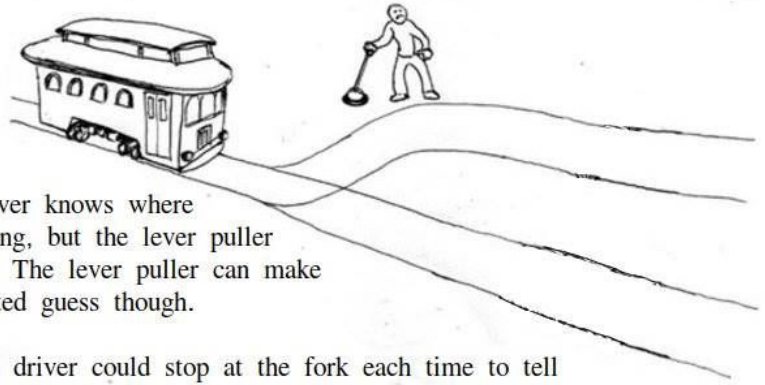
Branch prediction

Branch Prediction



Train driver knows where he is going, but the lever puller does not. The lever puller can make an educated guess though.

The train driver could stop at the fork each time to tell the lever puller which way he is going.

OR

The lever puller makes an educated guess. If he is wrong, the train driver stops, reverses, and then continues on the right track.

Backwards and forwards branches

```
while(guard) {
    ...
}
```

```
loop: bne s0 s1 end
...
j loop
end: ...
```

Backwards branches are *very frequently* taken (~90% by some counts)
Forwards branches are a coin flip without any other information

```
j start
loop: ...
...
start: beq s0 s1 loop
```

Software should be optimized such that the sequential code path is the most common path, with less-frequently taken code paths placed out of line. Software should also assume that backward branches will be predicted taken and forward branches as not taken, at least the first time they are encountered. Dynamic predictors should quickly learn any predictable branch behavior.

Unlike some other architectures, the RISC-V jump (JAL with *rd*=x0) instruction should always be used for unconditional branches instead of a conditional branch instruction with an always-true condition. RISC-V jumps are also PC-relative and support a much wider offset range than branches, and will not pressure conditional branch prediction tables.

Double-loop code

```
addi t0, x0, 0      // t0 = 0
addi t1, x0, 400    // t1 = 400
l1: addi t2, x0, 3 // t2 = 3
l2: addi t0, t0, 1 // t0++
addi t2, t2, -1     // t2--
bne t2, x0, l2      // loop while t2 > 0
slli t0, t0, 1      // t0 <<= 1
addi t1, t1, -1     // t1--
bne t1, x0, l1      // loop while t1 > 0
```

Dynamic branch prediction: CPU can choose to predict (keep executing as if) branch is taken OR branch is not taken

Flushes (pipeline or ROB) if it's wrong

How? Keep a Branch Prediction Buffer (Branch History Table) that maps branch instr addresses to predictions

1-bit BPB entry

Works great for outer loop (one misprediction and then 400 correct predictions)

Works less great for inner loop:

2-bit BPB entry

2 bits can keep track of 4 states: strong taken, weak taken, weak not taken, strong not taken

Keeps some of history (means branch prediction needs to be wrong twice instead of once before changing) – works better for the inner loop!

Two-level predictors

Correlated branches

```
if (x == 2) // branch A
    x = 0;
if (y == 2) // branch B
    y = 0;
if (x != y) // branch C
    …
```

A taken and B taken implies C not taken!
→ branch outcomes are often **not independent** of each other

What is the simplest possible way to try to predict correlated branches?

Global predictors

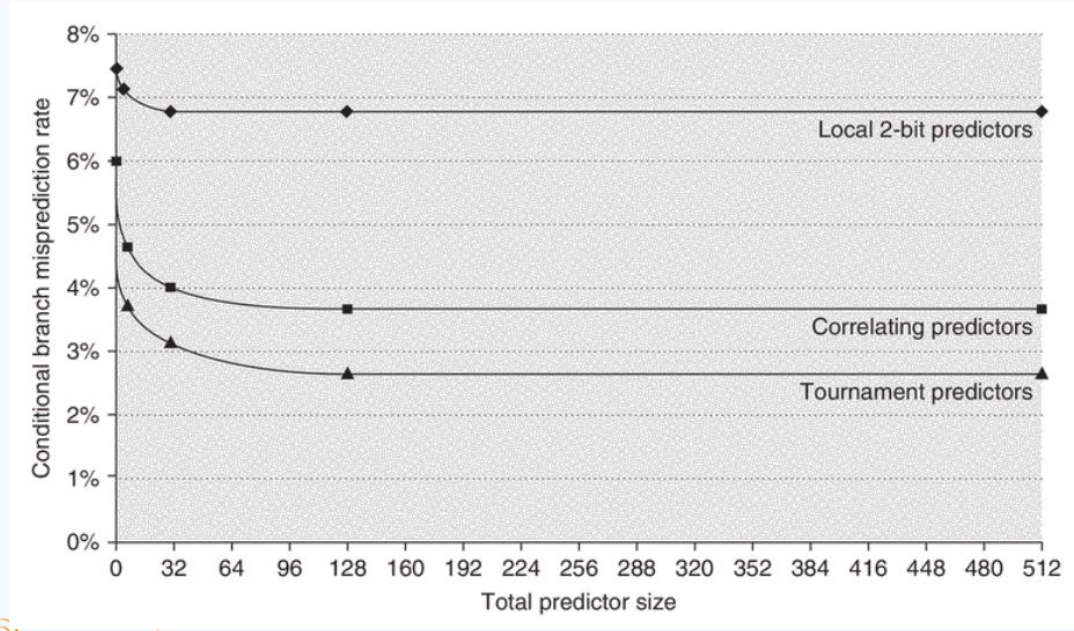
Correlating predictors

Tournament branch predictors

Run local predictors and global predictor

Keep track of which one is doing better (using eg 2-bit predictor!) and use that one for each branch

H&P fig. 3.4



Even if we had perfect branch prediction, what would we need in order to make sure that speculative execution is fast?

Branch target buffers

Cache for computing branch or jump target address (new PC)

Potentially faster to fetch next instruction

- Common in modern systems; unlike branch delay slots

Multiple ways to set this up (see Agner document)

- Multiple levels
- Different behavior for different types of branches/jumps

What is the theoretical best CPI for our OOO CPU (with or without speculation)?

Multiple-issue

Allows for multiple instructions to be issued at the same time

Dynamically (by processor): superscalar

- Multiple variations: in-order, OOO, OOO + speculative

Statically (by compiler): Very Long Instruction Word (VLIW), EPIC

- Next week

Potentially how many instructions can we issue at once if we have the following FUs? With what caveats?

- 1 load
- 1 store
- 2 integer ALUs
- 1 FP add/sub
- 1 FP mul/div

Issuing two instrs at once

lw t0 8(s0)

add t2, t0, t1

- What do the reservation stations/ROB look like?
- What does the hardware need to check in a *single cycle*?