Dynamic branch prediction

Data structures and approaches

Introduction

1-bit

2-bit

Conclusion

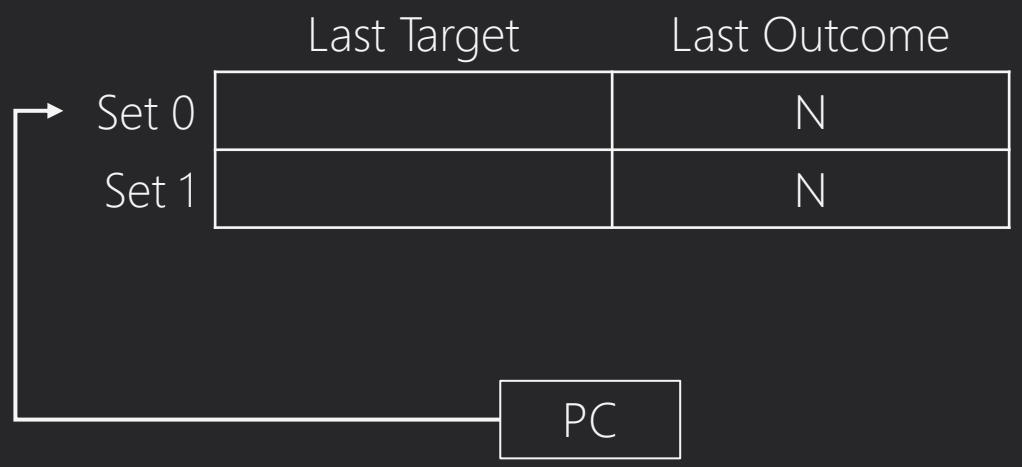
A branch example (n = 5)

```
sum = 0;

for(i = 0; i < n; i++)
     sum = sum + i;

return sum
```

```
add $r1, $zero, $zero
add $r2, $zero, $zero
addi $r3, $zero, 5
loop:
     beq $r2, $r3, return
     add $r1, $r1, $r2
     addi $r2, $r2, 1
     j loop

return:
     jr $ra
```

An example data structure

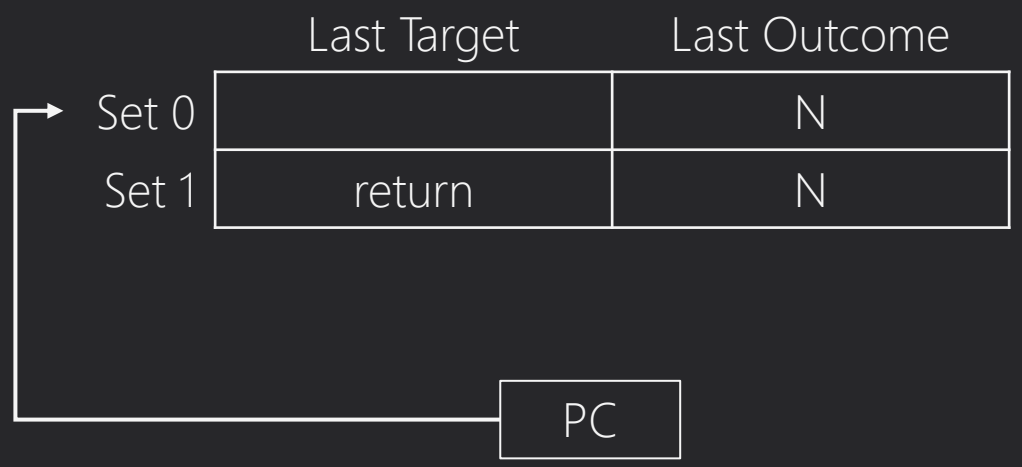
Data structure



Updates…

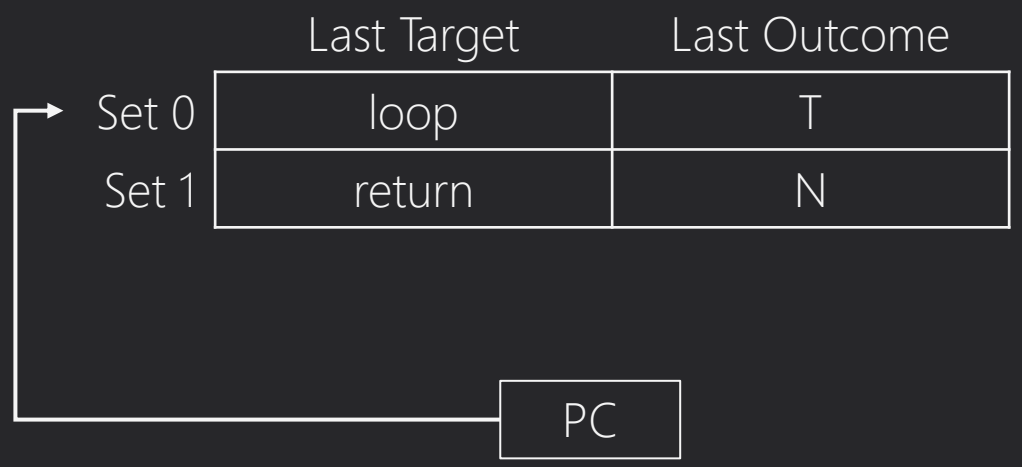
Instruction	… to target	… to direction
beq	After instruction fetch	After comparison
j	After instruction fetch	Always taken
jr	After register fetch	Always taken

An example execution (beq 1)



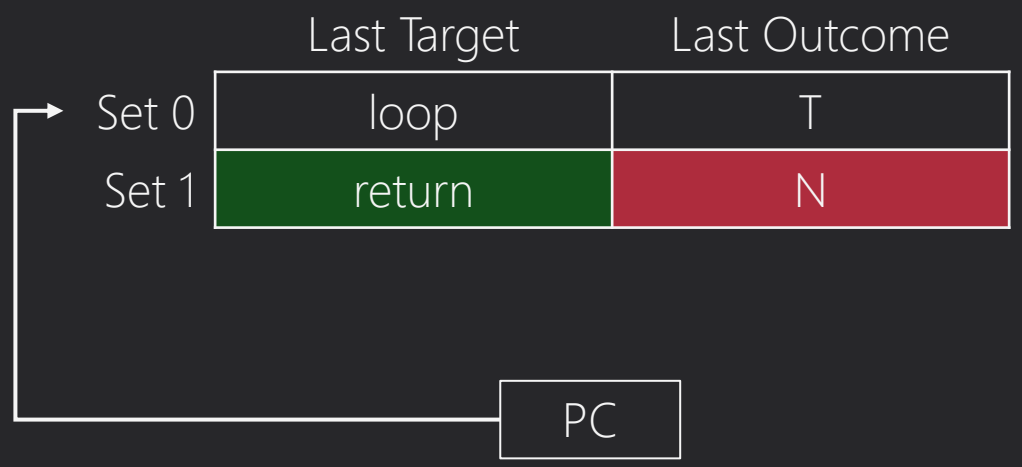

```
add $r1, $zero, $zero
add $r2, $zero, $zero
addi $r3, $zero, 5
loop:
     beq $r2, $r3, return
     add $r1, $r1, $r2
     addi $r2, $r2, 1
     j loop

return:
     jr $ra
```

An example execution (j 1)



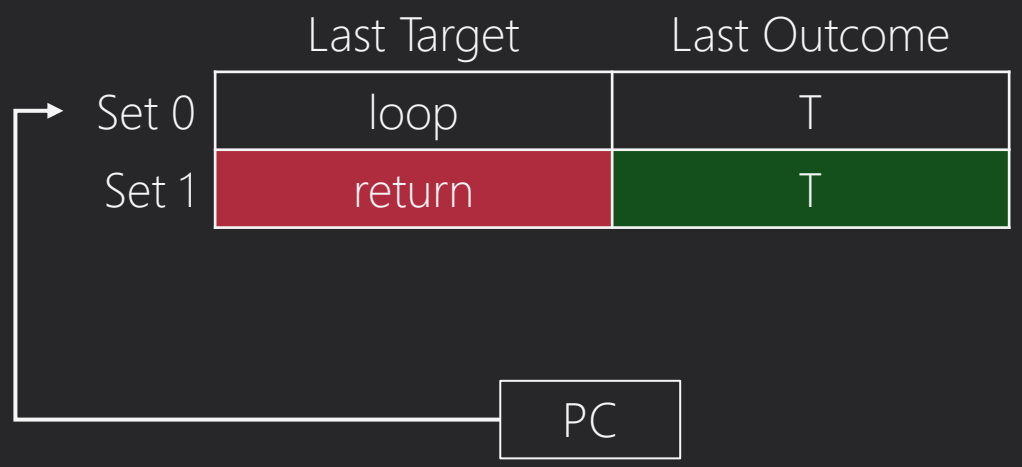

```
add $r1, $zero, $zero
add $r2, $zero, $zero
addi $r3, $zero, 5
loop:
     beq $r2, $r3, return
     add $r1, $r1, $r2
     addi $r2, $r2, 1
     j loop

return:
     jr $ra
```

An example execution (before beq 6)




```
add $r1, $zero, $zero
add $r2, $zero, $zero
addi $r3, $zero, 5
loop:
     beq $r2, $r3, return
     add $r1, $r1, $r2
     addi $r2, $r2, 1
     j loop

return:
     jr $ra
```

An example execution (jr 1)




```
add $r1, $zero, $zero
add $r2, $zero, $zero
addi $r3, $zero, 5
loop:
     beq $r2, $r3, return
     add $r1, $r1, $r2
     addi $r2, $r2, 1
     j loop

return:
     jr $ra
```

A branch direction timeline for beq

Value of i	State / Prediction	Outcome
0	N	N
1	N	N
2	N	N
3	N	N
4	N	N
5	N	T (misprediction)

$$Accuracy = \frac{5}{6} = 83.3\%$$

The return address stack (RAS)

- Return statements are easy to predict
  - Keep a separate data structure (the RAS) from BTB
- A function call (e.g., jal) pushes PC + 4 onto the RAS
  - Predict return (e.g., jr $ra) address is at the top of the RAS
- Add metadata to instruction cache (pre-decode bits)
  - 1-bit of meta-data: is this a return instruction?
  - 1-bit of meta-data: is this a conditional branch?

The nested loop

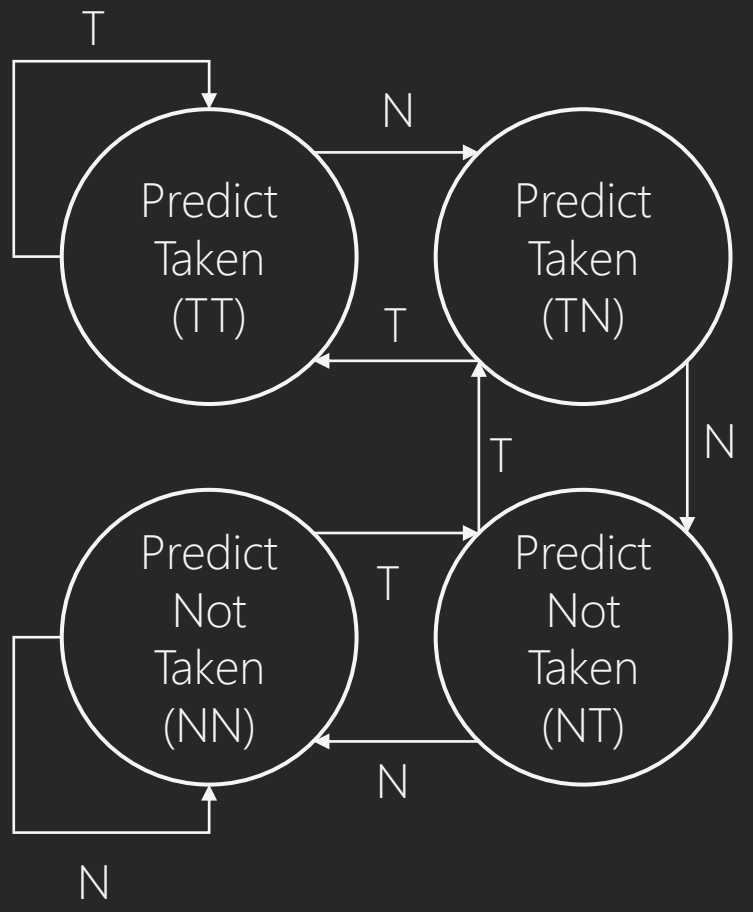
```
for(i =0; i < n; i++)
      for(j = 0; j < m; j++)
            // loop body
```

- Suppose n is very large (100)
- Suppose m is small (3)
- Let Taken (T) mean “enter loop body”

Value of i	Value of j	State / Prediction	Outcome
0	0	N	T
0	1	T	T
0	2	T	T
0	3	T	N
1	0	N	T
1	1	T	T
1	2	T	T
1	3	T	N
2	0	N	T
2	1	T	T
2	2	T	T
2	3	T	N

The two-bit saturating counter

- Store two bits rather than 1
- Two bits differentiate between four states
  - TT: Strongly taken
  - TN: Weakly taken
  - NT: Weakly not taken
  - NN: Strongly not taken
- Starting state: NN



The nested loop (2-bit saturating counters)

```
for(i =0; i < n; i++)
      for(j = 0; j < m; j++)
            // loop body
```

- Suppose n is very large (100)
- Suppose m is small (3)
- Let Taken (T) mean “enter loop body”

Value of i	Value of j	State / Prediction	Outcome
0	0	NN	T
0	1	NT	T
0	2	TN	T
0	3	TT	N
1	0	TN	T
1	1	TT	T
1	2	TT	T
1	3	TT	N
2	0	TN	T
2	1	TT	T
2	2	TT	T
2	3	TT	N

Conclusion

Branch targets

- The branch target buffer (BTB) acts as a cache
- It is useful to separate different types of branch instructions
  - Return Address Stack (RAS)
  - Pre-decode bits

Branch direction

- 1-bit counters are limited
- 2-bit counters are good predictors of nested loops
- What about other patterns?