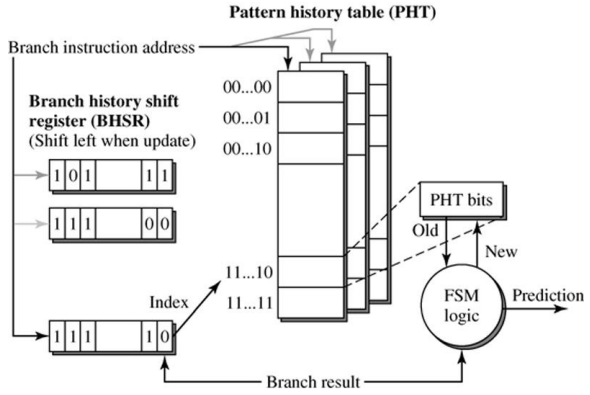
Two-Level Adaptive Branch Prediction



Two-level Adaptive Branch predictors: Taxonomy

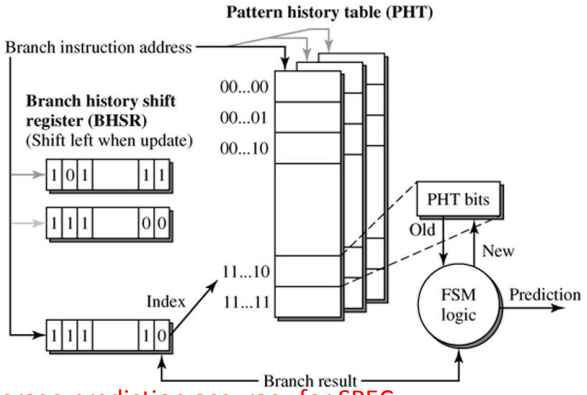
- Based on indices for branch history and pattern history
  - BHR: {G,P}: {Global history, Per-address history}
  - PHT: {g,p,s}: {Global, Per-address, Set}
    - g: use the BHR output as the address into the PHT
    - p: combine the BHR output with some bits from the PC
    - s: use an arbitrary hashing function for PHT addressing
  - 9 combinations: GAg, GAp, GAs, PAg, PAp, PAs, SAg, SAp and SAs
- Examples
  - Our global predictor so far is a GAp
  - Our local predictor so far was a PAp
- T. Yeh and Y. Patt. Two-Level Adaptive Branch Prediction. Intl. Symposium on Microarchitecture, November 1991.

2-Level Adaptive Branch Prediction [Yeh & Patt]



{G,P}: G for global history, P for local (individual history)
{g,p,s}: g for single PHT for all branches (PC bits not used for PHT indexing),
p a separate PHT selected by PC bits, s for shared PHT

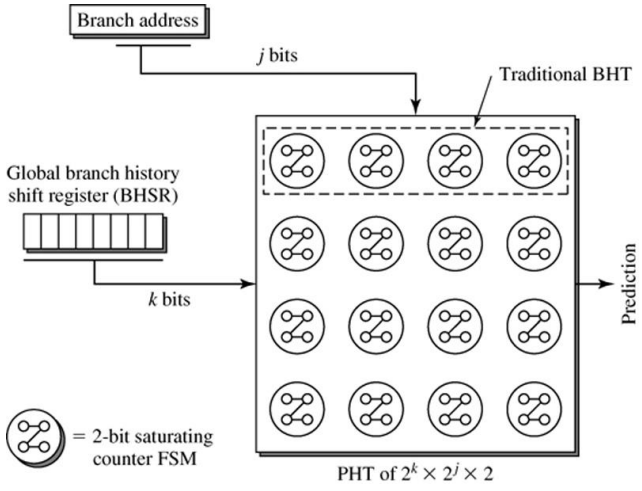
2-Level Adaptive Branch Prediction [Yeh & Patt]



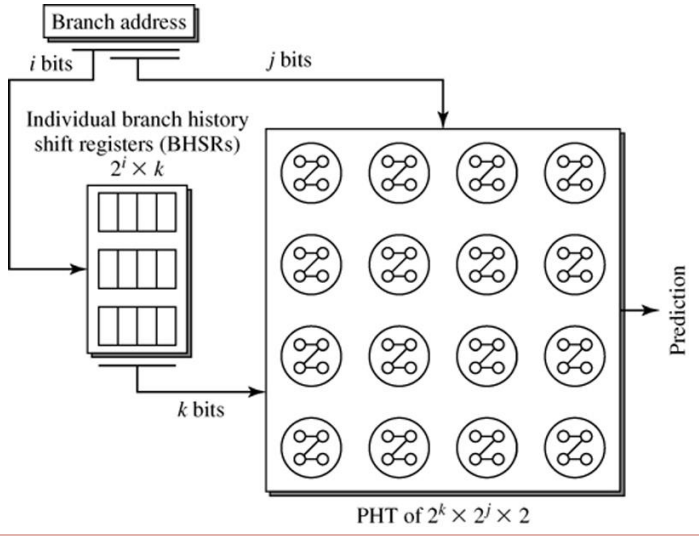
To achieve 97% average prediction accuracy for SPEC:

G (1) BHR: 18 bits;	g (1) PHT: 2^{18} x 2 bits	*total = 524 kbits*
P (512x4) BHR: 12 bits;	g (1) PHT: 2^{12} x 2 bits	*total = 33 kbits*
P (512x4) BHR: 6 bits;	s (512) PHT: 2^{6} x 2 bits	*total = 78 kbits*

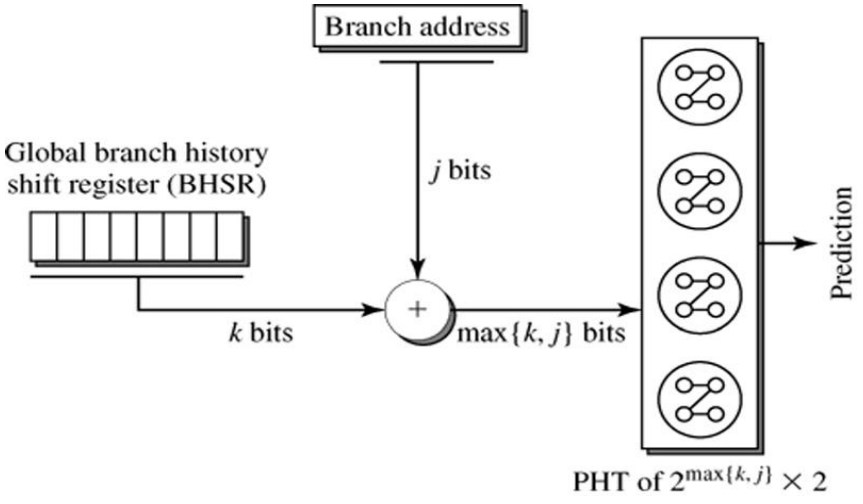
Example: Global BHSR Scheme (GAs)



Example: Per-Branch BHSR Scheme (PAs)



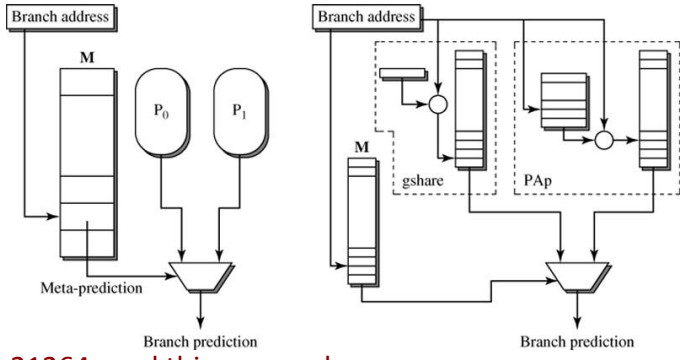
Gshare Branch Predictor [Scott McFarling]



Combining, Hybrid, or Tournament Branch Predictors

- What if different programs exhibit different patterns?
- Combining predictors: use multiple predictors
  - Each type tries to capture a particular program behavior
  - Use another history-based prediction scheme to "predict" which predictor should be used for a particular branch
  - You get the best of all worlds. This works quite well
- Variations:
  - Static prediction using software hints
  - Select from more than one alternative
    - Multi-hybrid and fusion predictors

Combining, Hybrid, or Tournament Branch Predictors

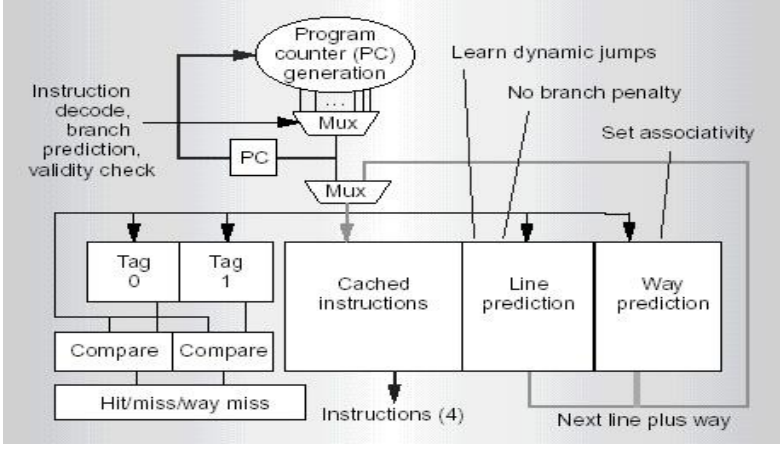


- E.g. Alpha 21264 used this approach
  - Predictor 1: a gshare with 12 bits of history (4K counters)
  - Predictor 2: a Pap with 1K history entries (10b) and 1K BHT
  - Selector: a 4K entry BHT

I-Cache and BTB Integration

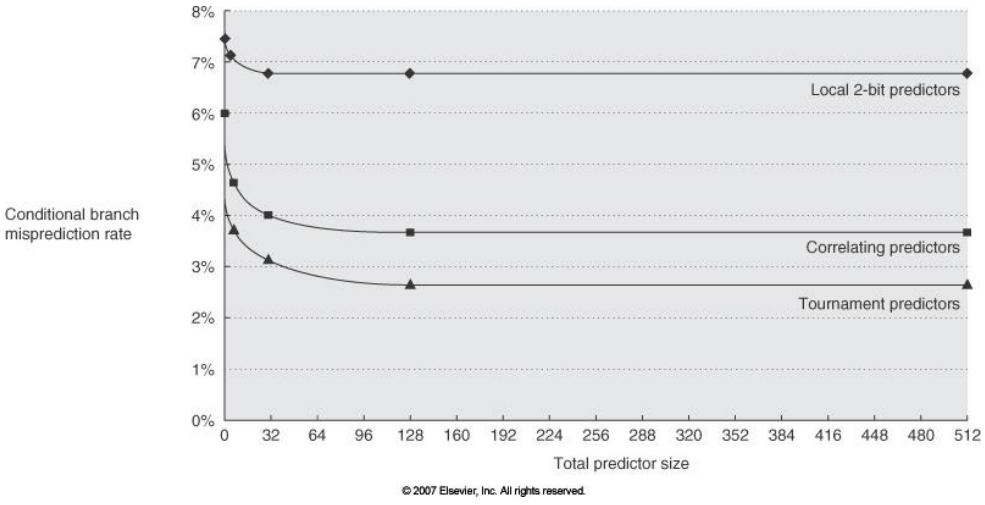
- Why does this make sense?
- Place a BTB entry in each cache line
  - Each cache line tells you which line to address next
  - Do not need the full PC, just an index the cache + a way select
  - This is called way & line prediction
- Implemented in Alpha 21264 processor
  - On refills, prediction value points to the next sequential fetch line.
  - Prediction is trained later on as program executes…
    - When correct targets are known…
  - Line prediction is verified in next cycle. If line-way prediction is incorrect, slot stage is flushed and PC generated using instruction info and direction prediction

Alpha 21264 Line & Way Prediction

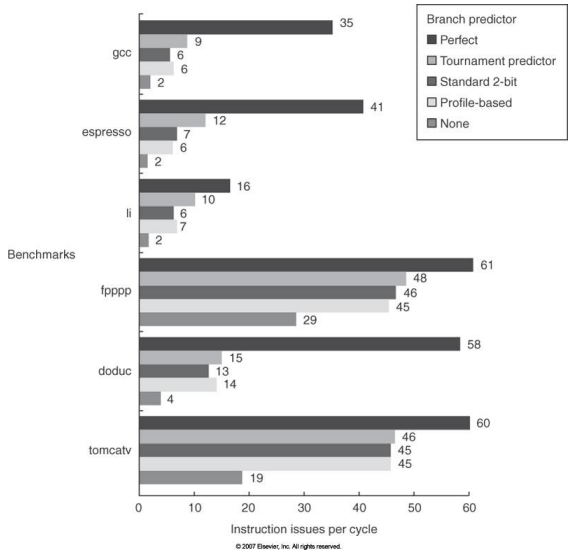


Source: IEEE Micro, March-April 1999

Comparison of Branch Predictor (SPEC'92)



Are We Done with Branch Predictors?

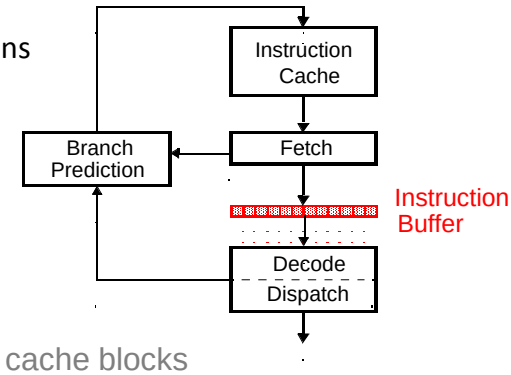


C. Wide Instruction Fetch

Electrical & Computer ENGINEERING

Impediments to Wide Fetching

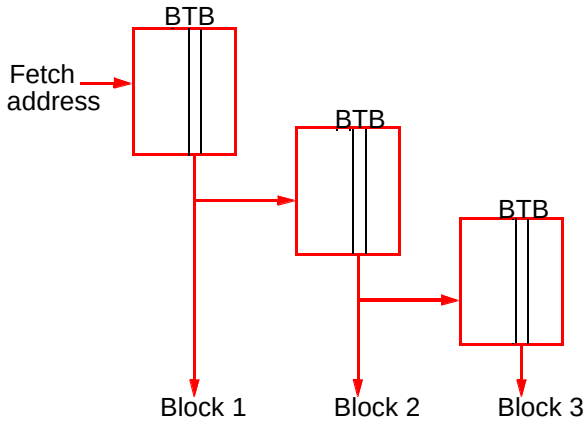
- Average Basic Block Size
  - integer code: 4-6 instructions
  - floating-point code: 6-10 instructions
- Three Major Challenges:
  - Multiple-Branch Prediction
  - Multiple Fetch Groups
  - Alignment and Collapsing



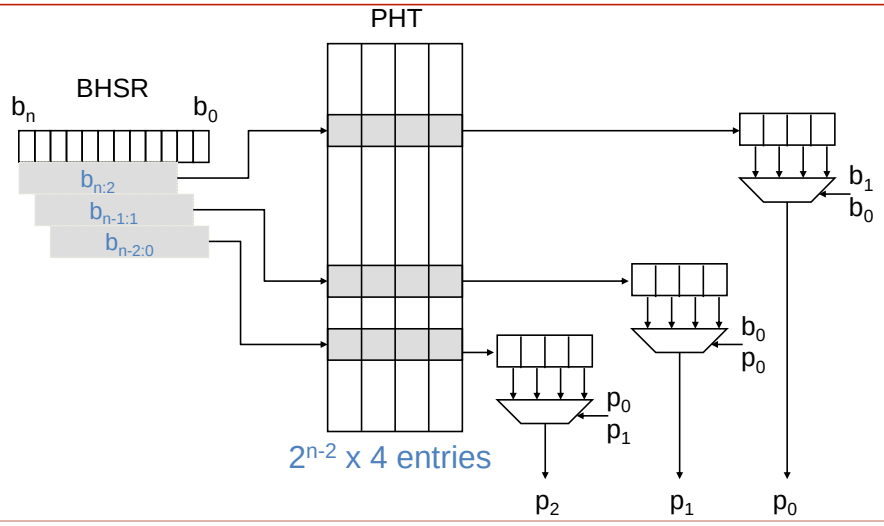
Cannot be solved with just longer cache blocks

Multiple Branch Predictions

- Issues with multiple branch predictions:
  - Latency resulting from sequential predictions
  - Later predictions based on stale/speculative history
  - Don't forget, 0.95x0.95x0.95=0.85

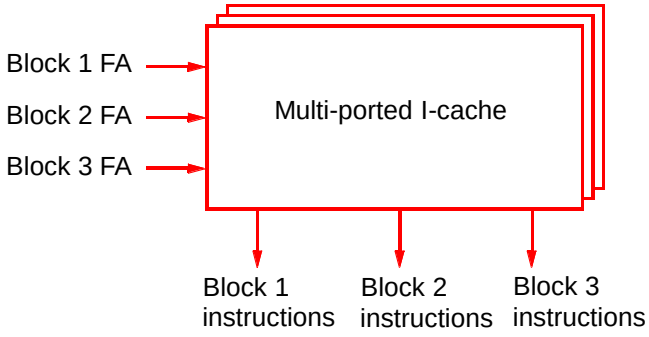


Examples of Multi-Branch Predictors



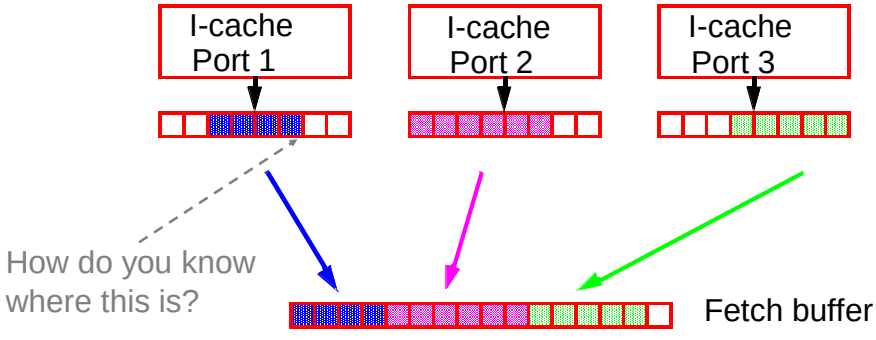
Multiple Predicted Taken Branches

- Issues with multiple taken branches:
  - Long latency with multiple sequential I-cache accesses
  - or, multi-ported I-cache with slower access latency
  - or, multi-banked I-cache to approximate multi-port

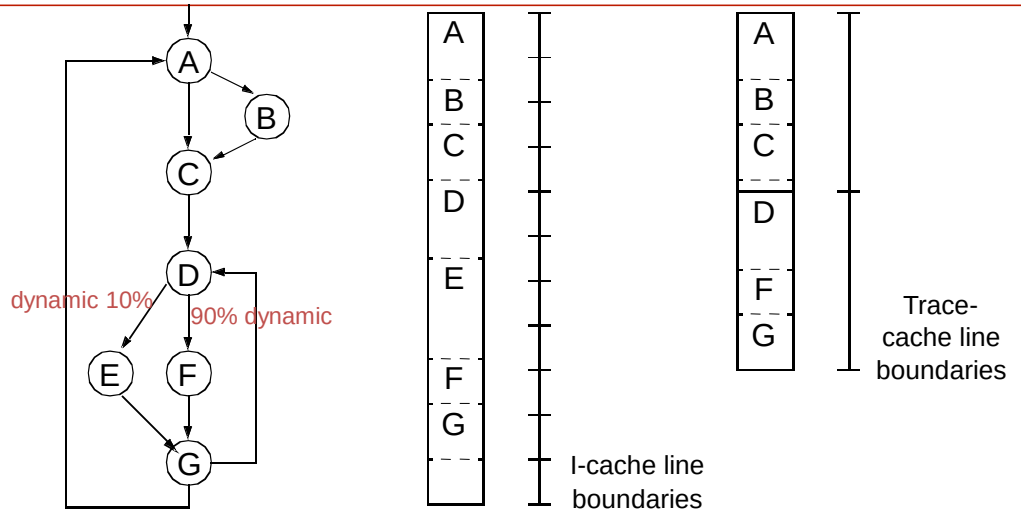


Instruction Alignment and Collapsing

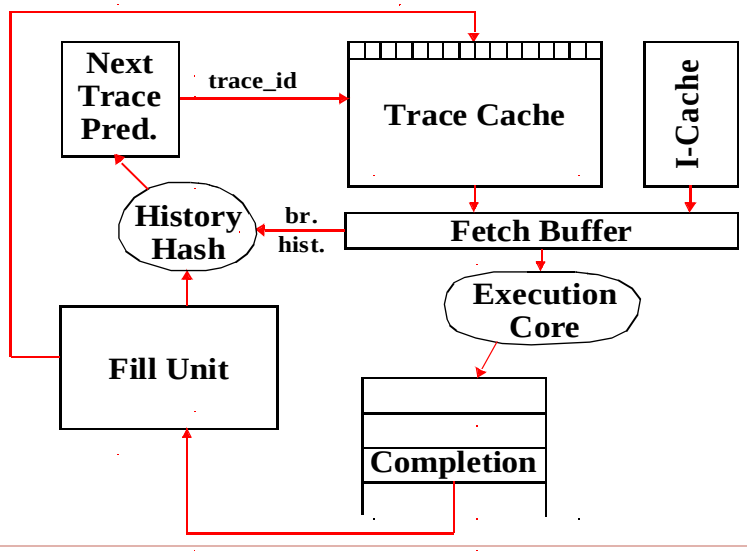
- Issues with alignment and collapsing:
  - Misalignment between fetch group and cache line.
  - Packing of variable-sized blocks into fetch buffer.



The Trace Cache Proposal



A Typical Trace Cache Organization



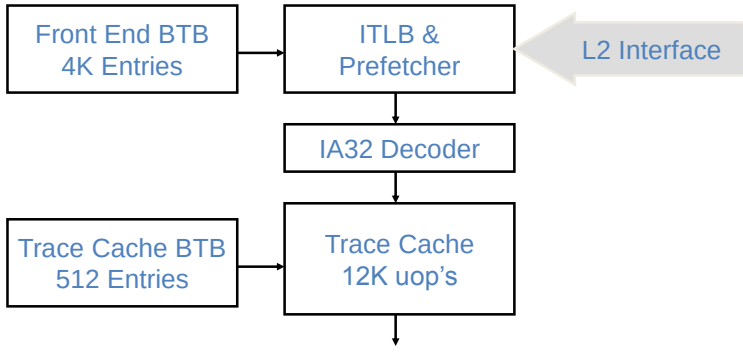
Trace Fill Unit

- Observe the dynamic execution sequence
- Gather instructions into a trace segment (or trace cache block)
- Some simple heuristics for forming trace segments
  - stop after collecting up to *N* instructions
    (*N* is the trace cache block size)
  - stop after *B* conditional branches
    (*B* is the limit of the multi-branch predictor)
  - stop after seeing an register-indirect jump
  - Don't split basic blocks
  - In some designs, unconditional and conditional branches can be dropped from the traces
- Can include pre-decoded dependence information
- Can even dynamically re-order instructions.

Intel Pentium 4 Trace Cache

- A 12K-uop trace cache replaces the L1 I-cache
- 6-uop per trace line, can include branches
- Trace cache returns 3-uop per cycle
- IA-32 decoder can be simpler and slower

Only needs to decode one IA-32 instruction per cycle



D. Still More on Branch Predictors

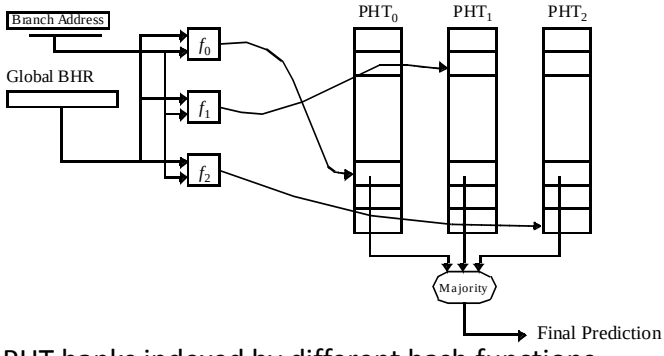
Causes for Mispredictions

- Fundamentally unpredictable branches
  - Cold miss, data dependent, …
- Training period
  - Need some time to warm up the predictor
  - The more patterns detected, the longer it takes to train
    - E.g. assume a global history predictor with 10 bits of history
    - Need to potentially train up to 2^10 entries for a specific branch
- Insufficient history to capture patterns
- Aliasing/interference
  - Branch direction predictors have limited capacity and no tags
  - Negative aliasing: two branches train same entry in opposite directions
  - Positive/neutral aliasing: two branches train same entry in same direction

Other Advanced Branch Predictors

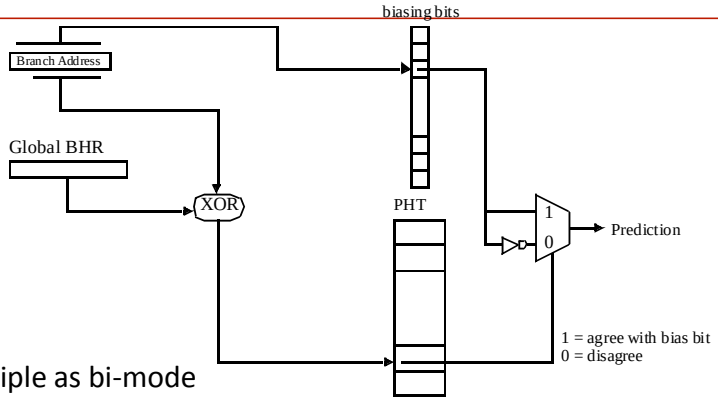
- Bi-mode predictor
  - Separate PHT for mostly taken and mostly non-taken branches (predictor to select)
  - Eliminates negative aliasing
- G-skew predictor (used in Alpha EV8)
  - Use multiple hash-functions into PHTs & vote on outcome
  - Reduce chance of negative interference affecting prediction
- Agree predictor
  - BTB gives you a basic prediction, extra PHT tells you if you should agree with the BTB
  - Biased branches have positive interference regardless of direction…
- YAGS
  - Keep a small tagged cache with branches that experience interference
- Other related ideas:
  - Branch filtering, selective branch inversion, alloyed history predictors, path history predictors, variable path length predictors, dynamic history length fitting predictors, loop counting predictors, percepton predictors, data-flow predictors, two-level predictors, analog circuit predictors, …

gskewed Predictor



- Multiple PHT banks indexed by different hash functions
  - Conflicting branch pair unlikely to conflict in more than one PHT
- Majority vote determines prediction
- Used in the cancelled Alpha 21464

Agree Predictor

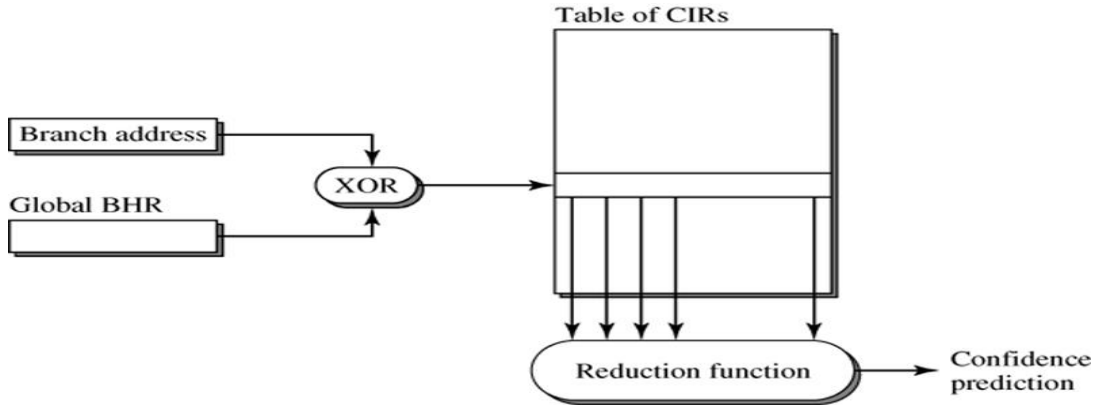


- Same principle as bi-mode
- PHT records whether branch bias matches outcome
  - Exploits 70-80% static predictability
- Used in in HP PA-8700

Prediction Confidence (Useful Tool for Speculation)

- Estimate if your prediction is likely to be correct
- Applications
  - Avoid fetching down unlikely path
    - Save time & power by waiting
  - Start executing down both paths (selective eager execution)
  - Switch to another thread (for multithreaded processors)
- Implementation
  - Naïve: don't use NT or TN states in 2-bit counters
  - Better: array of CIR (correct/incorrect registers)
    - Shift in if last prediction was correct/incorrect
    - Count the number of 0s to determine confidence
  - Many other implementations are possible
    - Using counters etc

Branch Confidence Prediction



Other Branch Prediction Related Issues

- Multi-cycle BTB
  - Keep fetching sequentially, repair later (bubbles for taken branches)
  - Need pipelined access though
- BTB & predictor in series
  - Get fast target/direction prediction from BTB only
  - After decoding, use predictor to verify BTB
    - Causes a pipeline mini-flush if BTB was wrong
  - This approach allows for a much larger/slower predictor
- BTB and predictor integration
  - Can merge BTB with the local part of a predictor
  - Can merge both with I-cache entries
- Predictor/BTB/RAS updates
  - Can you see any issue?