ECE 571 – Advanced Microprocessor-Based Design Lecture 10

Vince Weaver

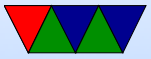
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

25 September 2024

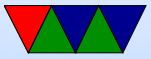
Announcements

- Don't forget HW#3
- RAPL also used for "TurboBoost"
- Reading: Hennessey and Patterson 5th edition (available online from UMaine Library) Chapter 3.3 Branch Prediction



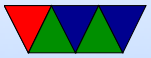
The Branch Problem

- 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 [cite?]
- What can you do to speed things up?



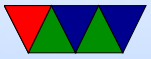
Branch Prediction

- One solution is speculative execution.
  Guess which way branch goes.
- Good branch predictors have a 95% or higher hit rate
- Downsides?
  If wrong, in-flight thrown out, have to replay.
- **Speculation wastes power**
- Also, it turns out, there are security issues



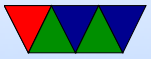
Speculative Execution Aside

- What do you do if guess wrong?
- What if an exception happens on the wrong path (pointer dereference)
- What is you hit another branch on the wrong path?
- What if you load/store memory on the wrong path?
- What if the wrong path leaks state?



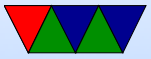
Branch Predictor Implementations

How would you implement a predictor?



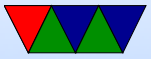
Static Prediction of Conditional Branches

- Backward taken
- Forward not taken
- Can be used as fallback when nothing else is available



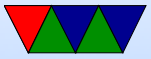
Common Access Patterns – For Loop

```
// loop with backward branch

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

    mov r1,#0
    b   label2
label:
    SOMETHING
    add r1,r1,#1
label2:
    cmp r1,#100
    bne label
```



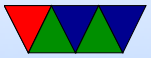
`for` Branch Behavior

- Cond branch executed 101 times (because checks before loop entry)
- No branch predictor – 101 stalls
- Other way to avoid problem (branch delay slot on MIPS)
- Static prediction BTFN – 100 times predicted right, 1 time wrong (exit)
- 99% correct predict rate (for this particular)
  Depends on iterations, 50% to 99+%



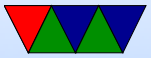
Common Access Patterns – While Loop

```
// loop with forward branch

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

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

     SOMETHING

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



`while` Branch Behavior

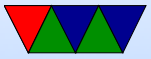
- Cond branch executed 101 times (because checks before loop entry)
- No branch predictor – 101 stalls (unless branch delay slot)
- Static BTFN prediction – 100 times predicted right, 1 time wrong (exit)
- 99% correct predict rate



Common Access Patterns – Do/While Loop

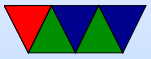
```
x=0; do { SOMETHING; x++;} while(x<100);
```

```
    mov r1,#0
label:
    SOMETHING

    add r1,r1,#1
    cmp r1,#100
    blt label

done:
```



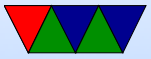
`while` Do/While Behavior

- No branch predictor – 100 stalls (unless branch delay slot)
- Static BTFN prediction – 99 times predicted right, 1 time wrong (exit)
- 99% correct predict rate



Notes

- Optimizing compiler will optimize all above to same for loop (tried it). Why?
- Because loop unrolling becomes possible?

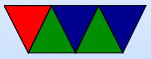


Common Access Patterns – If/Then

A lot harder to predict than loops

```
if (x) { FOO } else { BAR}
```

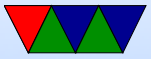
```
   cmp r1,#0
   beq else
then:
   FOO
   b done
else:
   BAR
done:
```



Avoiding Branches in If/Then

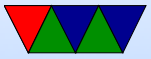
- You can try to avoid branches with fancy coding
- Some chips have conditional move instructions (x86)
- ARM32 has conditional/prefixed execution

```
ARM:
    cmp r1,#0
    FOOeq
    BARne
```

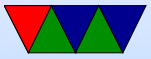


Common Access Patterns – If/Then Behavior

- If x is true, static = 100%, if x is false, 0%
- Assuming completely random, average 50% miss rate
- ARM can use conditional execution/predication to avoid this in simple scenarios



How can we Improve Things?



Branch Prediction Hints

- Give compiler (or assembler) 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
- Linux did this – but turns out people can be bad at hinting