

static branch prediction

forward (target > PC) not taken; backward taken

intuition: loops:

```
LOOP: ...  
      ...  
      je LOOP
```

```
LOOP: ...  
      jne SKIP_LOOP  
      ...  
      jmp LOOP  
SKIP_LOOP:
```

predict: repeat last

PC of branch

0x40042A

hash function

index *prediction/
last result?*

0 taken (1)

1 not taken (0)

2 taken (1)

3 taken (1)

...

14 not taken (0)

15 taken (1)

predict: repeat last

PC of branch

0x40042A

hash function

index *prediction/
last result?*

0

taken (1)

1

not taken (0)

2

taken (1)

3

taken (1)

...

...

14

not taken (0)

typical choice: some bits of branch address
for our example: will use bits 4-7

predict: repeat last

PC of branch

0x40042A

hash function

<i>index</i>	<i>prediction/ last result?</i>
0	taken (1)
1	not taken (0)
2	taken (1)
3	taken (1)
...	...
14	not taken (0)
15	taken (1)

predict: repeat last

PC of branch

0x40042A

hash function

<i>index</i>	<i>prediction/ last result?</i>
0	taken (1)
1	not taken (0)
2	taken (1)
3	taken (1)
...	...
14	not taken (0)
15	taken (1)

prediction
to fetch stage

predict: repeat last

PC of branch

0x40042A

hash function

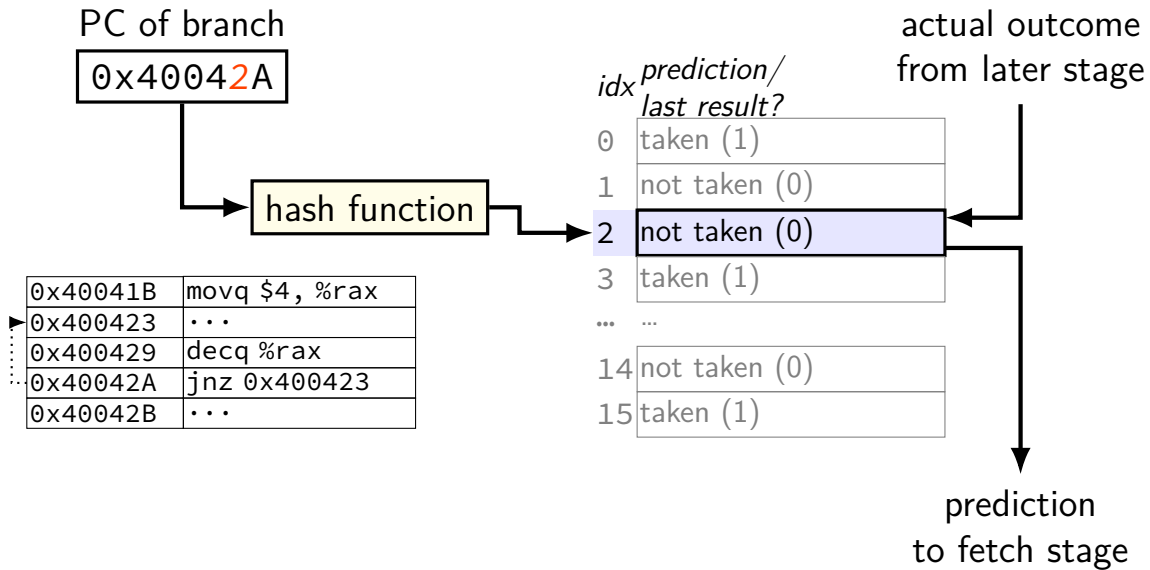
*index prediction/
last result?*

0	taken (1)
1	not taken (0)
2	taken (1)
3	taken (1)
...	...
14	not taken (0)
15	taken (1)

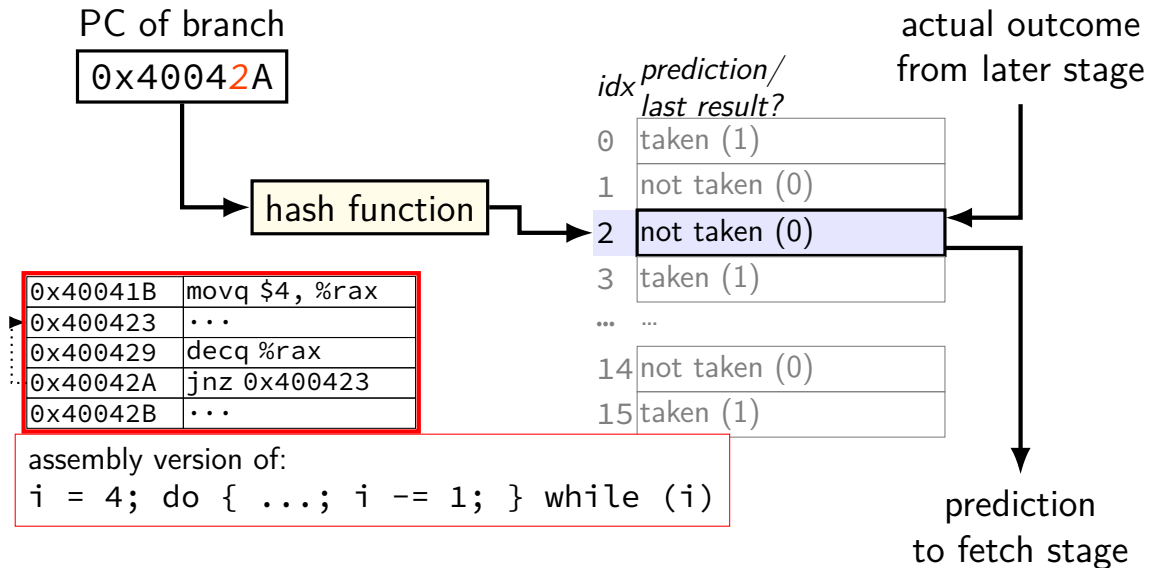
actual outcome
(from later stage)

prediction
to fetch stage

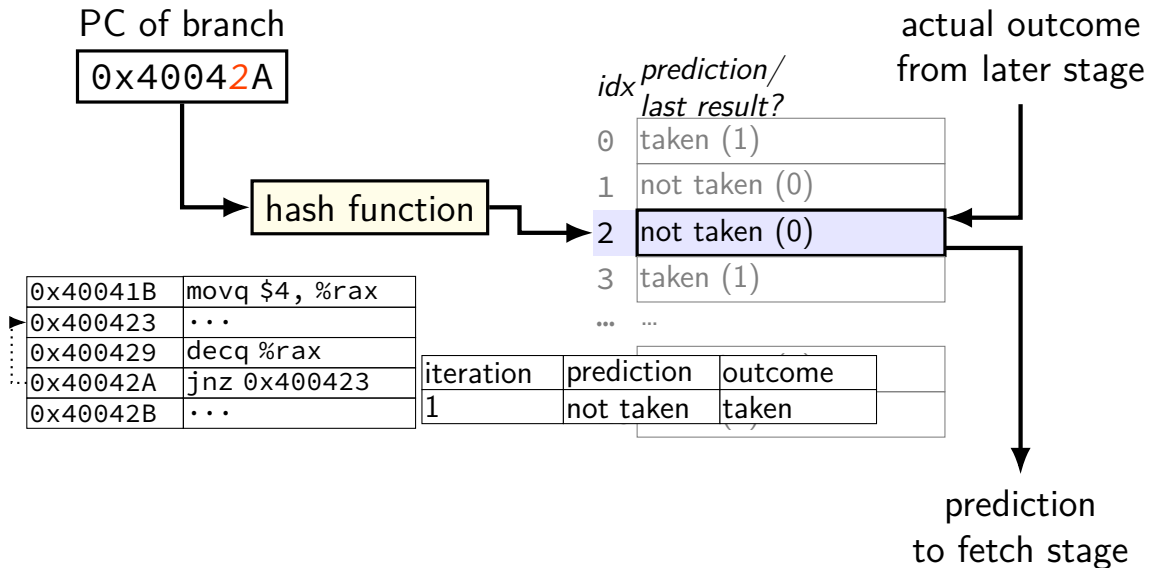
example



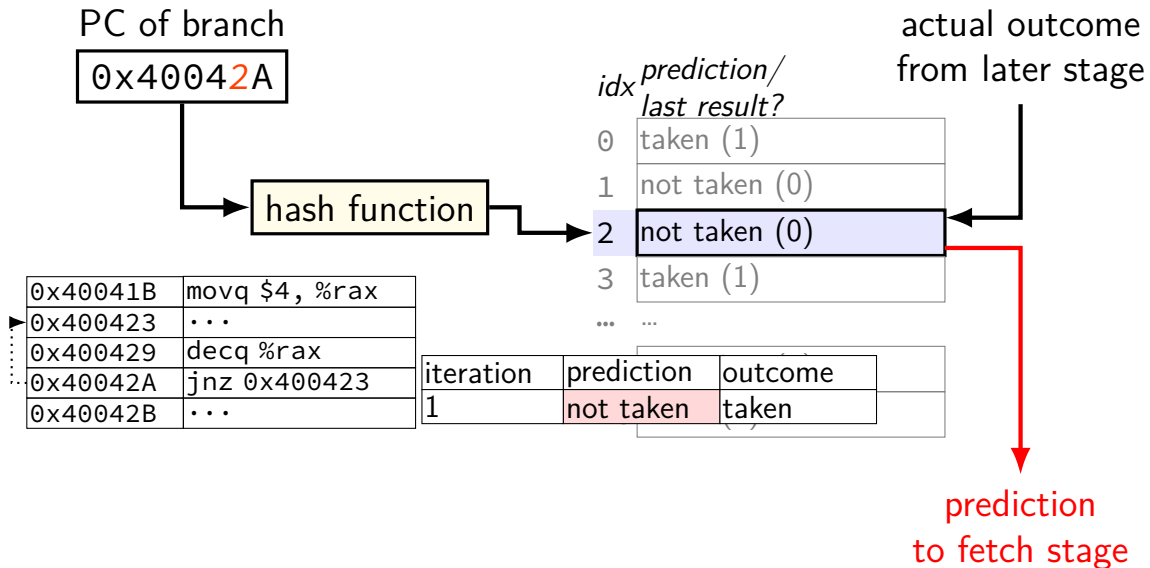
example



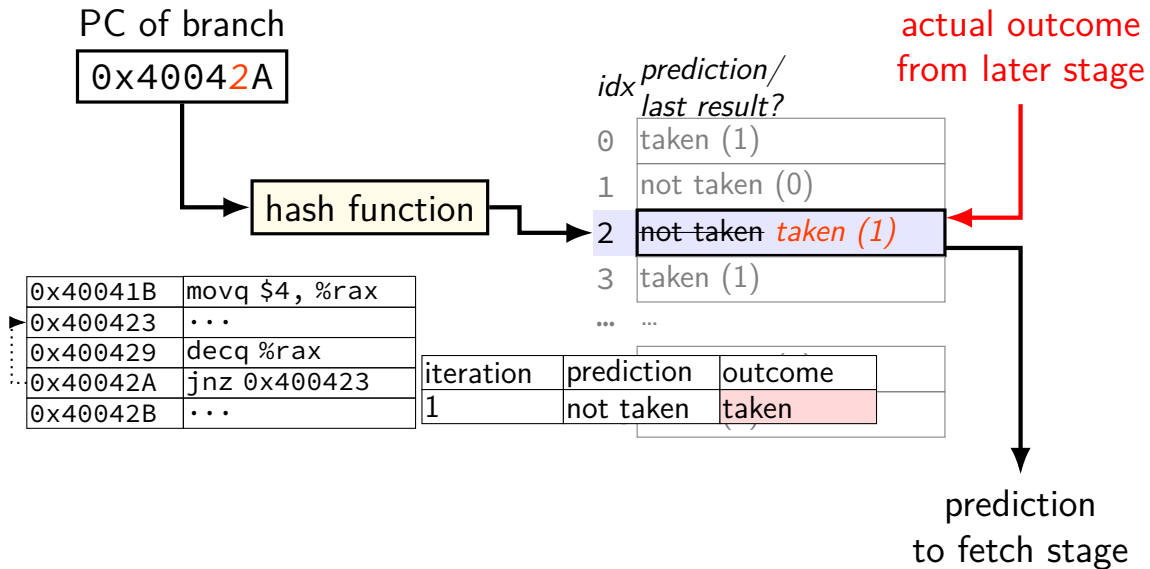
example



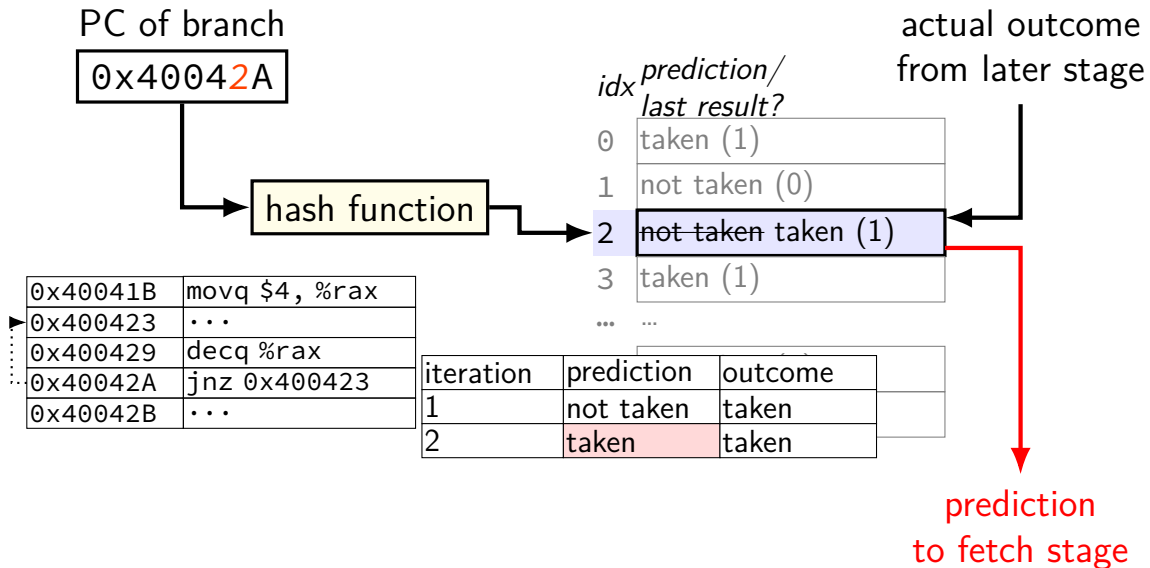
example



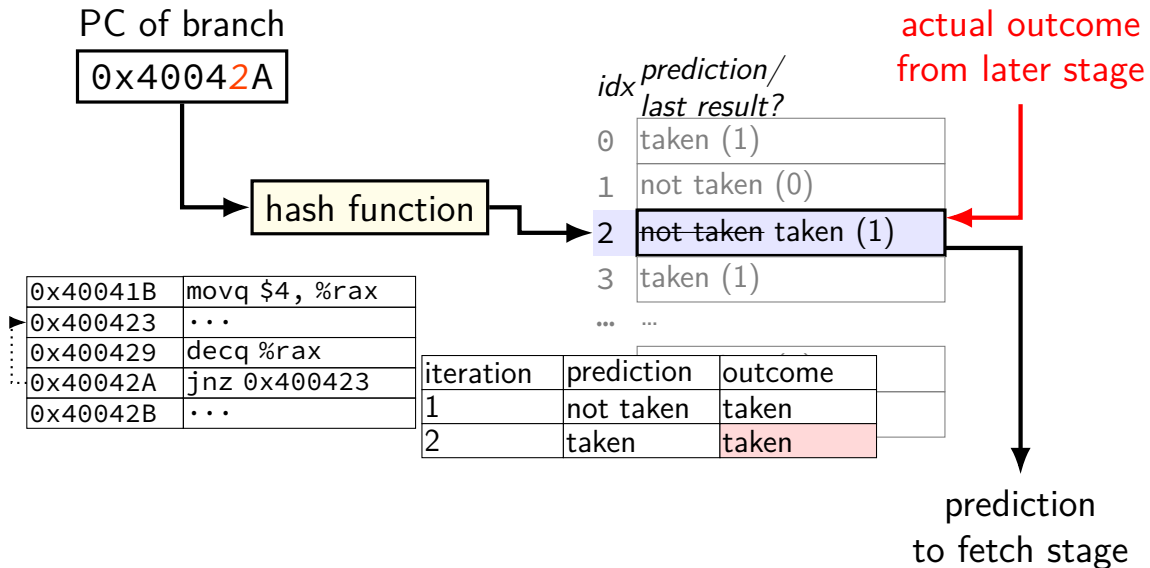
example



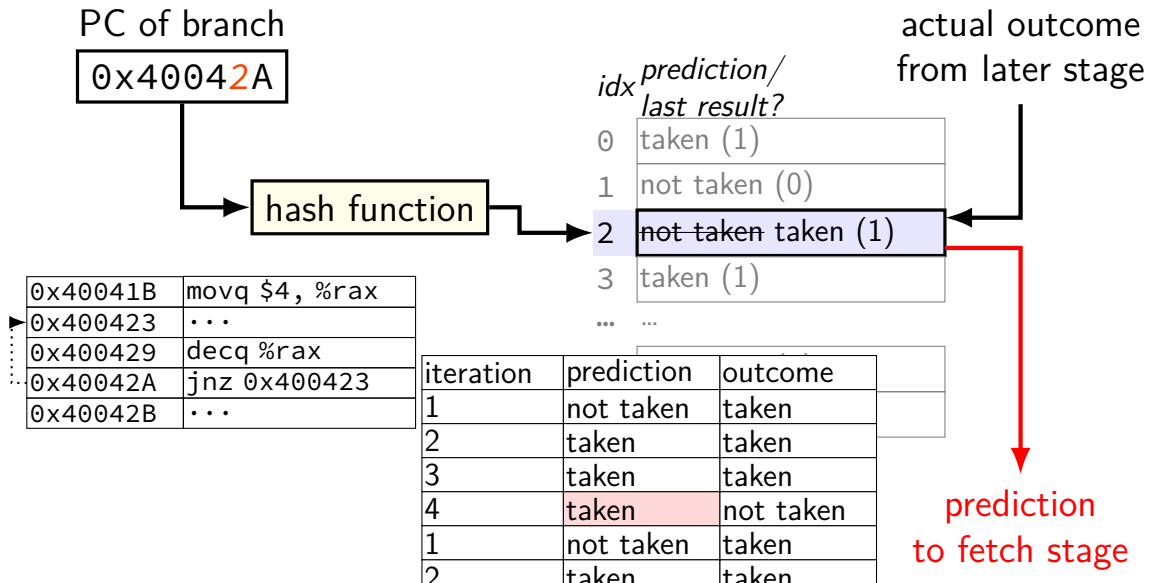
example



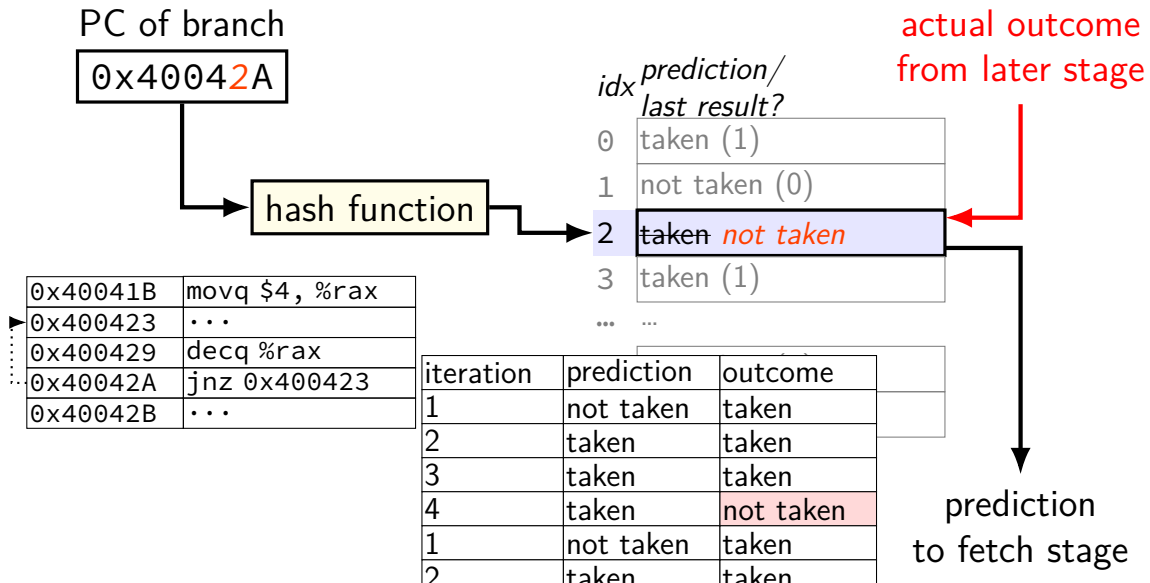
example



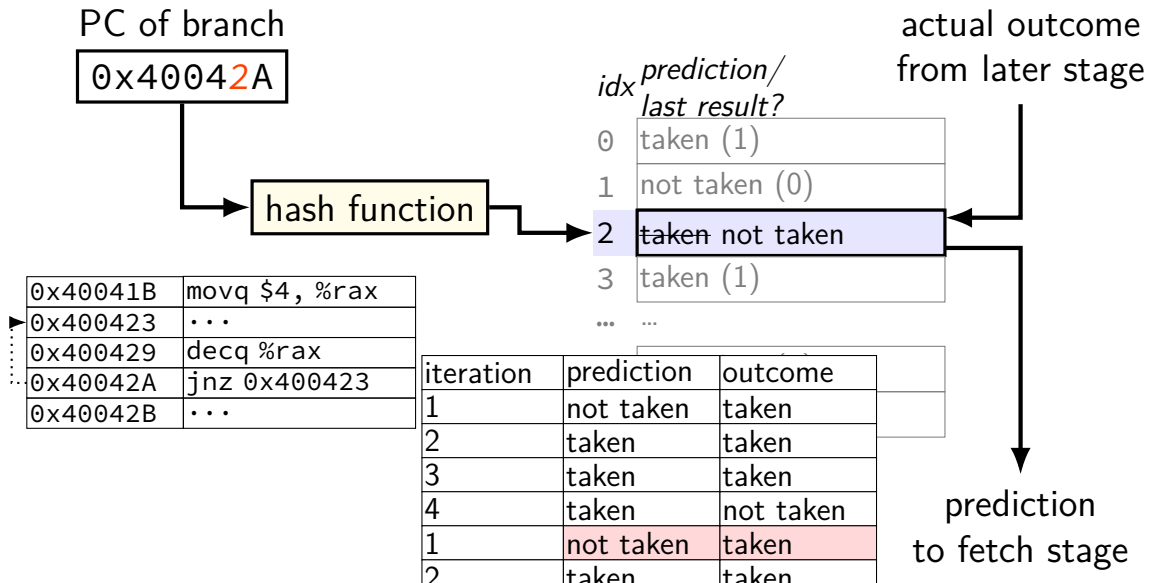
example



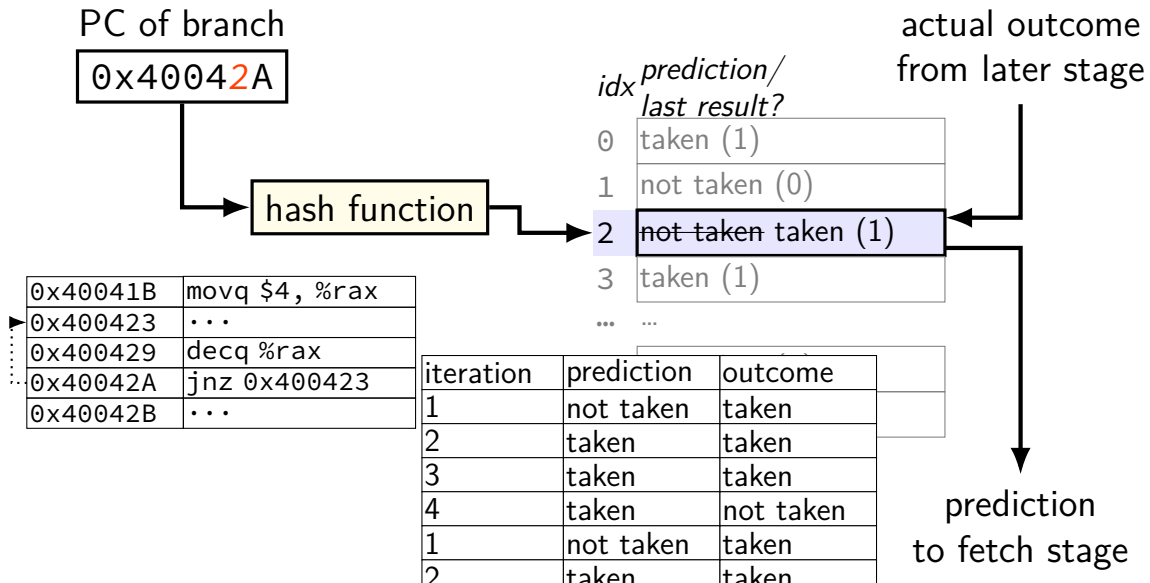
example



example



example



collisions?

two branches could have same hashed PC

nothing in table tells us about this

versus direct-mapped cache: had *tag bits* to tell

is it worth it?

adding tag bits makes table *much* larger and/or slower

but does anything go wrong when there's a collision?

collision results

possibility 1: both branches usually taken

no actual conflict — prediction is better(!)

possibility 2: both branches usually not taken

no actual conflict — prediction is better(!)

possibility 3: one branch taken, one not taken

performance probably worse

1-bit predictor for loops

predicts first and last iteration wrong

example: branch to beginning — but same for branch from beginning to end

everything else correct

exercise (pt 1)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate for $i \% 3 == 0$?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

exercise (pt 1)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate for $i \% 3 == 0$?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

i =	branch	pred	outcome	correct?
0	mod 3	???	<i>T</i>	???
1	mod 3	<i>T</i>	F	no
2	mod 3	F	F	yes
3	mod 3	F	T	no
...	...	...	...	...

exercise (pt 1)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate for $i \% 3 == 0$?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

i =	branch	pred	outcome	correct?
0	mod 3	???	T	???
1	mod 3	T	<i>F</i>	no
2	mod 3	<i>F</i>	F	yes
3	mod 3	F	T	no
...	...	...	...	...

exercise (pt 1)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate for $i \% 3 == 0$?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

i =	branch	pred	outcome	correct?
0	mod 3	???	T	???
1	mod 3	T	F	no
2	mod 3	F	<i>F</i>	yes
3	mod 3	<i>F</i>	T	no
...	...	...	...	...

exercise (pt 2)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate for $i == 50$?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

exercise (full)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

exercise (full)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

i =	branch	pred	outcome	correct?
0	mod 3	???	<i>T</i>	???
1	== 50	???	F	???
1	mod 3	<i>T</i>	F	—
2	== 50	F	F	✓
...	...	...	...	...

exercise (full)

use 1-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional jump misprediction rate?

```
int i = 0;
while (true) {
    if (i % 3 == 0)
        goto next;
    ...
next:
    i += 1;
    if (i == 50)
        break;
}
```

i =	branch	pred	outcome	correct?
0	mod 3	???	T	???
1	== 50	???	<i>F</i>	???
1	mod 3	T	F	—
2	== 50	<i>F</i>	F	✓
...	...	...	...	...

exercise soln (1)

i=	branch	predicted	outcome	correct?
0	mod 3	???	T	???
1	== 50	???	N	???
1	mod 3	T	N	
2	== 50	N	N	✓
2	mod 3	N	N	✓
3	== 50	N	N	✓
3	mod 3	N	T	
4	== 50	N	N	✓
...	...	...	...	...
48	mod 3	N	T	
49	== 50	N	N	✓
49	mod 3	T	N	
50	== 50	N	T	
0	mod 3	N	T	
1	== 50	T	N	

mod 3: correct for i=2,5,8,...,49 (16/50)
break: correct for i=2,3,...,48 (48/50)
overall: 64/100

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```

exercise soln (1)

i=	branch	predicted	outcome	correct?
0	mod 3	???	T	???
1	== 50	???	N	???
1	mod 3	T	N	
2	== 50	N	N	✓
2	mod 3	N	N	✓
3	== 50	N	N	✓
3	mod 3	N	T	
4	== 50	N	N	✓
...	...	...	...	...
48	mod 3	N	T	
49	== 50	N	N	✓
49	mod 3	T	N	
50	== 50	N	T	
0	mod 3	N	T	
1	== 50	T	N	

mod 3: correct for i=2,5,8,...,49 (16/50)
break: correct for i=2,3,...,48 (48/50)
overall: 64/100

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```

exercise soln (1)

i=	branch	predicted	outcome	correct?
0	mod 3	???	T	???
1	== 50	???	N	???
1	mod 3	T	N	
2	== 50	N	N	✓
2	mod 3	N	N	✓
3	== 50	N	N	✓
3	mod 3	N	T	
4	== 50	N	N	✓
...	...	...	...	...
48	mod 3	N	T	
49	== 50	N	N	✓
49	mod 3	T	N	
50	== 50	N	T	
0	mod 3	N	T	
1	== 50	T	N	

mod 3: correct for i=2,5,8,...,49 (16/50)
break: correct for i=2,3,...,48 (48/50)
overall: 64/100

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```


exercise soln (1)

i=	branch	predicted	outcome	correct?
0	mod 3	???	T	???
1	== 50	???	N	???
1	mod 3	T	N	
2	== 50	N	N	✓
2	mod 3	N	N	✓
3	== 50	N	N	✓
3	mod 3	N	T	
4	== 50	N	N	✓
...	...	...	...	...
48	mod 3	N	T	
49	== 50	N	N	✓
49	mod 3	T	N	
50	== 50	N	T	
0	mod 3	N	T	
1	== 50	T	N	

mod 3: correct for i=2,5,8,...,49 (16/50)
break: correct for i=2,3,...,48 (48/50)
overall: 64/100

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```

exercise soln (1)

i=	branch	predicted	outcome	correct?
0	mod 3	???	T	???
1	== 50	???	N	???
1	mod 3	T	N	
2	== 50	N	N	✓
2	mod 3	N	N	✓
3	== 50	N	N	✓
3	mod 3	N	T	
4	== 50	N	N	✓
...	...	...	...	...
48	mod 3	N	T	
49	== 50	N	N	✓
49	mod 3	T	N	
50	== 50	N	T	
0	mod 3	N	T	
1	== 50	T	N	

mod 3: correct for i=2,5,8,...,49 (16/50)
break: correct for i=2,3,...,48 (48/50)
overall: 64/100

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```

branch target buffer

what if we can't decode LABEL from machine code for `jmp LABEL` or `jle LABEL` fast?

will happen in more complex pipelines

what if we can't decode that there's a `RET`, `CALL`, etc. fast?

BTB: cache for branch targets

idx	valid	tag	ofst	type	target	(more info?)
0x00	1	0x400	5	Jxx	0x3FFFF3	...
0x01	1	0x401	C	JMP	0x401035	---
0x02	0	---	---	---	---	---
0x03	1	0x400	9	RET	---	...
...	...	...	...	...	...	...
0xFF	1	0x3FF	8	CALL	0x404033	...

valid	...
1	...
0	...
0	...
0	...
...	...
0	...

```
0x3FFFF3:  movq %rax, %rsi
0x3FFFF7:  pushq %rbx
0x3FFFF8:  call 0x404033
0x400001:  popq %rbx
0x400003:  cmpq %rbx, %rax
0x400005:  jle 0x3FFFF3
...
0x400031:  ret
...
```

BTB: cache for branch targets

idx	valid	tag	ofst	type	target	(more info?)
0x00	1	0x400	5	Jxx	0x3FFFF3	...
0x01	1	0x401	C	JMP	0x401035	---
0x02	0	---	---	---	---	---
0x03	1	0x400	9	RET	---	...
...	...	...	...	...	...	...
0xFF	1	0x3FF	8	CALL	0x404033	...

valid	...
1	...
0	...
0	...
0	...
...	...
0	...

```
0x3FFFF3:  movq %rax, %rsi
0x3FFFF7:  pushq %rbx
0x3FFF8:   call 0x404033
0x400001:  popq %rbx
0x400003:  cmpq %rbx, %rax
0x400005:  jle 0x3FFFF3
...
0x400031:  ret
...
```

BTB: cache for branch targets

idx	valid	tag	ofst	type	target	(more info?)
0x00	1	0x400	5	Jxx	0x3FFFF3	...
0x01	1	0x401	C	JMP	0x401035	---
0x02	0	---	---	---	---	---
0x03	1	0x400	9	RET	---	...
...	...	...	...	...	...	...
0xFF	1	0x3FF	8	CALL	0x404033	...

valid	...
1	...
0	...
0	...
0	...
...	...
0	...

```
0x3FFFF3:  movq %rax, %rsi
0x3FFFF7:  pushq %rbx
0x3FFFF8:  call 0x404033
0x400001:  popq %rbx
0x400003:  cmpq %rbx, %rax
0x400005:  jle 0x3FFFF3
...
0x400031:  ret
...
```

indirect branch prediction

`jmp *%rax` or `jmp *(%rax, %rcx, 8)`

BTB can provide a prediction

but can do better with more context

example—predict based on other recent computed jumps

good for polymorphic method calls

table lookup with `Hash(last few jmps)`

instead of `Hash(this jmp)`

beyond 1-bit predictor

devote *more space* to storing history

main goal: *rare exceptions don't immediately change prediction*

example: branch taken 99% of the time

1-bit predictor: wrong about 2% of the time

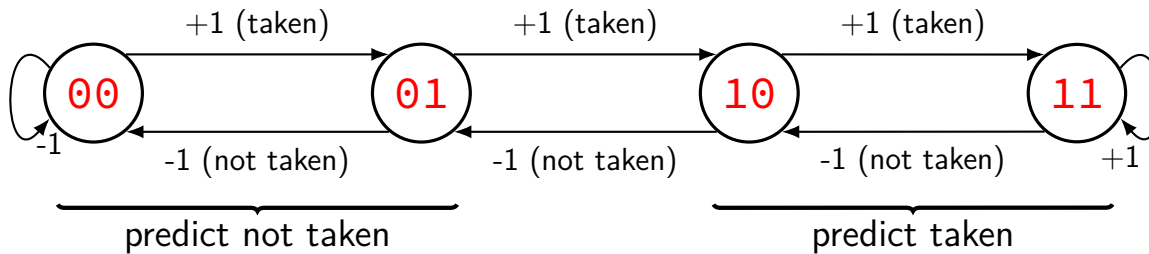
- 1% when branch not taken

- 1% of taken branches right after branch not taken

new predictor: wrong about 1% of the time

- 1% when branch not taken

2-bit saturating counter



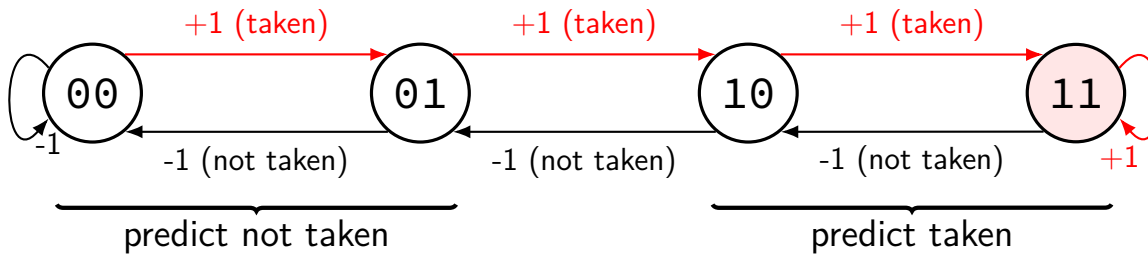
PC of branch

0x40042A

hash function

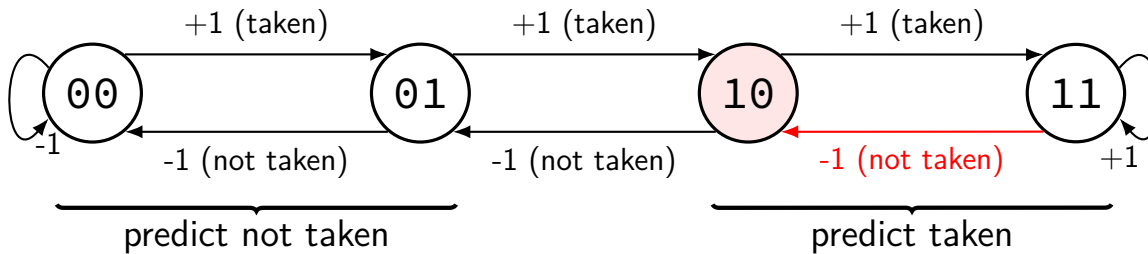
index	counter
0	11
1	01
2	11
...	...
14	10

2-bit saturating counter



branch always taken:
value increases to 'strongest' taken value

2-bit saturating counter



branch almost always taken, then not taken once:
still predicted as taken

example

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

iter.	table before	prediction	outcome	table after
1	01	not taken	taken	10
2	10	taken	taken	11
3	11	taken	taken	11
4	11	taken	not taken	10
1	10	taken	taken	11
2	11	taken	taken	11
3	11	taken	taken	11
4	11	taken	not taken	10
1	10	taken	taken	11
...	...	...	...	...

generalizing saturating counters

2-bit counter: ignore one exception to taken/not taken

3-bit counter: ignore more exceptions

000 $\leftrightarrow$ 001 $\leftrightarrow$ 010 $\leftrightarrow$ 011 $\leftrightarrow$ 100 $\leftrightarrow$ 101 $\leftrightarrow$ 110 $\leftrightarrow$ 111

000-011: not taken

100-111: taken

exercise

use 2-bit predictor on this loop

executed in outer loop (not shown) many, many times

what is the conditional branch misprediction rate?

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```

exercise soln (1)

i=	branch	predicted	outcome	correct?
0	mod 3	01 (N)	T	
1	break	01 (N)	N	✓
1	mod 3	10 (T)	N	
2	break	00 (N)	N	✓
2	mod 3	01 (N)	N	✓
3	break	00 (N)	N	✓
3	mod 3	00 (N)	T	
4	break	00 (N)	N	✓
...	...	...	...	...
48	mod 3	00 (N)	T	
49	break	00 (N)	N	✓
49	mod 3	01 (N)	N	✓
50	break	00 (N)	T	
0	mod 3	00 (N)	T	
1	break	01 (N)	N	✓

mod 3: correct for i=1,2,4,5,7,8,...,49
(33/50)

mod 3: ends up always predicting not taken

break: correct for i=2,3,...,48

(49/50)

break: ends up always predicting not taken

overall: 82/100

```
int i = 0;
while (true) {
    if (i % 3 == 0) goto next;
    ...
next:
    i += 1;
    if (i == 50) break;
}
```

branch patterns

```
i = 4;  
do {  
    ...  
    i -= 1;  
} while (i != 0);
```

typical pattern for jump to top of do-while above:

TTTN TTTN TTTN TTTN TTTN...(T = taken, N = not taken)

goal: take advantage of recent pattern to make predictions

just saw 'NTTTNT'? predict T next

'TNTTTN'? predict T; 'TTNTTT'? predict N next

local pattern predictor (incomplete)

PC of branch

0x40042A

hash function

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

4-iter loop: TTTN TTTN TTTN ...

idx *recent pattern*

0	NNNNNN
1	NNTNTT
2	TTTTNT
3	TTTTTT
...	...
14	NTNTTN
15	NNTTTT

actual outcome

from commit(?) stage

???

convert to
prediction
???

prediction
to fetch stage

local pattern predictor (incomplete)

PC of branch

0x40042A

hash function

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

4-iter loop: TTTN TTTN TTTN ...

iter.	pattern tbl before	predicted	outcome	pattern tbl after
1	TTTTNT	???	taken	TTTNTT

idx recent
pattern

0	NNNNNN
1	NNTNTT
2	TTTTNT
3	TTTTTT
...	...
14	NTNTTN
15	NNTTTT

actual outcome

from commit(?) stage

???

convert to
prediction

???

prediction
to fetch stage

local pattern predictor (incomplete)

PC of branch

0x40042A

hash function

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

4-iter loop: TTTN TTTN TTTN ...

iter.	pattern tbl before	predicted	outcome	pattern tbl after
1	TTTTNT	???	taken	TTTNTT

idx recent
pattern

0	NNNNNN
1	NNTNTT
2	TTTTNT
3	TTTTTT
...	...
14	NTNTTN
15	NNTTTT

actual outcome
from commit(?) stage

???

convert to
prediction
???

prediction
to fetch stage

local pattern predictor (incomplete)

PC of branch

0x40042A

hash function

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

idx recent
pattern

0	NNNNNN
1	NNTNTT
2	TTTNTT
3	TTTTTT
...	...
14	NTNTTN
15	NNTTTT

actual outcome

from commit(?) stage

???

convert to
prediction

???

prediction
to fetch stage

4-iter loop: TTTN TTTN TTTN ...

iter.	pattern tbl before	predicted	outcome	pattern tbl after
1	TTTTNT	???	taken	TTTNTT
2	TTTNTT	???	taken	TTNTTT

local pattern predictor (incomplete)

PC of branch

0x40042A

hash function

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

idx recent
pattern

0	NNNNNN
1	NNTNTT
2	TTTNTT <i>T</i>
3	TTTTTT
...	...
14	NTNTTN
15	NNTTTT

actual outcome

from commit(?) stage

???

convert to
prediction

???

prediction
to fetch stage

4-iter loop: TTTN TTTN TTTN ...

iter.	pattern tbl before	predicted	outcome	pattern tbl after
1	TTTTNT	???	taken	TTTNTT
2	TTTNTT	???	taken	TTNTTT
3	TTNTTT	???	taken	TNTTTT

local pattern predictor (incomplete)

PC of branch

0x40042A

hash function

0x40041B	movq \$4, %rax
0x400423	...
0x400429	decq %rax
0x40042A	jz 0x400423
0x40042B	...

idx recent
pattern

0	NNNNNN
1	NNTNTT
2	TTTNTTT
3	TTTTTT
...	...
14	NTNTTN
15	NNTTTT

actual outcome

from commit(?) stage

???

convert to
prediction

???

prediction
to fetch stage

4-iter loop: TTTN TTTN TTTN ...

iter.	pattern tbl before	predicted	outcome	pattern tbl after
1	TTTTNT	???	taken	TTTNTT
2	TTTNTT	???	taken	TTNTTT
3	TTNTTT	???	taken	TNTTTT
4	TNTTTT	???	not taken	NTTTTN

recent pattern to prediction?

easy cases:

just saw TTTTTT: predict T

just saw NNNNNN: predict N

just saw TNTNTN: predict T

hard cases:

TTNTTTT

predict T? loop with many iterations
(NTTTTTTTNTTTTTTTNTTTTTT...)

predict T? if statement mostly taken
(TTTTNTTNTTTTTTTTTTTNTTTT...)

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTN
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern
NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter
00
00
...
10
...
11
...
01
01
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTN
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern

NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter

00
00
...
10
...
11
...
01
01
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTNT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern
NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter
00
00
...
10
...
11
...
01
01 10
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTNT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern
NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter
00
00
...
10
...
11
...
01 10
11

prediction
to fetch sta

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTNTT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern
NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter
00
00
...
10
...
11
...
01 10
01 10
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTNTT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern

NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter

00
00
...
10
...
11
...
01 10
01 10
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTT NTTT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern
NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter
00
00
...
10
...
11
...
01 10
01 10
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT
4	NTTT	01	not taken	taken	10	TTTT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTNTTT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern

NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter

00
00
...
10 11
...
11
...
01 10
01 10
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT
4	NTTT	01	not taken	taken	10	TTTT

prediction
to fetch sta

history of history

PC of branch

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTT TNTT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern
NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTT

counter
00
00
...
10 11
...
11
...
01 10
01 10
11

prediction
to fetch sta

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTT	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT
4	NTTT	01	not taken	taken	10	TTTT

history of history

PC of branch

0x40042A

hash

idx recent
pattern

0	NNNN
1	TNTT
2	TTTT NTTT
3	TTTT
...	...
14	NTTN
15	TTTT

actual outcome
from commit(?) stage

pattern

NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter

00
00
...
10 11
...
11
...
01 10
01 10 11
11

prediction
to fetch sta

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT
4	NTTT	01	not taken	taken	10	TTTT

history of history

PC of branch

actual outcome
from commit(?) stage

0x40042A

hash

idx	recent pattern
0	NNNN
1	TNTT
2	TTTT NTTT
3	TTTT
...	...
14	NTTN
15	TTTT

pattern

NNNN
NNNT
...
NTTT
...
TNTT
...
TTNT
TTTN
TTTT

counter

00
00
...
10 11
...
11
...
01 10
01 10 11
11

iter.	branch to pat. tbl before	pat. to counter before	predict	actual	pat. to counter after	branch to pat. tbl after
1	TTTN	01	not taken	taken	10	TTNT
2	TTNT	01	not taken	taken	10	TNTT
3	TNTT	11	taken	taken	11	NTTT
4	NTTT	01	not taken	taken	10	TTTT

prediction
to fetch sta

local patterns and collisions (1)

```
i = 10000;  
do {  
    p = malloc(...);  
    if (p == NULL) goto error; // BRANCH 1  
    ...  
} while (i-- != 0); // BRANCH 2
```

what if branch 1 and branch 2 hash to same table entry?

local patterns and collisions (1)

```
i = 10000;  
do {  
    p = malloc(...);  
    if (p == NULL) goto error;  // BRANCH 1  
    ...  
} while (i-- != 0); // BRANCH 2
```

what if branch 1 and branch 2 hash to same table entry?

pattern: TNTNTNTNTNTNTNTNT...

actually no problem to predict!

local patterns and collisions (2)

```
i = 10000;  
do {  
    if (i % 2 == 0) goto skip; // BRANCH 1  
    ...  
    p = malloc(...);  
    if (p == NULL) goto error; // BRANCH 2  
skip: ...  
} while (i-- != 0); // BRANCH 3
```

what if branch 1 and branch 2 and branch 3 hash to same table entry?

local patterns and collisions (2)

```
i = 10000;  
do {  
    if (i % 2 == 0) goto skip; // BRANCH 1  
    ...  
    p = malloc(...);  
    if (p == NULL) goto error; // BRANCH 2  
skip: ...  
} while (i-- != 0); // BRANCH 3
```

what if branch 1 and branch 2 and branch 3 hash to same table entry?

pattern: TTNNTTNNTTNNTTNNTT

also no problem to predict!

local patterns and collisions (3)

```
i = 10000;  
do {  
    if (A) goto one // BRANCH 1  
    ...  
one:  
    if (B) goto two // BRANCH 2  
    ...  
two:  
    if (A or B) goto three // BRANCH 3  
    ...  
    if (A and B) goto three // BRANCH 4  
    ...  
three:  
    ... // changes A, B  
} while (i-- != 0);
```

what if branch 1-4 hash to same table entry?

better for prediction of branch 3 and 4

global history predictor: idea

one predictor idea: ignore the PC

just record taken/not-taken pattern for all branches

lookup in big table like for local patterns

global history predictor (1)

branch history register

NTTT

<i>pat</i>	<i>counter</i>
NNNN	00
NNNT	00
...	...
NTTT	10
TNNN	01
TNNT	10
TNTN	11
...	...
TTTN	10
TTTT	11

outcome
from
commit(?)

prediction
to fetch stage

global history predictor (1)

```

i = 10000;
do {
    if (i % 2 == 0) goto skip;
    ...
    if (p == NULL) goto error;
skip:
    ...
} while (i-- != 0);
    
```

branch history register

NTTT

pat

NNNN

NNNT

...

NTTT

TNNN

TNNT

TNTN

...

TTTN

TTTT

counter

00

00

...

10

01

10

11

...

10

11

outcome
from
commit(?)

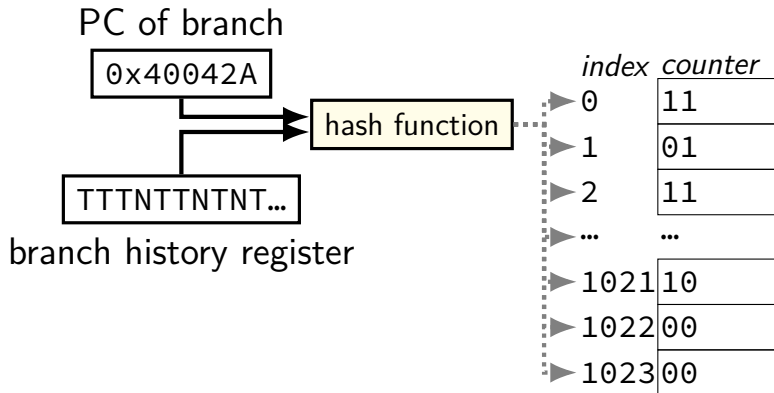
prediction
to fetch stage

iter./ branch	history before	counter before	predict	outcome	counter after	history after
0/mod 2	NTTT	10	taken	taken	11	TTTT
0/loop	TTTT			taken		TTTT
1/mod 2	TTTT			not taken		TTTN
1/error	TTTN			not taken		TTNN
1/loop	TNNT			taken		NNTT

correlating predictor

global history *and* local info good together

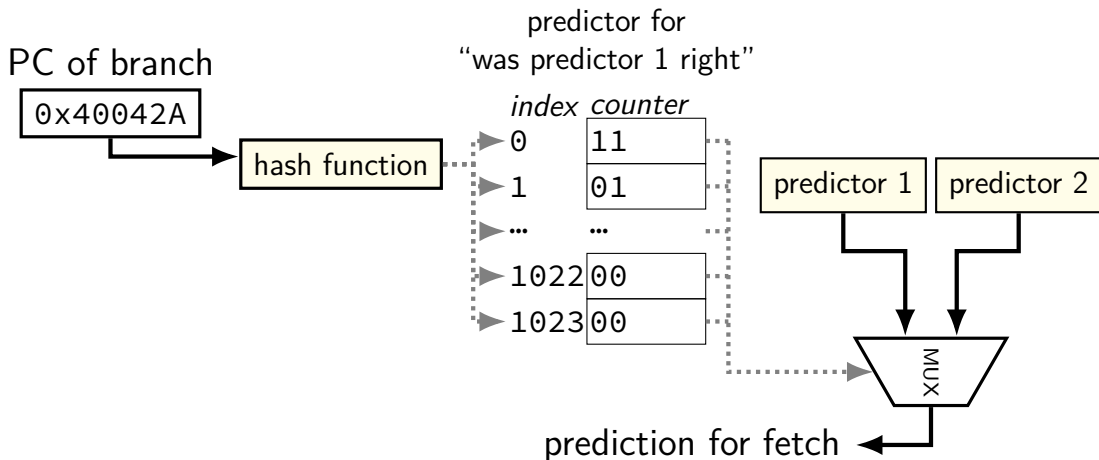
one idea: *combine history register + PC* (“gshare”)



mixing predictors

different predictors good at different times

one idea: have two predictors, + predictor to predict which is right



loop count predictors (1)

```
for (int i = 0; i < 64; ++i)  
    ...
```

can we predict this perfectly with predictors we've seen

yes — local or global history with 64 entries

but this is very important — more efficient way?

loop count predictors (2)

loop count predictor idea: look for NNNNNNT+repeat (or TTTTTTN+repeat)

track for each possible loop branch:

- how many repeated Ns (or Ts) so far

- how many repeated Ns (or Ts) last time before one T (or N)

- something to indicate this pattern is useful?

known to be used on Intel

benchmark results

from 1993 paper

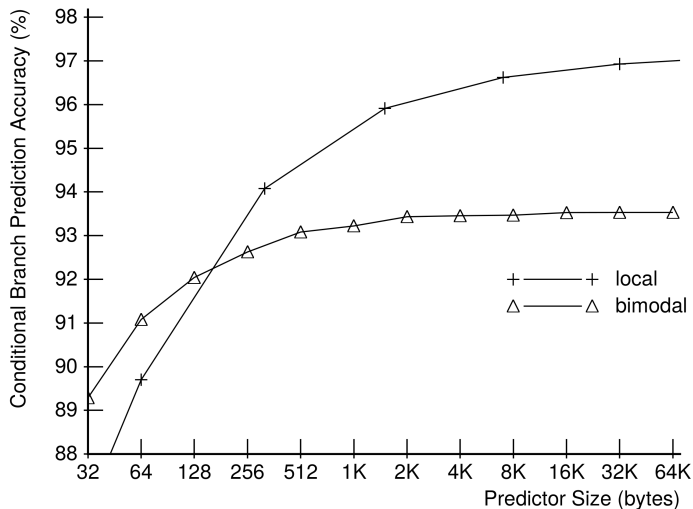
(not representative of modern workloads?)

rate for conditional branches on benchmark

variable table sizes

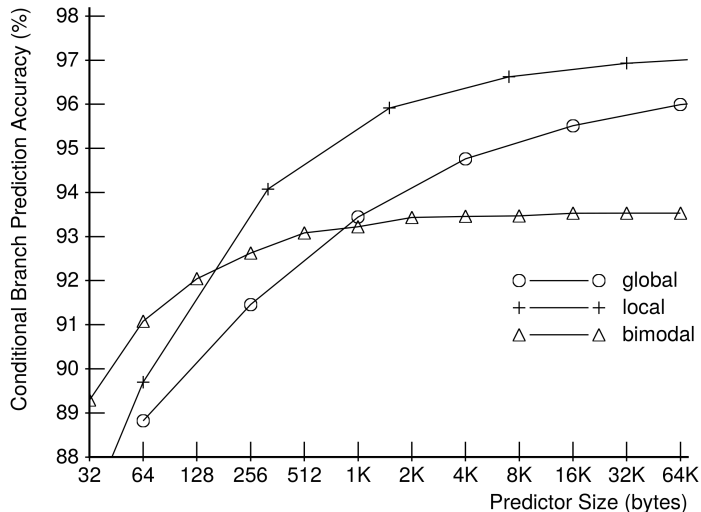
2-bit ctr + local history

from McFarling, “Combining Branch Predictors” (1993)



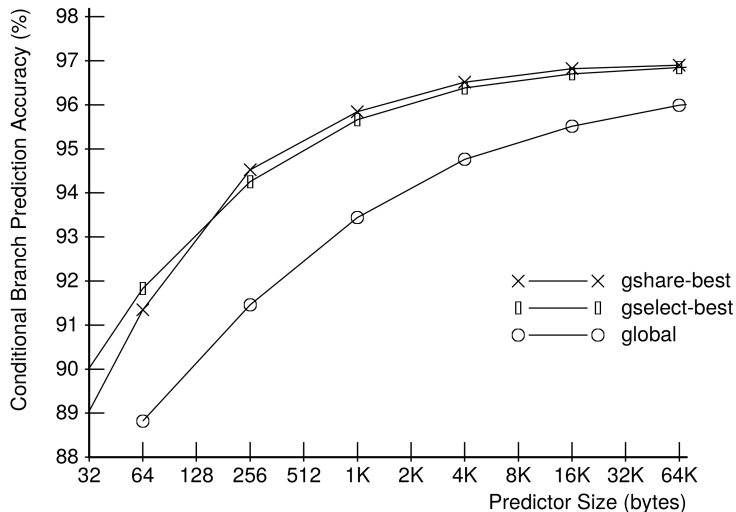
2-bit (bimodal) + local + global hist

from McFarling, "Combining Branch Predictors" (1993)



global + hash(global+PC) (gshare/gselect)

from McFarling, "Combining Branch Predictors" (1993)



real BP?

details of modern CPU's branch predictors often not public
but...

Google Project Zero blog post with reverse engineered details

<https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html>

for RE'd BTB size:

<https://xania.org/201602/haswell-and-ivy-btb>

reverse engineering Haswell BPs

branch target buffer

- 4-way, 4096 entries

- ignores bottom 4 bits of PC?

- hashes PC to index by shifting + XOR

- seems to store 32 bit offset from PC (not all 48+ bits of virtual addr)

indirect branch predictor

- like the global history + PC predictor we showed, but...

- uses history of recent branch addresses instead of taken/not taken

- keeps some info about last 29 branches

what about conditional branches??? loops???

- couldn't find a reasonable source

backup slides

backup slides

exercise: static prediction

```
.global foo
foo:
    xor %eax, %eax // eax ← 0
foo_loop_top:
    test $0x1, %edi
    je foo_loop_bottom // if (edi & 1 == 0) goto for_loop_bottom
    add %edi, %eax
foo_loop_bottom:
    dec %edi // edi = edi - 1
    jg for_loop_top // if (edi > 0) goto for_loop_top
    ret
```

suppose `%edi = 3` (initially)

and using forward-not-taken, backwards-taken strategy:

how many mispredictions for `je`? for `jg`?