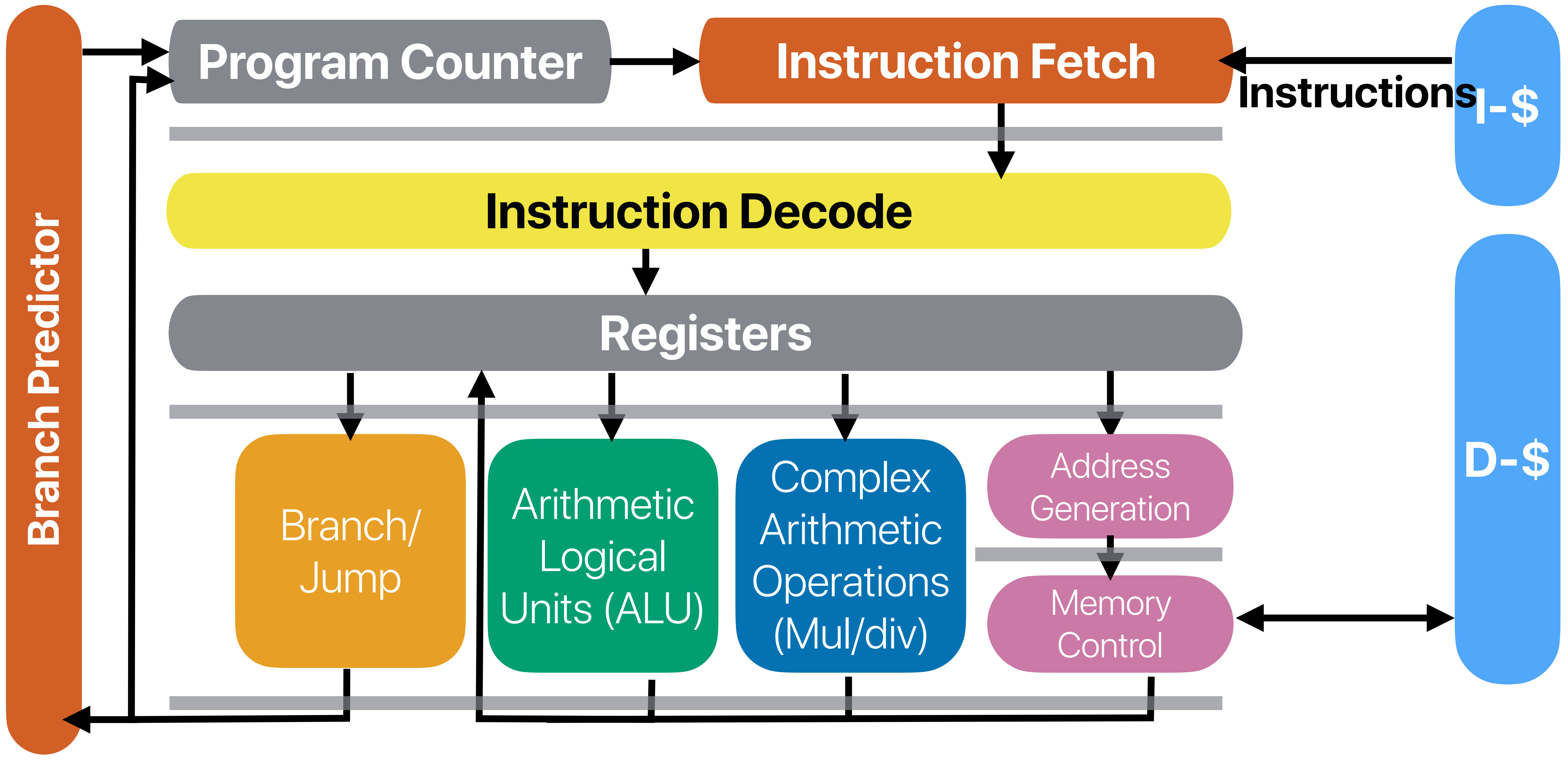
Modern Processor Design (III): Whenever You're Ready

Hung-Wei Tseng

Recap: Microprocessor with a “branch predictor”



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

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

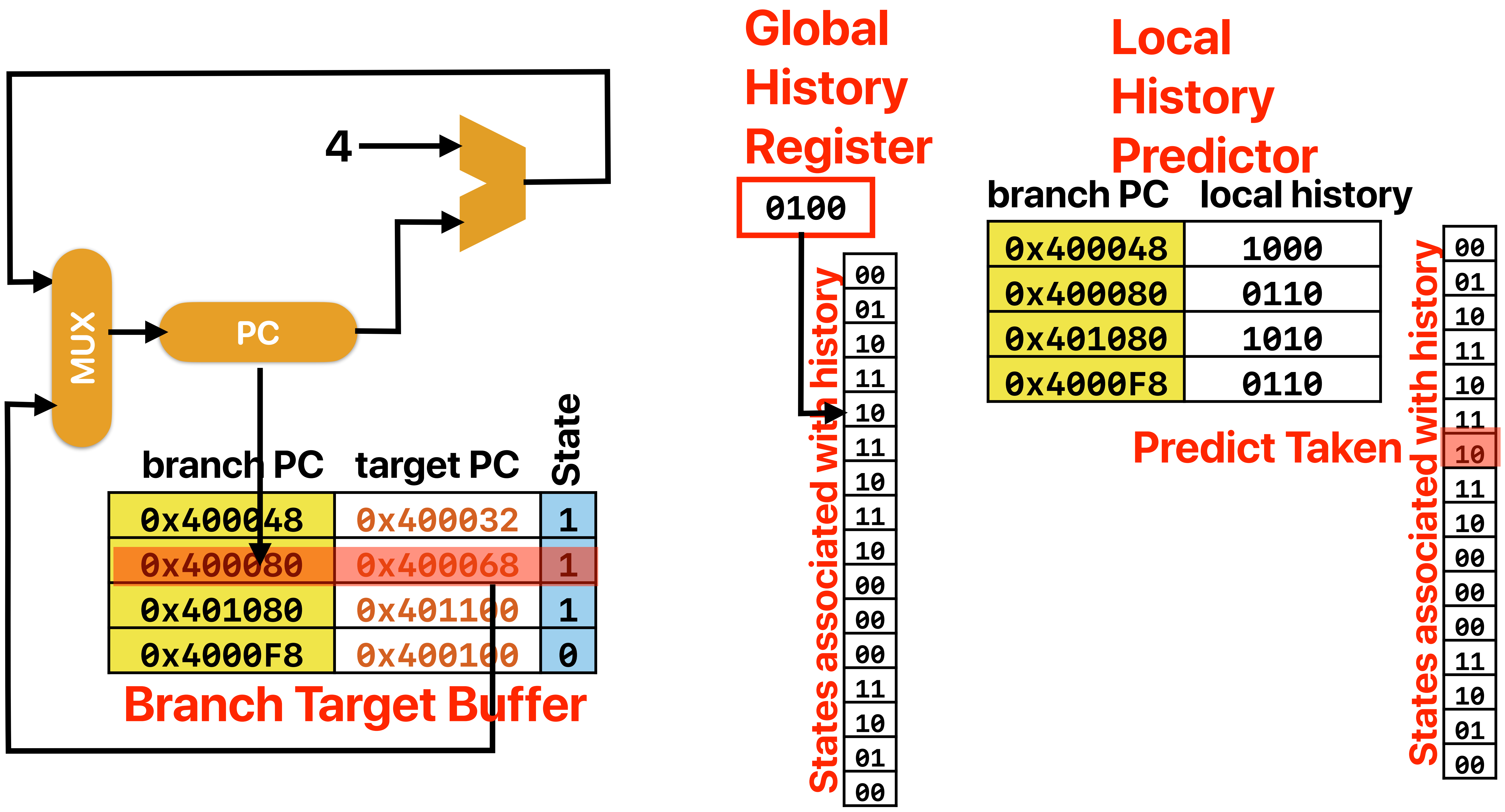
- The cost of not to predict a branch is to stall until the data dependency is resolved — 34 cycles on modern intel processors and 23 on AMD processors
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict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

Outline

- Hybrid predictors (cont.)
- Data hazards
- Hardware optimizations for data hazards

Hybrid predictors

Tournament Predictor


4
MUX
PC
States associated with history
States associated with history
Predict Taken

Branch Target Buffer

branch PC	target PC	State
0x400048	0x400032	1
0x400080	0x400068	1
0x401080	0x401100	1
0x4000F8	0x400100	0

Global History Register

```
0100
```

States associated with history
00
01
10
11
10
11
10
11
10
00
00
00
11
10
01
00

Local History Predictor

branch PC	local history
0x400048	1000
0x400080	0110
0x401080	1010
0x4000F8	0110

States associated with history
00
01
10
11
10
11
10
11
10
00
00
00
11
10
01
00

Tournament Predictor

- The state predicts “which predictor is better”
  - Local history
  - Global history
- The predicted predictor makes the prediction
- Tournament predictor is a “hybrid predictor” as it takes both local & global information into account

Perceptron

Jiménez, Daniel, and Calvin Lin. "Dynamic branch prediction with perceptrons." Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. IEEE, 2001.
The following slides are excerpted from https://www.jilp.org/cbp/Daniel-slides.PDF by Daniel Jiménez

Branch Prediction is Essentially an ML Problem

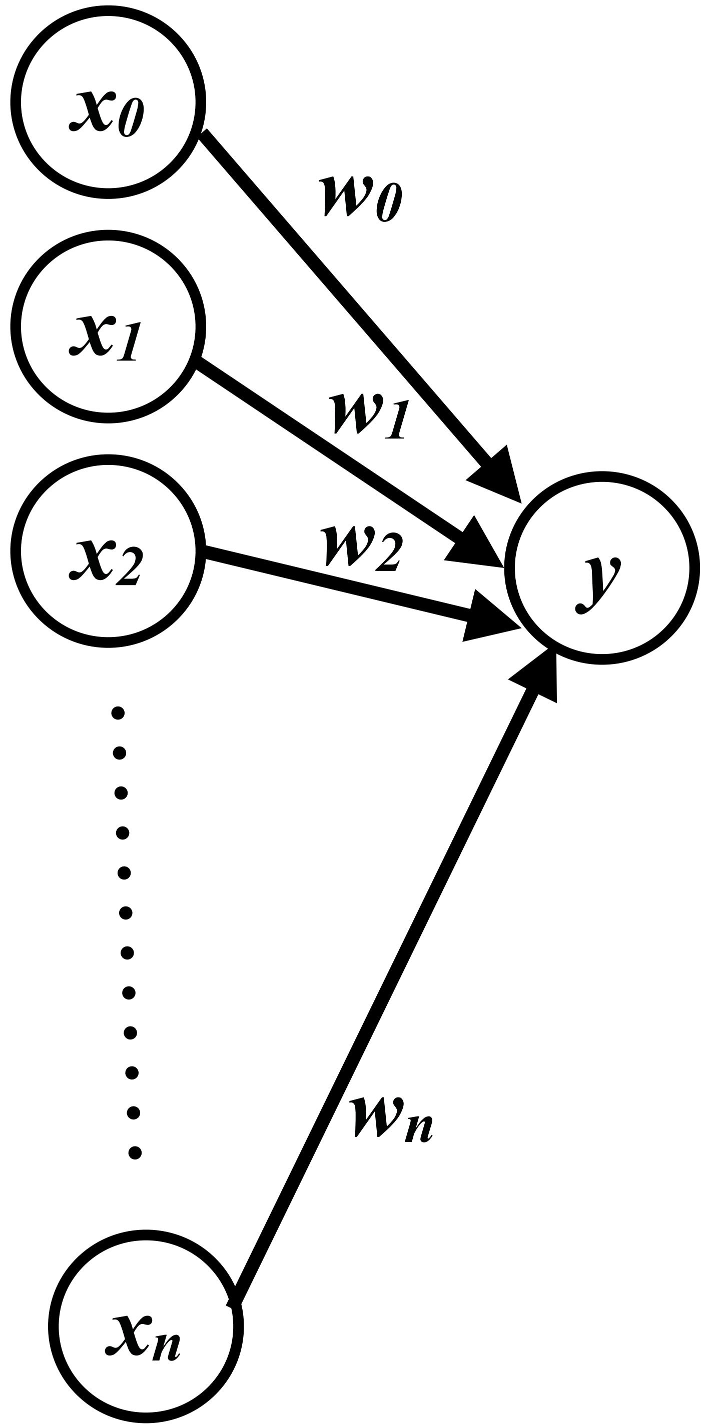
- The machine learns to predict conditional branches
- Artificial neural networks
    - Simple model of neural networks in brain cells
    - Learn to recognize and classify patterns

Mapping Branch Prediction to NN

- The inputs to the perceptron are branch outcome histories
  - Just like in 2-level adaptive branch prediction
  - Can be global or local (per-branch) or both (alloyed)
  - Conceptually, branch outcomes are represented as
    - +1, for taken
    - -1, for not taken
- The output of the perceptron is
  - Non-negative, if the branch is predicted taken
  - Negative, if the branch is predicted not taken
- Ideally, each static branch is allocated its own perceptron

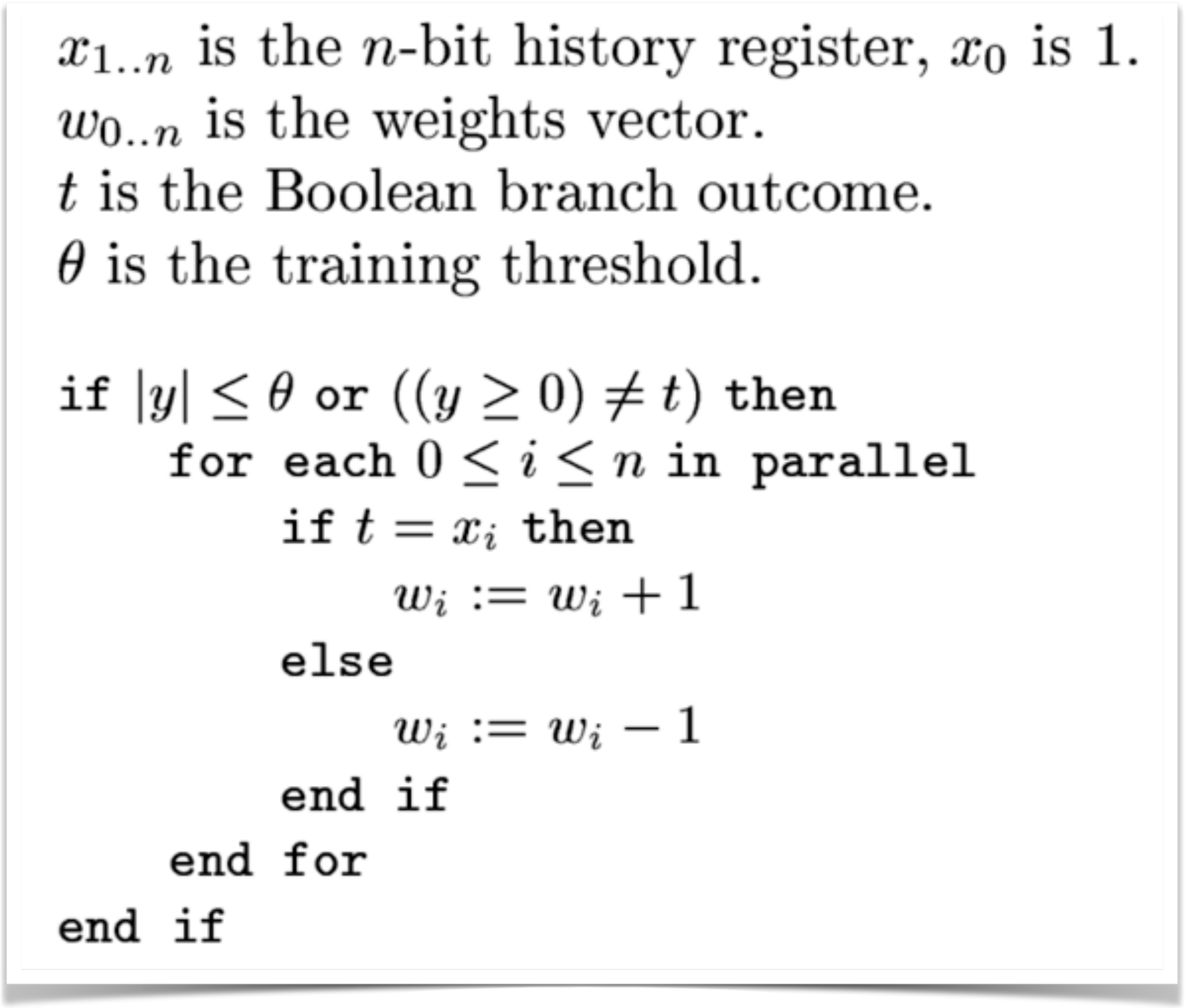
Mapping Branch Prediction to NN (cont.)

- Inputs (x's) are from branch history and are -1 or +1
- n + 1 small integer weights (w's) learned by on-line training
- Output (y) is dot product of x's and w's; predict taken if y = 0
- Training finds correlations between history and outcome



$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

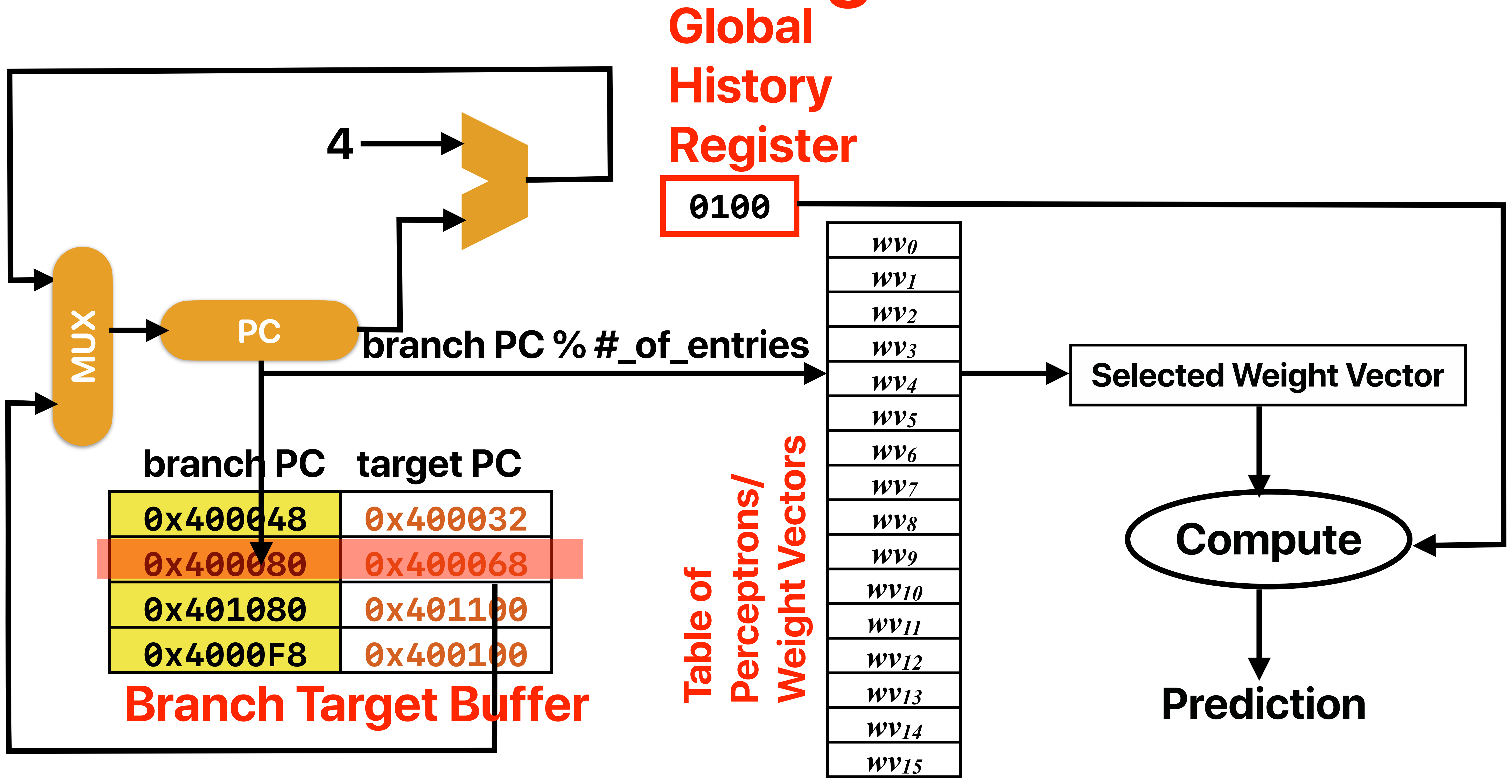
Training Algorithm



$x_{1..n}$ is the n-bit history register, x_0 is 1.
$w_{0..n}$ is the weights vector.
t is the Boolean branch outcome.
θ is the training threshold.

```
if $|y| \leq \theta$ or $((y \geq 0) \neq t)$ then
    for each $0 \leq i \leq n$ in parallel
        if $t = x_i$ then
            $w_i := w_i + 1$
        else
            $w_i := w_i - 1$
        end if
    end for
end if
```

Predictor Organization



branch PC	target PC
0x400048	0x400032
0x400080	0x400068
0x401080	0x401100
0x4000F8	0x400100

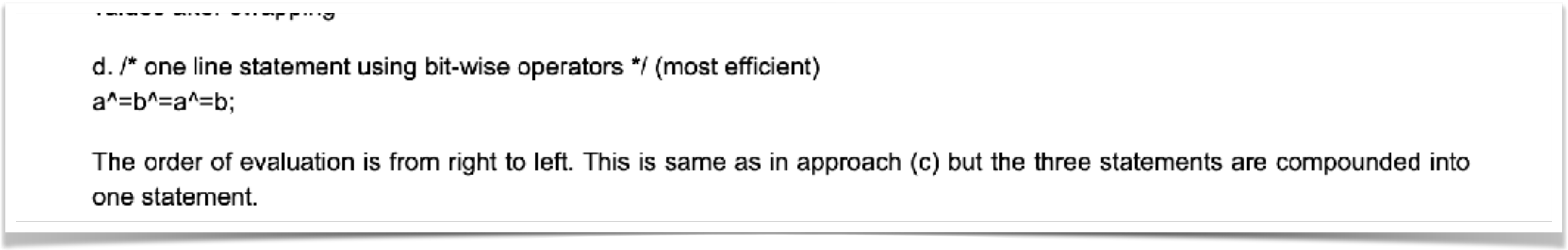
Branch predictors in processors

- The Intel Pentium MMX, Pentium II, and Pentium III have local branch predictors with a local 4-bit history and a local pattern history table with 16 entries for each conditional jump.
- Global branch prediction is used in Intel Pentium M, Core, Core 2, and Silvermont-based Atom processors.
- Tournament predictor is used in DEC Alpha, AMD Athlon processors
- The AMD Ryzen multi-core processor's Infinity Fabric and the Samsung Exynos processor include a perceptron based neural branch predictor.

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 pretty accurate branch predictors — if the code itself is predictable

Recap: But A is faster!


values after swapping

d. /* one line statement using bit-wise operators */ (most efficient)
a^=b^=a^=b;

The order of evaluation is from right to left. This is same as in approach (c) but the three statements are compounded into one statement.

A

```
void regswap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

B

```
void xorswap(int* a, int* b) {
    *a ^= *b = *a = *b;
}
```

Data hazards

Data hazards

- An instruction currently in the pipeline cannot receive the "logically" correct value for execution
- Data dependencies
  - The output of an instruction is the input of a later instruction
  - **May sometimes** result in data hazard if the later instruction that consumes the result is still in the pipeline

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

How many data dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?

```
movl     (%rdi), %eax
movl     (%rsi), %edx
movl     %edx, (%rdi)
movl     %eax, (%rsi)
```

```
int temp = *a;
*a = *b;
*b = temp;
```

A. 1
B. 2
C. 3
D. 4
E. 5

How many dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?

```
movl    (%rdi), %eax
movl    (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```

```c
int temp = *a;
*a = *b;
*b = temp;
```

A. 1
[✓ B. 2]
C. 3
D. 4
E. 5

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

How many data dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?

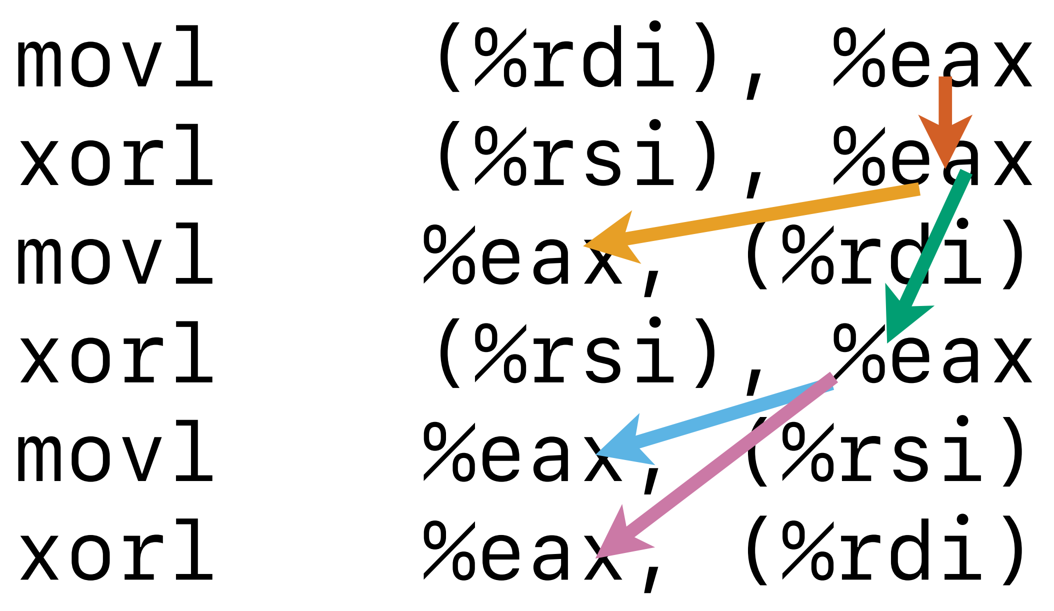
```
movl    (%rdi), %eax
xorl    (%rsi), %eax
movl    %eax, (%rdi)
xorl    (%rsi), %eax
movl    %eax, (%rsi)
xorl    %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

A. 1
B. 2
C. 3
D. 4
E. 5

How many dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?



```
movl    (%rdi), %eax
xorl    (%rsi), %eax
movl    %eax, (%rdi)
xorl    (%rsi), %eax
movl    %eax, (%rsi)
xorl    %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

A. 1
B. 2
C. 3
D. 4
[✓ E. 5]

Data hazards

```
① movl     (%rdi), %eax
② movl     (%rsi), %edx
③ movl    %edx, (%rdi)
④ movl    %eax, (%rsi)
```

	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(4)	(3)	(2)	(1)				
5		(4)	(3)	(2)	(1)			
6			(4)	(3)	(2)	(1)		
7				(4)	(3)	(2)	(1)	
8					(4)	(3)	(2)	
9						(4)	(3)	
10							(4)	
11								
12								
13								
14								

%edx does not have our desired value

%eax does not have our desired value

Solution 1: Let’s try “stall” again

- Whenever the input is not ready when the consumer is decoding, just stall — the consumer stays at ID.

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

Data hazards?

- How many cycles do we have to stall in the following x86 instructions to get the expected output if a memory operation (assume 100% cache hit rate) takes 5 cycles?

```
① movl     (%rdi), %eax
② movl     (%rsi), %edx
③ movl    %edx, (%rdi)
④ movl    %eax, (%rsi)
```

A. 1
B. 2
C. 3
D. 4
E. 5

Data hazards?

- How many cycles do we have to stall in the following x86 instructions to get the expected output if a memory operation (assume 100% cache hit rate) takes 5 cycles?

```
① movl    (%rdi), %eax
② movl    (%rsi), %edx
③ movl    %edx, (%rdi)
④ movl    %eax, (%rsi)
```

A. 1
B. 2
C. 3
D. 4
[✓ E. 5]

	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(4)	(3)	(2)	(1)				
5	(4)	(3)		(2)	(1)			
6	(4)	(3)			(2)	(1)		
7	(4)	(3)				(2)	(1)	
8	(4)	(3)					(2)	(1)
9	(4)	(3)						(2)
10		(4)	(3)					
11			(4)	(3)				
12				(4)	(3)			
13					(4)	(3)		
14						(4)		(3)
15								(4)

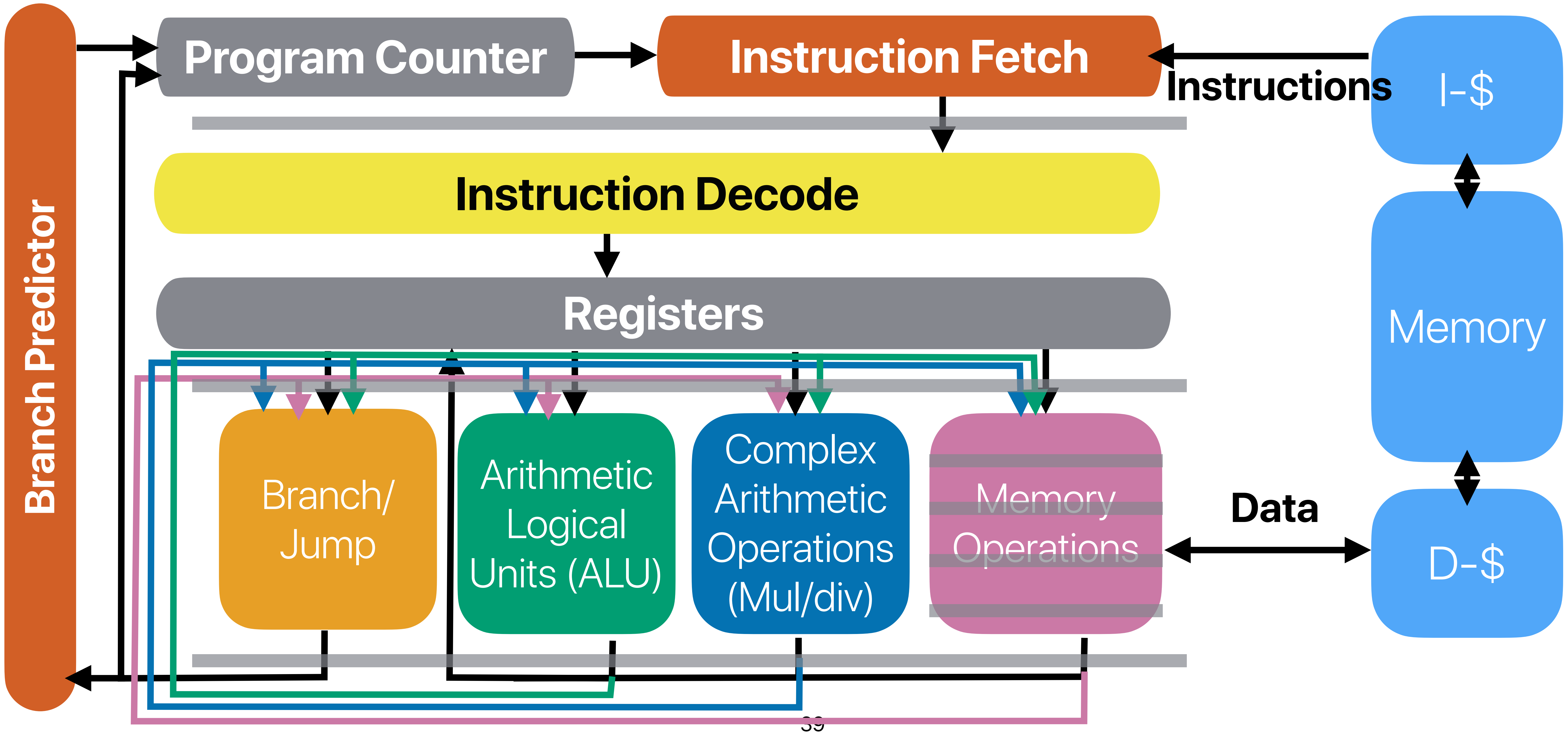
5 cycles stalls

we have the value for %edx already!
Why another cycle?

Solution 2: Data forwarding

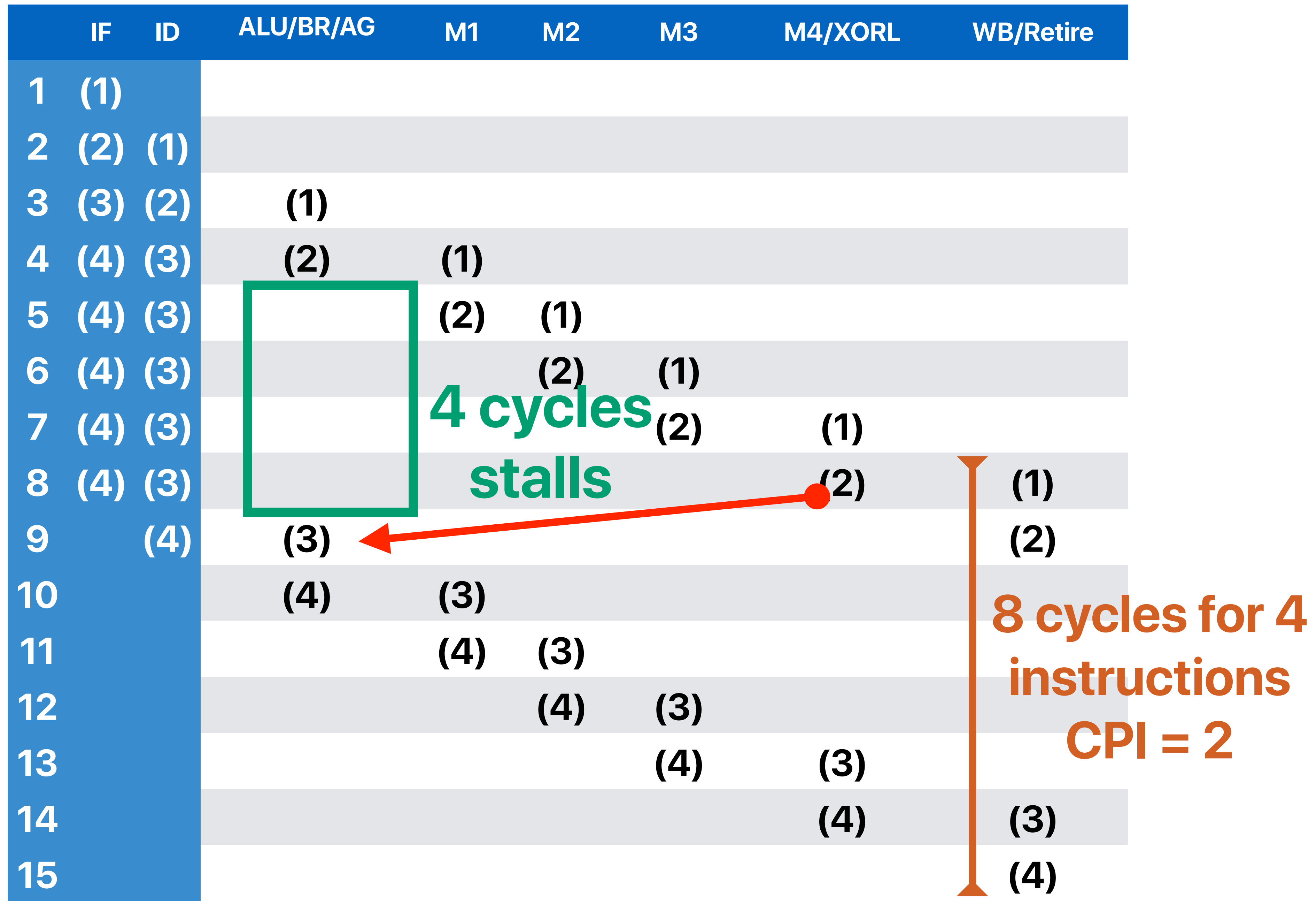
- Add logics/wires to forward the desired values to the demanding instructions

Data “forwarding”



The effect of data forwarding

```
① movl     (%rdi), %eax
② movl     (%rsi), %edx
③ movl     %edx, (%rdi)
④ movl     %eax, (%rsi)
```



	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(4)	(3)	(2)	(1)				
5	(4)	(3)		(2)	(1)			
6	(4)	(3)			(2)	(1)		
7	(4)	(3)				(2)	(1)	
8	(4)	(3)					(2)	(1)
9		(4)	(3)					(2)
10			(4)	(3)				
11				(4)	(3)			
12					(4)	(3)		
13						(4)	(3)	
14							(4)	(3)
15								(4)

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

How many of data hazards w/ Data Forwarding?

- How many pairs of back-to-back data dependences in the following x86 instructions will result in stalls even with data forwarding and both memory operations & xorl take 5 cycles?

```
① movl     (%rdi), %eax
② xorl     (%rsi), %eax
③ movl     %eax, (%rdi)
④ xorl     (%rsi), %eax
⑤ movl     %eax, (%rsi)
⑥ xorl     %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

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

How many of data hazards w/ Data Forwarding?

- How many pairs of back-to-back data dependences in the following x86 instructions will result in stalls even with data forwarding and both memory operations & xorl take 5 cycles?

```
① movl    (%rdi), %eax
② xorl    (%rsi), %eax
③ movl    %eax, (%rdi)
④ xorl    (%rsi), %eax
⑤ movl    %eax, (%rsi)
⑥ xorl    %eax, (%rdi)
```

A. 0
B. 1
C. 2
[✓ D. 3]
E. 4

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(3)	(2)		(1)				
5	(3)	(2)			(1)			
6	(3)	(2)				(1)		
7	(3)	(2)					(1)	
8	(4)	(3)	(2)					(1)
9	(4)	(3)		(2)				
10	(4)	(3)			(2)			
11	(4)	(3)				(2)		
12	(4)	(3)					(2)	
13	(5)	(4)	(3)					(2)
14	(6)	(5)	(4)	(3)				
15	(6)	(5)		(4)	(3)			
16	(6)	(5)			(4)	(3)		
17	(6)	(5)				(4)		
18	(6)	(5)					(3)	
19	(6)	(5)					(4)	(3)
20		(6)	(5)					(4)
21			(6)	(5)				
22				(6)	(5)			
23					(6)	(5)		
24						(6)	(5)	
25							(6)	(5)
26								(6)

19 cycles for 6 instructions
CPI = 3.17!

Takeaways: data hazards

- More data dependencies, more likelihood of data hazards
- Stalls and data forwarding can both address data hazards to generate correct code execution results — but not very efficient

Let's extend the example a bit...

```
for(i = 0; i < count; i++) {
    int64_t temp = a[i];
    a[i] = b[i];
    b[i] = temp;
} .L9:
  ①  movq    (%rdi,%rax), %rsi
  ②  movq    (%rcx,%rax), %r8
  ③  movq    %r8, (%rdi,%rax)
  ④  movq    %rsi, (%rcx,%rax)
  ⑤  addq    $8, %rax
  ⑥  cmpq    %r9, %rax
  ⑦  jne     .L9
  ⑧  movq    (%rdi,%rax), %rsi
  ⑨  movq    (%rcx,%rax), %r8
  ⑩  movq    %r8, (%rdi,%rax)
  ⑪  movq    %rsi, (%rcx,%rax)
  ⑫  addq    $8, %rax
  ⑬  cmpq    %r9, %rax
  ⑭  jne     .L9
```

	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(4)	(3)	(2)	(1)				
5	(4)	(3)		(2)	(1)			
6	(4)	(3)			(2)	(1)		
7	(4)	(3)				(2)	(1)	
8	(4)	(3)					(2)	(1)
9	(5)	(4)	(3)					(2)
10	(6)	(5)	(4)	(3)				
11	(7)	(6)	(5)	(4)	(3)			
12	(8)	(7)	(6)		(4)	(3)		
13	(9)	(8)	(7)			(4)	(3)	
14	(10)	(9)	(8)				(4)	(3)
15	(11)	(10)	(9)	(8)				(4)
16	(11)	(10)		(9)	(8)			(5)
17	(11)	(10)			(9)	(8)		(6)
18	(11)	(10)				(9)	(8)	(7)
19	(11)	(10)					(9)	(8)
20	(12)	(11)	(10)					(9)
21	(13)	(12)	(11)	(10)				
22	(14)	(13)	(12)	(11)	(10)			
23		(14)	(13)	(12)	(11)	(10)		
24			(14)	(13)	(12)	(11)	(10)	

11 cycles for 7 instructions

CPI = 1.57

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

The effect of code optimization

- By reordering which pair of the following instruction stream can we reduce stalls without affecting the correctness of the code?

```
① movq    (%rdi,%rax), %rsi
② movq    (%rcx,%rax), %r8
③ movq    %r8, (%rdi,%rax)
④ movq    %rsi, (%rcx,%rax)
⑤ addq    $8, %rax
⑥ cmpq    %r9, %rax
⑦ jne     .L9
```

A. (1) & (2)

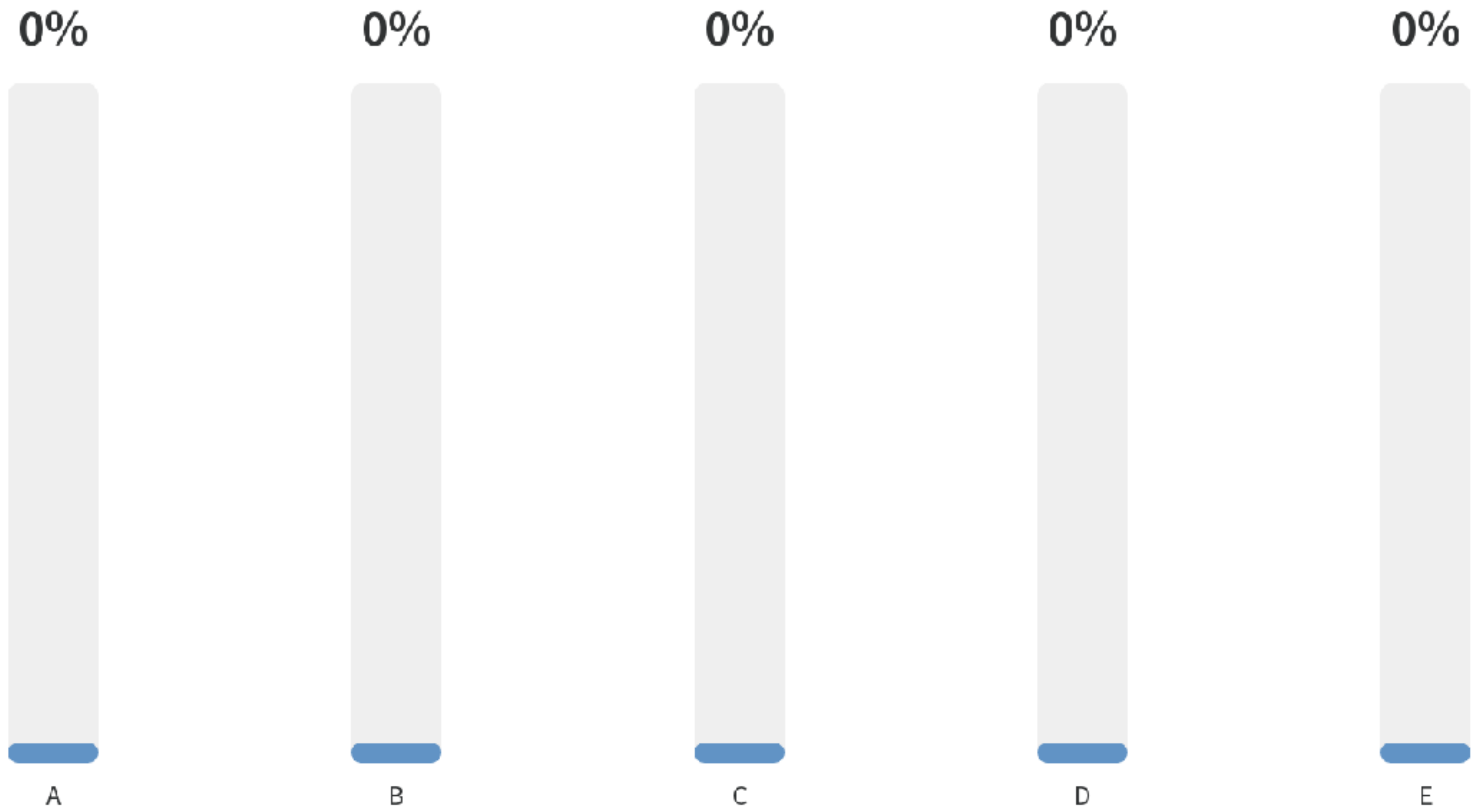
B. (2) & (3)

C. (3) & (5)

D. (4) & (6)

E. No ordering can help reduce the stalls

0



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

The effect of code optimization

- By reordering which pair of the following instruction stream can we reduce stalls without affecting the correctness of the code?

```
① movq    (%rdi,%rax), %rsi
② movq    (%rcx,%rax), %r8
③ movq    %r8, (%rdi,%rax)
④ movq    %rsi, (%rcx,%rax)
⑤ addq    $8, %rax
⑥ cmpq    %r9, %rax
⑦ jne     .L9
```

A. (1) & (2)

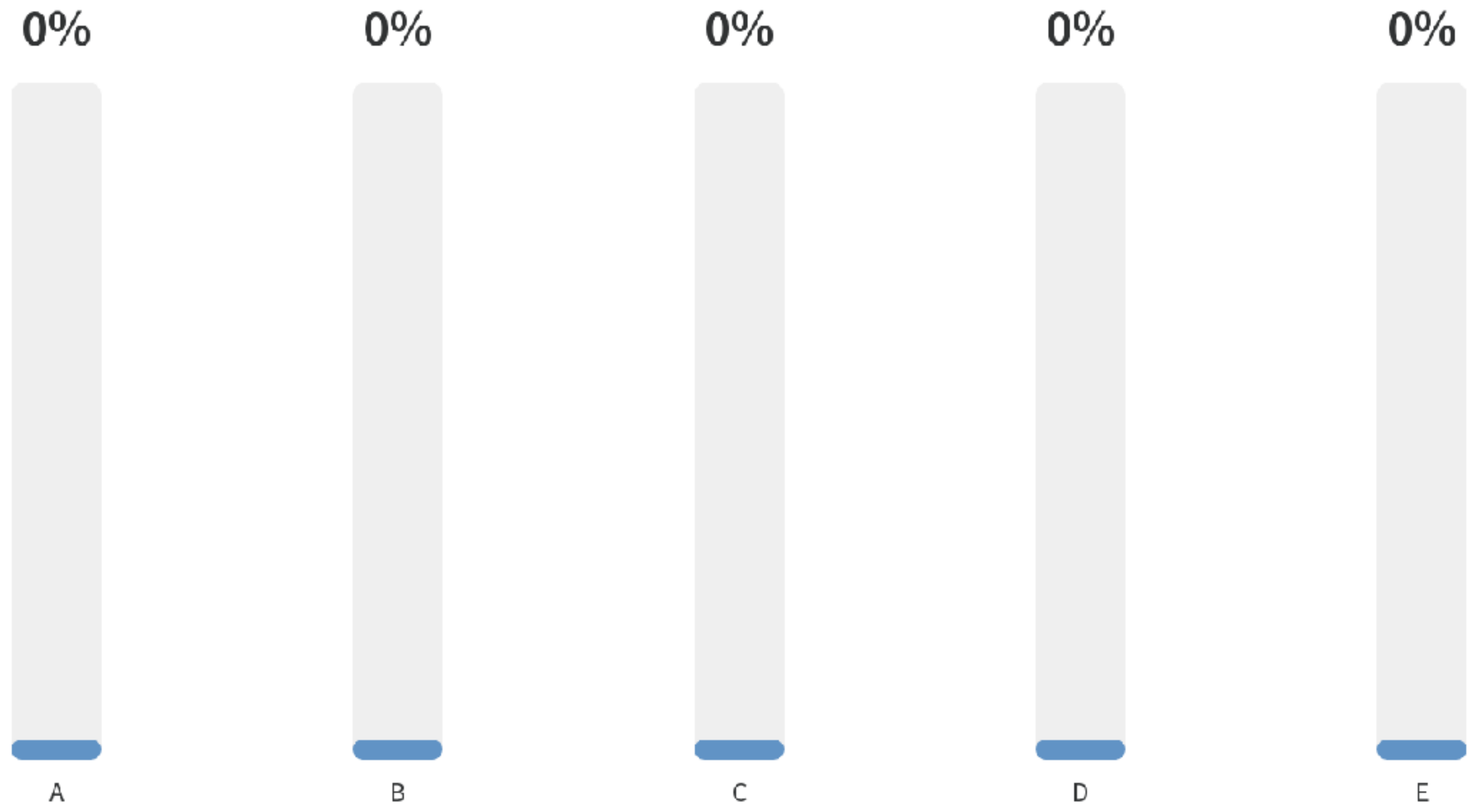
B. (2) & (3)

C. (3) & (5)

D. (4) & (6)

E. No ordering can help reduce the stalls

0



The effect of code optimization

- By reordering which pair of the following instruction stream can we reduce stalls without affecting the correctness of the code?

```
① movq    (%rdi,%rax), %rsi
② movq    (%rcx,%rax), %r8
③ movq    %r8, (%rdi,%rax)
④ movq    %rsi, (%rcx,%rax)
⑤ addq    $8, %rax
⑥ cmpq    %r9, %rax
⑦ jne     .L9
```

[✓ A. (1) & (2)]

B. (2) & (3)

C. (3) & (5)

D. (4) & (6)

E. No ordering can help reduce the stalls

Compiler optimization

```
for(i = 0; i < count; i++) {
    int64_t temp = a[i];
    a[i] = b[i];
    b[i] = temp;
}
```

```
movq     (%rdi,%rax), %rsi
movq     (%rcx,%rax), %r8
movq     %r8, (%rdi,%rax)
movq     %rsi, (%rcx,%rax)

cmpq     %r9, %rax
jne      .L9
movq     (%rdi,%rax), %rsi
movq     (%rcx,%rax), %r8
movq     %r8, (%rdi,%rax)
movq     %rsi, (%rcx,%rax)
addq     $8, %rax
cmpq     %r9, %rax
jne      .L9
```

```
.L9:
①   movq     (%rcx,%rax), %r8
②   movq     (%rdi,%rax), %rsi
③   addq     $8, %rax
④   movq     %r8, -8(%rdi,%rax)
⑤   movq     %rsi, -8(%rcx,%rax
⑥   cmpq     %r9, %rax
⑦   jne      .L9
⑧   movq     (%rcx,%rax), %r8
⑨   movq     (%rdi,%rax), %rsi
⑩   addq     $8, %rax
⑪   movq     %r8, -8(%rdi,%rax)
⑫   movq     %rsi, -8(%rcx,%rax
⑬   cmpq     %r9, %rax
⑭   jne      .L9
```

	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(4)	(3)	(2)	(1)				
5	(5)	(4)	(3)	(2)	(1)			
6	(5)	(4)			(2)	(1)		
7	(5)	(4)				(2)	(1)	
8	(6)	(5)	(4)				(2)	(1)
9	(7)	(6)	(5)	(4)				(2)
10	(8)	(7)	(6)	(5)	(4)			(3)
11	(9)	(8)	(7)		(5)	(4)		
12	(10)	(9)	(8)			(5)	(4)	
13	(11)	(10)	(9)	(8)			(5)	(4)
14	(11)	(10)	(10)	(9)	(8)			(5)
15	(11)	(10)			(9)	(8)		(6)
16	(11)	(10)				(9)	(8)	(7)
17	(12)	(11)	(11)				(9)	(8)
18	(12)	(11)	(12)	(11)				(9)
19	(13)	(12)	(13)	(12)	(11)			(10)
20	(14)	(13)	(14)		(12)	(11)		
21		(14)				(12)	(11)	
22							(12)	(11)
23								(12)
24								(13)
25								(14)

9 cycles for 7 instructions

CPI = 1.29

Missing opportunities

```
for(i = 0; i < count; i++) {
    int64_t temp = a[i];
    a[i] = b[i];
    b[i] = temp;
}
```

Compiler can only do this when it's 100% for sure count is always an even number! — loop unrolling

Compilers are limited by the number of registers available to the software!

```
:
movq    (%rcx,%rax), %r8
movq    (%rdi,%rax), %rsi
addq    $8, %rax
movq    %r8, -8(%rdi,%rax)
movq    %rsi, -8(%rcx,%rax)
cmpq    %r9, %rax
jne     .L9
movq    (%rcx,%rax), %r8
movq    (%rdi,%rax), %rsi
addq    $8, %rax
movq    %r8, -8(%rdi,%rax)
movq    %rsi, -8(%rcx,%rax
cmpq    %r9, %rax
jne     .L9
```

```
.L9:
①  movq    (%rcx,%rax), %r8
②  movq    (%rdi,%rax), %rsi
③  addq    $8, %rax
④  movq    %r8, -8(%rdi,%rax)
⑤  movq    %rsi, -8(%rcx,%rax
⑥  movq    (%rcx,%rax), %r8
⑦  movq    (%rdi,%rax), %rsi
⑧  cmpq    %r9, %rax
⑨  jne     .L9
⑩  addq    $8, %rax
⑪  movq    %r8, -8(%rdi,%rax)
⑫  movq    %rsi, -8(%rcx,%rax
⑬  cmpq    %r9, %rax
⑭  jne     .L9
```

	IF	ID	ALU/BR/AG	M1	M2	M3	M4/XORL	WB/Retire
1	(1)							
2	(2)	(1)						
3	(3)	(2)	(1)					
4	(4)	(3)	(2)	(1)				
5	(5)	(4)	(3)	(2)	(1)			
6	(5)	(4)			(2)	(1)		
7	(5)	(4)				(2)	(1)	
8	(6)	(5)	(4)				(2)	(1)
9	(7)	(6)	(5)	(4)				(2)
10	(8)	(7)	(6)	(5)	(4)			(3)
11	(9)	(8)	(7)	(6)	(5)	(4)		
12	(10)	(9)	(8)	(7)	(6)	(5)	(4)	
13	(11)	(10)	(9)		(7)	(6)	(5)	(4)
14	(12)	(11)	(10)			(7)	(6)	(5)
15	(13)	(12)	(11)	(10)			(7)	(6)
16	(14)	(13)	(12)	(11)	(10)			(7)
17		(14)	(13)	(12)	(11)	(10)		(8)
18			(14)	(13)	(12)	(11)	(10)	(9)
19				(14)	(13)	(12)	(11)	(10)
20					(14)	(13)	(12)	(11)
21						(14)	(13)	(12)
22							(14)	(13)
23								(14)

7 cycles for 7 instructions

CPI = 1

Limitations of Compiler Optimizations

- If the hardware (e.g., pipeline changes), the same compiler optimization may not be that helpful
- The compiler can only optimize on static instructions, but cannot optimize dynamic instruction
  - Compiler cannot predict branches
  - Compiler does not know if cache has the data/instructions

Takeaways: data hazards

- More data dependencies, more likelihood of data hazards
- Stalls and data forwarding can both address data hazards to generate correct code execution results — but not very efficient
- Compiler optimizations can help, but to a limited extent

Announcements

- **Assignment 4** due this **Saturday**
  - Please reaccept the invitation again — we have to scrap the original one due to permission issues
  - You may still keep your current content — rename the folder on datahub to a different name, copy your answers to the newly created notebook
- **Reading Quiz 7** due **Wednesday** before the lecture

Computer
Science &
Engineering

つづく