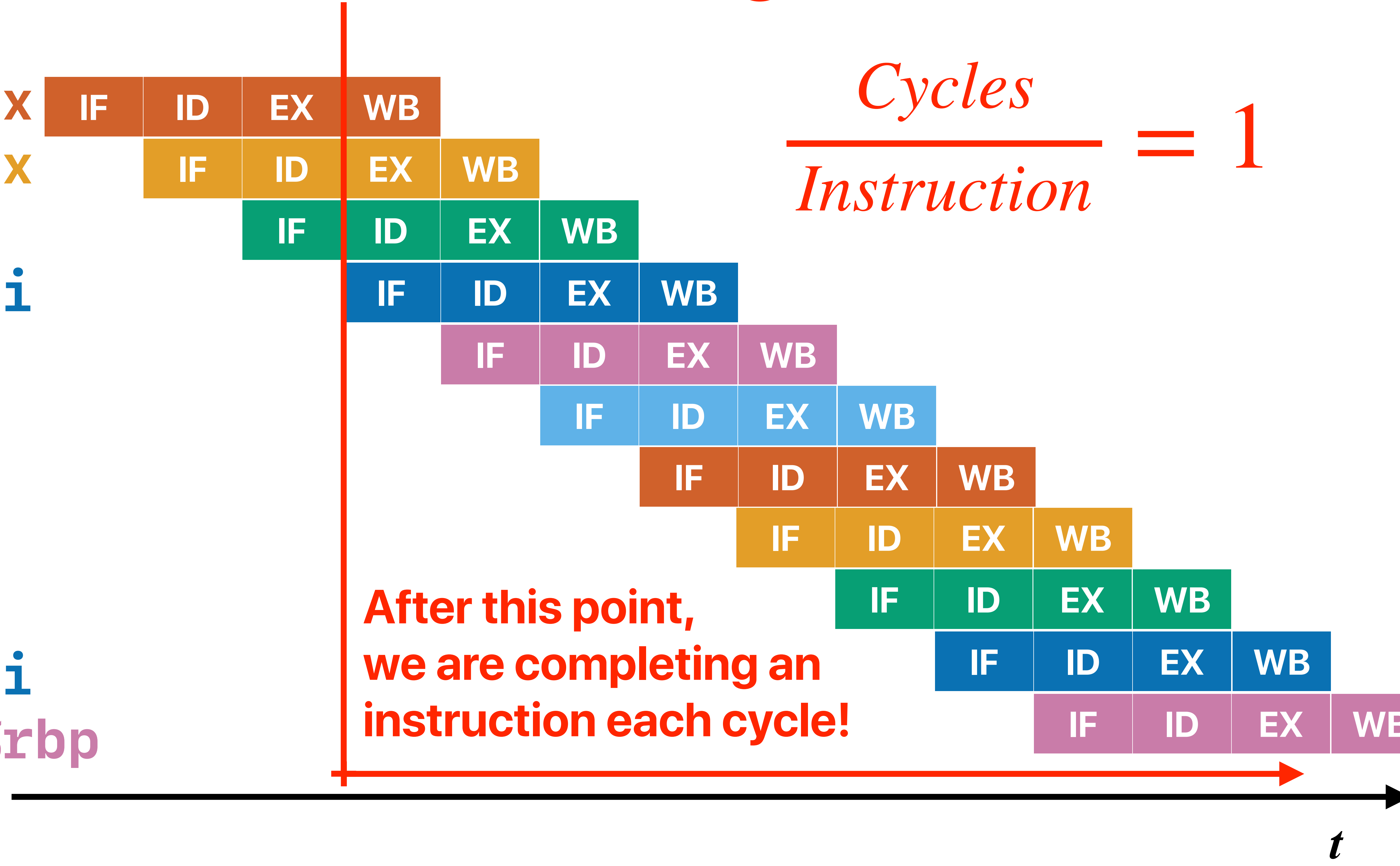
Modern Processor Design (2): I guess I just feel like...

Hung-Wei Tseng

Recap: Pipelining

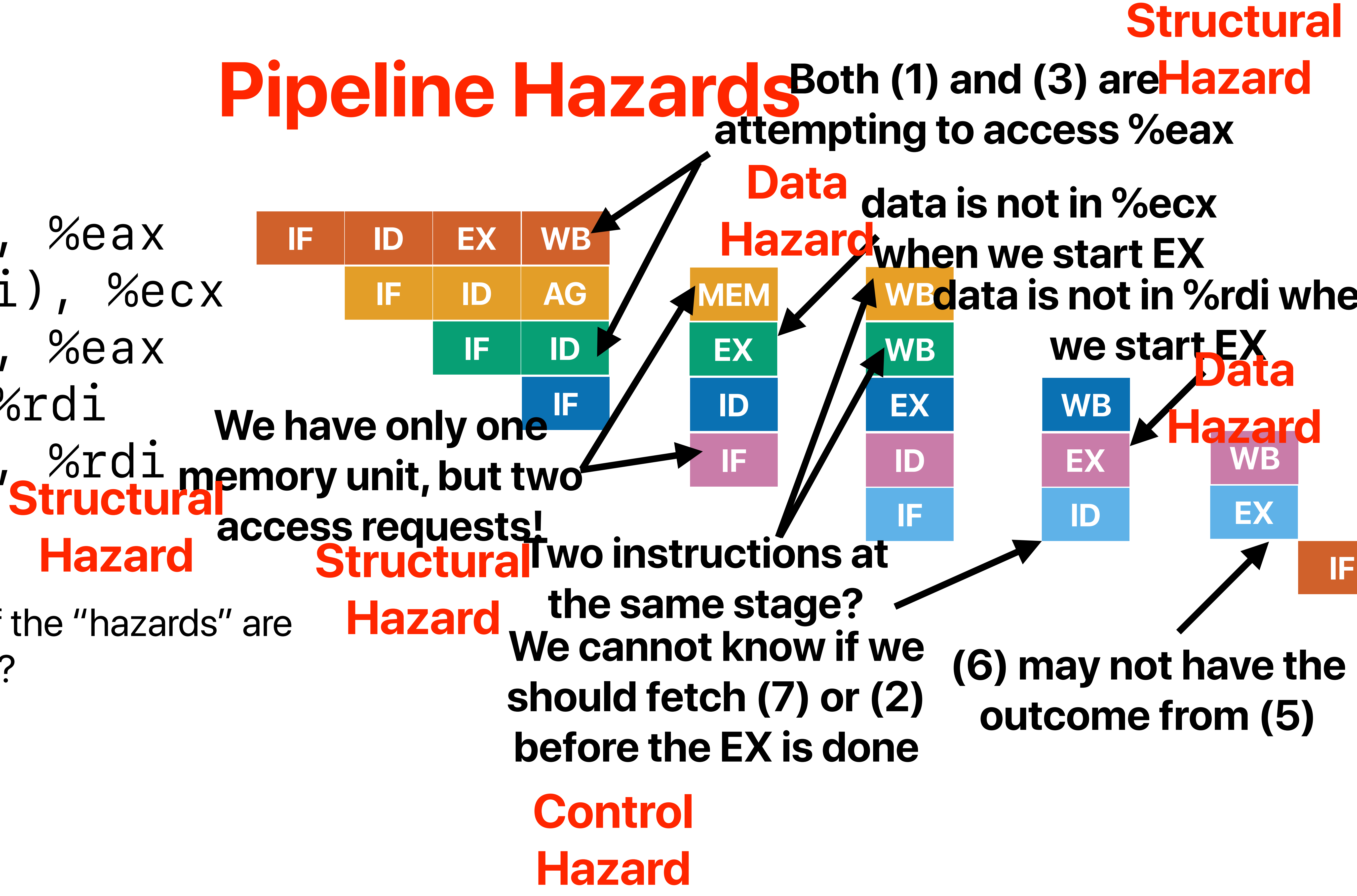
```
addl   %eax, %eax
addl   %rdi, %ecx
addq   $4, %r11
testl  %esi, %esi
movl   $10, %edx
pushq  %r12
pushq  %rbp
pushq  %rbx
subq   $8, %rsp
addl   %rsi, %rdi
movslq  %eax, %rbp
```



$$\frac{Cycles}{Instruction} = 1$$

Pipeline Hazards

```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
⑥ jne  .L3
⑦ ret
```



- How many of the “hazards” are data hazards?
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Recap: Structural Hazards

- Force later instructions to stall
- Improve the pipeline unit design to allow parallel execution
  - Write-first, read later register files
  - Split L1-Cache

Outline

- The cost of control hazards
- Dynamic branch predictions
  - Local predictor — 2 bit
  - Global predictor — 2-level
  - Hybrid predictors
    - Tournament
    - Perceptron

Control Hazards

How does the code look like?

```
for (unsigned i = 0; i < size; ++i) {
    if (data[i] < threshold)
        call_when_true(&a[i]);
    else
        call_when_false(&a[i]);
}
```

// taken when true

// taken when true

Branch taken simply means we are using branch target address as the next address

```
.LFB16:
    endbr64
    testl %esi, %esi
    jle   .L10
    movslq   %esi, %rsi
    pushq %r12
    leaq  (%rdi,%rsi,8), %r12
    pushq %rbp
    movslq   %edx, %rbp
    pushq %rbx
    movq  %rdi, %rbx
    jmp   .L5
    .p2align 4,,10
    .p2align 3
.L15:
    call  call_when_true@PLT
    addq  $8, %rbx
```

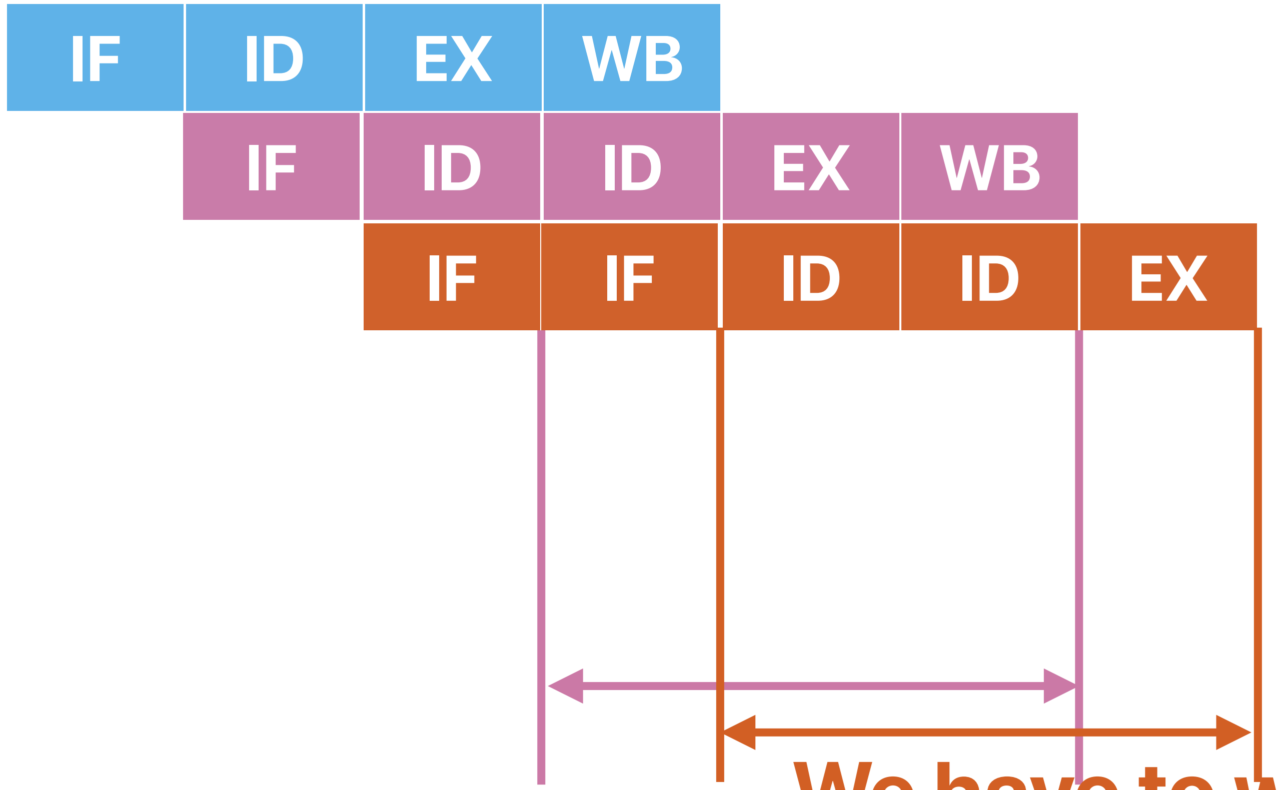
Branch taken

Branch taken

```
    cmpq  %r12, %rbx
    je .L14
.L5:
    movq  %rbx, %rdi
    cmpq  %rbp, (%rbx)
    jl .L15
    call  call_when_false@PLT
    addq  $8, %rbx
    cmpq  %r12, %rbx
    jne   .L5
.L14:
    popq  %rbx
    xorl  %eax, %eax
    popq  %rbp
    popq  %r12
    ret
```

Why is "branch" problematic in performance?

```
① addq $8, %rbx
② cmpq %r12, %rbx
③ jne  .L5
```



The latency of executing the cmpq instruction

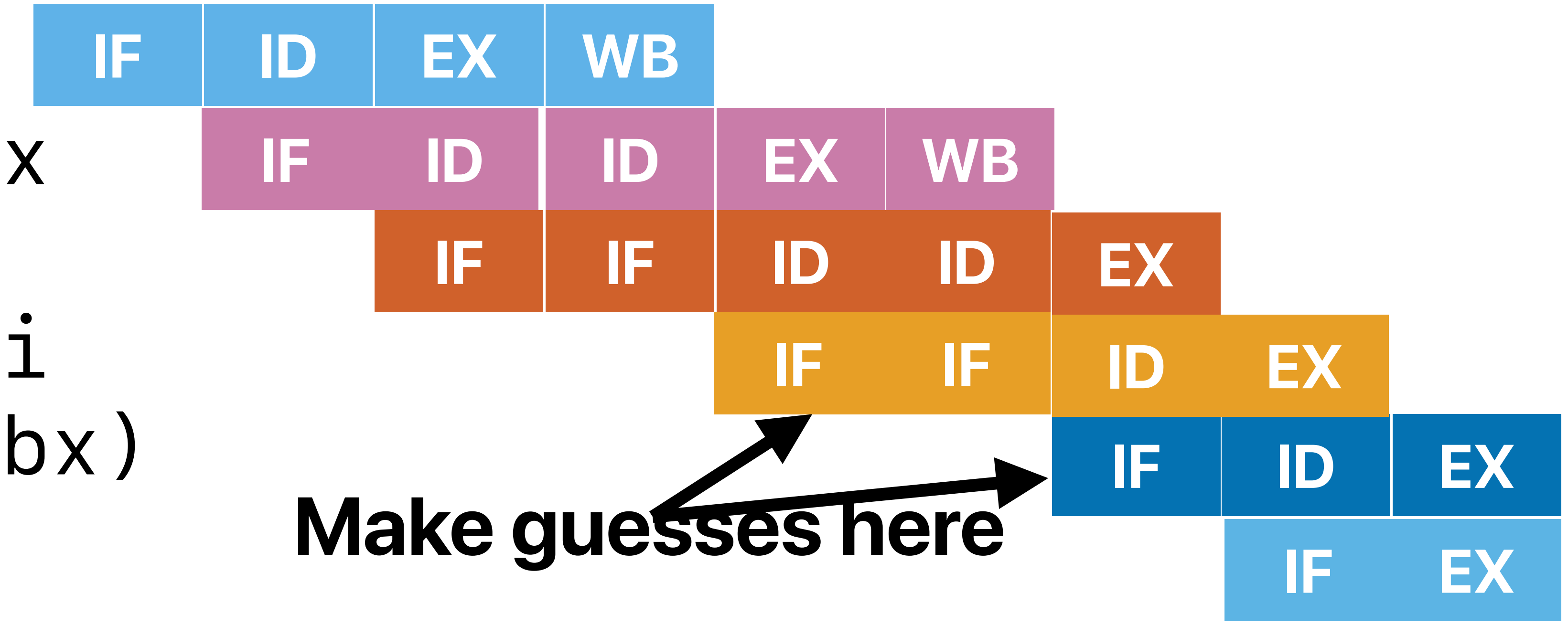
We have to wait almost as long as the latency of the previous instruction to make a decision — we cannot fetch anything before that

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved

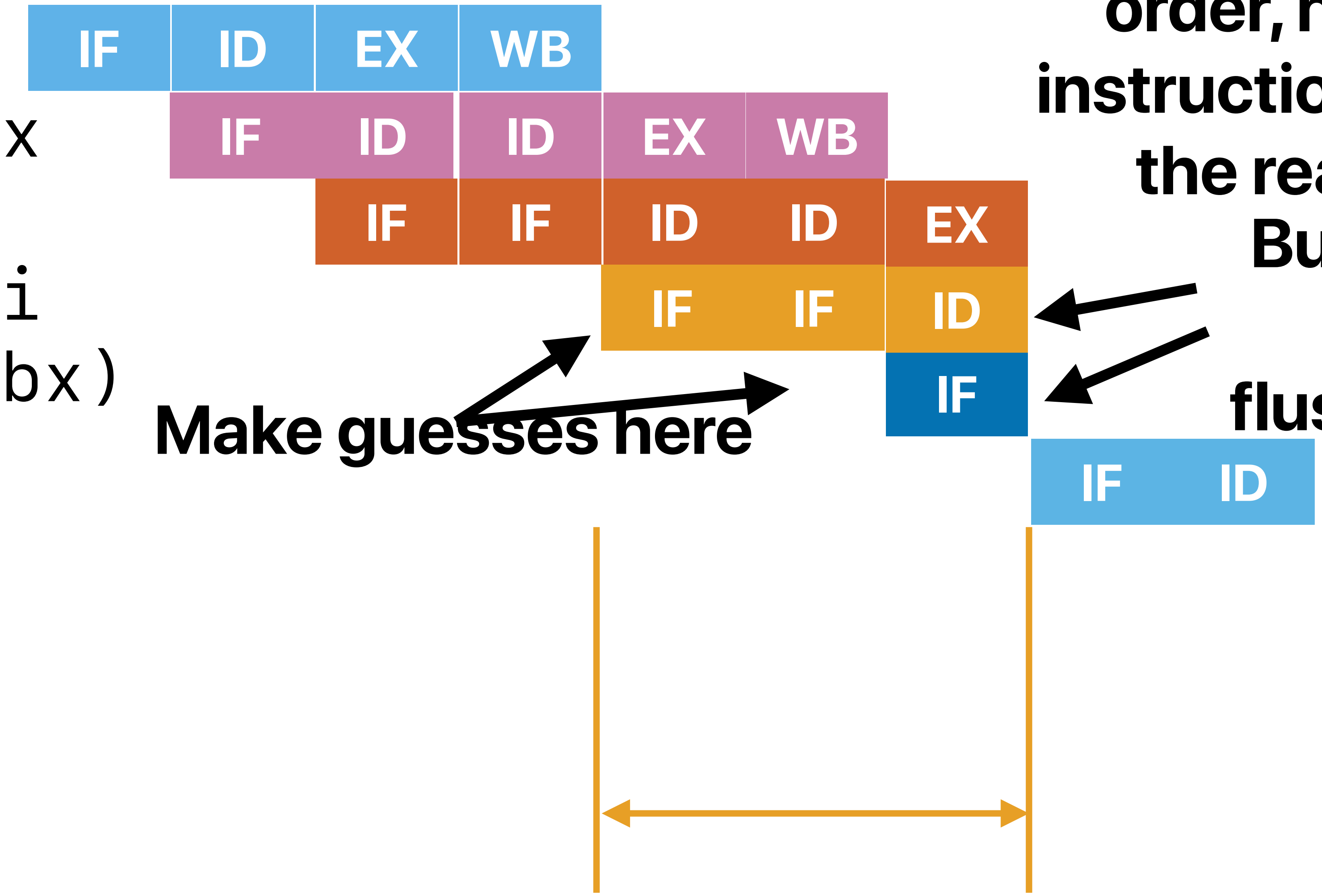
Prediction: What if we guessed right?

```
① 	addq  $8, %rbx
② 	cmpq  %r12, %rbx
③ 	jne   .L5
④ .L5:	movq  %rbx, %rdi
⑤ 	cmpq  %rbp, (%rbx)
```



Prediction: What if we are wrong?

```
①        addq  $8, %rbx
②        cmpq  %r12, %rbx
③        jne   .L5
④ .L5:   movq  %rbx, %rdi
⑤        cmpq  %rbp, (%rbx)
⑥ .L14:  popq  %rbx
```

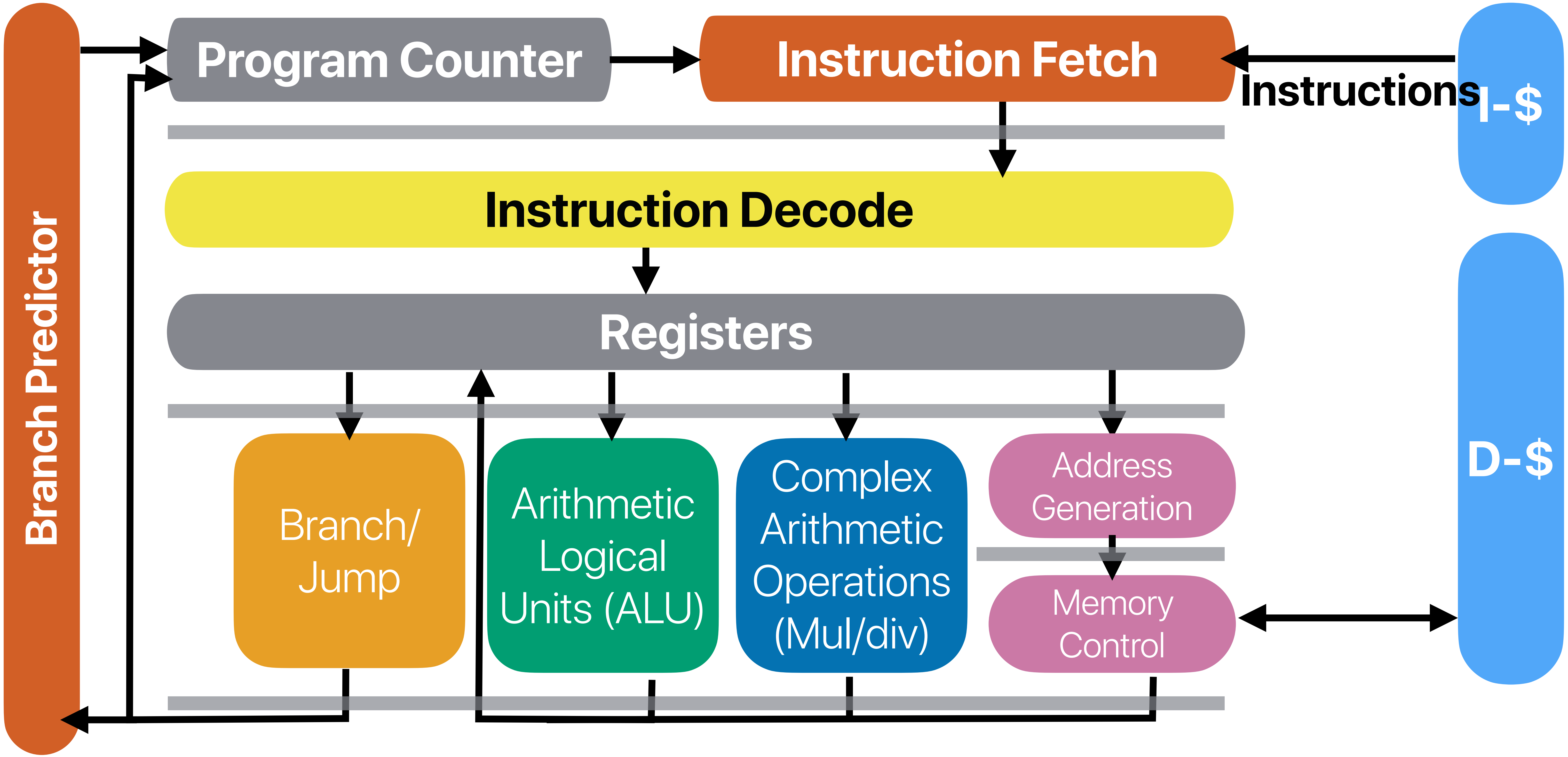


Since we execute in-order, no predicted instructions will modify the real registers

Bubble the rest stages by flushing pipeline registers

We still only wasted three cycles — same as not doing anything

Microprocessor with a "branch predictor"



Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!

https://www.pollev.com/hungweitseng close in 1:30

What should branch prediction "predict"

- How many of the following statements are true regarding the why is branch can lead to serious performance issues
  - ① The result value of the previous instruction generating the input to the branch
  - ② The direction of the branch (i.e., taken or not-taken)
  - ③ The target address of the branch
  - ④ The forth-through address of the branch
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

What should branch prediction "predict"

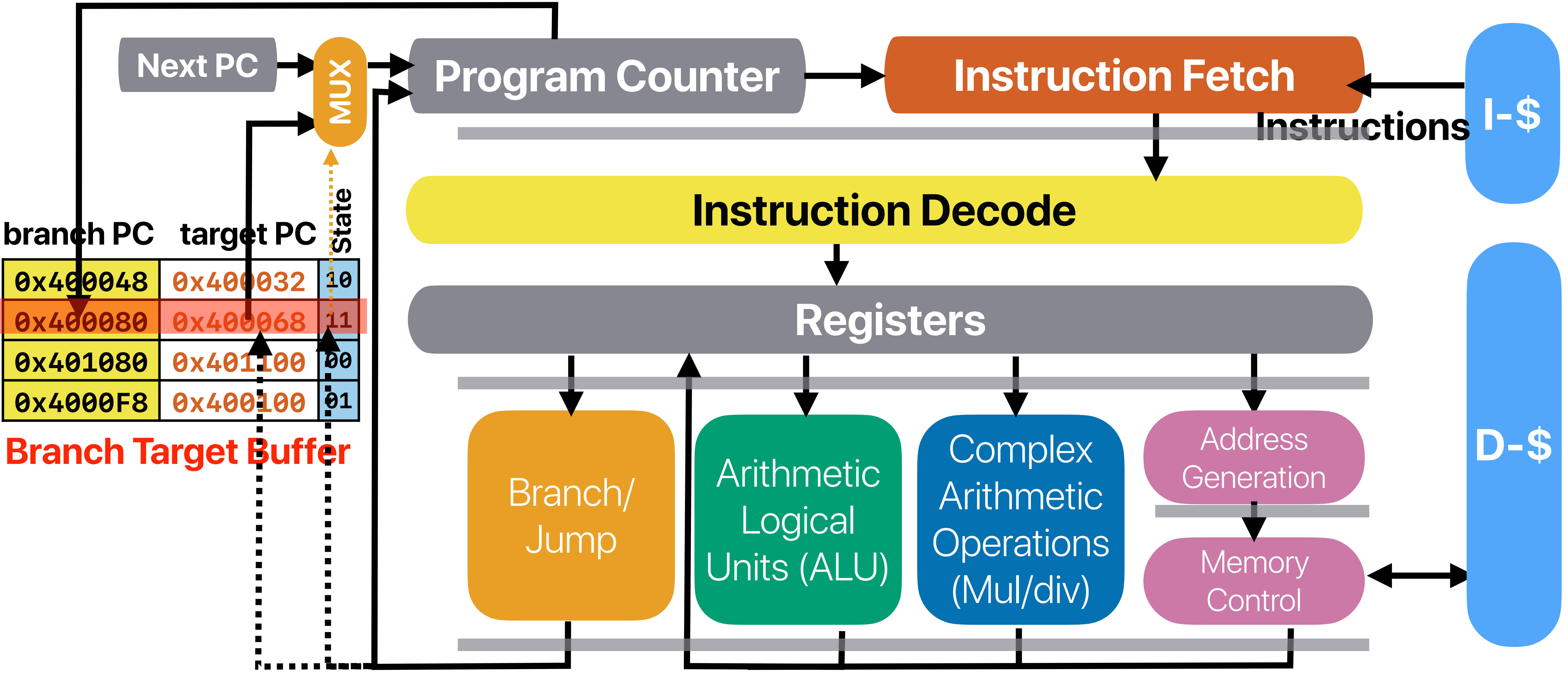
- How many of the following statements are true regarding the why is branch can lead to serious performance issues
  ① The result value of the previous instruction generating the input to the branch
  ② The direction of the branch (i.e., taken or not-taken)
  ③ The target address of the branch
  ④ The forth-through address of the branch

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

What are the "outcome" of the branch?
- **Taken, not-take** **You need to predict that — history/states**
- **Target address, if taken**

You need a cheatsheet for that — branch target buffer

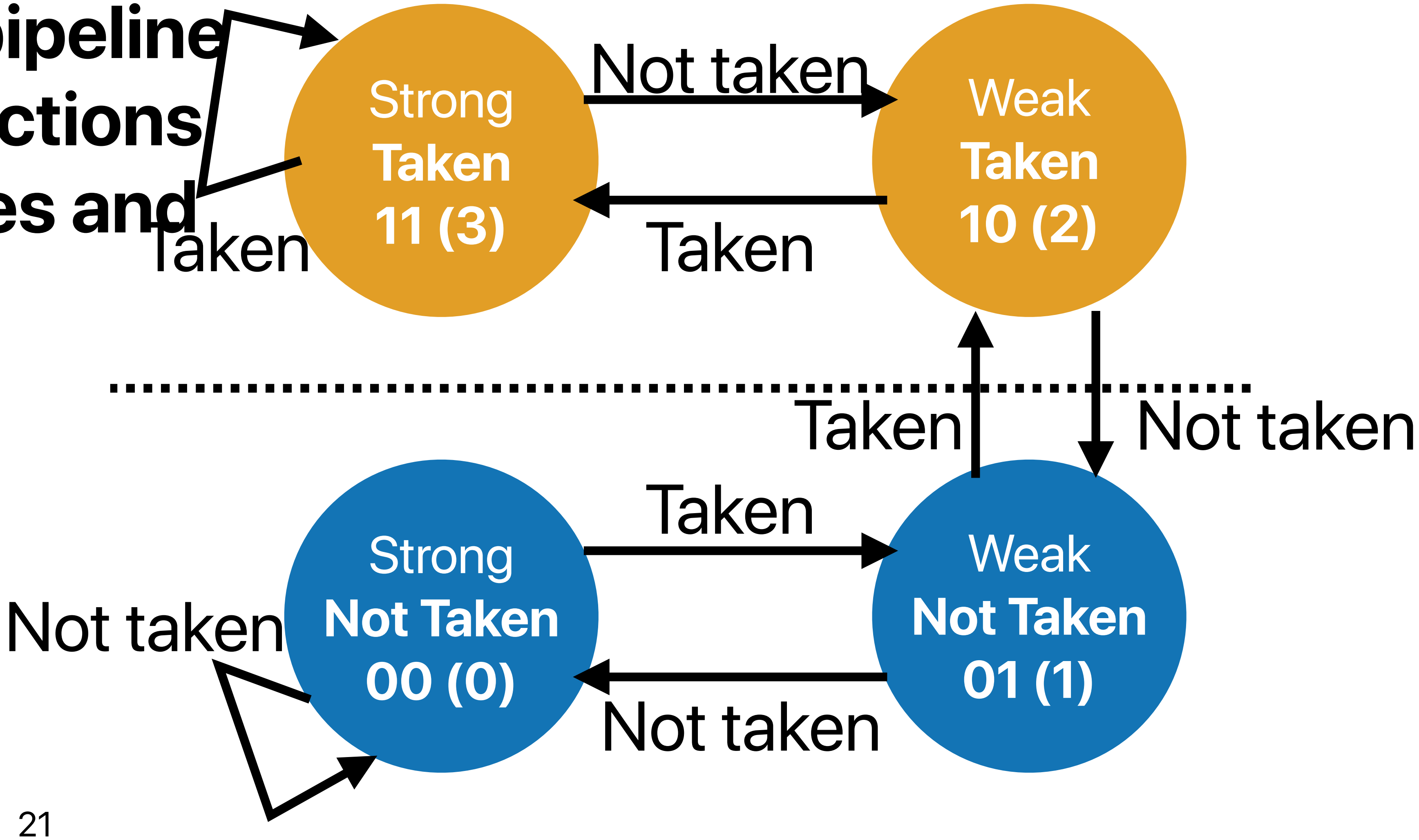
Detail of a basic dynamic branch predictor



2-bit/Bimodal local predictor

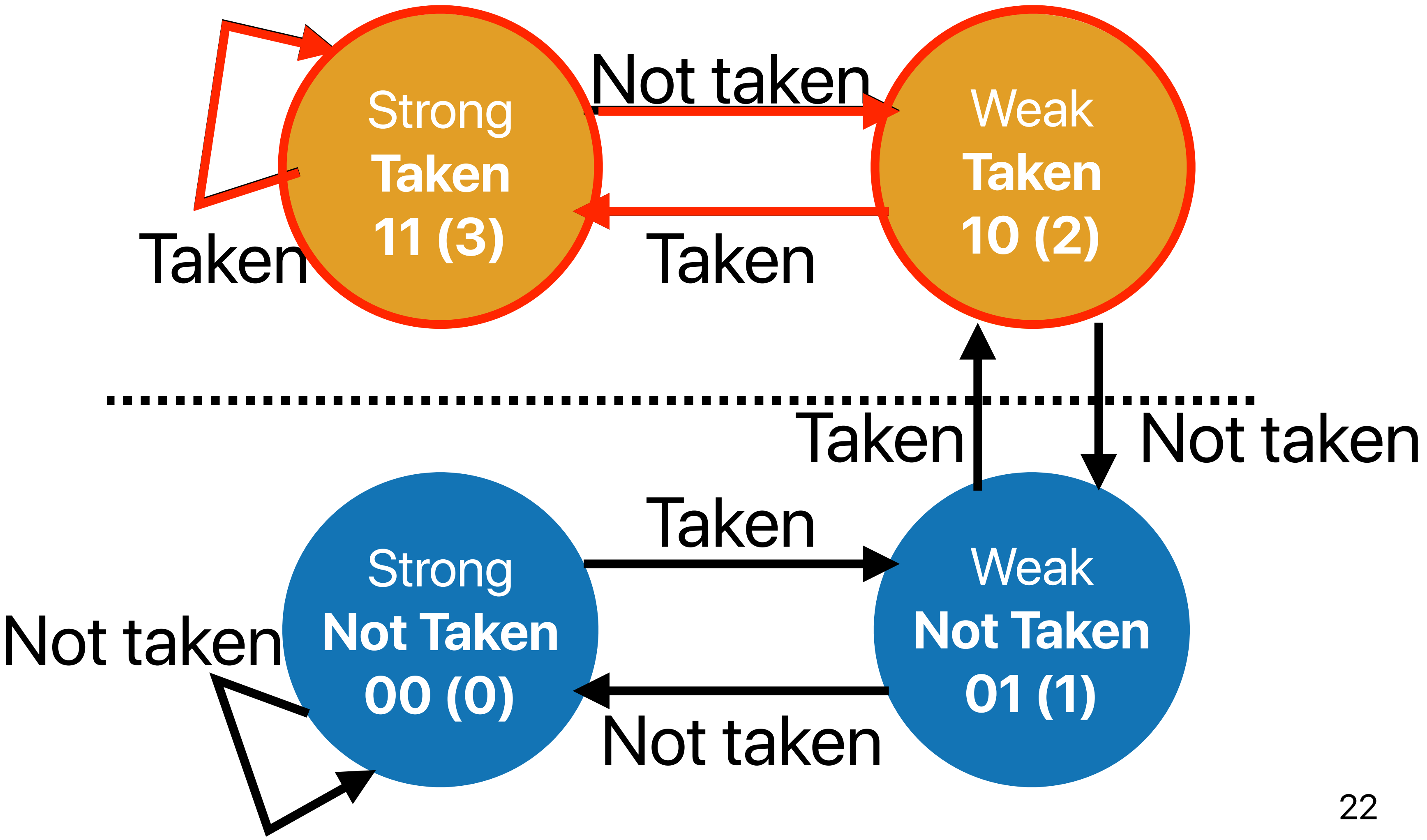
- Local predictor — every branch instruction has its own state
- 2-bit — each state is described using 2 bits
- Change the state based on **actual** outcome
- If we guess right — no penalty
- **If we guess wrong — flush (clear pipeline registers) for mis-predicted instructions that are currently in IF and ID stages and reset the PC**

	branch PC	target PC	State
	0x400048	0x400032	10
Predict Taken	0x400080	0x400068	11
	0x401080	0x401100	00
	0x4000F8	0x400100	01



2-bit local predictor

```
i = 0;
do {
      sum += a[i];
} while(++i < 10);
```



i	state	predict	actual
1	10	T	T
2	11	T	T
3	11	T	T
4-9	11	T	T
10	11	T	NT

90% accuracy!

https://www.pollev.com/hungweitseng close in 1:30

Demo revisited

- Assume that we have a 2-bit local predictor and the values in data is randomly distributed in the number space, what's the branch prediction accuracy of branch X when option is "0" and "1". You may also assume the predictors' states start with 0s.

	Without sorting	After sorting
A	100%	0%
B	50%	0%
C	50%	50%
D	50%	100%
E	0%	100%

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)  // Branch X
            sum ++;
    }
}
```

Demo revisited

- Assume that we have a 2-bit local predictor and the values in data is randomly distributed in the number space, what's the branch prediction accuracy of branch X when option is "0" and "1". You may also assume the predictors' states start with 0s.

	Without sorting	After sorting
A	100%	0%
B	50%	0%
C	50%	50%
D	50%	100%
E	0%	100%

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)  // Branch X
            sum ++;
    }
}
```

Demo revisited

- Assume that we have a 2-bit local predictor and the values in data is randomly distributed in the number space, what's the branch prediction accuracy of branch X when option is "0" and "1". You may also assume the predictors' states start with 0s.

	Without sorting	After sorting
A	100%	0%
B	50%	0%
C	50%	50%
D	50%	100%
E	0%	100%

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)  // Branch X
```

	Without sorting	With sorting
The prediction accuracy of X before threshold	50%	100%
The prediction accuracy of X after threshold	50%	100%

Demo revisited

If there is no branch predictor on the processor, the code w/ sorting will be slower — but every processor has branch predictors now

	Without sorting	After sorting
A	100%	0%
B	50%	0%
C	50%	50%
D	50%	100%
E	0%	100%

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)  // Branch X
```

	Without sorting	With sorting
The prediction accuracy of X before threshold	50%	100%
The prediction accuracy of X after threshold	50%	100%

How can we evaluate the cost of mis-predicted branches?

- Compare the number of mis-predictions
- Calculate the difference of cycles
- We can get the “average CPI” of a mis-prediction!

Demo revisited: evaluating the cost of mis-predicted branches

- Compare the number of mis-predictions
- Calculate the difference of cycles
- We can get the “average CPI” of a mis-prediction!

34 cycles on Intel Alder Lake

23 cycles on AMD Zen 3

Could be more expensive than cache misses

https://www.pollev.com/hungweitseng close in 1:30

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

(assume all states started with 00)

A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
        a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

Can we do a better job?

(assume all states started with 00)

A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

For branch Y, almost 100%,
For branch X, only 50%

i	branch?	state	prediction	actual
0	X	00	NT	T
1	Y	00	NT	T
1	X	01	NT	NT
2	Y	01	NT	T
2	X	00	NT	T
3	Y	10	T	T
3	X	01	NT	NT
4	Y	11	T	T
4	X	00	NT	T
5	Y	11	T	T
5	X	01	NT	NT
6	Y	11	T	T
6	X	00	NT	T
7	Y	11	T	T

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved — 34 cycles on modern intel processors and 23 on AMD processors
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict based on prior history
  - Local predictor — make prediction based on the state of each branch instruction

Two-level global predictor

Marius Evers, Sanjay J. Patel, Robert S. Chappell, and Yale N. Patt. 1998. An analysis of correlation and predictability: what makes two-level branch predictors work. In Proceedings of the 25th annual international symposium on Computer architecture (ISCA '98).

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
        a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

(assume all states started with 00)

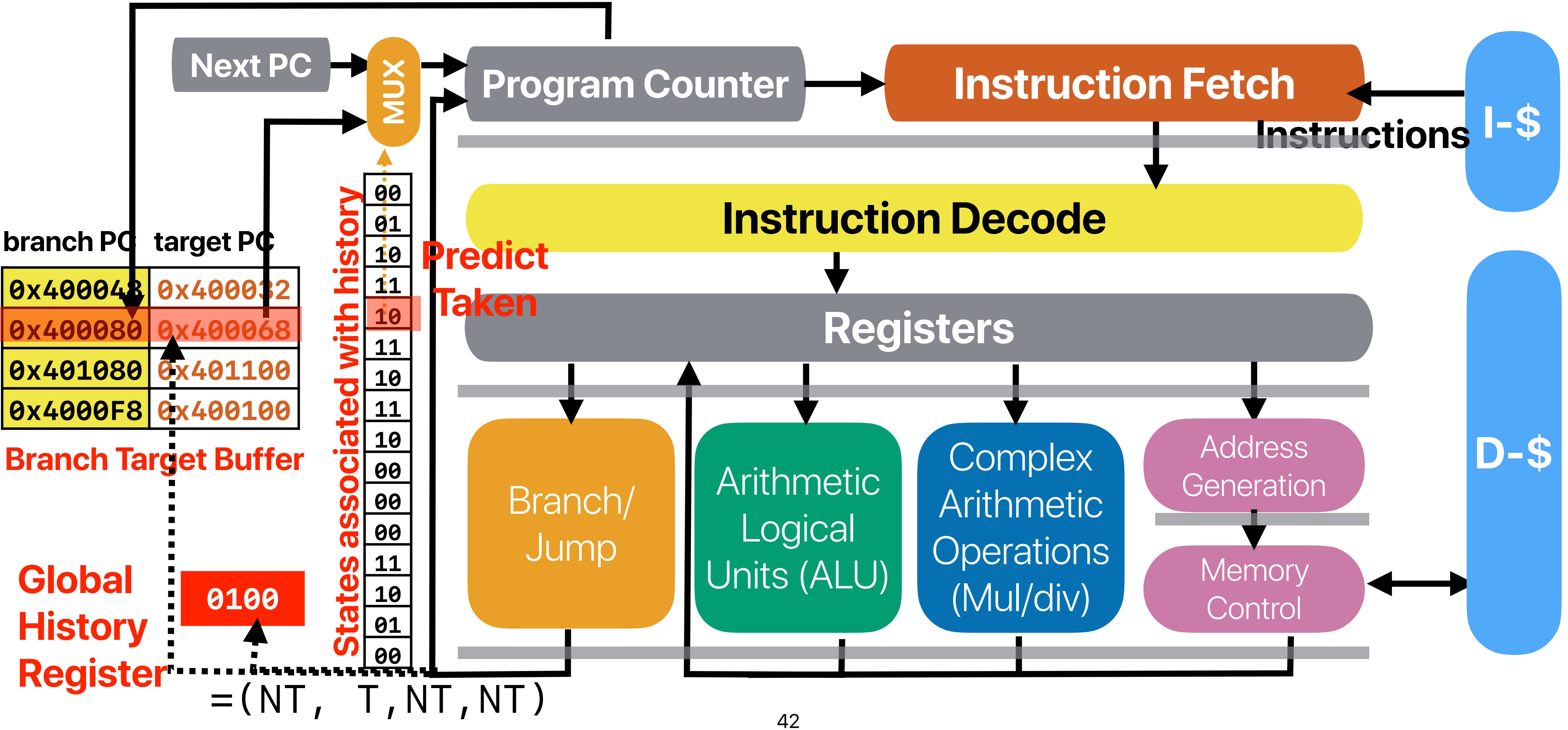
A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

i	branch?	state	prediction	actual
0	X	00	NT	T
1	Y	00	NT	T
1	X	01	NT	NT
2	Y	01	NT	T
2	X	00	NT	T
3	Y	10	[illegible]	T
3	X	01	NT	NT
4	Y	11	T	T
4	X	00	NT	T
5	Y	11	T	T
5	X	01	NT	NT
6	Y	11	T	T
6	X	00	NT	T
7	Y	11	T	T

This pattern repeats all the time!

For branch Y, almost 100%,
For branch X, only 50%

Detail of a basic dynamic branch predictor



Performance of GH predictor

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
        a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

i	branch?	GHR	state	prediction	actual
0	X	000	00	NT	T
1	Y	001	00	NT	T
1	X	011	00	NT	NT
2	Y	110	00	NT	T
2	X	101	00	NT	T
3	Y	011	00	NT	T
3	X	111	00	NT	NT
4	Y	110	01	NT	T
4	X	101	01	NT	T
5	Y	011	01	NT	T
5	X	111	00	NT	NT
6	Y	110	10	T	T
6	X	101	10	T	T
7	Y	011	10	T	T
7	X	111	00	NT	NT
8	Y	110	11	T	T
8	X	101	11	T	T
9	Y	011	11	T	T
9	X	111	00	NT	NT
10	Y	110	11	T	T
10	X	101	11	T	T
11	Y	011	11	T	T

Near perfect after this

https://www.pollev.com/hungweitseng close in 1:30

Better predictor?

- Consider two predictors — (L) 2-bit local predictor with unlimited BTB entries and (G) 4-bit global history with 2-bit predictors. How many of the following code snippet would allow (G) to outperform (L)?

I

```
i = 0;
do {
    if( i % 10 != 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100);
```

II

```
i = 0;
do {
    a[i] += i;
} while ( ++i < 100);
```

III

```
i = 0;
do {
    j = 0;
    do {
      sum += A[i*2+j];
    }
    while( ++j < 2);
} while ( ++i < 100);
```

IV

```
i = 0;
do {
    if( rand() %2 == 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)
```

A. 0
B. 1
C. 2
D. 3
E. 4

Better predictor?

- Consider two predictors — (L) 2-bit local predictor with unlimited BTB entries and (G) 4-bit global history with 2-bit predictors. How many of the following code snippet would allow (G) to outperform (L)?

I

about the same

```
i = 0;
do {
    if( i % 10 != 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100);
```

II

about the same

```
i = 0;
do {
    a[i] += i;
} while ( ++i < 100);
```

III

```
i = 0;
do {
    j = 0;
    do {
      sum += A[i*2+j];
    }
    while( ++j < 2);
} while ( ++i < 100);
```

IV

L could be better

```
i = 0;
do {
    if( rand() %2 == 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)
```

A. 0

B. 1

C. 2

D. 3

E. 4

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved — 34 cycles on modern intel processors and 23 on AMD processors
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict based on prior history
  - Local predictor — make predictions based on the state of each branch instruction
  - Global predictor — make predictions based on the state from all branches
  - Both are not perfect

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved — 34 cycles on modern intel processors and 23 on AMD processors
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict based on prior history
    - Local predictor — make predictions based on the state of each branch instruction
    - Global predictor — make predictions based on the state from all branches
    - Both are not perfect — hybrid predictors
        - Tournament
        - Perceptron
    - All modern processors have branch predictors!

Announcements

- Assignment #3 — due this Saturday
- Reading quiz #6 due next Monday
- Assignment #4 will be up on Sunday and we have covered the material this week so you should start immediately after we released that

Computer
Science &
Engineering

142

つづく