ECE 571 – Advanced Microprocessor-Based Design Lecture 11

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

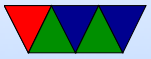
27 September 2024

Announcements

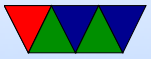
- HW#4 will be posted (branch predictors)
  will use AMD and ARM machines

- Interesting article on AMD branch predictor

  `https://grsecurity.net/amd_branch_mispredictor_just_set_it_and_forget_it`

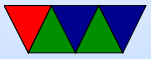


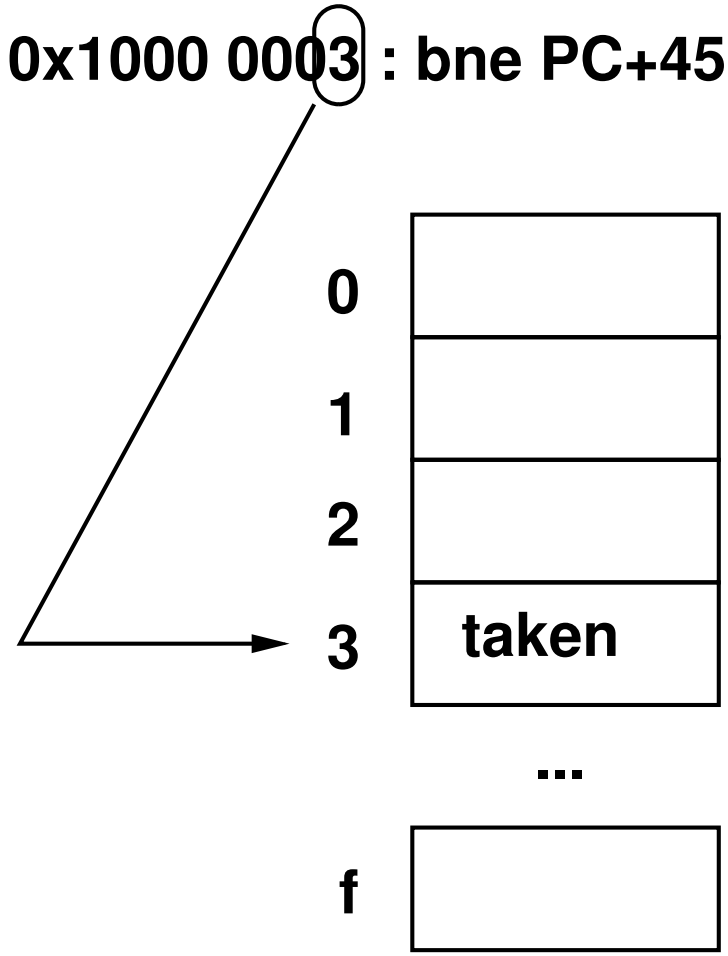
Dynamic Branch Prediction

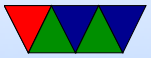


One-bit Branch History Table

- Table, likely indexed by lower bits of instruction
  (Why low bits and not high bits?)
- Can have more advanced indexing to avoid aliasing
  no-tag bits, unlike caches aliasing does not affect correct program execution
- One-bit indicating what the branch did last time
- Update when a branch miss happens

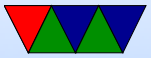



0x1000 0003 : bne PC+45
0
1
2
3
taken
...
f



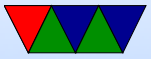
Branch History Table Behavior

- Two misses for each time through loop. Wrong at exit of loop, then wrong again when restarts. (so actually worse than static on loops)
- If/Then potentially better than static if long runs of true/false but can be worse if completely random.



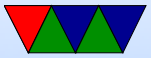
Aliasing

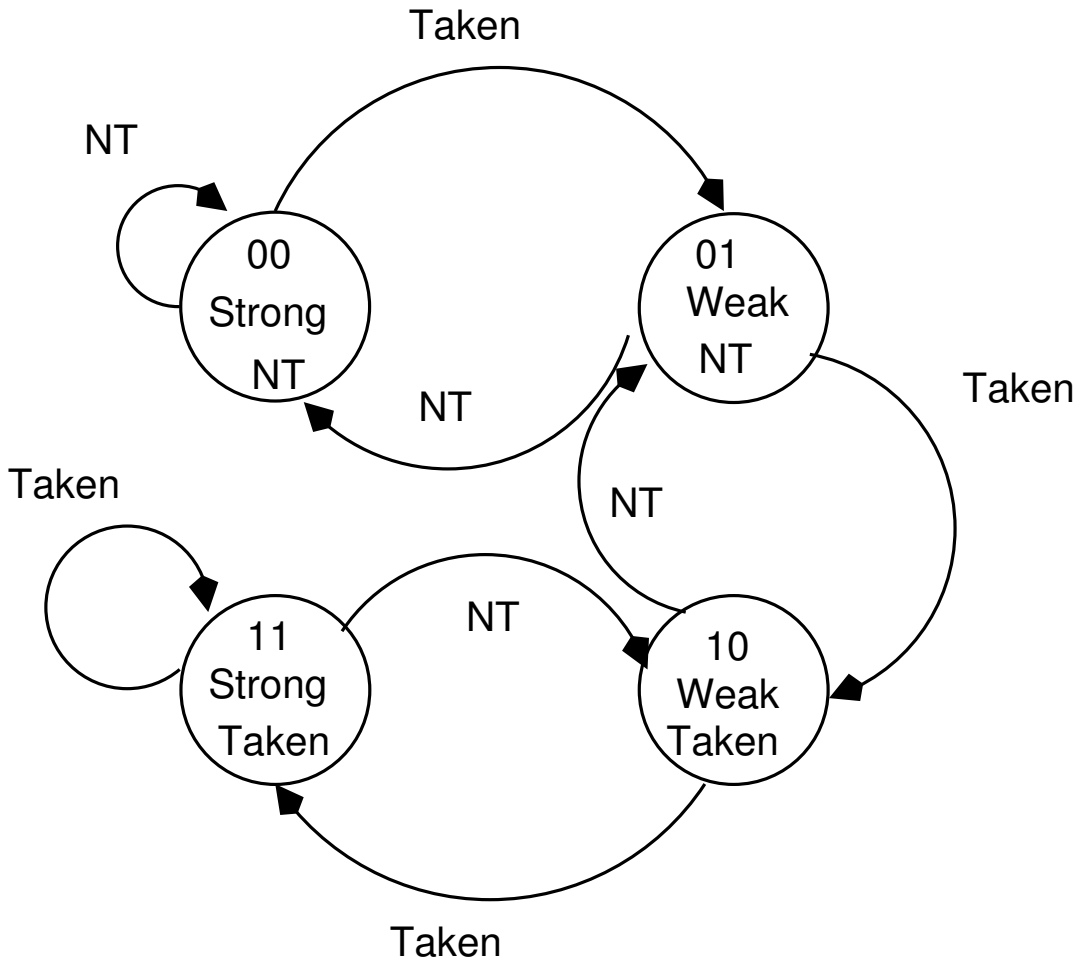
- Is it bad? Good?
- Does the O/S need to save on context switch?
- Do you need to save if entering low-power state?

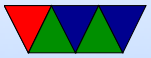


Two-bit saturating counter

- Hysterisis
- Use saturating 2-bit counter
- If 3/2, predict taken, if 1,0 not-taken. Takes two misses or hits to switch from one extreme to the next, letting loops take only one mispredict.
- Needs to be updated on every branch, not just for a mispredict

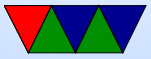



Taken
NT
00
Strong
NT
01
Weak
NT
Taken
NT
NT
Taken
11
Strong
Taken
NT
10
Weak
Taken
Taken

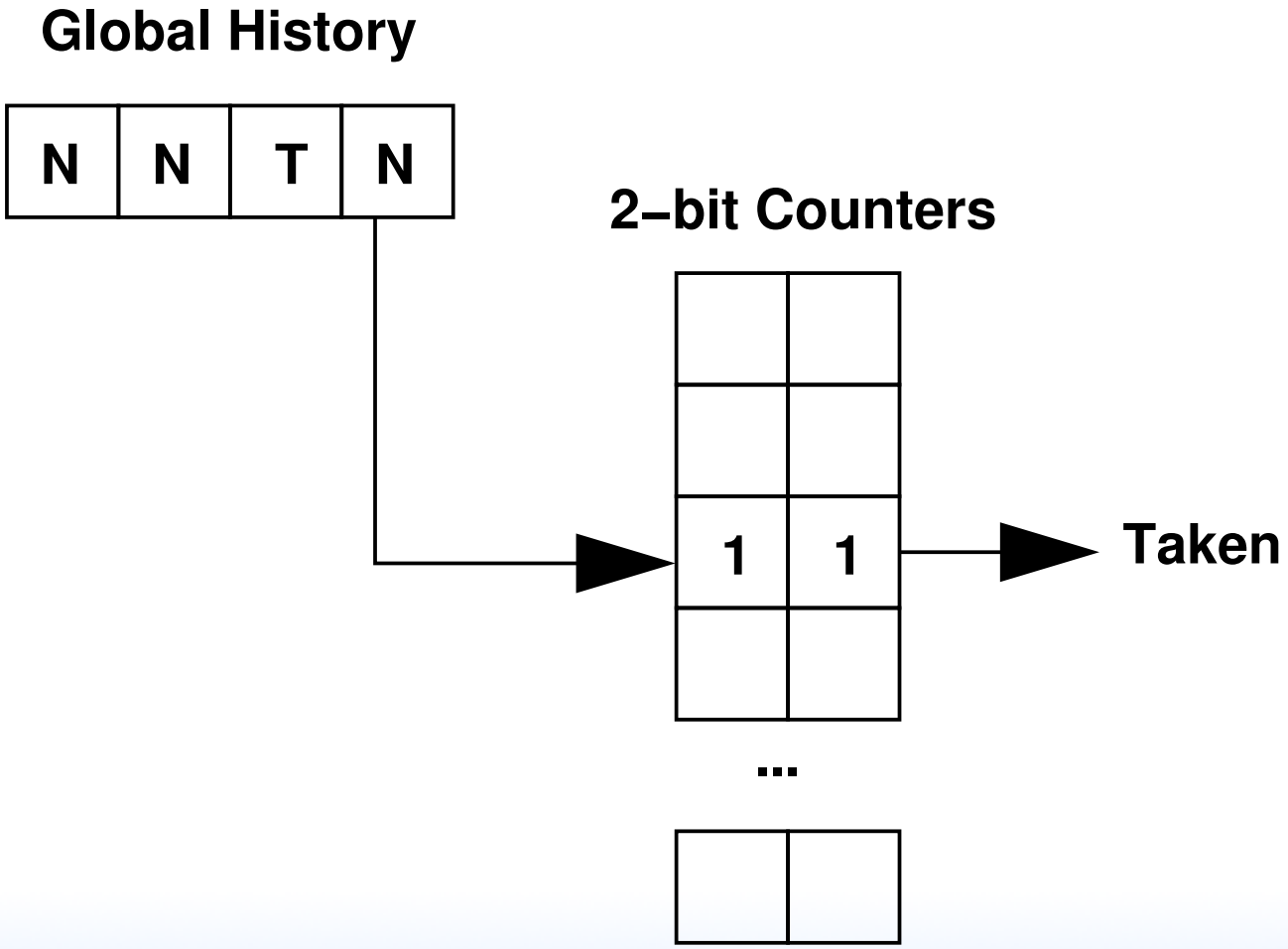


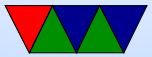
Branch History

- Can use branch history as index into tables
- Use a shift register to hold history
- Global vs Local
  - Global: history is all branches
  - Local: store branch history on a branch by branch basis

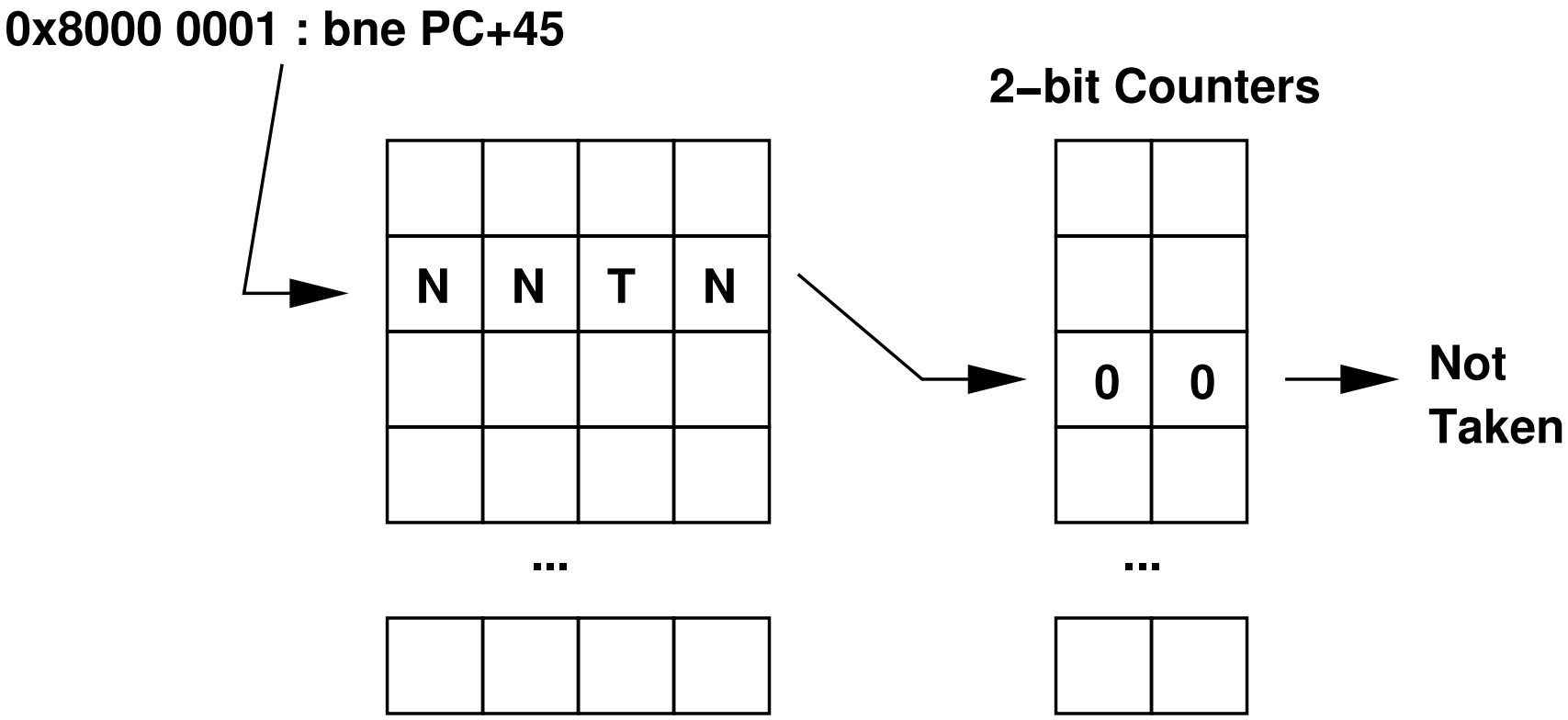


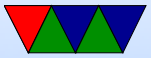
Global Predictor





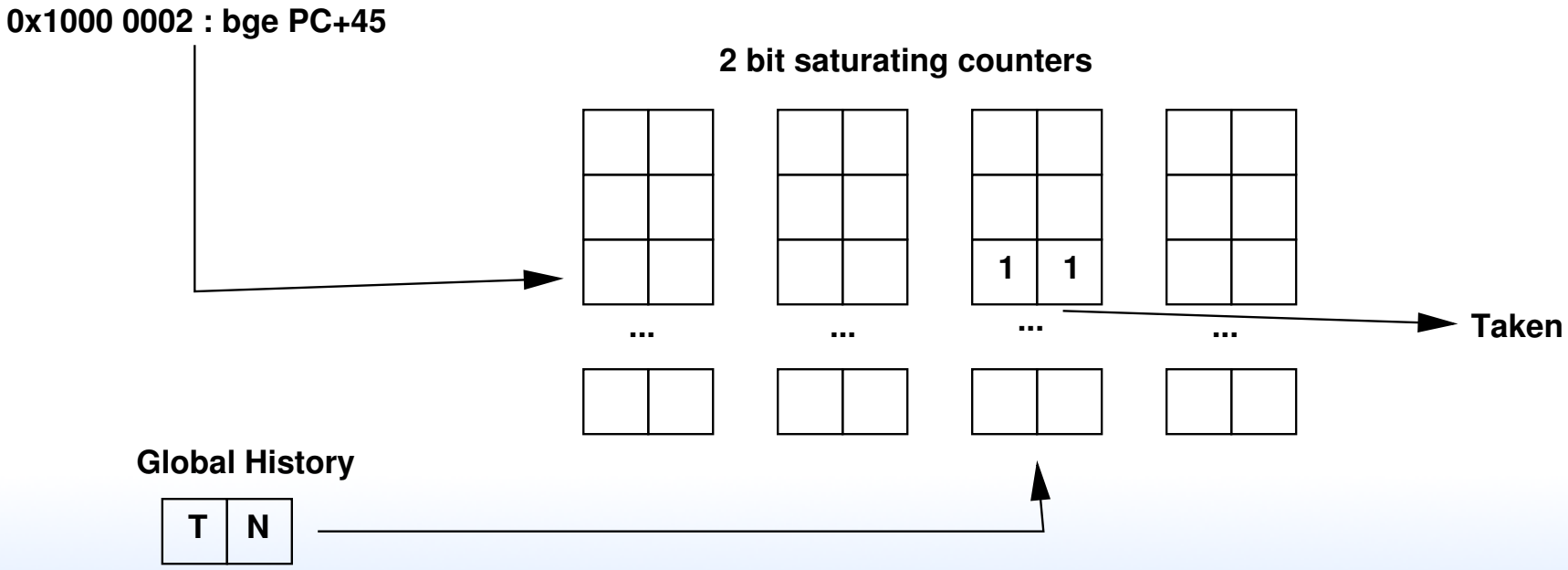
Local Predictor

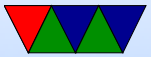




Correlating / Two Level Predictors

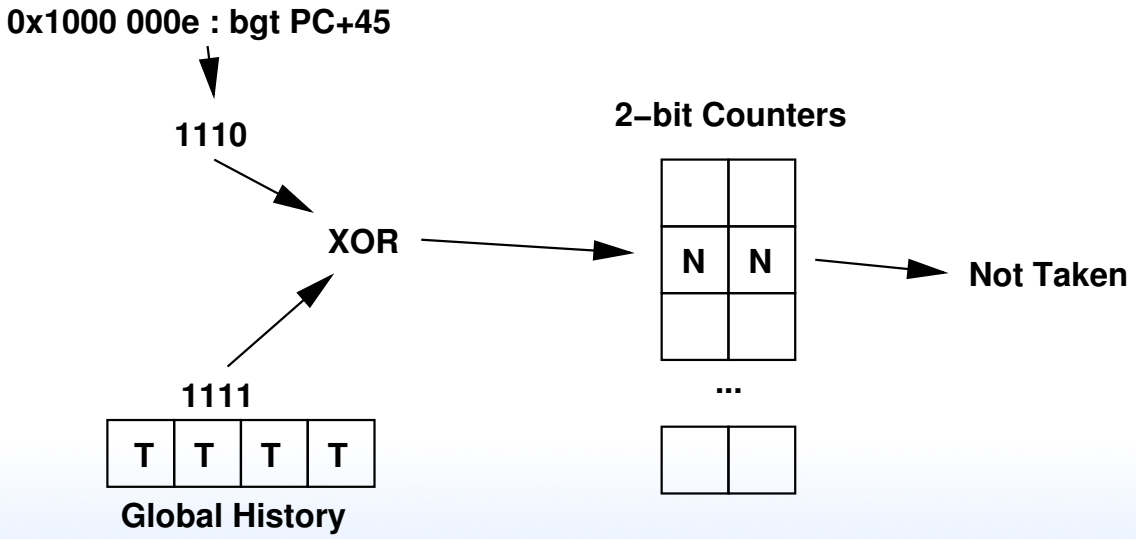
- Take history into account.
  Break branch prediction for a branch out into multiple locations based on history.

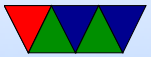




gshare

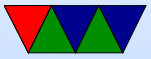
- Xors the global history with the address bits to get which line to use.
- Benefits of 2-level without the extra circuitry





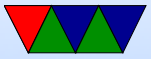
Tournament Predictors

- Which to use? Local or global?
- Have both. How to know which one to use? Predict it!
- 2-bit counter remembers which was best.



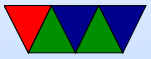
Multilevel

- What if it takes 20 cycles to do a prediction?
- Have two, fast and slow?
- Can start speculating with simple/fast predictor
- Have a bigger/slower one that takes a few more cycles that can over-ride the fast/early one if needed

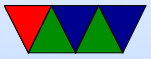


Perceptron

- Type of AI/Machine Learning/Neural network that gives yes/no answer
- There are actually Branch Prediction Competitions
- The winner the past few times has been a "Perceptron" predictor
- Samsung M1 processor at Hotchips'16 says it has neural-net branch predictor
- AMD Zen uses hashed perceptron
- Modern Intel re-written with Haswell, but to what?

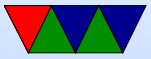


Rumored TAGE or ITTAGE (which use tags instead of pure hashes to avoid aliasing) and possibly perceptron?



Branch Predictor Implementations

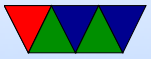
- Agner Fogg document, interesting. Found a bug in 2-bit saturating counters on P1
- Haswell re-written from scratch, unknown. Too many branches close together can cause problems



Cortex A9 Branch Predictor

From the Manual:

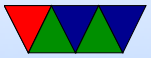
- two-level prediction mechanism, comprising: a two-way BTAC of 512 entries organized as two-way x 256 entries
- a Global History Buffer (GHB) with 4096 2-bit predictors
- a return stack with eight 32-bit entries.
- It is also capable of predicting state changes from ARM to Thumb, and from Thumb to ARM.



Core2 i7 Branch Predictor

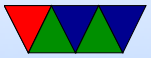
From H&P:

- Small 1 prediction/cycle predictor
- Larger (but slower) backup predictor that can correct the first if it's wrong
- Tournament predictor made up of
  - 2-bit
  - Global history
  - Loop-exit



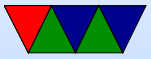
Branch Target Buffer (BTB)

- Instruction fetcher needs to know next instruction to fetch even before instructions decoded
- Predicts the actual destination of addresses.
- Indexed by whole PC. May be looking up before even know it is a branch instruction.
- Only need to store predicted-taken branches. (Why? Because not-taken fall through as per normal).



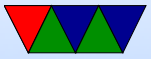
What Uses BTB?

- Originally indirect jumps (jmp reg, bl reg, ret)
- Some modern processors might use it for all branches
- The instruction fetcher might be independent of execution unit, so has to make decisions on what blocks of instructions to fetch next. Could actually be faster to predict than to wait/calculate the destination.



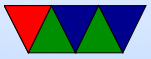
Return Address Stack

- Function calls can confuse BTB. Multiple locations branching to same spot. Which return address should be predicted?
- Keep a stack of return addresses for function calls
- Playing games with size optimization and fallthrough/tail optimization can confuse.



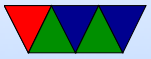
Comparing Predictors

- Branch miss rate not enough
- Usually the total number of bits needed is factored in
- May also need to keep track of logic needed if it is complex.
- Speed: what if it takes 20 cycles to do a prediction? Have two, fast and slow? Override fast result once get slow?



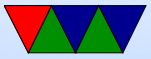
Security

- Speculative execution can lead to information leaks
- Best way to manipulate speculative execution paths is to manipulate BTB contents
- Recent processor changes have instructions for flushing or otherwise modifying BTB state

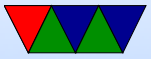


Spectre Security Vulnerability

- Problem when Branch Predictor state can leak information
- Side-channels – can find out what a processor is doing even if you shouldn't. Often by timing how long things take, as algorithms and processors can take varying amounts of time to do things.
- Speculative execution – what happens when you speculate wrong. Do the mis-speculated instructions leave any trace once they are thrown out? Ideally, no...



- For example, if down the wrong path a memory access happens, this could load a value from the cache, and even though the code is backed out, the values loaded into cache might not be backed out, and this can be determined by timing attacks "implicit caching"
- Example code that can reliably train a branch to miss or hit in the branch predictor.
- Use this to read out entire process's memory space. Is this an issue? What if browser/javascript could read out all memory of the browser (location and content of other tabs, passwords, etc).

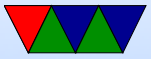


- Need code to flush cache. Cflush instruction, or clever access patterns
- Variants
  - Variant 1 – bounds checking

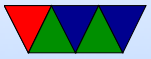
```
if (index<0x400) {
     a=array[index]; // index points to value you care a
     if (index&0x1) b=array[somevalue];
}
// check to see if somevalue is in cache after this
```

  - Variant 2 – branch steering

    Have an indirect branch, where target is from memory. Knock that memory location out of cache. It will then

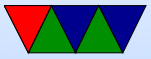


stall, causing the processor speculate for 100 cycles
  - Variant 3 – Meltdown (talk about later, Intel bug)
- Mitigations
  - Turn off high-accuracy counters in Javascript
  - Firmware patches to processor (but cause crashing?) IBRS (Indirect branch-related speculation)
  - retpoline (google), compiler change way indirect branches work



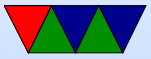
More on mitigations

- retpoline, redirect indirect branches to a fake call which uses return-address stack which is less vulnerable, and have a trap infinite loop after that will stop speculation
- firmware updates added ipbp barriers to flush brpred state



Adjusting Predictor on the Fly

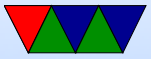
- "chicken bit"
- Some processors let you configure predictor at runtime
  MIPS R12000 let you, ARM possibly does.
- Why is this useful?
  In theory if you have a known workload you can pick the one that works best.
  Also if realtime you want something that is deterministic, like static prediction.
  Also Good for simulator validation



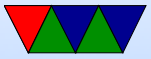
Example

Code in perf_event validation tests for generic events.

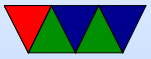
`http://web.eece.maine.edu/~vweaver/projects/perf_events/validation/`



Example Results

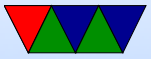


```
Part 1
Testing a loop with 1500000 branches (100 times):
On a simple loop like this, miss rate should be very small.
Adjusting domain to 0,0,0 for ARM
Average number of branch misses: 685

Part 2
Adjusting domain to 0,0,0 for ARM

Testing a function that branches based on a random number
   The loop has 7710798 branches.
   500000 are random branches; 250699 of those were taken
Adjusting domain to 0,0,0 for ARM

Out of 7710798 branches, 291081 were mispredicted
Assuming a good random number generator and no freaky luck
The mispredicts should be roughly between 125000 and 375000

Testing ‘‘branch-misses’’ generalized event...                PASSED
```



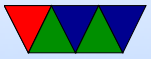
Some last branch predictor things

- Can turn off branch prediction on some machines. Most notably on the ARM1176 chip in a Raspberry Pi.
- Though trust-zone can make this difficult



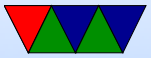
Branch Predictor Energy

- How much Energy does a branch predictor take?
- Often modeled as memory. Only in more complex setups does the logic take much space.
- How much die area (how many bits) (leakage)
- How often are they updated
- Do you need to store branch history on context switch?
- What happens if you turn off branch predictor? Will your code still run?



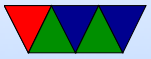
Branch Predictor Energy

- Parikh, Skadron, Zhang, Barcella, Stan
- 4 concerns:
  1. Accuracy. Not affect power, but performance
  2. Configuration (may affect power)
  3. Number of lookups
  4. Number of updates
- Tradeoff power vs time.
- brpred can be size of small cache, 10% of power
- Can use banking to mitigate



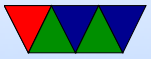
Branch Predictors

- can watch icache, not activate predictor if no branches
- Pipeline gating, keep track of each predicted branch confidence. If confidence hits certain threshold, stop speculating. Show this may or may not be good.
- Integer code, large predictors good
- FP, tight loops, predictors not as important.



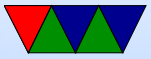
Branch Predictor Evaluation

- (Strasser, 1999). Simulation, small branch predictor can help energy.
- (Co, Weikle, Skadron) Formula for break even point. Leakage matters, what brpred hides is stall cycles.
- SEPAS: A Highly Accurate Energy-Efficient Branch Predictor (Baniasadi, Moshovos. ISLPED 2004). Once a branch prediction reaches steady state (unlikely to change) stop accessing/updating predictor, saving



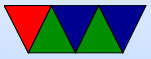
energy.

- Low Power/Area Branch Prediction Using Complementary Branch Predictors (Sendag, Yi, Chuang, Lija. IPDPS 2008)
  Complementary Branch Predictor to handle the tough cases.



Branch Predictors and Code Type

- What type of code is easiest to predict?
  Regular Loops
  Often found in large scientific "floating-point" workloads
- What is hard to predict? User input, random data/parsing with lots of conditional branches.
  "integer benchmarks", things like compilers, parsers, compression



Value Prediction

- Can we use this mechanism to help other performance issues?
  What about caches?

- Can we predict values loaded from memory?

- Load Value Prediction. You can, sometimes with reasonable success, but apparently not worth trouble as no vendors have ever implemented it.