Importance of Branch Prediction

- DLX/MIPS R2000 -- branch hazard of 1 cycle, 1 instruction issued per cycle
  - delayed branch
- next generation – 2-3 cycle hazard, 1-2 instructions issued per cycle
  - cost of branch misprediction goes up
- Pentium Pro -- 17+ cycle misprediction penalty, 3+ instructions issued per cycle
  - HUGE penalty for mispredicting a branch

Branch Prediction

- Easiest (static prediction)
  - always not taken, always taken
  - forward not taken, backward always taken
  - compiler predicted (branch likely, branch not likely)
- Next easiest
  - remember last taken/not taken per branch (1 bit)
  - per branch approximated
    - per I cache line
    - use part of address
  - what happens on a loop?

2-bit branch prediction

- has 4 states instead of 2, allowing for more information about tendencies.
  
  ![2-bit branch prediction diagram]

- Loops?
Two different 2-bit schemes

- Strongly Taken: 11
- Weakly Taken: 10
- Weakly Not Taken: 01
- Strongly Not Taken: 00

Branch History Table

- has limited size
- 2 bits by N (e.g. 4K)
- uses low bits of branch address to choose entry

- what happens when table too small?
- what about even/odd branch?

2-bit prediction accuracy

Can We Do Better?

- Can we get more information dynamically than just the history of this branch?
- We can look at patterns (2-level predictor) for a particular branch:
  - last eight branches 00100100, then it is a good guess that the next one is “1” (taken)

- even/odd branch?
Can We Do Better?

• Correlating Branch Predictors also look at other branches for clues
  if (i == 0)
  ...
  if (i > 7)
  ...
• Typically use two indexes
  – Global history register $\rightarrow$ history of last $m$ branches (e.g., 0100011)
  – branch address

Correlating Branch Predictors

• The global history register is a shift register that records the last $n$ branches (of any address) encountered by the processor.

Two-level correlating branch predictors

• Can use both the PC address and the GHR

2-level branch prediction

• If we concatenate the GHR and the PC, we get...
  • This is a (2,2) scheme (2 bits of global history, 2-bit predictors)
  • Could also XOR GHR and PC (gshare).
  – Eg, a typical gshare might xor 12 bits of PC, 12 bits of GHR, into a 4K-entry table.
Performance of 2-level Correlating Branch Prediction

- Combining branch predictors use multiple schemes and a voter to decide which one typically does better for that branch.

But...

- When do we need to do the prediction to avoid any control hazards on a correct prediction?
- A taken/not taken prediction only helps us if...?

Branch Target Buffers

- predict the location of branches in the instruction stream
- predict the destination of branches
BTB Operation

- use PC (all bits) for lookup
  - match implies this is a branch
- if match and predict bits \( \Rightarrow \) taken, set PC to predicted PC
- if branch predict wrong, must recover (same as branch hazards we’ve already seen)
  - but what about dynamically scheduled processor??
- if decode indicates branch when no BTB match, two choices:
  - look up prediction now and act on it
  - just predict not taken
- when branch resolved, update BTB (at least prediction bits, maybe more)

BTB Performance

- Two things that can go wrong
  - didn’t predict the branch (misfetch)
  - mispredicted a branch (mispredict)
- Suppose BTB hit rate of 85% and predict accuracy of 90%, misfetch penalty of 2 cycles and mispredict penalty of 5 cycles, what is average branch penalty?
- Can use both BTB with BPT
  - have no prediction bits in BTB (why is that a good idea?)
  - presence of PC in BTB indicates a lookup in BPT to predict whether the branch will go to destination address in BTB.

What about indirect jumps/returns?

- BPT does really well with conditional jumps
- BTB does really well with unconditional jumps (jump, jal, etc.)
- Indirect jumps often jump to different destinations, even from the same instruction. Indirect jumps most often used for return instructions.
- Return easily handled by a stack.
  - jal \( \Rightarrow \) push PC+4
  - return \( \Rightarrow \) predict jump to address on top of stack, pop stack

Real BP -- PowerPC 620

- 256-entry 2-way set-associative BTB
- 2048-entry BHT
- return-address stack
**Pentium Pro**

- 512-entry BTB 4-way set-associative
- 2-level predictor (1st level in BTB, one per set, 4 bits)
- return stack

![Pentium Pro Diagram]

**Branch Prediction Key Points**

- The better we predict, the behinder we get.
- 2-bit predictors capture tendencies well.
- Correlating predictors improve accuracy, particularly when combined with 2-bit predictors.
- Accurate branch prediction does no good if we don’t know there was a branch to predict.
- BTB identifies branches in (or before) IF stage.
- BTB combined with branch prediction table identifies branches to predict, and predicts them well.

**Compaq/Digital Alpha 21264**

- next-cache-line field in I-cache replaces BTB
- return address stack

![Compaq/Digital Alpha 21264 Diagram]