ECE 571 – Advanced Microprocessor-Based Design Lecture 11

Vince Weaver

http://www.eece.maine.edu/∼vweaver

vincent.weaver@maine.edu

19 February 2013

Metrics

- Cache miss rate misses/total. For various levels, including TLB
- IPC and CPU. IPC (higher better). CPI (lower better) dependent on other parts of chip. Instruction-Level Parallelism. Higher than 1 IPC.
- Branch miss rate

Maximum – IPC

- In theory, on single-issue pipelined processor maximum IPC is 1.0
- With multi-issue this can be increased.
- The Pandaboard can in theory issue 3 (maybe 4?) instructions per cycle (although the decoder can only decode 2?)
  So maximum IPC might be 3.0?
- There are limits to which instructions can issue (ALU,

FP, etc) and any branch or cache miss will hurt IPC, as well as high-latency instructions

IPC for Cache Examples

Cache example from last time:

- First naive implementation 0.13 IPC
- Second, swapped, 0.34 IPC
- ATLAS: 0.87 IPC

Branch Prediction – Background

- With a pipelined processor, you may have to stall waiting until branch outcome known before you can correctly fetch the next instruction

- Conditional branches are common. On average every 5th instruction?

- One solution is speculative execution.
  Guess which way branch goes.
  If wrong, in-flight thrown out, have to replay.

- How can get around this?
  Try to speculate multiple paths? (rapidly increases).
- Good branch predictors have a 95% or higher hit rate
- **Speculation wastes power**

Static Prediction of Conditional Branches

- Backward taken
- Forward not taken
- Can be used as fallback when there's not more info

Common Access Patterns – For Loop

```
for(i=0;i<100;i++) SOMETHING;

    mov r1,#100
label:
    SOMETHING
    add r1,r1,#-1
    bne label
```

Common Access Patterns – While Loop

Optimizing compiler may translate this to a for loop (why?)

```
x=0; while(x<100) { SOMETHING; x++;}

    mov r1,#0
label:
    cmp r1,#100
    bge done
    SOMETHING
    bne label

    add r1,r1,#1
    b label
done:
```

Common Access Patterns – If/Then

ARM can use predication to avoid this.

```
if (x) { foo } else { bar}

  cmp r1,#0
  bne else
then:
  foo
  b done
else:
    bar
done:
```

Branch Prediction Hints

- likely() (maps to `__builtin_expect()`)
- unlikely()
- on some processors, (p4) hint for static
- others, just move unlikely blocks out of way for better L1I$ performance

Dynamic Branch Prediction

Branch History Table

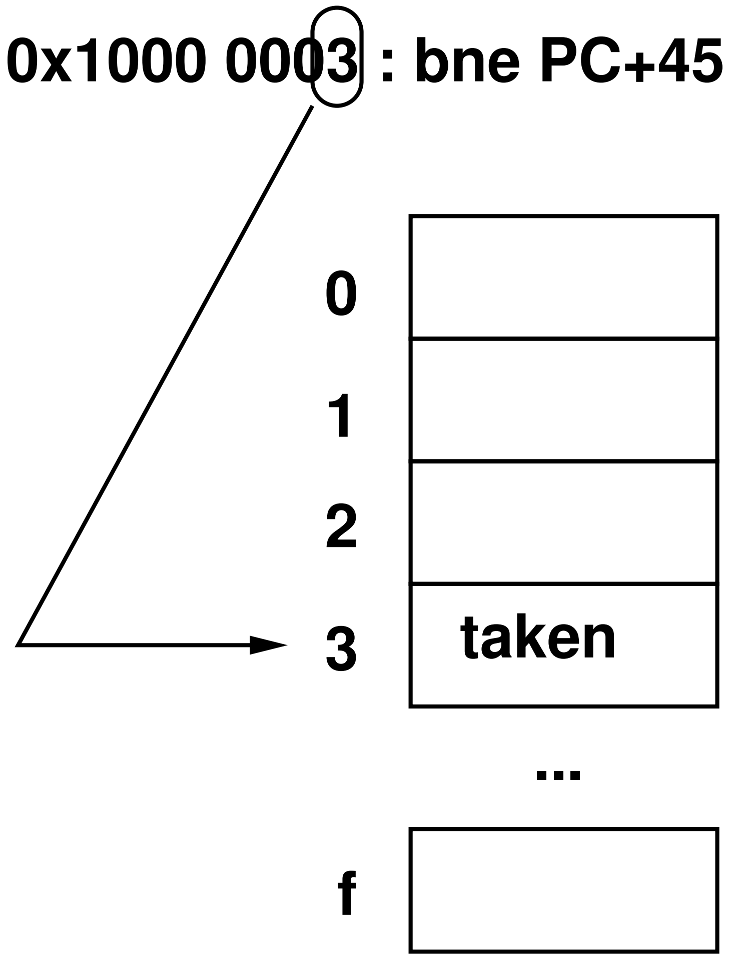
- table, likely indexed by lower bits of instruction can have more advanced indexing to avoid aliasing no-tag bits, unlike caches aliasing does not affect correct program execution
- one-bit indicating what the branch did last time
- update when a branch miss happens
- two misses for each time through loop. Wrong at exit of loop, then wrong again when restarts.

Aliasing

- Is it bad? Good?

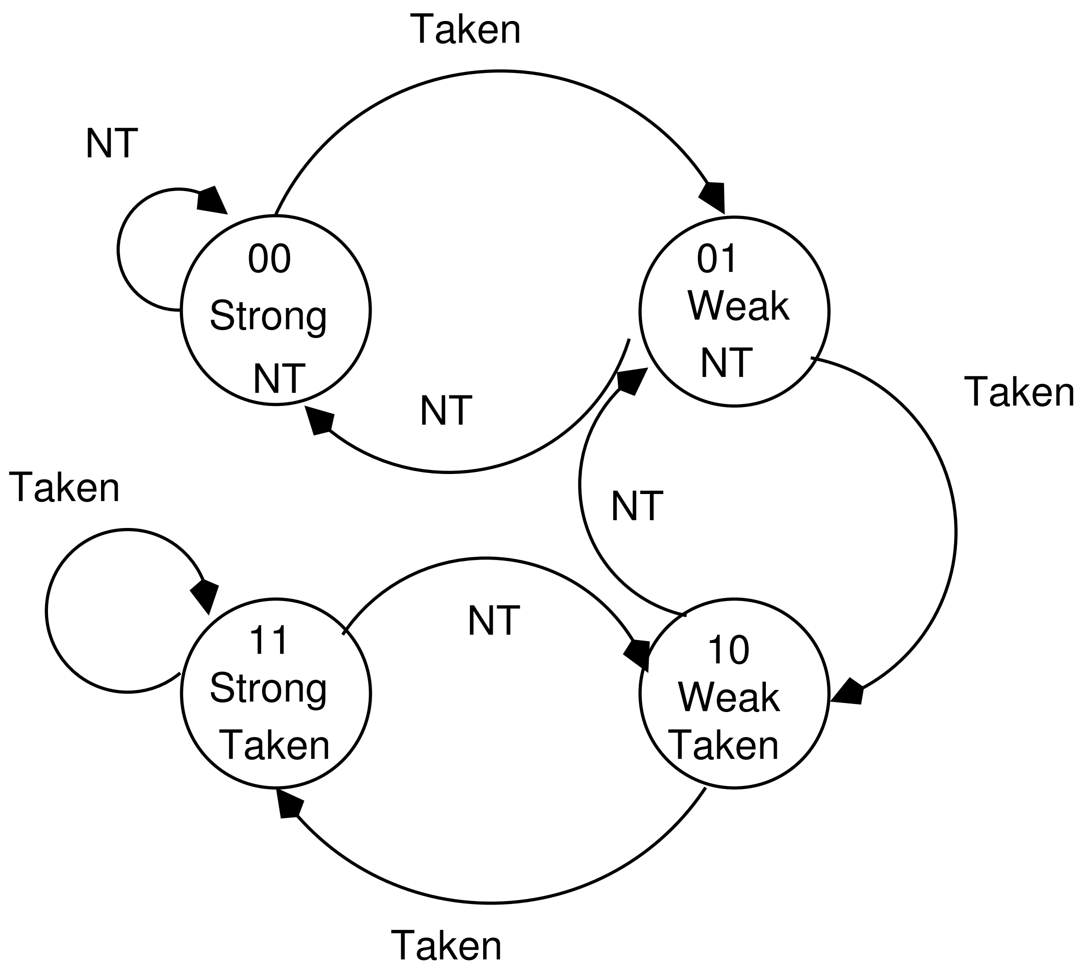
- Does the O/S need to save on context switch?

- Do you need to save if entering low-power state?


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

Two-bit saturating counter

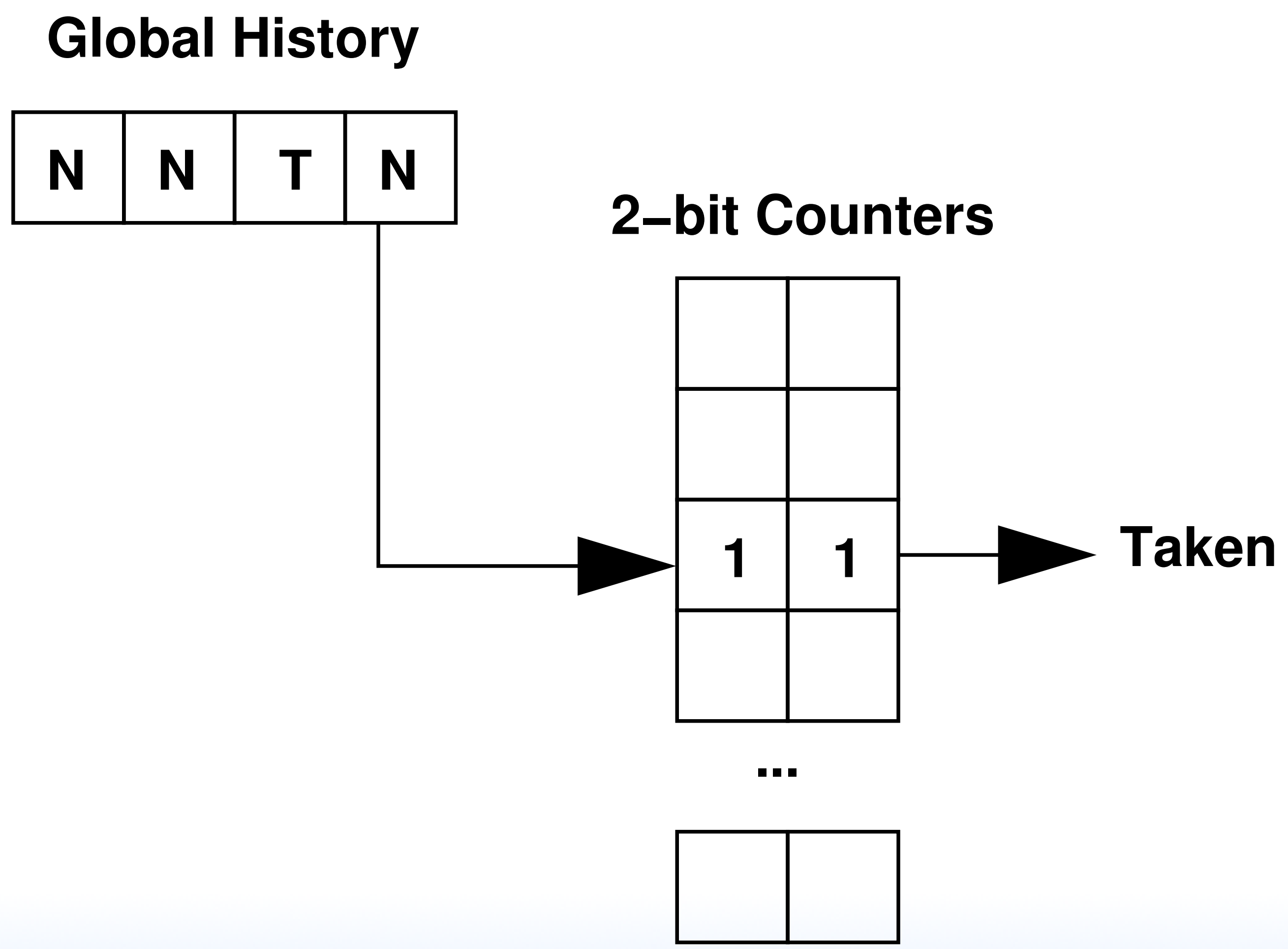
- Use saturating 2-bit counter
- If 3/2, predict taken, if 1,0 not-taken. Takes two misses or hits to switch from one extreme to the next, letting loops take only one mispredict.
- Needs to be updated on every branch, not just for a mispredict


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

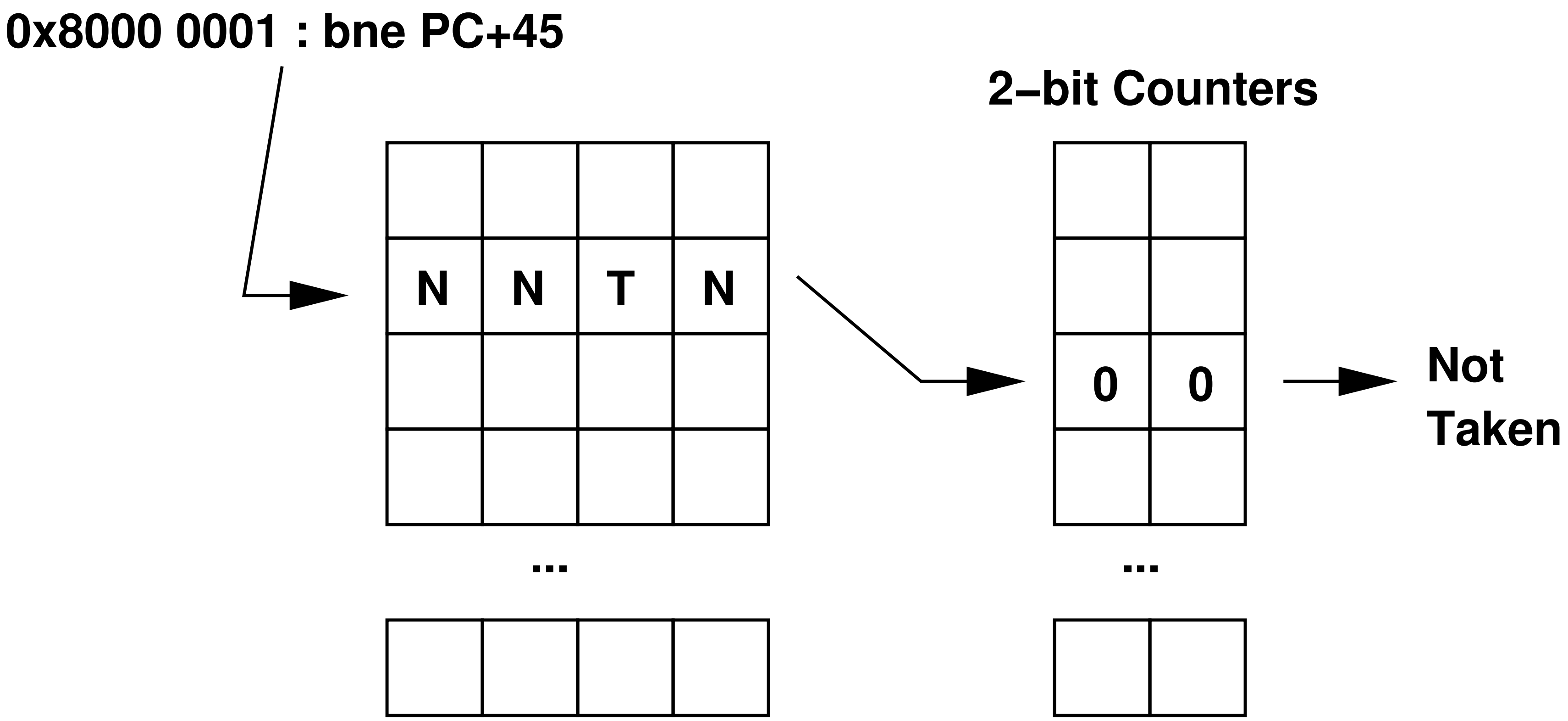
Local vs Global History

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

Global Predictor

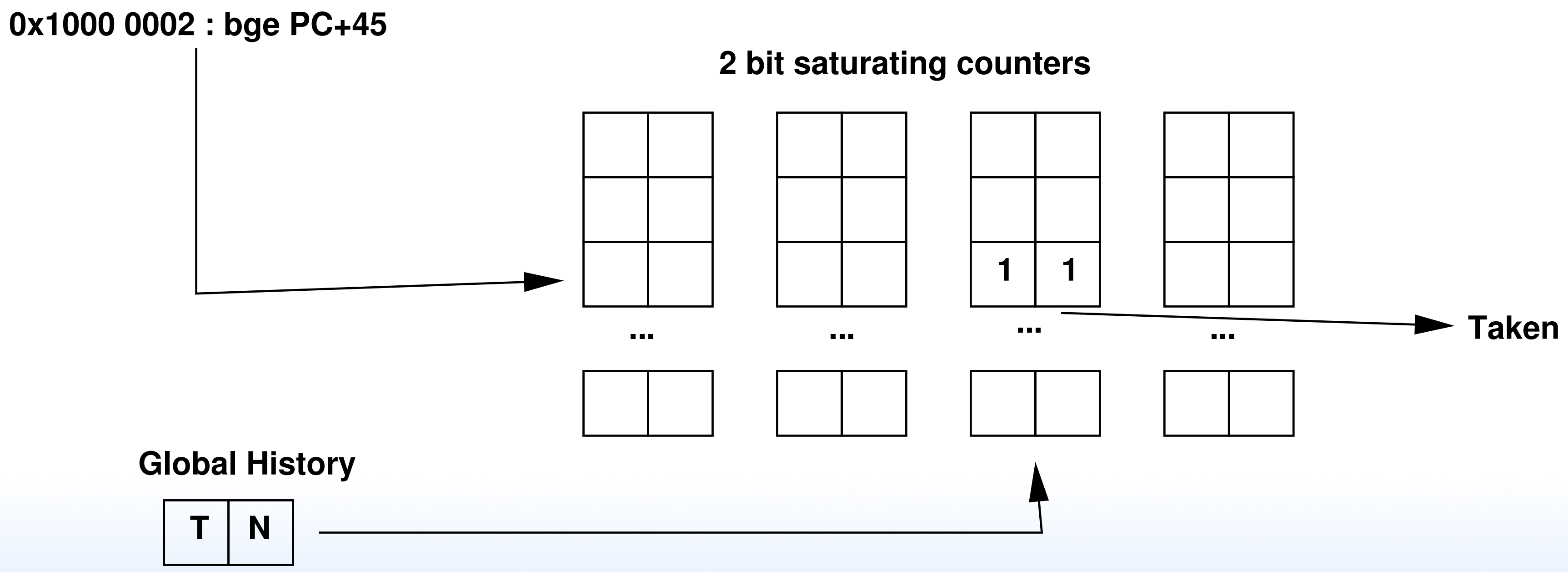


Local Predictor



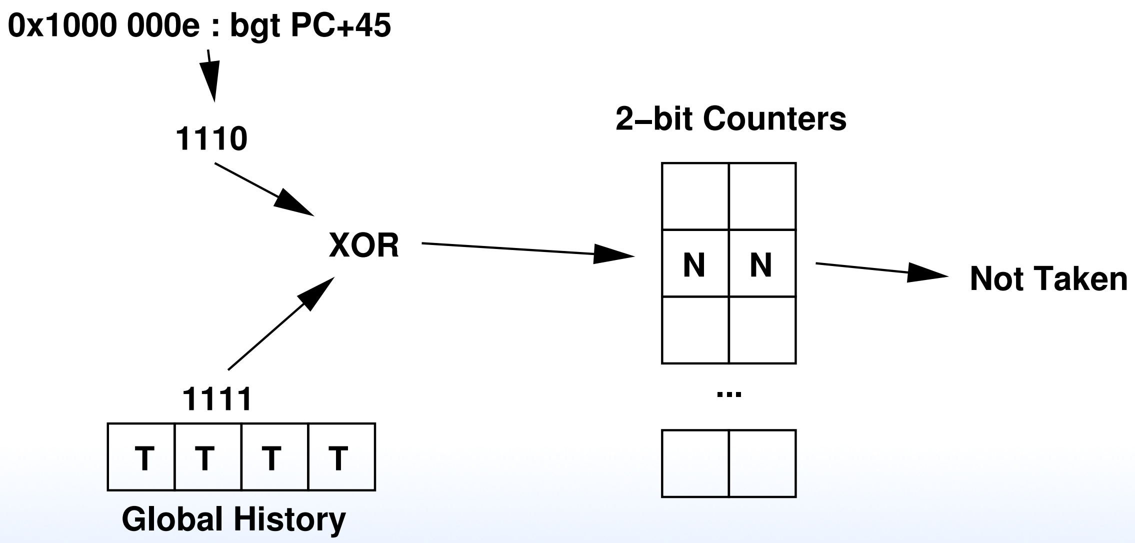
Correlating / Two Level Predictors

- 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.

Perceptron

- There are actually Branch Prediction Competitions
- The winner the past few times has been a “Perceptron” predictor
- Neural Networks

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.

Branch Target Buffer

- 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).

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.

Adjusting Predictor on the Fly

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

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.

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
```

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.