Dynamic Branch Prediction

Prediction changes as program behavior changes

More important today -- why?

Improve the prediction

- branch prediction buffer with:
  (predict branch taken if branched the last time & vice versa)
  - 1-bit branch prediction (the original, just a model now)
  - 2-bit branch prediction (A M S)
  - simple correlating predictor (X)

Determine the target earlier

- branch target buffer (X)
- branch folding (S)
- return address stack (A M X)

Reduce misprediction penalty

- fetch both instruction streams (IBM mainframes)
- resume cache (M)

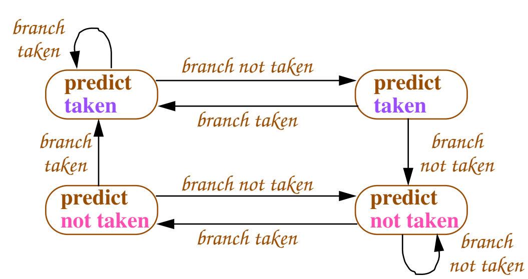
Improve the Prediction

Predict the branch-taken if branched the last time & vice versa

1-bit branch prediction buffer

- small cache indexed in the fetch stage by the lower bits of the address of a branch instruction
- contains a prediction bit/address
- toggle the bit if the prediction was wrong
- target address still calculated in the decode state

Two-bit branch prediction for loops: Jim Smith (A M S)



- **2 bit saturating counter:** increment when taken, decrement when not taken

Improve the Prediction

Correlating branch prediction: Yeh & Patt (X)

- the rationale: some branch outcomes are correlated

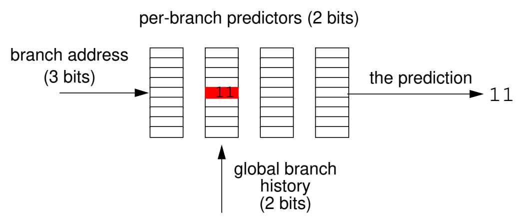
example: same cond var	*example: related cond var*
`if (d==0)`	`if (d==0)`
`...`	`b=1;`
`if (d==0)`	`if (b==1)`

  so having the prediction depend on only 1 branch outcome might produce bad predictions

  ⇒ use a **history of the past m branches**

  represents a path through the program

- how it works



  - global history is a **shift register**: shift left in the new branch outcome

Improve the Prediction

Combination of predictors (McFarland)

- local, per-branch 2-bit prediction, accessed by the PC
- global prediction based on the last *m* branches
- indicator of which had been the best predictor for this local/global particular pair (2 bits again)
- Alpha 21264
  - ~5% misprediction on SPEC95
  - 35K bits, 2% of die

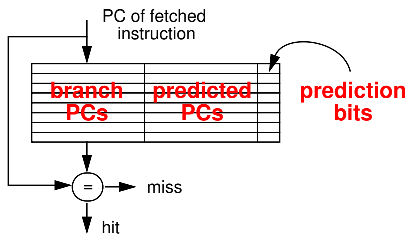
Predictor classification

- (m,n) predictors
  - m = history bits, number of branches in the history
  - n = prediction bits
- (0,1) = 1-bit branch prediction buffer
- (4,2) = Pentium Pro scheme

Determine the Target Earlier

Branch target buffer (BTB) (X)

- cache that stores: the PCs of branches *(the tag)*
  the predicted target address *(data)*
  optional branch prediction bits *(data)*
- accessed by PC address in fetch stage
  if hit: address was for a branch instruction
  fetch the target instruction if prediction bits say branch



- **no** branch delay if: branch found in BTB
  prediction is correct
  (assume BTB update can be done in next cycles)

Determine the Target Earlier

Return address stack (A M X)

- return address pushed on a call, popped on a return
- used for returns that are indirect jumps (85%)
- best for procedures that are called from multiple call sites
  - BTB would predict address of the return from the last call
- if big enough, can predict returns perfectly
  (typically 4-12 entries; A: 12 entries, M: 1 entry; p: 4 entries)

Determine the Target Earlier

Branch folding: Ditzel & McClellan)

- store target instructions in the BTB
- include unconditional branches in the BTB
- execute those instructions ***instead of*** the unconditional branch

$\Longrightarrow$ 0-time unconditional branches

- originally designed as a "next address field" in the I-cache on the Crisp microprocessor
- used on S and A21264 I-caches

Reduce the Misprediction Penalty

Fetch both instruction streams

- requires dual-ported instruction caches
- requires interleaved memory
- IBM high-end mainframes

Reduce the Misprediction Penalty

Resume cache

- technique used when don't know the target address until the end of decoding, so there is a one-cycle bubble in the pipeline
- save the sequential instructions in a cache
  - processor state also saved
- on a misprediction, reload these instructions from the resume cache
- shaves a cycle off he misprediction penalty

Today's Branch Prediction Strategy

Static and dynamic branch prediction work together
(S X, but assume all)

Predicting

- 2-bit prediction table (A: 2K entries, M: 512 entries)
- 2 bits/every 2 instructions in the I-cache (S: Ultra1)
- correlating branch prediction (X: 4 history bits)
  - Alpha 21264 will have a predictor for local & global predictors

BTB

- 512 entries, 4-way set associative (X)
- next address/every 4 instructions in the I-cache (S)
  - "address" = I-cache entry & set
  - Alpha 21264 will have this too

Return address stack (A M P X)

Resume cache (M)

Profiling feedback to the linker to do **instruction placement** (X)

Squash the branch delay cycle if a stall later down the pipeline (A)

Calculating the Cost of Branches

Factors to consider:

- branch frequency (every 4-6 instructions)
- misprediction rate
  - on SPEC92: A: 85% M: 87% S: 88% X:90%
  - on SPEC95: Alpha 21264: 95%
- misprediction penalty
  (A: 5 cycles; S: 5 cycles; X:15 cycles on average)

Static vs. Dynamic Branch Prediction

Today's processors rely on **dynamic** schemes for better prediction

- pipelines deeper

  ⇒ misprediction penalty greater
- multiple instructions issued at the same time (superscalar)

  ⇒ branch occurs almost every cycle

  ⇒ cost in instructions flushed in higher
- perhaps branches occur more often with modular, object-oriented programming styles

  ⇒ problem crops up more often
- other forms of pipeline stalling being addressed

  ⇒ portion of CPI due to branch delays is relatively more important

  what design principle applies here?
- hardware cost is small compared to the performance gain

Prediction Research

Predicting variable values

Predicting load addresses

Predicting many levels of branches