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

Hung-Wei Tseng

Recap: how does compiler implement if-else?

```
for(i=0;i<size;i++)
{
    if (data[i] % 2 != 0)
        call_when_true(&data[i]);
    else
        call_when_false(&data[i]);
}
return 0;
```

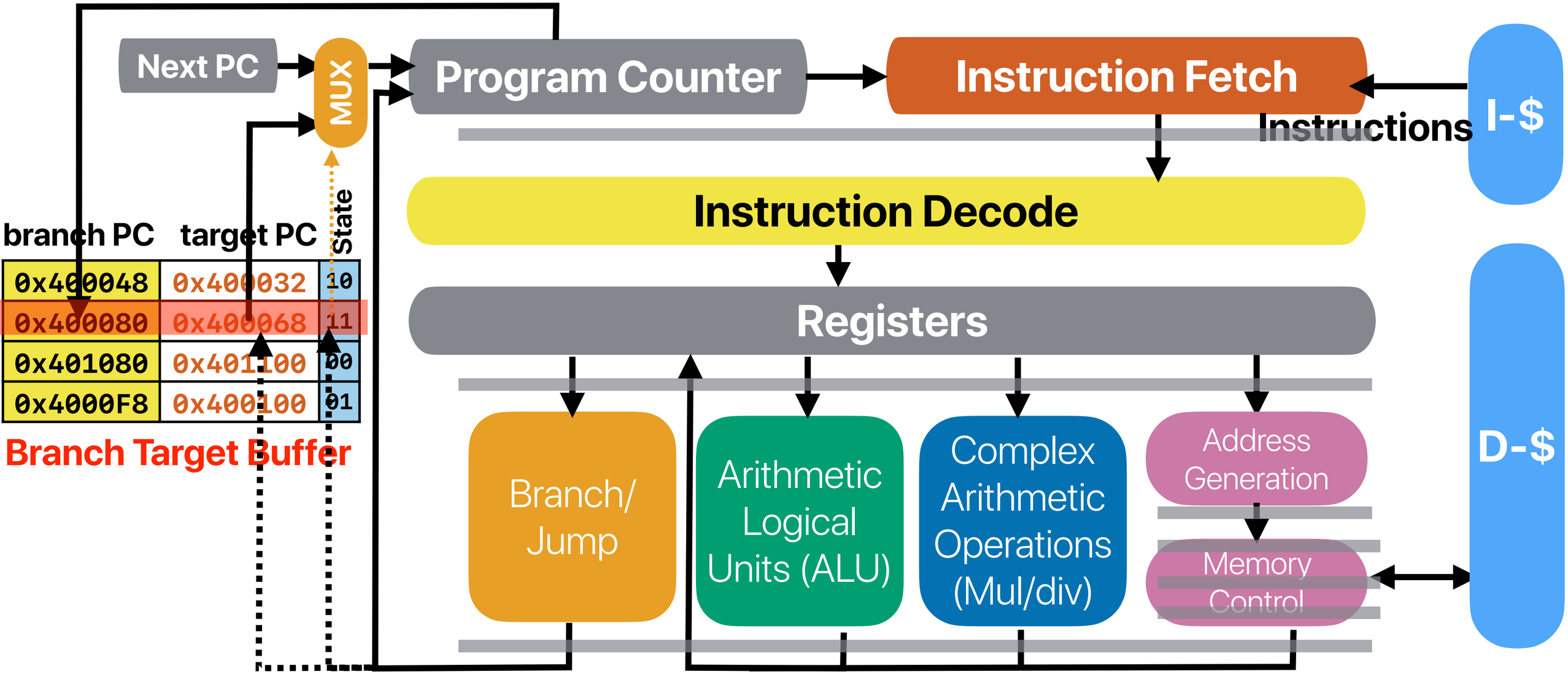
go to L10 when data[i] % 2 == 1 // branch taken if it's false

branch taken (changing address to L11) if the evaluation is not true

```
.L12:
    movl    -32(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    movq    -8(%rbp), %rax
    addq    %rdx, %rax
    movq    (%rax), %rax
    andl    $1, %eax
    testq   %rax, %rax
    je  .L10
    movl    -32(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    movq    -8(%rbp), %rax
    addq    %rdx, %rax
    movq    %rax, %rdi
    call    call_when_true
    jmp .L11
.L10:
    movl    -32(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    movq    -8(%rbp), %rax
    addq    %rdx, %rax
    movq    %rax, %rdi
    call    call_when_false
.L11:
    addl    $1, -32(%rbp)
.L9:
    movl    -32(%rbp), %eax
    cmpl    -24(%rbp), %eax
    jl  .L12
    movl    $0, %eax
    leave
```

In if-else statement, true can mean "not taken"

Detail of a basic dynamic branch predictor



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)

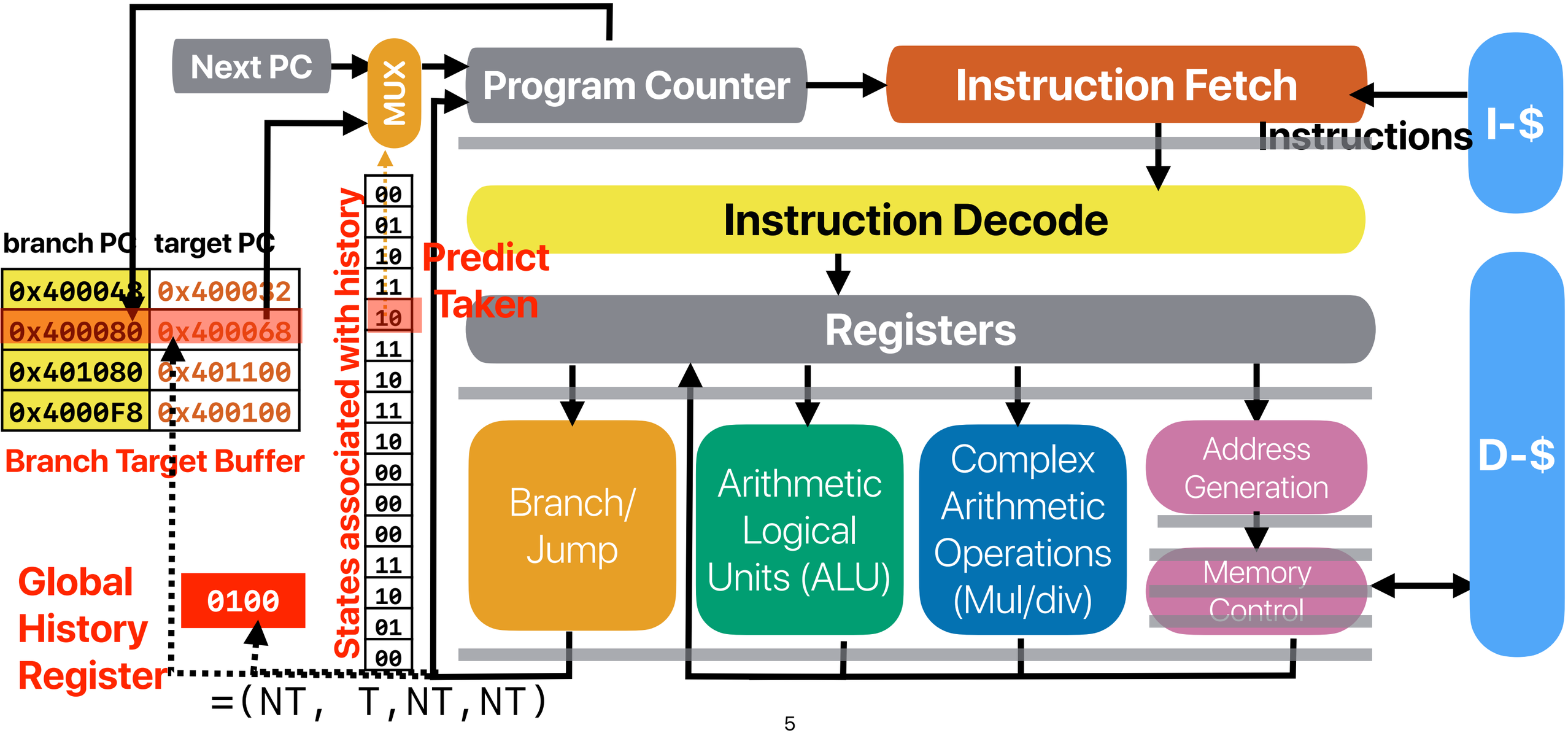
A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

Can we do a better job?

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

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

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

24 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

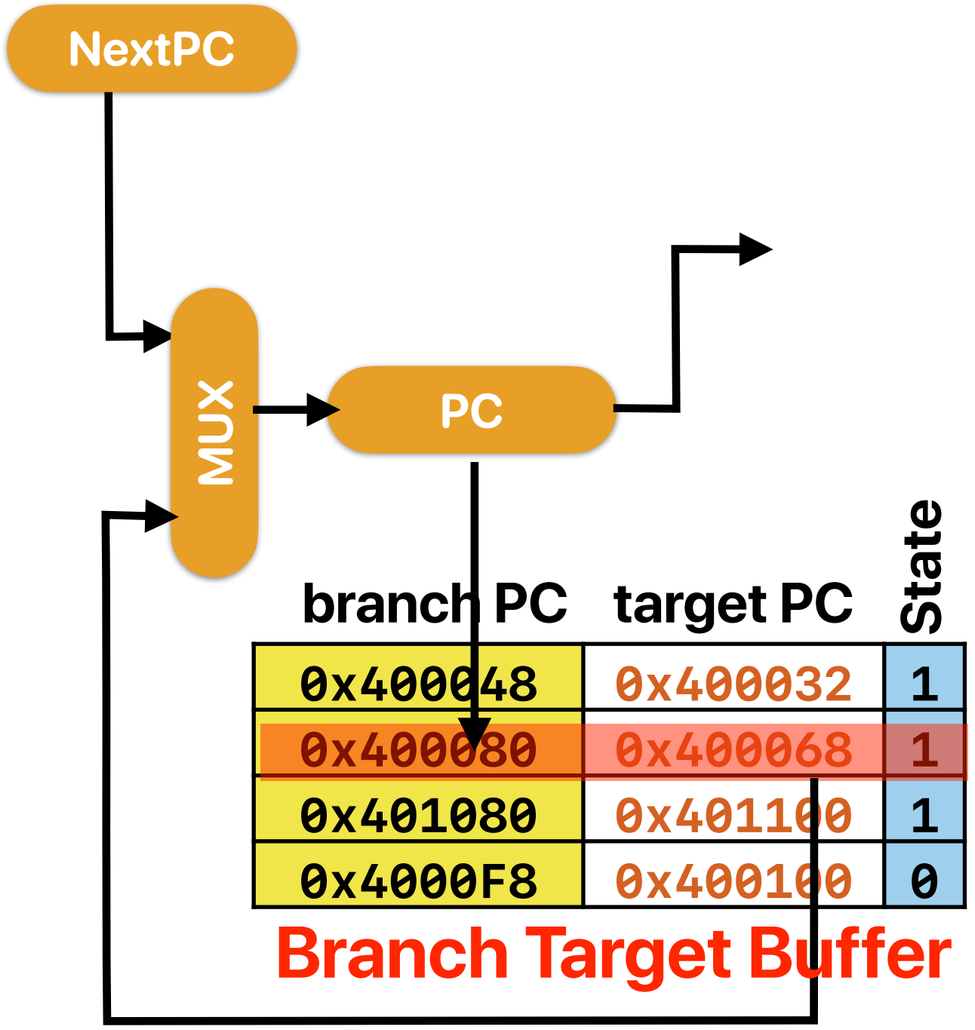
- The cost of not to predict a branch is to stall until the data dependency is resolved — 34 cycles on modern intel processors and 24 on AMD processors
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict 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

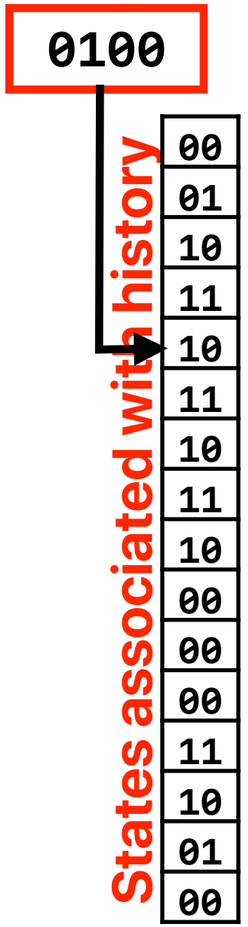
- Hybrid predictors (cont.)
- Data hazards
- Hardware optimizations for data hazards

Hybrid predictors

Tournament Predictor



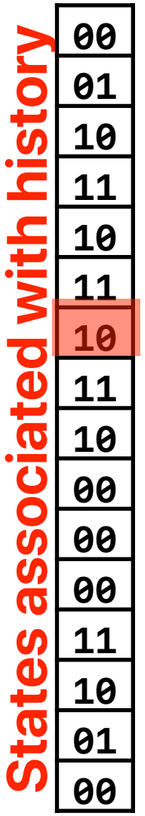
Global History Register



Local History Predictor

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

Predict Taken



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

TAGE

André Seznec. The L-TAGE branch predictor. Journal of Instruction Level Parallelism (http://wwwjilp.org/vol9), May 2007.

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);
```

about the same

II

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

about the same

III

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

IV

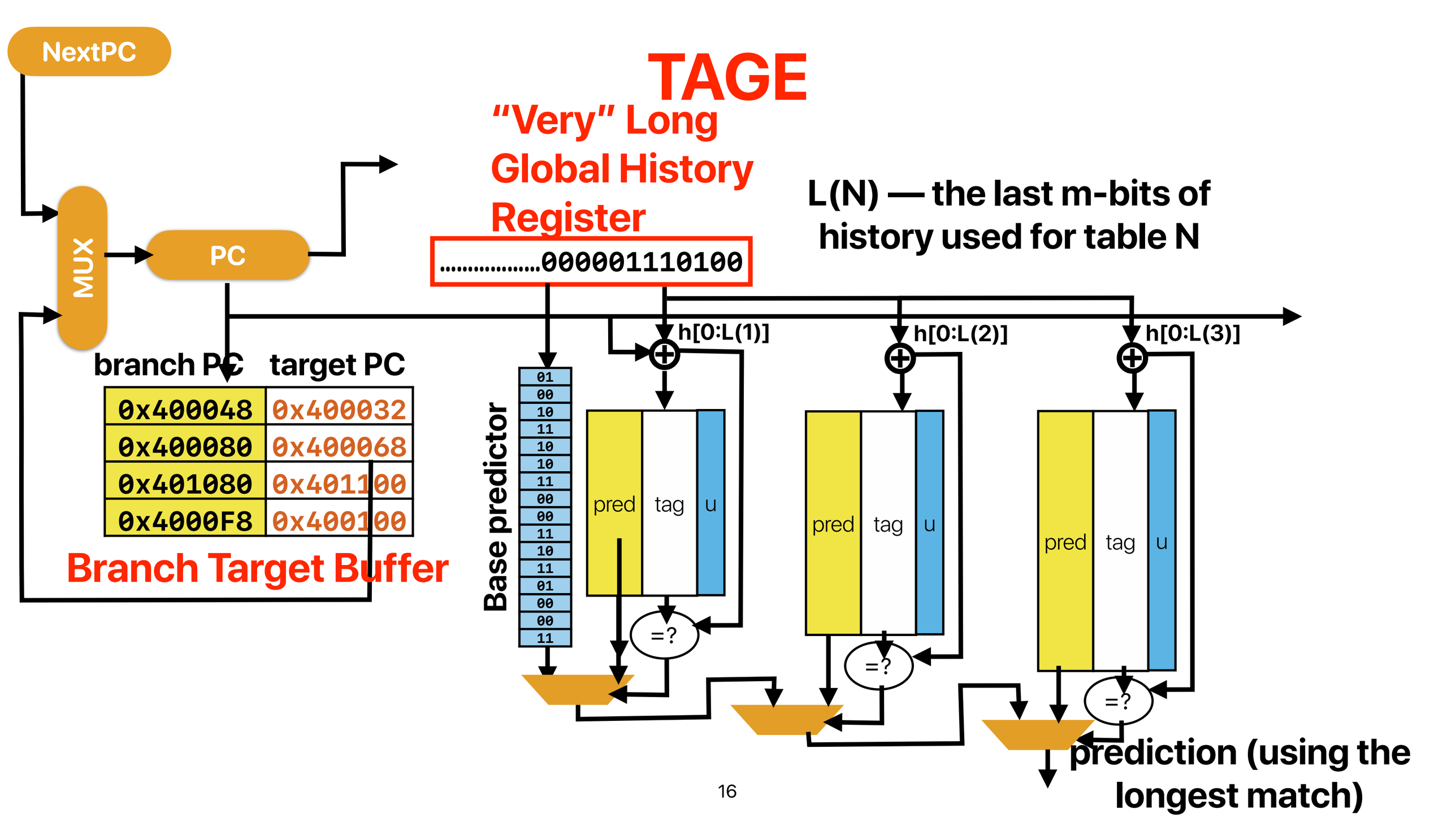
```
i = 0;
do {
    if( rand() %2 == 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)
```

L could be better

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

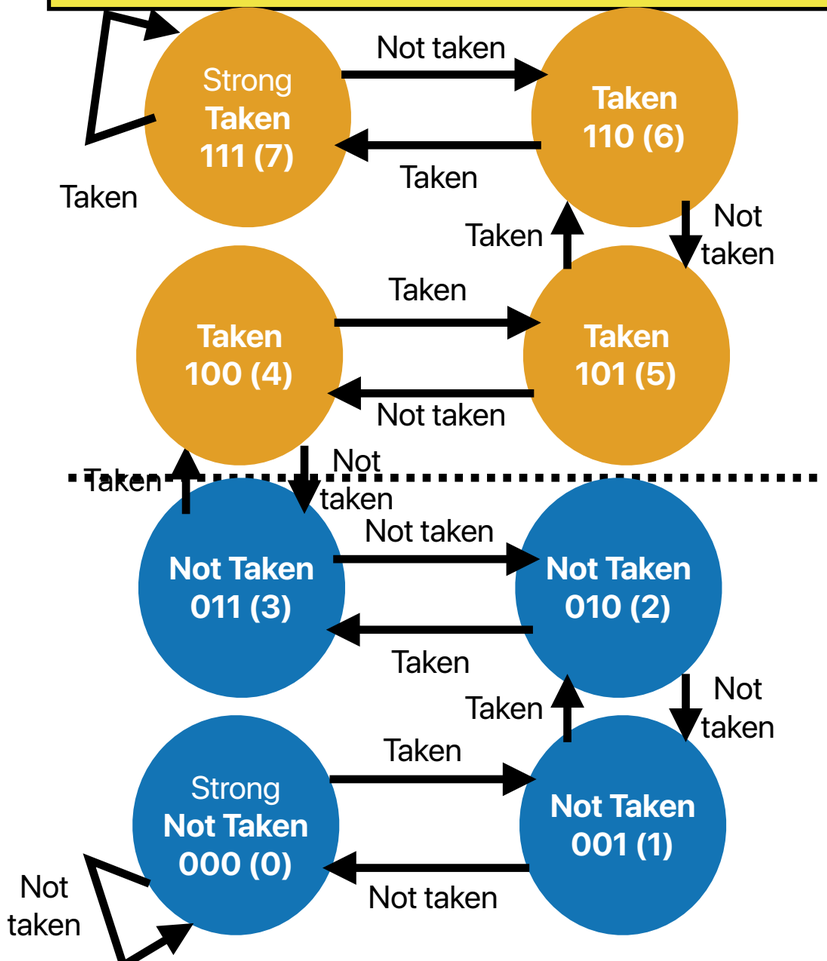
different branch needs different length of history

global predictor can work if the history is long enough!


TAGE
"Very" Long Global History Register
L(N) — the last m-bits of history used for table N
NextPC
MUX
PC
.....000001110100
branch PC
target PC
0x400048
0x400032
0x400080
0x400068
0x401080
0x401100
0x4000F8
0x400100
Branch Target Buffer
Base predictor
01
00
10
11
10
10
11
00
00
11
10
11
01
00
00
11
h[0:L(1)]
h[0:L(2)]
h[0:L(3)]
pred
tag
u
=?
prediction (using the longest match)

What's inside each table?

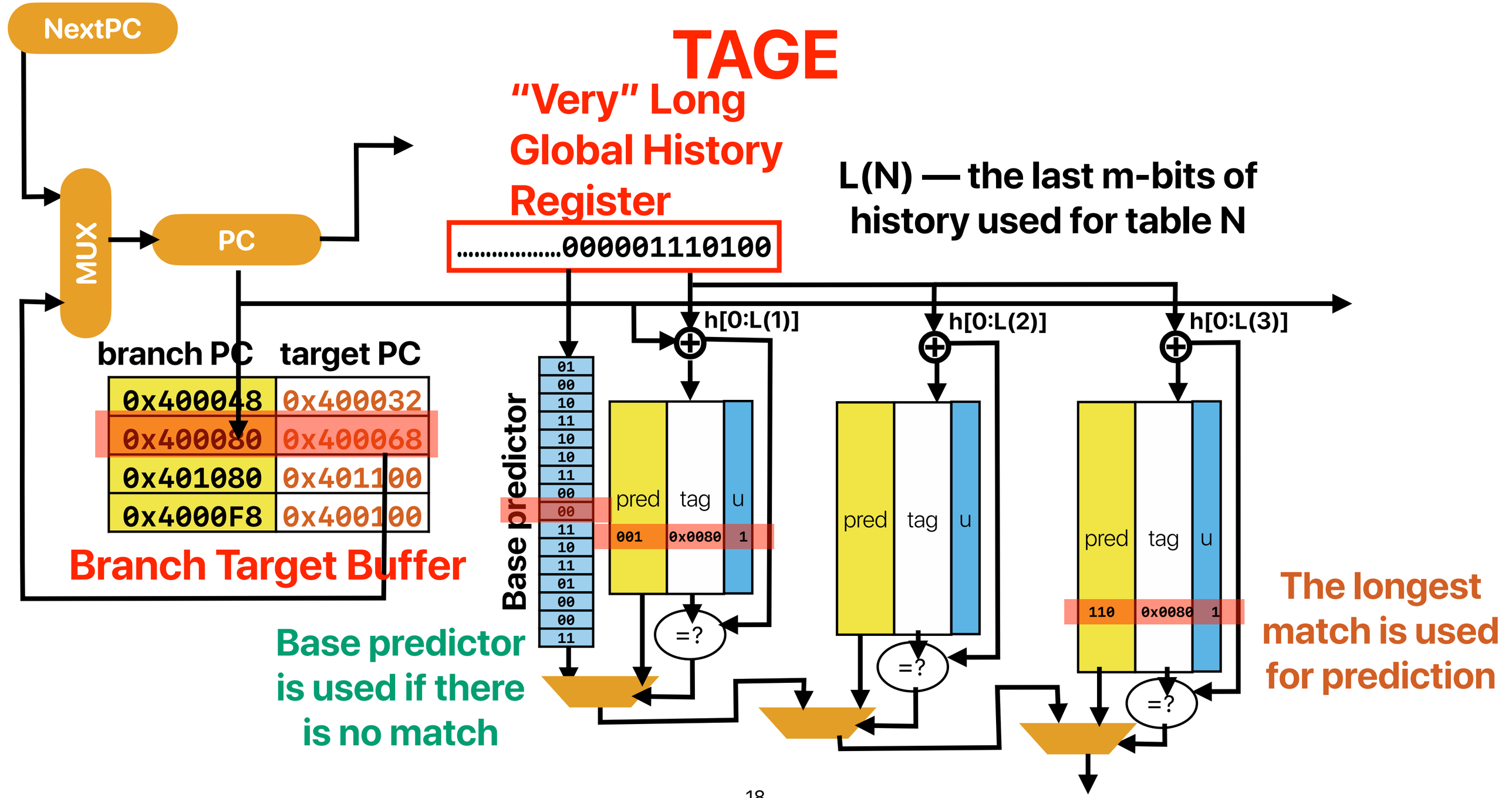
pred (3-bit counter)	tag (partial branch PC)	u (usefulness)



$$if\ prediction(alt\_predictor) \neq prediction(pred):$$

$$if\ prediction(pred) = actual\ result: u = u + 1$$

$$if\ prediction(pred) \neq actual\ result: u = u - 1$$


NextPC
MUX
PC
TAGE
"Very" Long Global History Register
.....000001110100
L(N) — the last m-bits of history used for table N
branch PC
target PC
0x400048
0x400032
0x400080
0x400068
0x401080
0x401100
0x4000F8
0x400100
Branch Target Buffer
Base predictor
01
00
10
11
10
10
11
00
00
11
10
11
01
00
00
11
h[0:L(1)]
h[0:L(2)]
h[0:L(3)]
pred
tag
u
001
0x0080
1
110
0x0080
1
=?
Base predictor is used if there is no match
The longest match is used for prediction

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

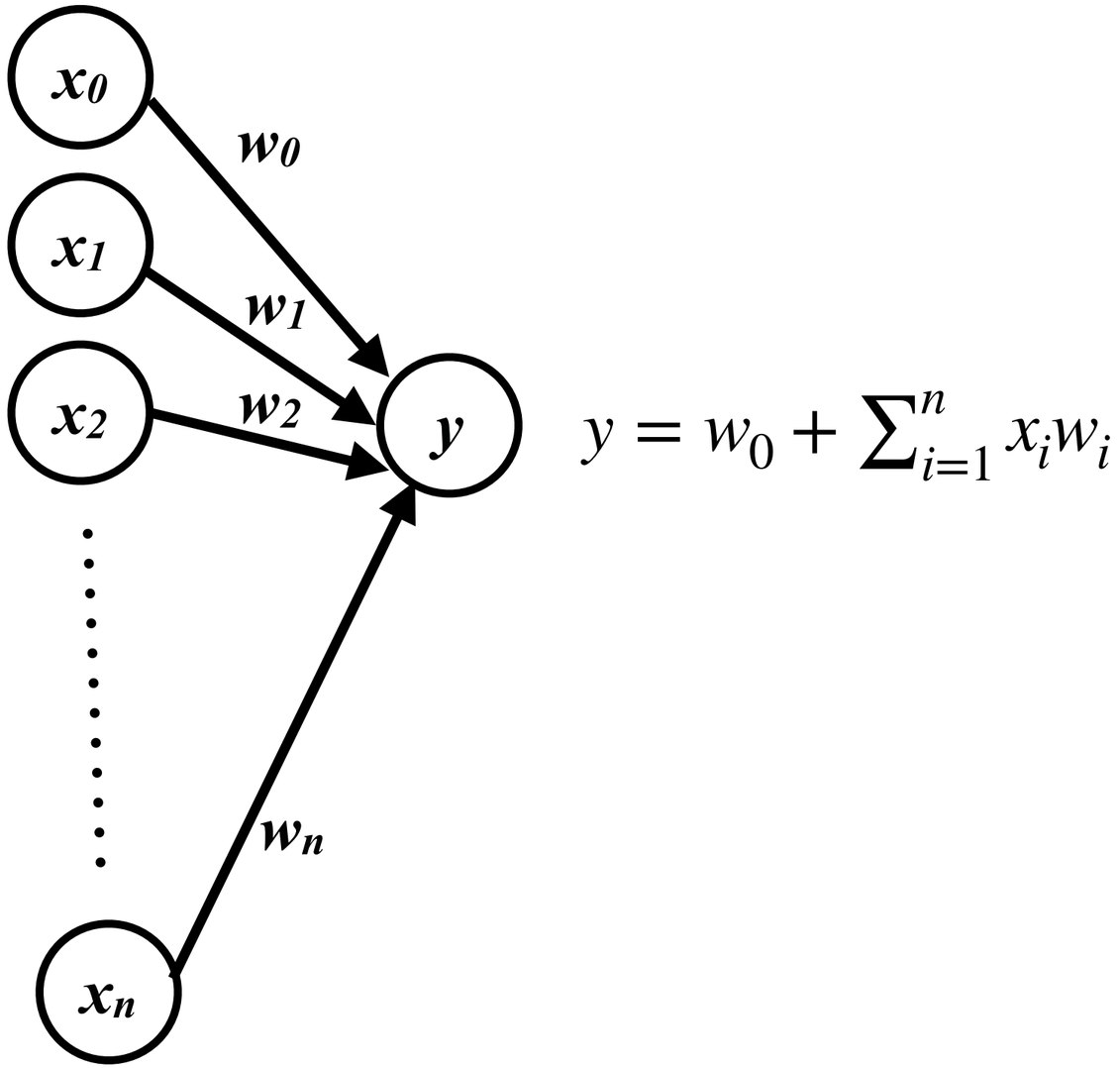
- The machine learns to predict conditional branches
- Artificial neural networks
  - Simple model of neural networks in brain cells
  - Learn to recognize and classify patterns

Mapping Branch Prediction to NN

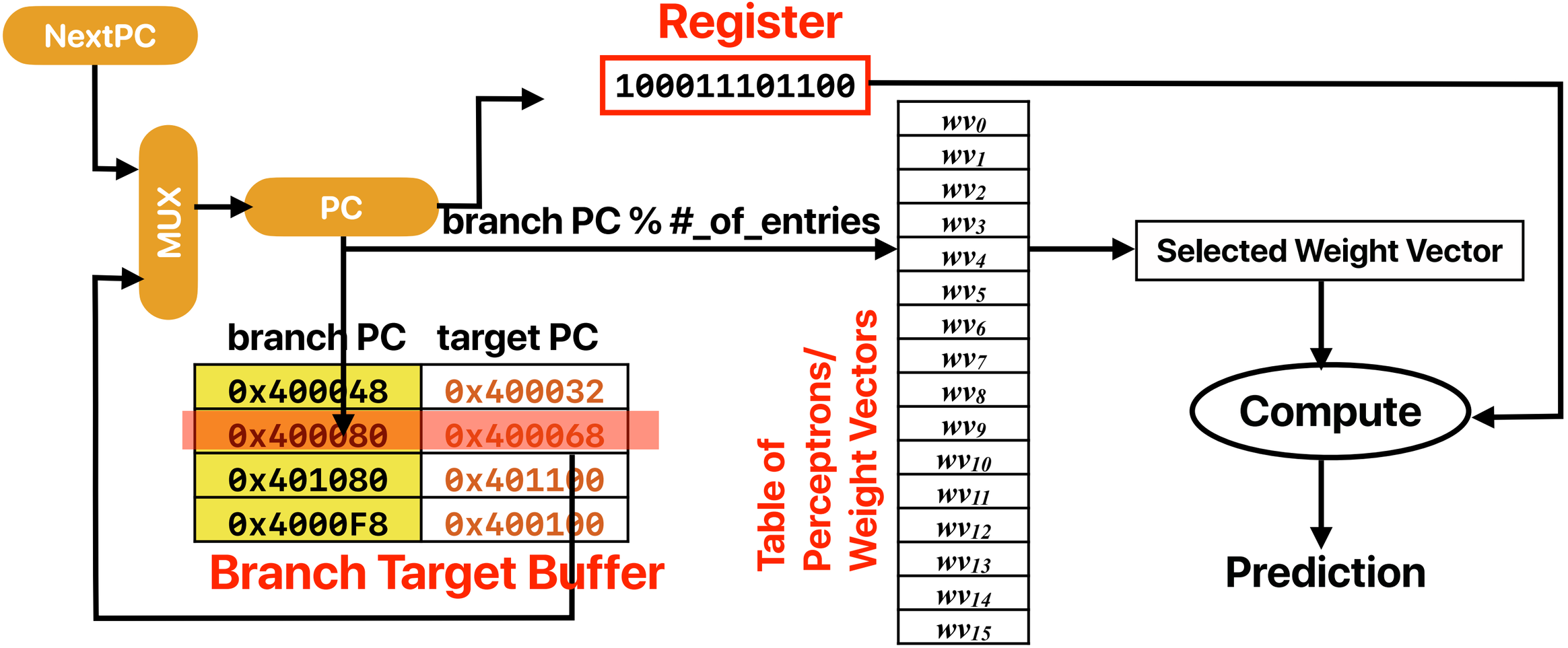
- The inputs to the perceptron are branch outcome histories
  - Just like in 2-level adaptive branch prediction
  - Can be global or local (per-branch) or both (alloyed)
  - Conceptually, branch outcomes are represented as
    - +1, for taken
    - -1, for not taken
- The output of the perceptron is
  - 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



Predictor Organization



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| ≤ θ or ((y ≥ 0) ≠ t) then
    for each 0 ≤ i ≤ 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
```

Global History `1 0 0 0 1 1 1 0 1 1 0 0`

Branch X Taken

Global History `1 1 1 0 1 1 0 0 1 0 0 0`

Branch X Taken

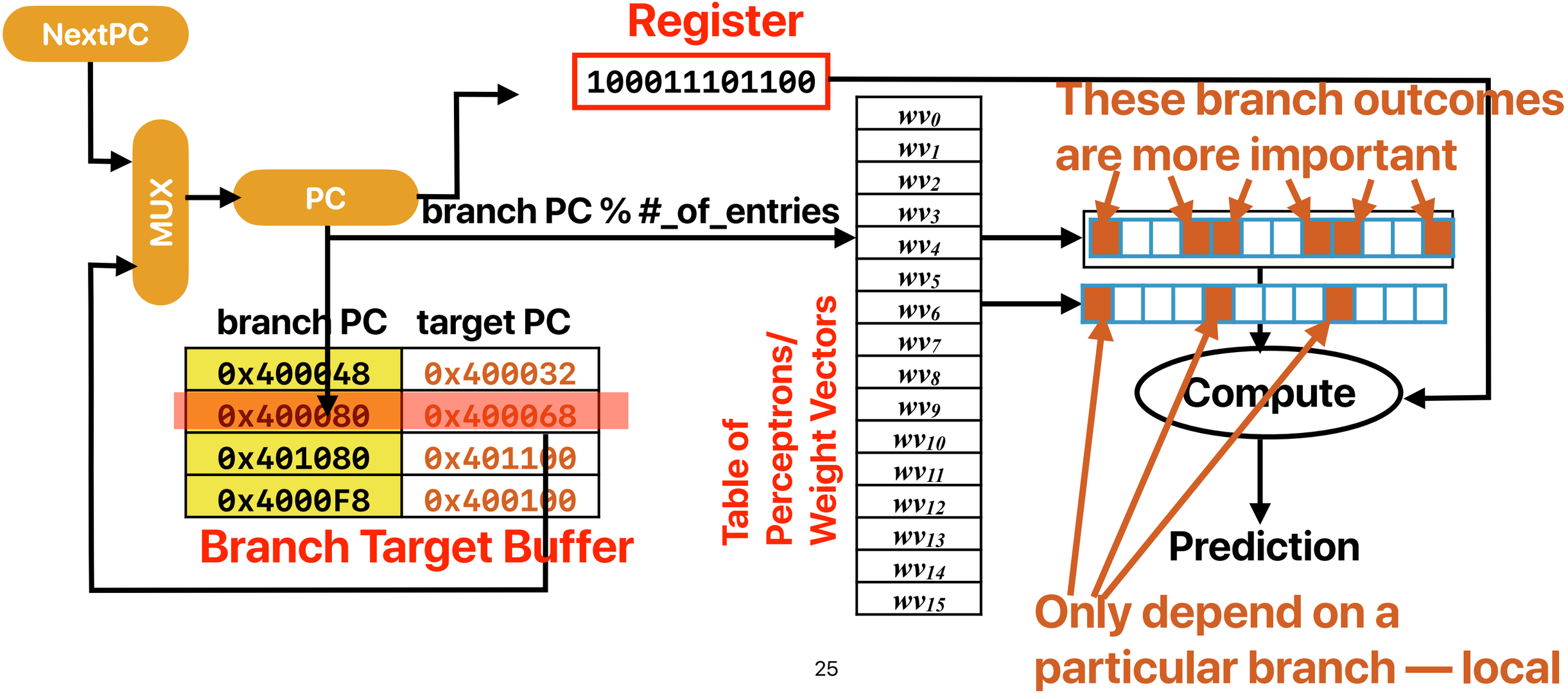
Global History `1 1 0 0 1 0 0 0 1 1 1 0`

Branch X Taken

Global History `1 0 0 0 1 1 1 0 1 1 0 0`

Branch X Taken

Predictor Organization



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.

Demo revisited

```
SELECT count(*) FROM TABLE WHERE val >= A;
```

```
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)
            sum ++;
    }
}
```

option = 1 is faster!!!, this is just a one-time cost and you don't do this after the data is loaded

	Without sorting	With sorting
The prediction accuracy of X before threshold	60%-70%	100%
The prediction accuracy of X after threshold	60%-70%	100%

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

Design decisions in real practice

- AMD Zen 2 (RyZen 3000 series processors) adopts a design with first level predictor using perceptron and using TAGE for the 2nd level. What such a design decision implies about the characteristics of TAGE and Perceptron?
  - ① Perceptron takes longer to train than TAGE
  - ② Perceptron takes longer to predict than TAGE
  - ③ Perceptron is more accurate than TAGE
  - ④ Perceptron's performance improves less given more area
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

perceptron predictors. For this reason, TAGE was a good choice for [redacted] L2 predictor while keeping perceptron as the L1 predictor for [redacted].

TAGE vs Perceptron

0

A

0%

B

0%

C

0%

D

0%

E

0%

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

Design decisions in real practice

- AMD Zen 2 (RyZen 3000 series processors) adopts a design with first level predictor using perceptron and using TAGE for the 2nd level. What such a design decision implies about the characteristics of TAGE and Perceptron?
  - ① Perceptron takes longer to train than TAGE
  - ② Perceptron takes longer to predict than TAGE
  - ③ Perceptron is more accurate than TAGE
  - ④ Perceptron's performance improves less given more area
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

perceptron predictors. For this reason, TAGE was a good choice for [redacted] L2 predictor while keeping perceptron as the L1 predictor for [redacted].

TAGE vs Perceptron — Group

0

A

0%

B

0%

C

0%

D

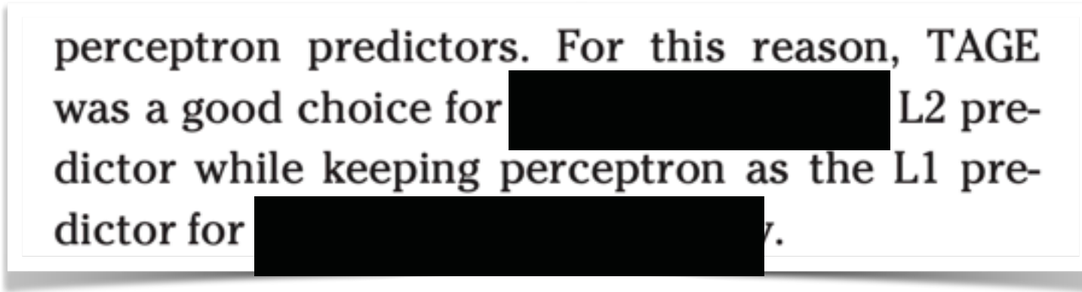
0%

E

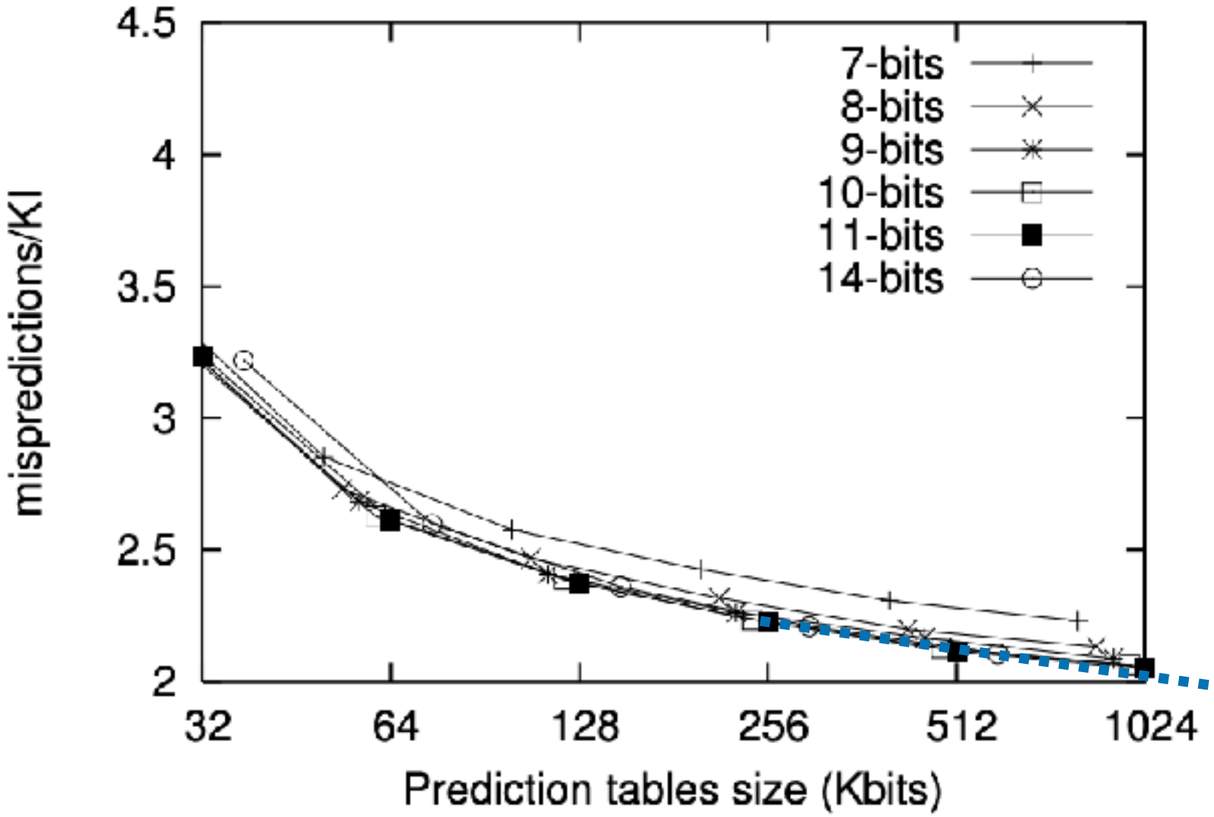
0%

Design decisions in real practice

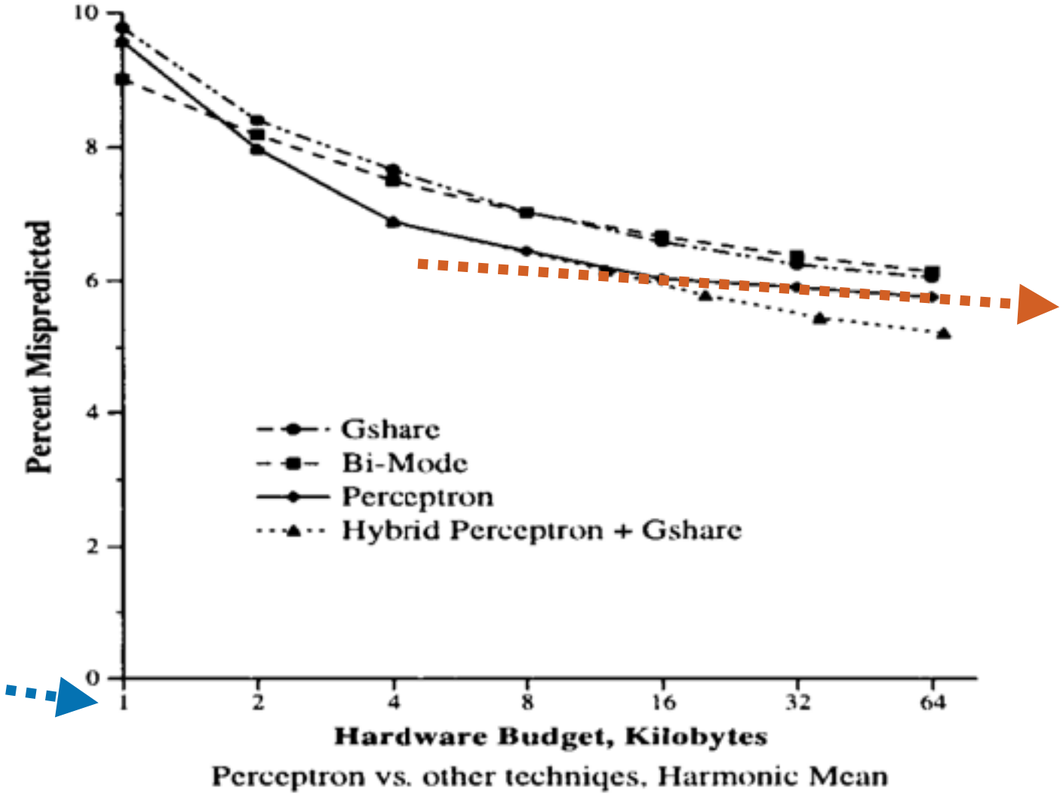
- AMD Zen 2 (RyZen 3000 series processors) adopts a design with first level predictor using perceptron and using TAGE for the 2nd level. What such a design decision implies about the characteristics of TAGE and Perceptron?
  - ① Perceptron takes longer to train than TAGE
  - ② Perceptron takes longer to predict than TAGE
  - ③ Perceptron is more accurate than TAGE
  - ④ Perceptron's performance improves less given more area
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4


perceptron predictors. For this reason, TAGE was a good choice for [redacted] L2 predictor while keeping perceptron as the L1 predictor for [redacted].

Area efficiency between TAGE and Perceptron



TAGE



Perceptron

How good is prediction using perceptrons?

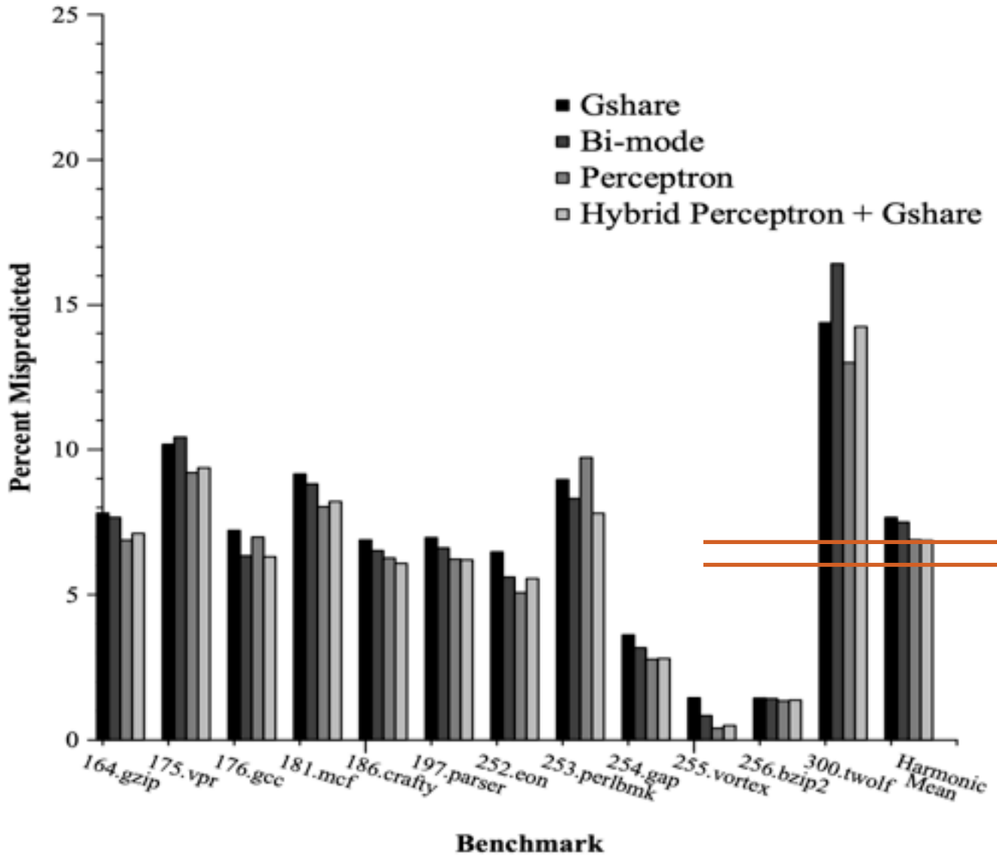


Figure 4: Misprediction Rates at a 4K budget. The perceptron predictor has a lower misprediction rate than *gshare* for all benchmarks except for `186.crafty` and `197.parser`.

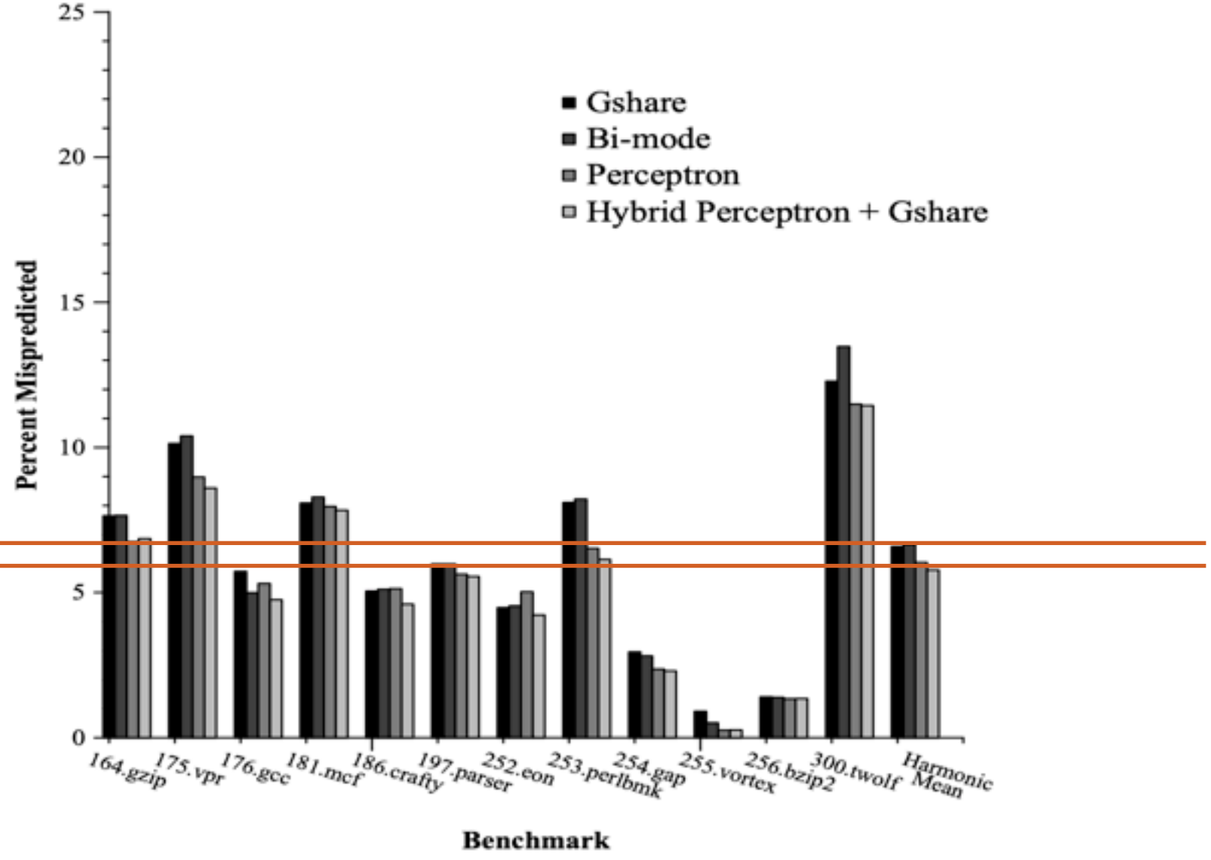
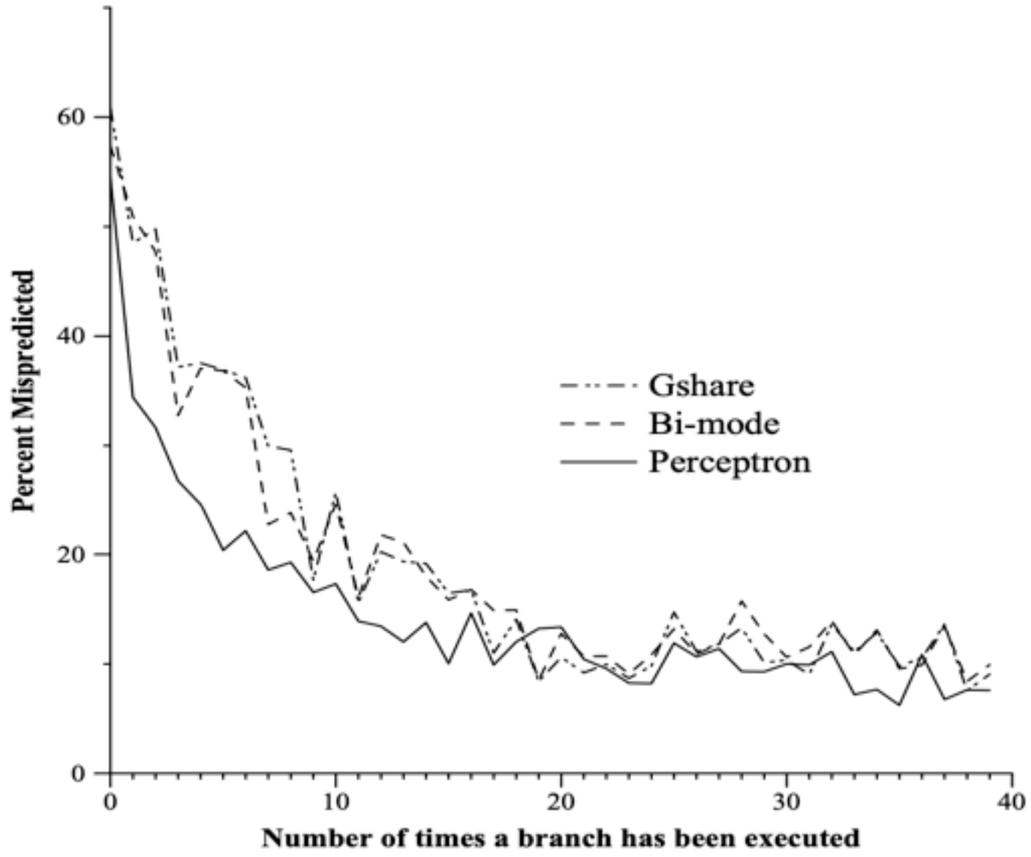


Figure 5: Misprediction Rates at a 16K budget. Gshare outperforms the perceptron predictor only on `186.crafty`. The hybrid predictor is consistently better than the PHT schemes.

History/training for perceptrons



Hardware budget in kilobytes	History Length		
	gshare	bi-mode	perceptron
1	6	7	12
2	8	9	22
4	8	11	28
8	11	13	34
16	14	14	36
32	15	15	59
64	15	16	59
128	16	17	62
256	17	17	62
512	18	19	62

Table 1: Best History Lengths. This table shows the best amount of global history to keep for each of the branch prediction schemes.

AMD Zen 2's design experience

PREDICTION, FETCH, AND DECODE

The in-order front-end of the Zen 2 core includes branch prediction, instruction fetch, and decode. The branch predictor in Zen 2 features a two-level conditional branch predictor. To increase prediction accuracy, the L2 predictor has been upgraded from a perceptron predictor in Zen to a tagged geometric history length (TAGE) predictor in Zen 2.[5] TAGE predictors provide high accuracy per bit of storage capacity. However, they do multiplex read data from multiple tables, requiring a timing tradeoff versus perceptron predictors. For this reason, TAGE was a good choice for the longer-latency L2 predictor while keeping perceptron as the L1 predictor for best timing at low latency.

D. Suggs D. Bouvier M. Subramony and K. Lepak "Zen 2" Hot Chips vol. 31 2019.

Design decisions in real practice

- AMD Zen 2 (RyZen 3000 series processors) adopts a design with first level predictor using perceptron and using TAGE for the 2nd level. What such a design decision implies about the characteristics of TAGE and Perceptron?

① Perceptron takes longer to train than TAGE

no — based on the paper

② Perceptron takes longer to predict than TAGE

short — otherwise won't be in L1

③ Perceptron is more accurate than TAGE

less accurate — otherwise won't need an L2

④ Perceptron's performance improves less given more area

A. 0

no — based on the paper

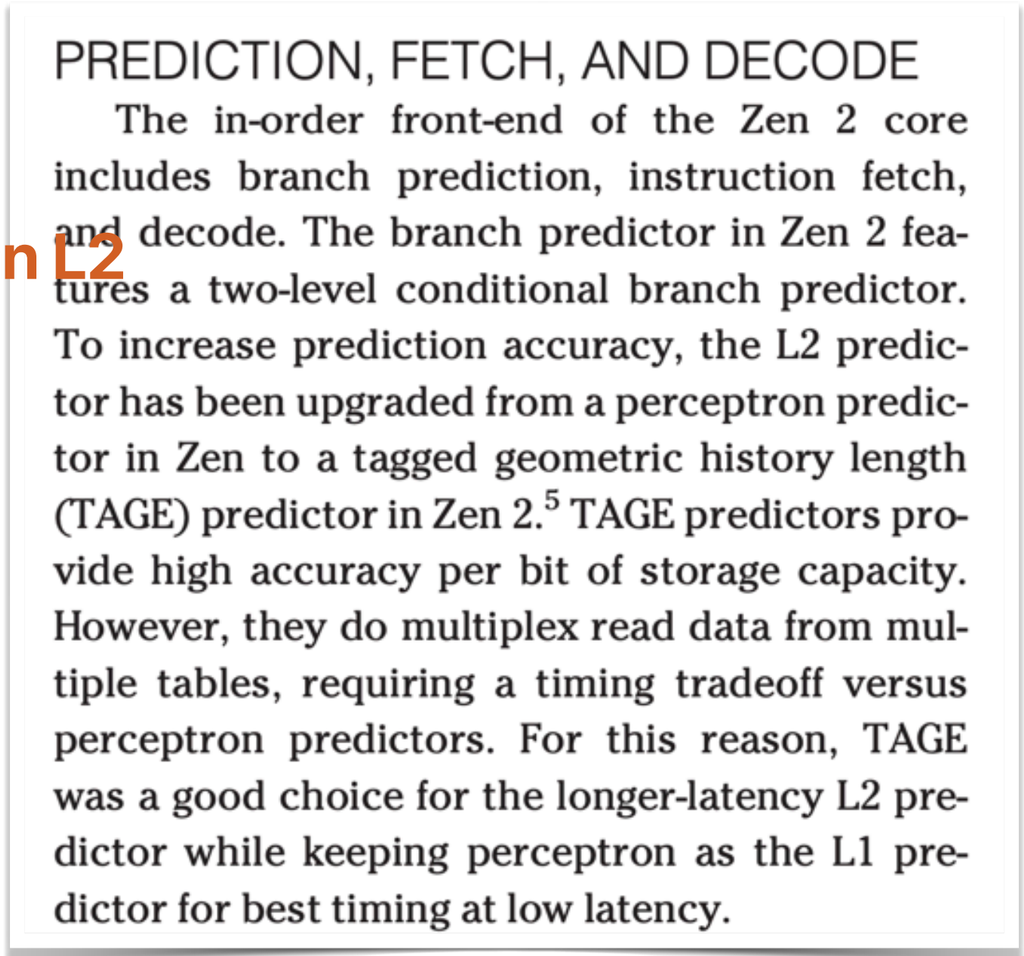
B. 1

C. 2

D. 3

E. 4

D. Suggs D. Bouvier M. Subramony and K. Lepak "Zen 2" Hot Chips vol. 31 2019.



PREDICTION, FETCH, AND DECODE

The in-order front-end of the Zen 2 core includes branch prediction, instruction fetch, and decode. The branch predictor in Zen 2 features a two-level conditional branch predictor. To increase prediction accuracy, the L2 predictor has been upgraded from a perceptron predictor in Zen to a tagged geometric history length (TAGE) predictor in Zen 2.[5] TAGE predictors provide high accuracy per bit of storage capacity. However, they do multiplex read data from multiple tables, requiring a timing tradeoff versus perceptron predictors. For this reason, TAGE was a good choice for the longer-latency L2 predictor while keeping perceptron as the L1 predictor for best timing at low latency.

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!

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

How many data dependencies do we have?

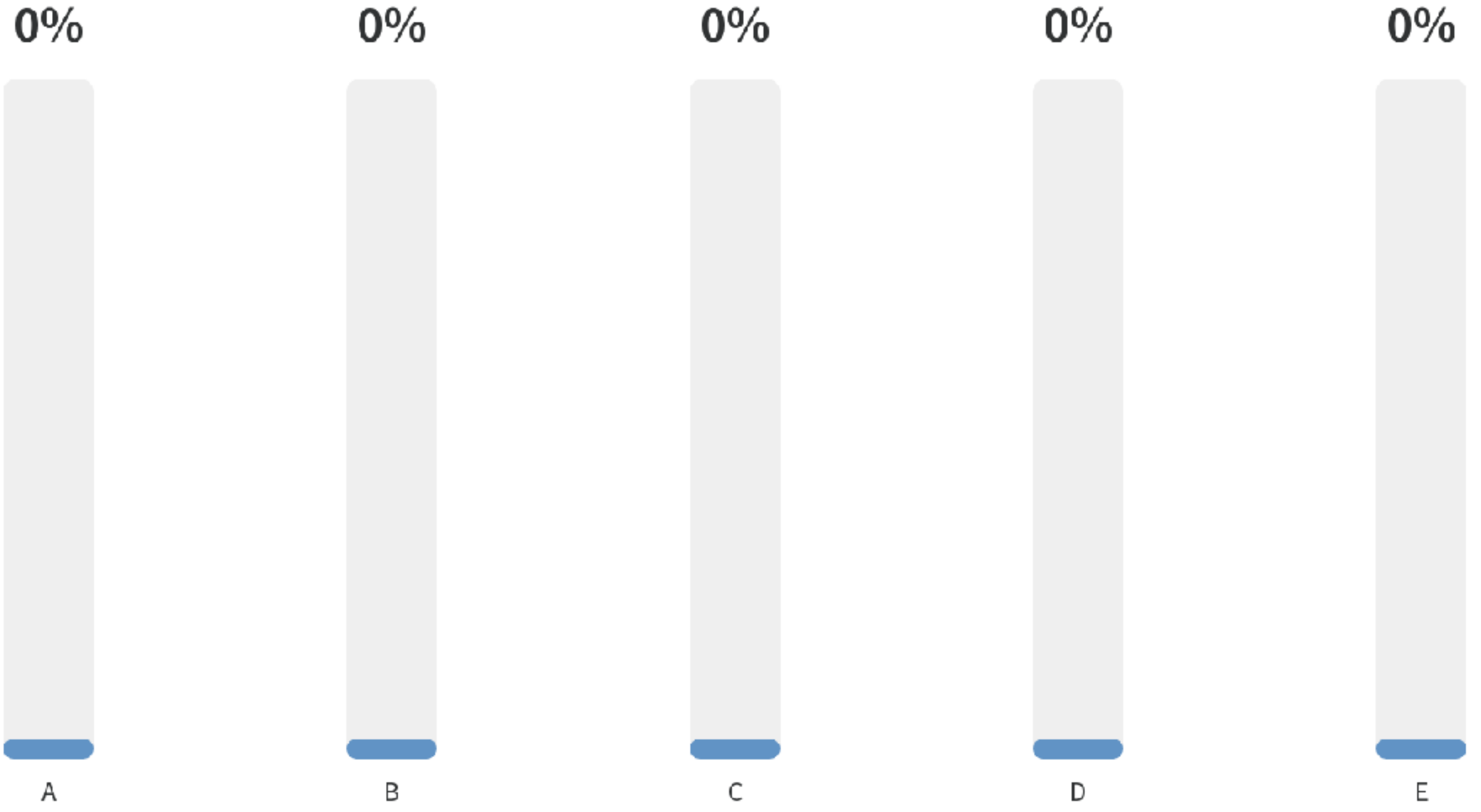
- How many pairs of data dependences are there in the following x86 instructions?

```
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

0



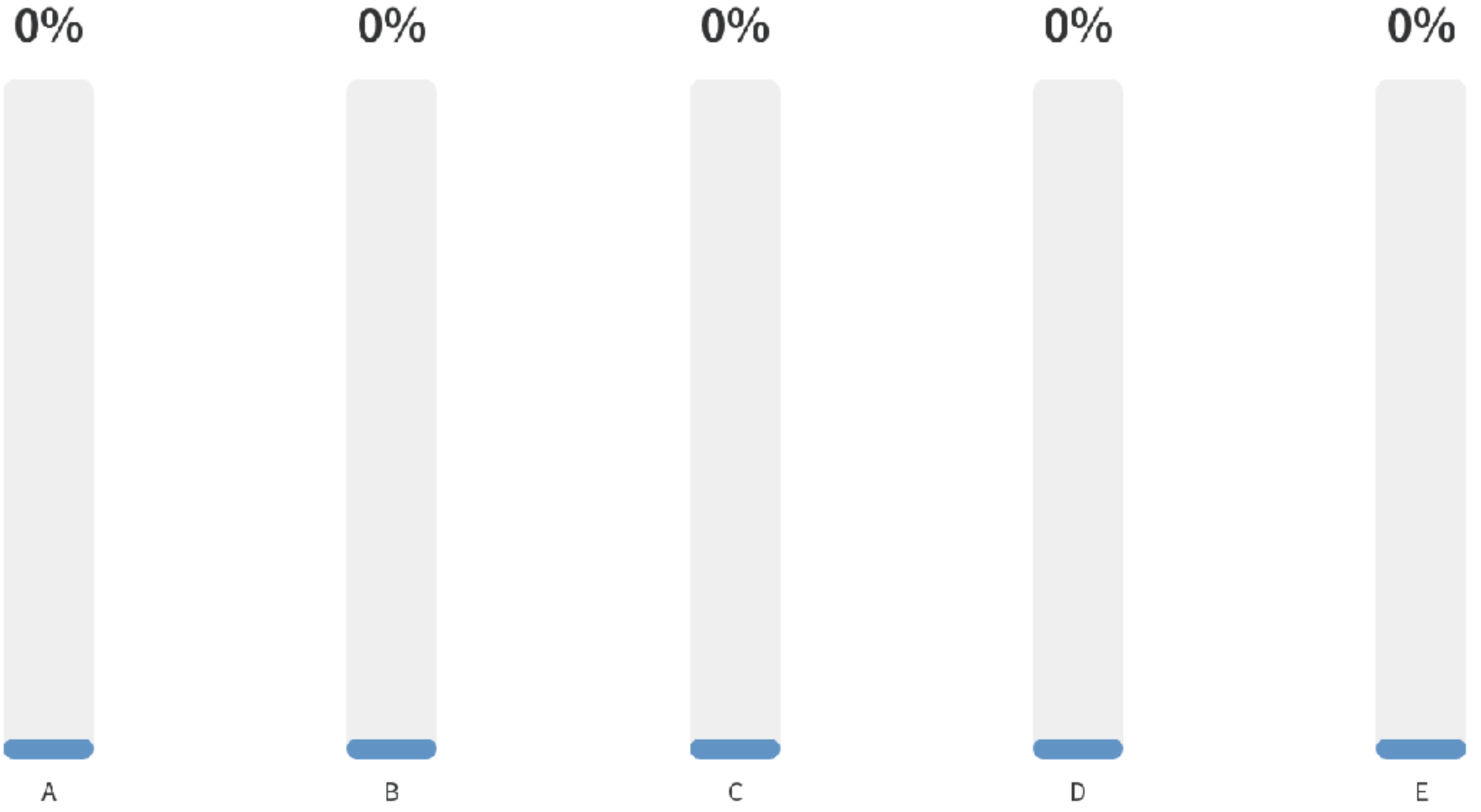
How many data dependencies do we have?

- 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)
```

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

0



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)
```

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

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

How many data dependencies do we have?

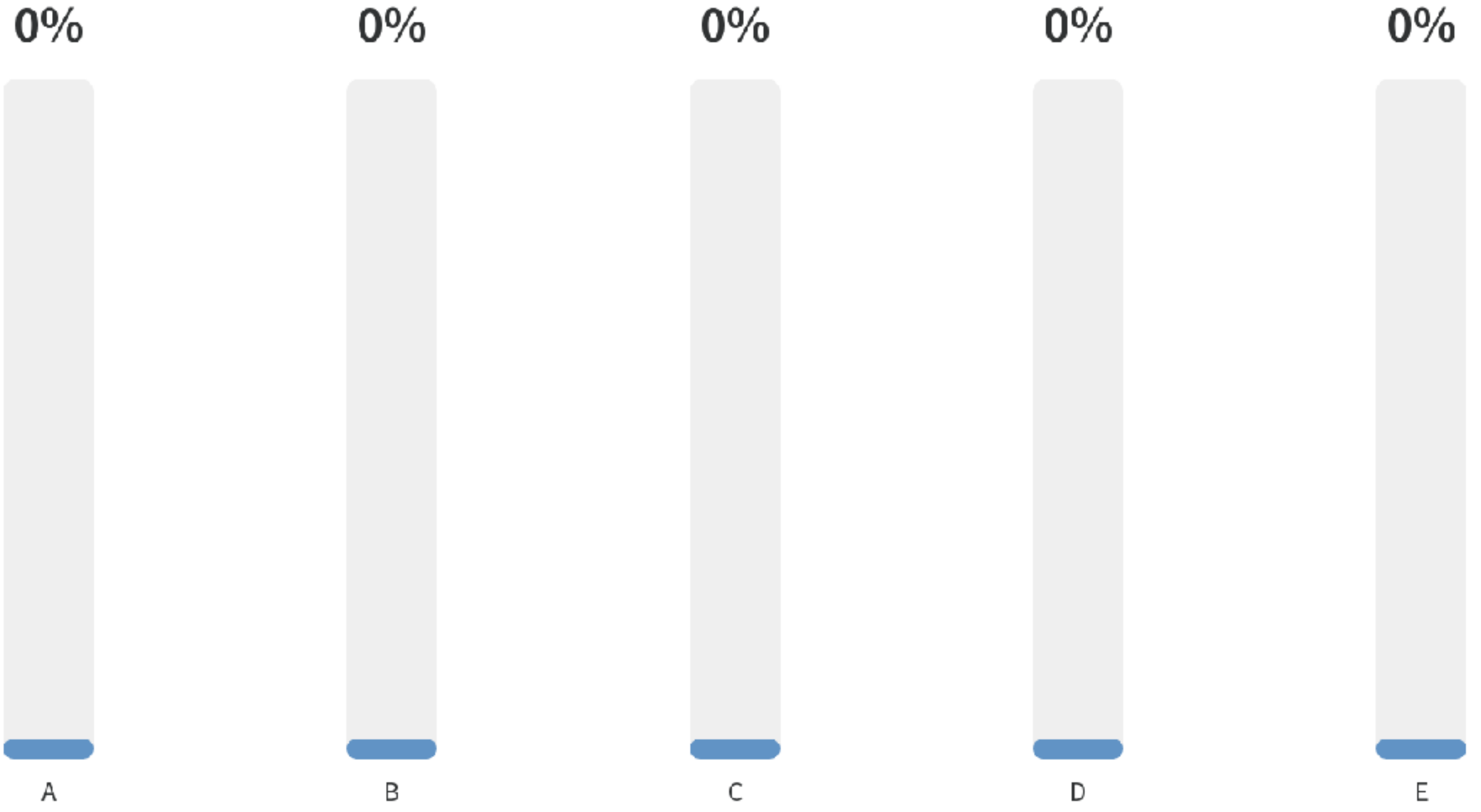
- How many pairs of data dependences are there in the following x86 instructions?

```
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

0



How many data dependencies do we have?

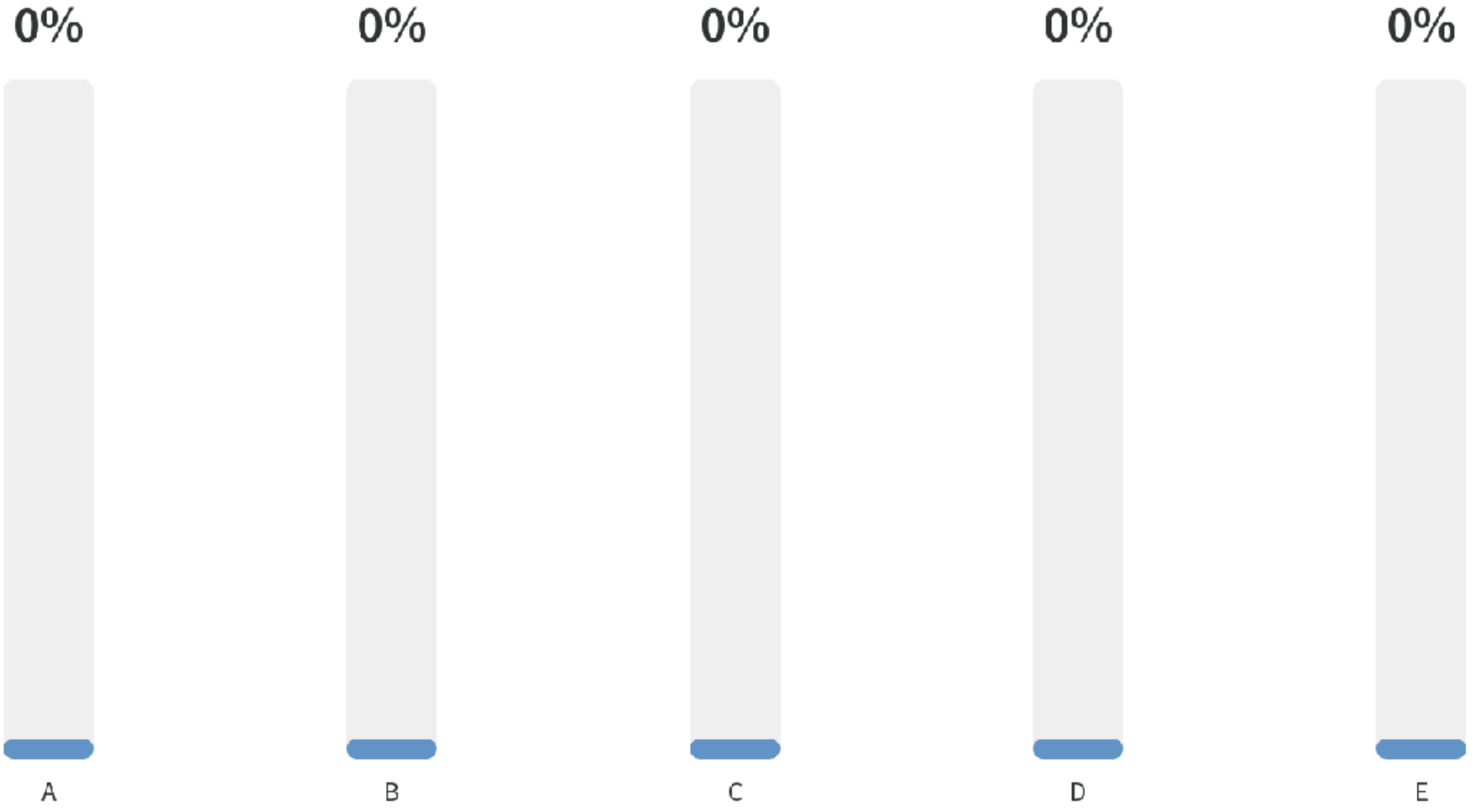
- How many pairs of data dependences are there in the following x86 instructions?

```
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

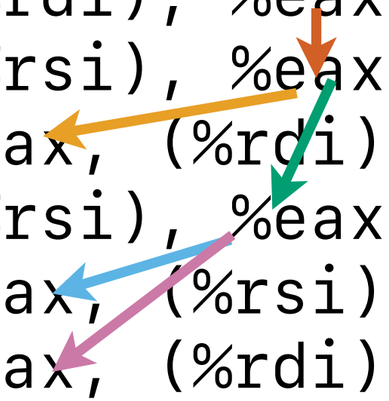
0



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.

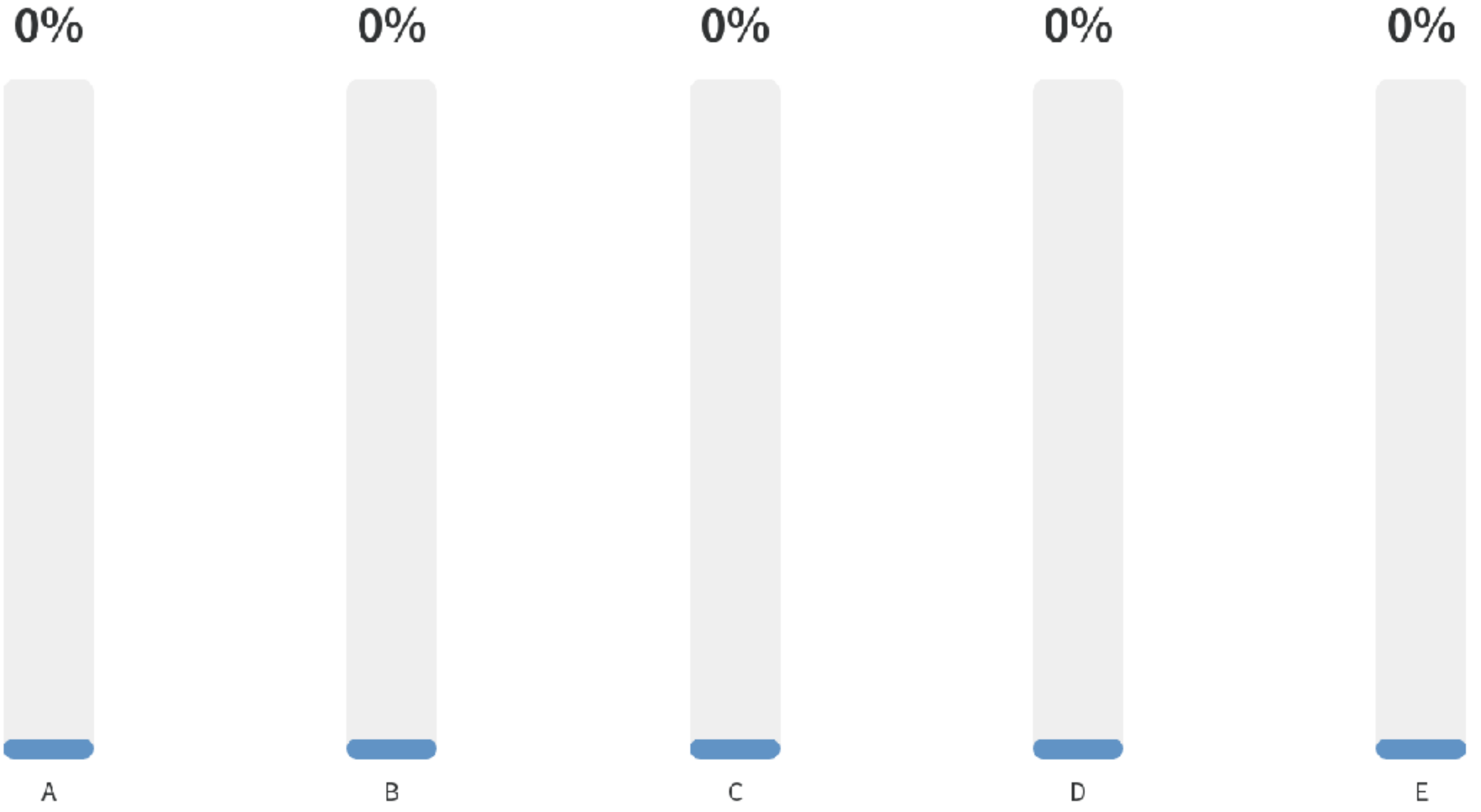
Data hazards?

- 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

0



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

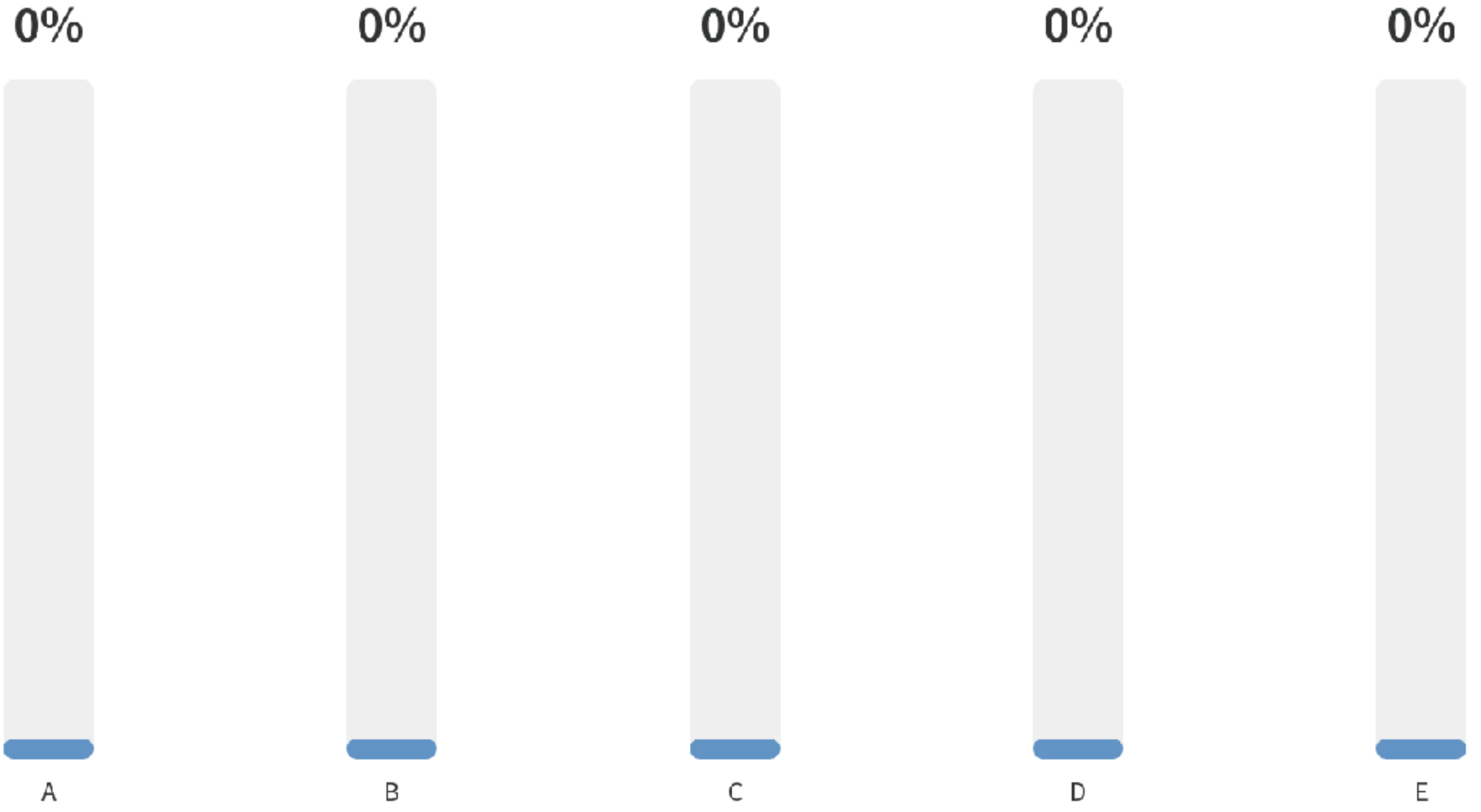
Data hazards?

- 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

0

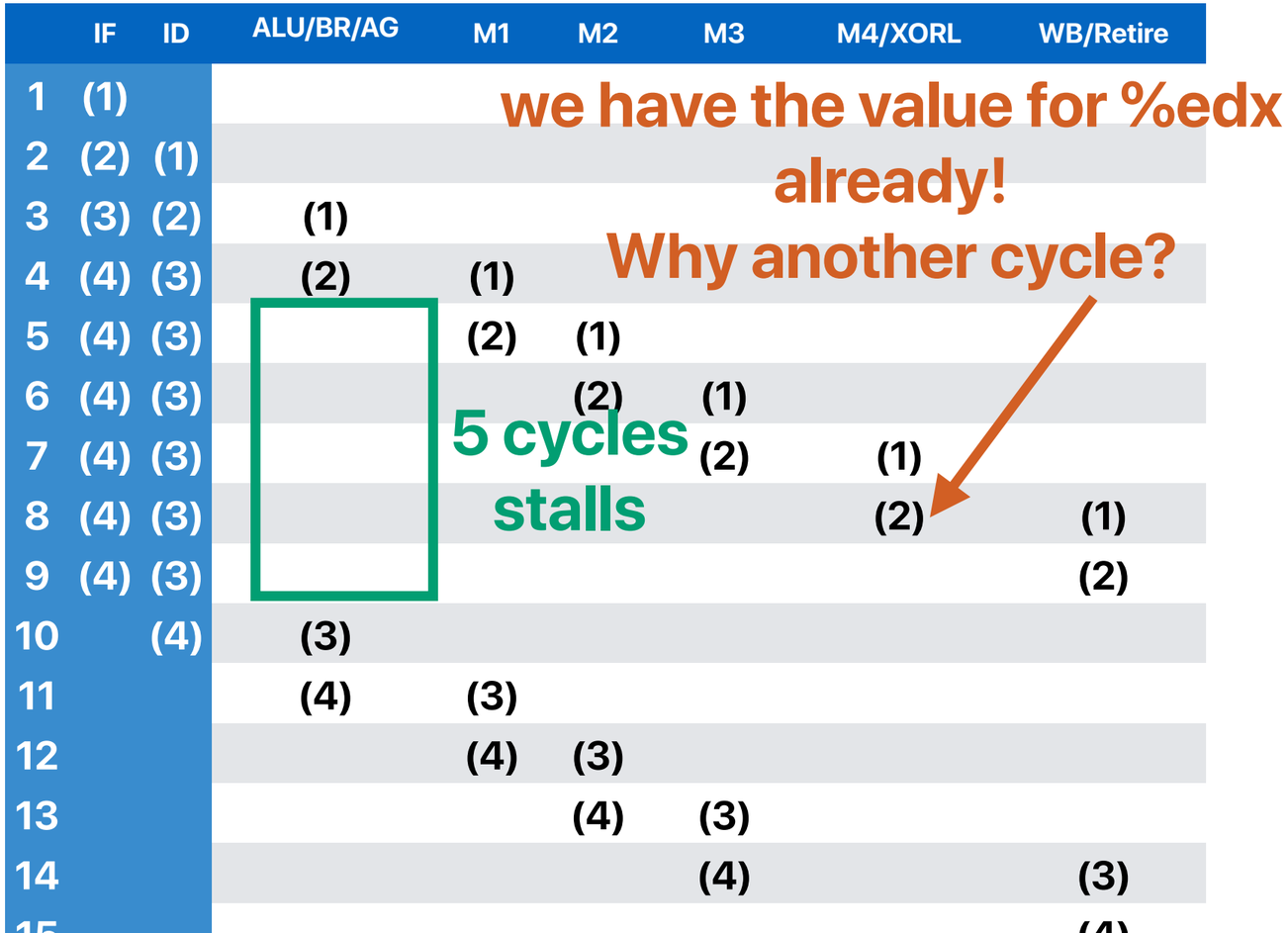


Data hazards?

- 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)
```

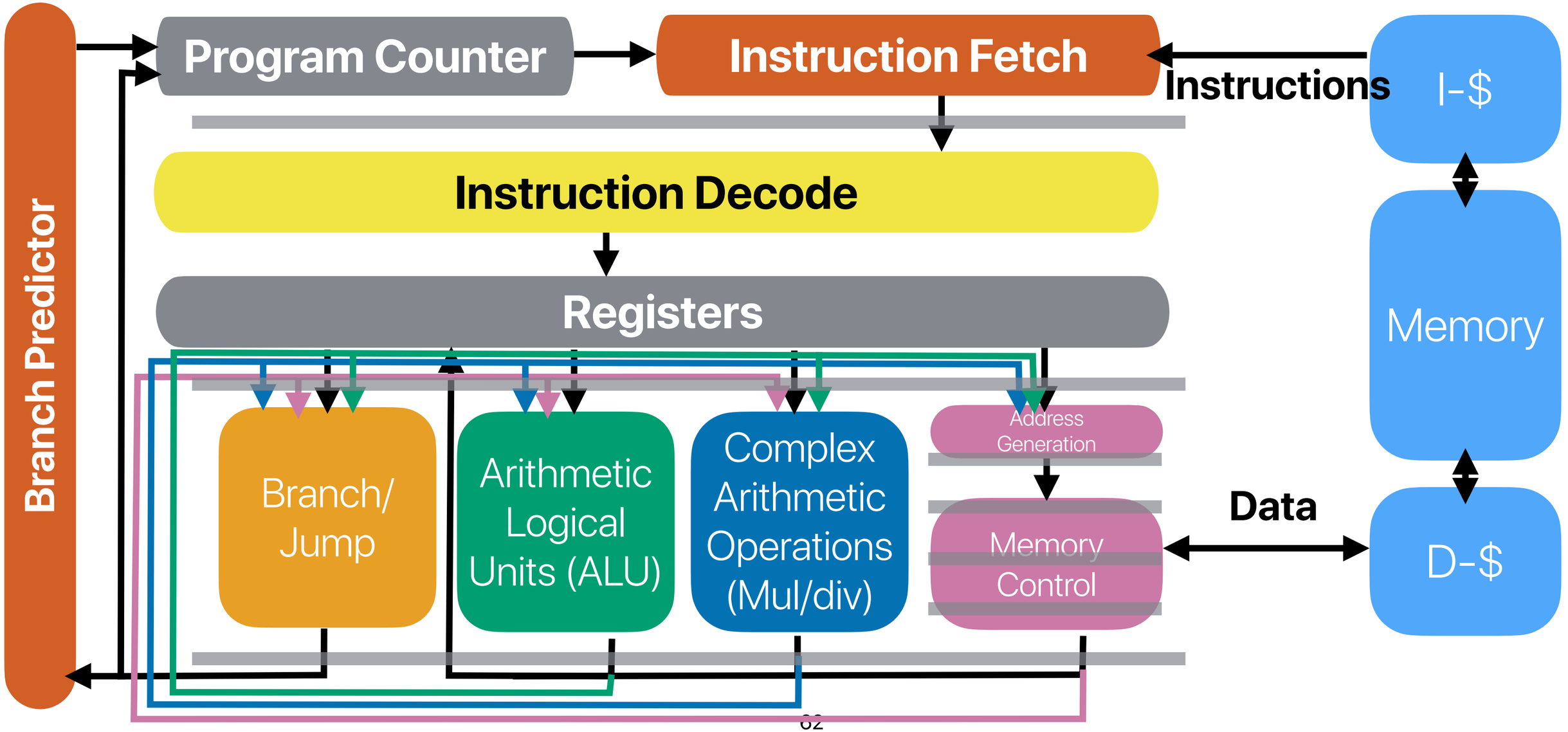
A. 1
B. 2
C. 3
D. 4
E. 5



Solution 2: Data forwarding

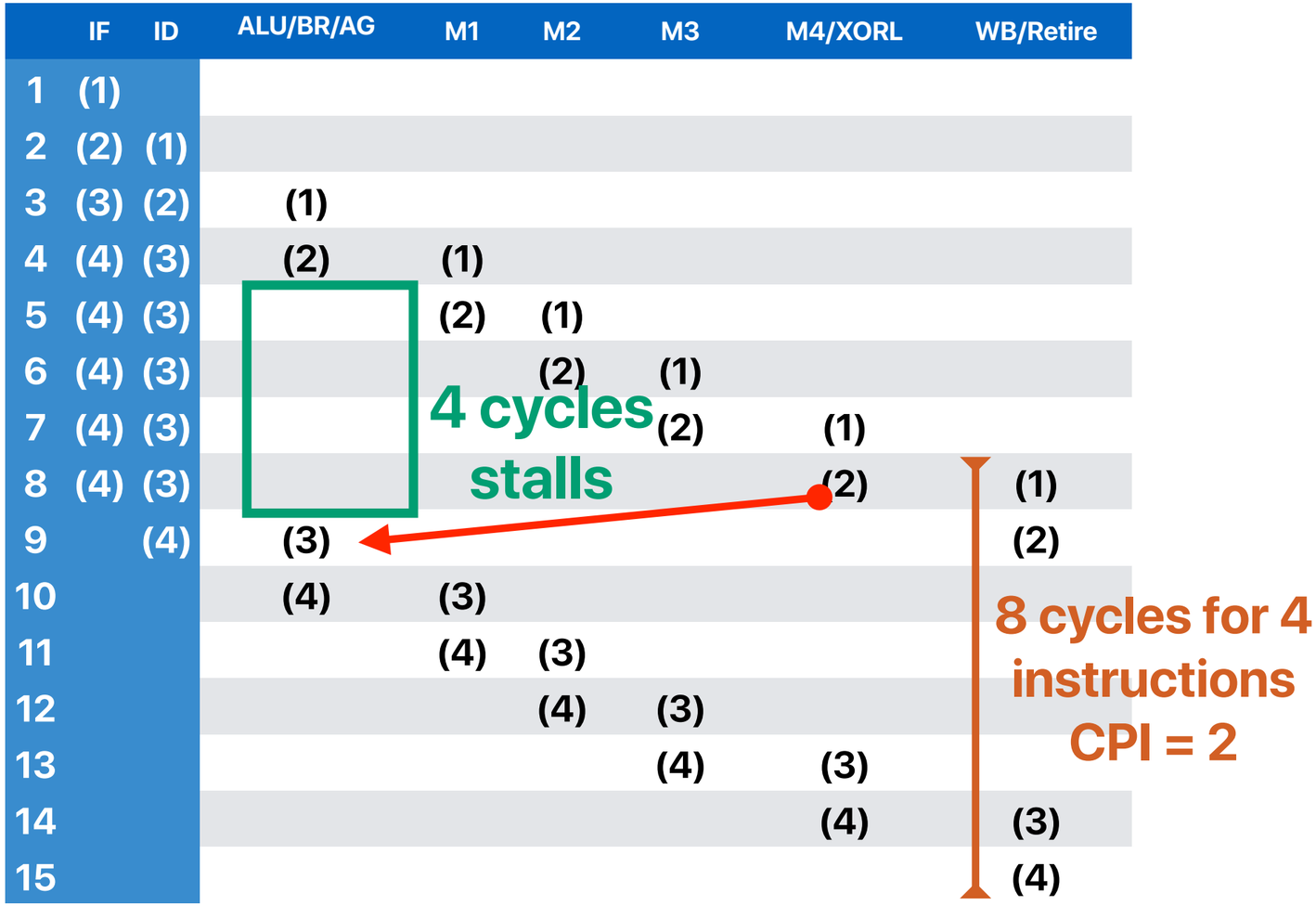
- Add logics/wires to forward the desired values to the demanding instructions

Data "forwarding"



The effect of data forwarding

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



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

How many of data hazards w/ Data Forwarding?

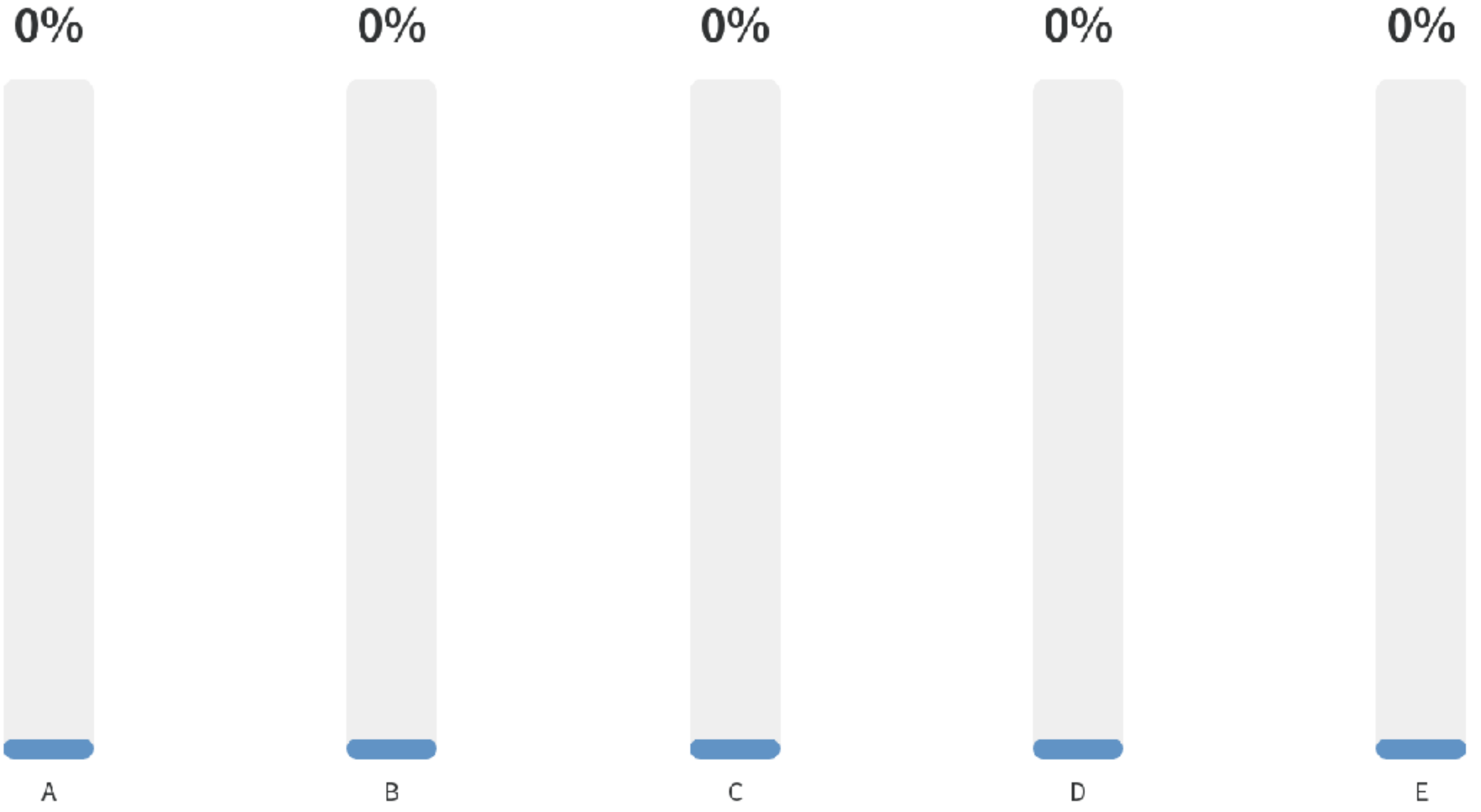
- How many pairs of back-to-back data dependences in the following x86 instructions will result in stalls even with data forwarding and both memory operations & xorl take 5 cycles?

```
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

0



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

How many of data hazards w/ Data Forwarding?

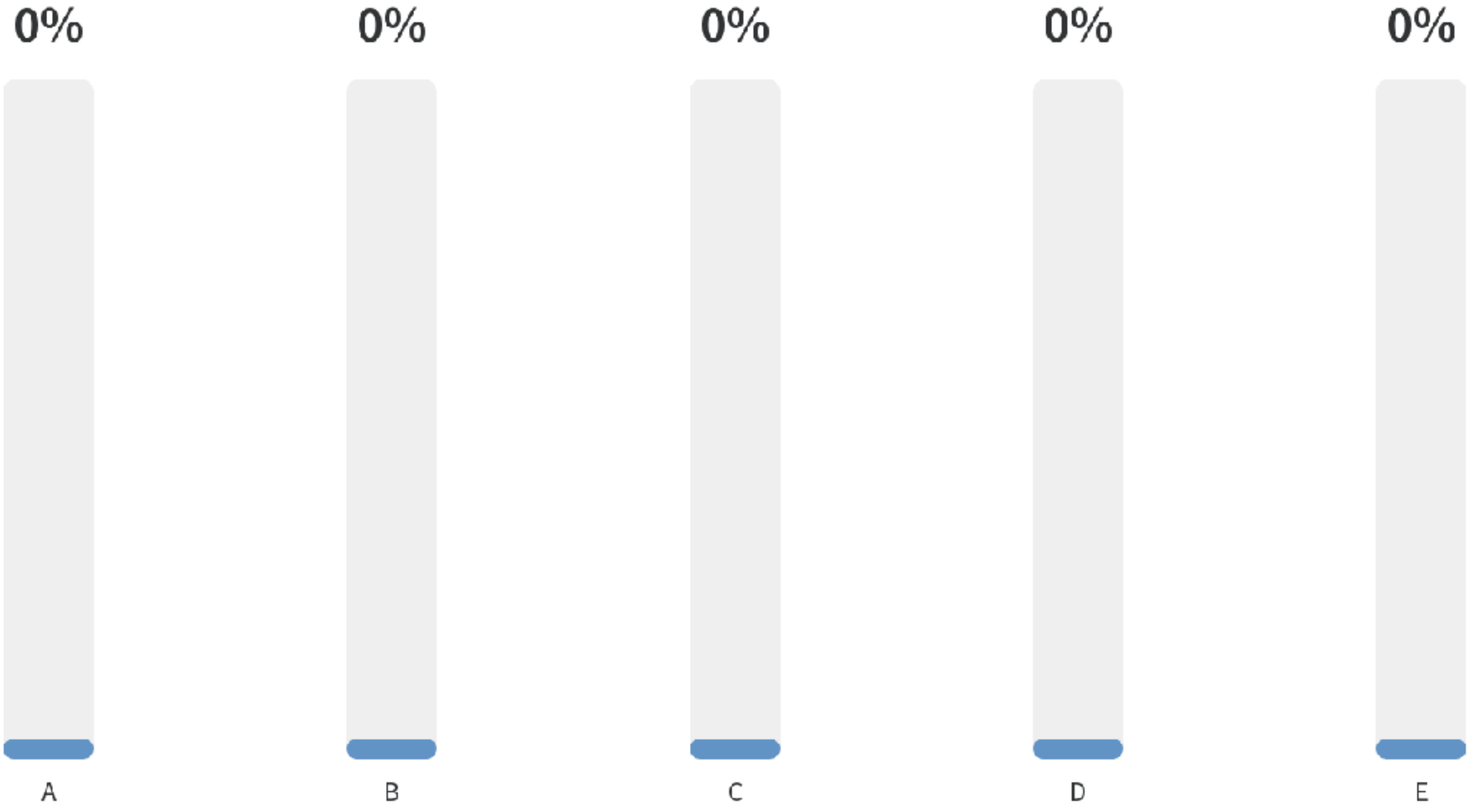
- How many pairs of back-to-back data dependences in the following x86 instructions will result in stalls even with data forwarding and both memory operations & xorl take 5 cycles?

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

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

0



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

	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

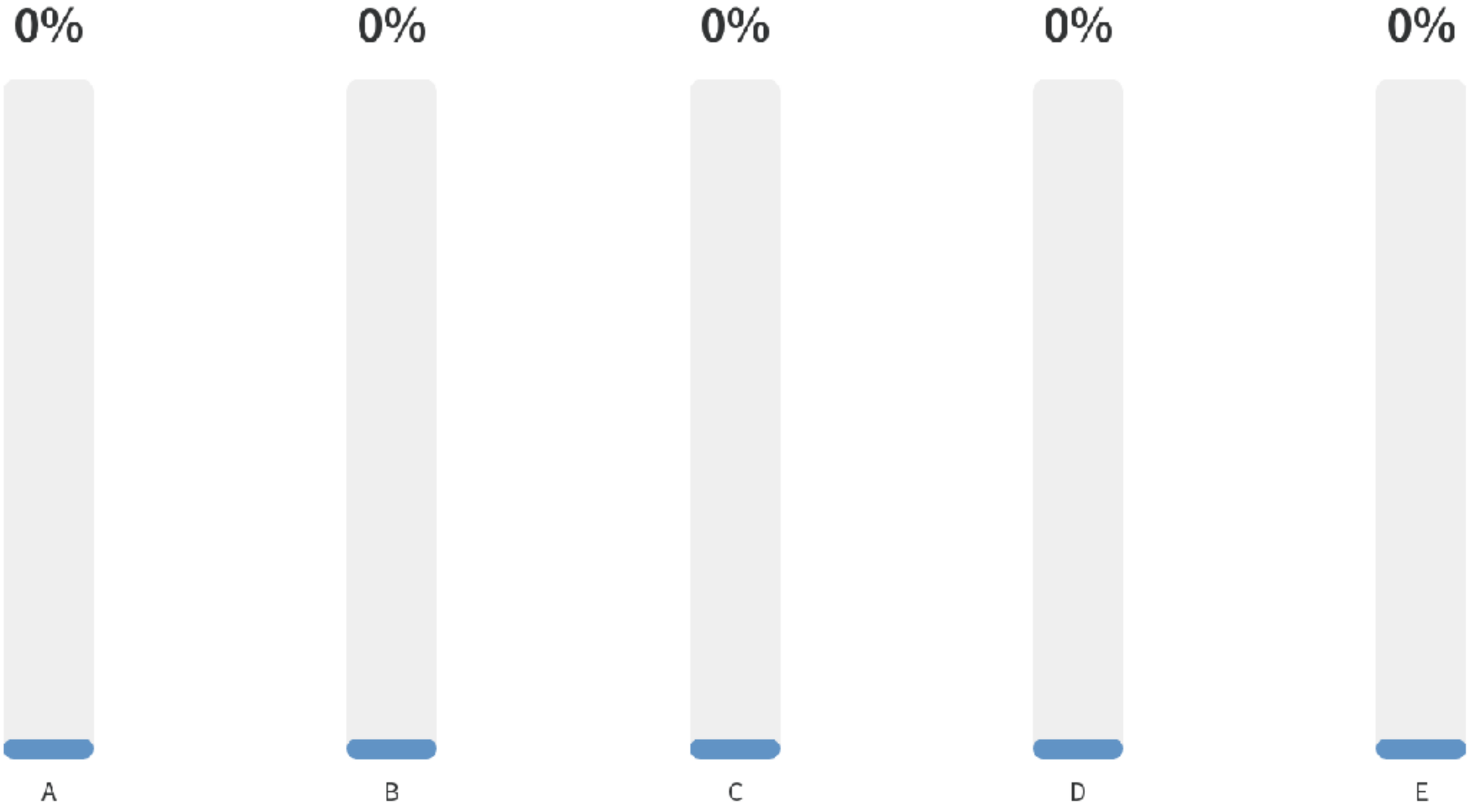
The effect of code optimization

- 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



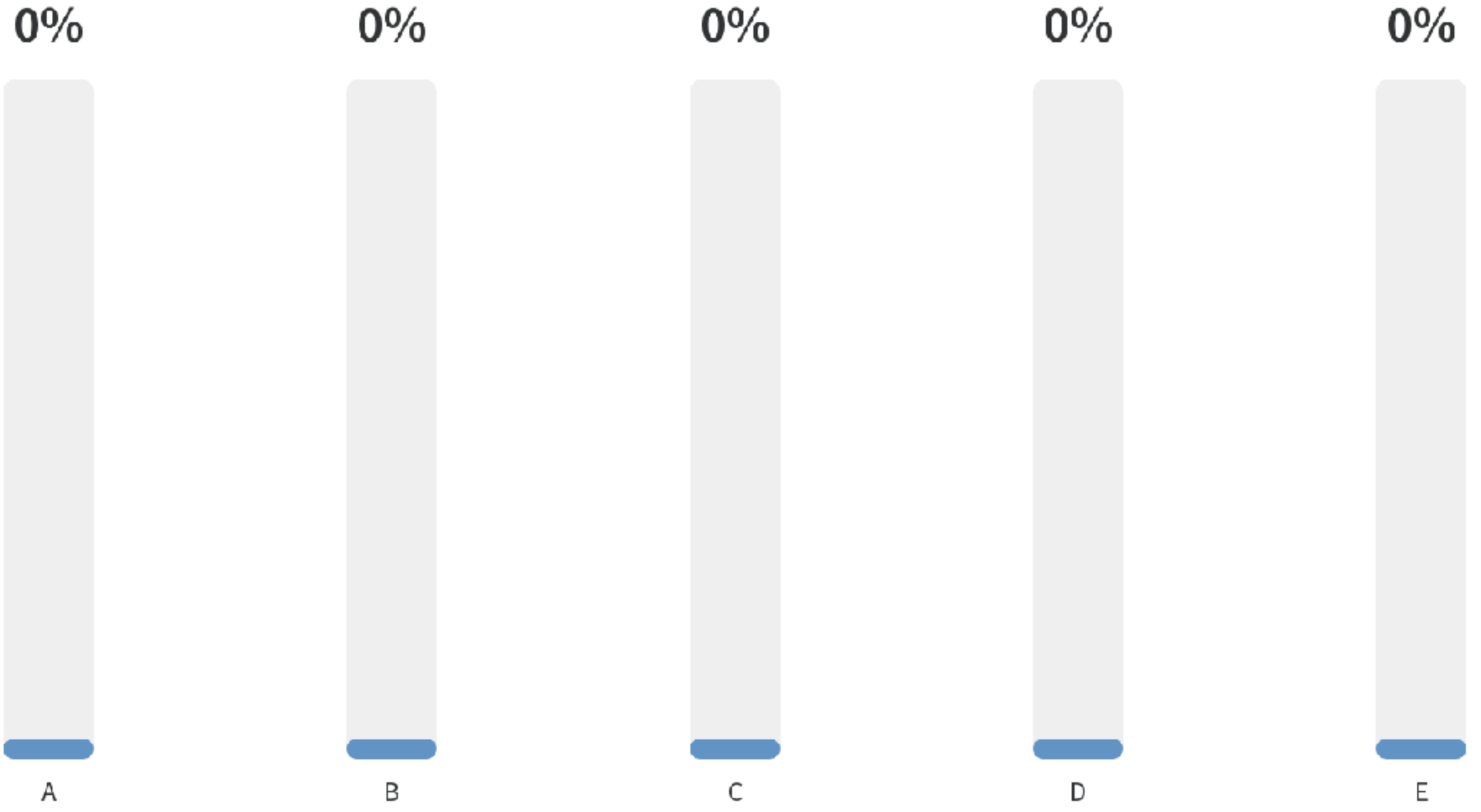
The effect of code optimization

- 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

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)
addq    $8, %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	(7)	(6)	(5)				(2)	(1)
9	(7)	(6)	(5)	(4)				(2)
10	(8)	(7)	(6)	(5)	(4)			(3)
11	(9)	(8)	(7)	(6)	(5)			
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

Missing opportunities

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

Processor can predict what should happen and unroll the loop "dynamically"

```
:
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	(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

Dynamic instruction scheduling/ Out-of-order (OoO) execution

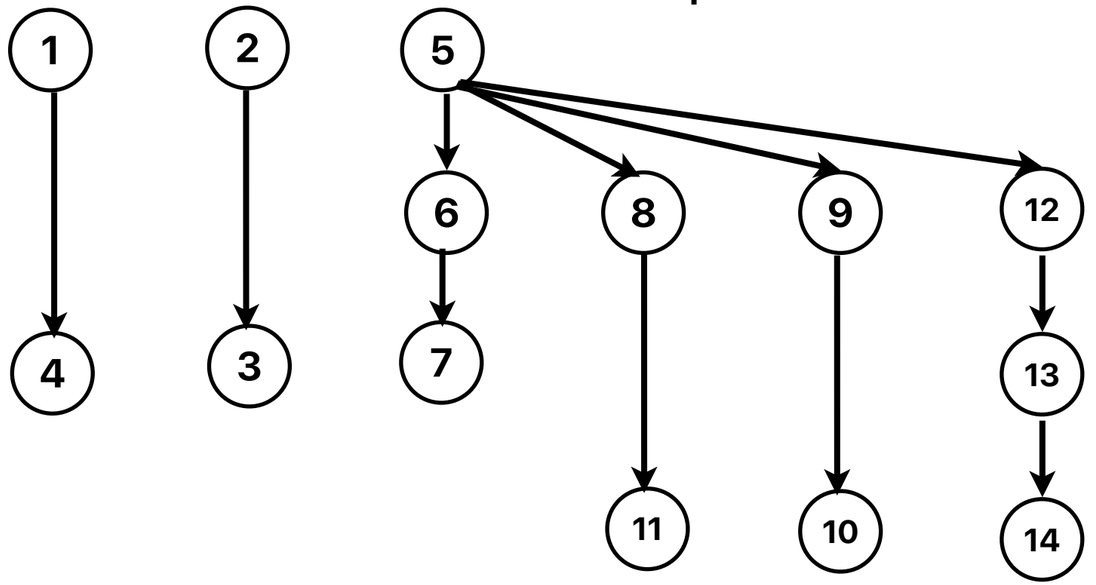
What do you need to execution an instruction?

- Whenever the instruction is decoded — put decoded instruction somewhere
- Whenever the inputs are ready — **all data dependencies are resolved**
- Whenever the target functional unit is available

Scheduling instructions: based on data dependencies

- Draw the data dependency graph, put an arrow if an instruction depends on the other.

```
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
```



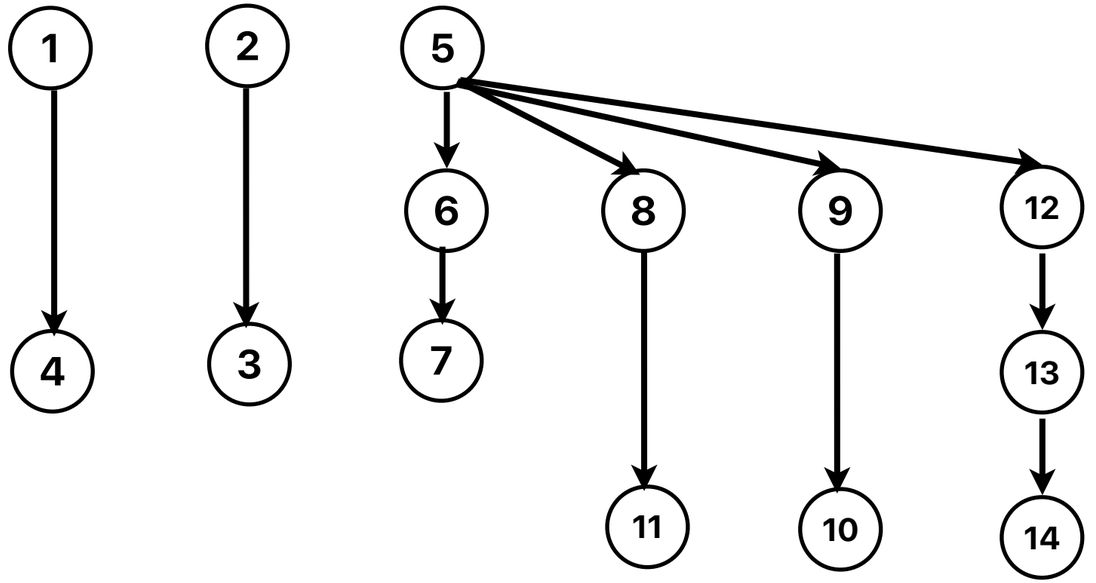
- **In theory**, instructions without dependencies can be executed in parallel or out-of-order
- Instructions with dependencies (on the same path) can never be reordered

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

If we can predict the future …

- Consider the following dynamic instructions:

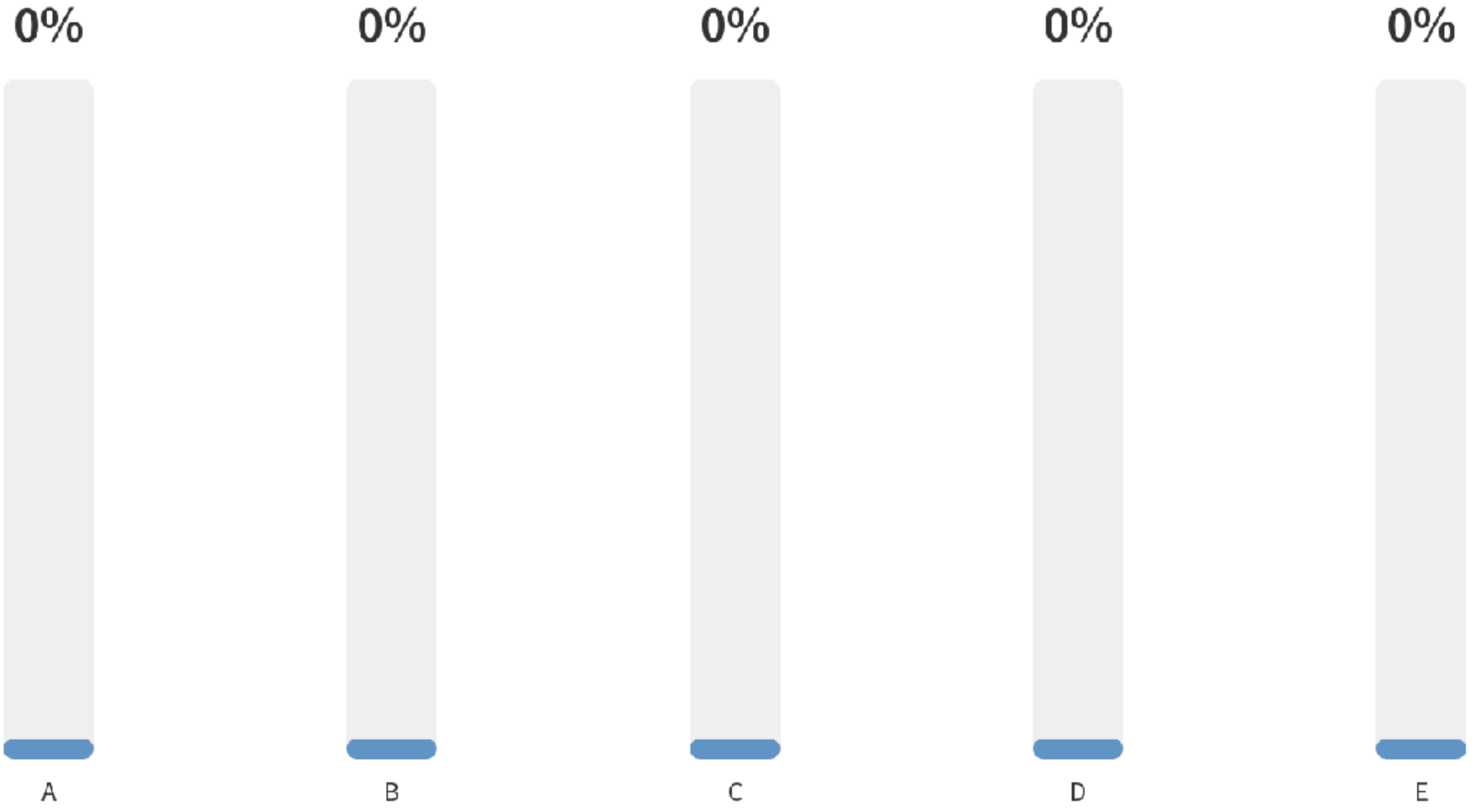
```
① movq    (%rdi,%rax), %rsi
② movq    (%rcx,%rax), %r8
③ movq    %r8, (%rdi,%rax)
④ movq    %rsi, (%rcx,%rax)
⑤ addq    $8, %rax
⑥ cmpq    %r9, %rax
⑦ jne     .L9
⑧ movq    (%rdi,%rax), %rsi
⑨ movq    (%rcx,%rax), %r8
⑩ movq    %r8, (%rdi,%rax)
⑪ movq    %rsi, (%rcx,%rax)
⑫ addq    $8, %rax
⑬ cmpq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



Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

A. (1) and (2)
B. (3) and (5)
C. (4) and (6)
D. (6) and (8)
E. (3) and (8)

0

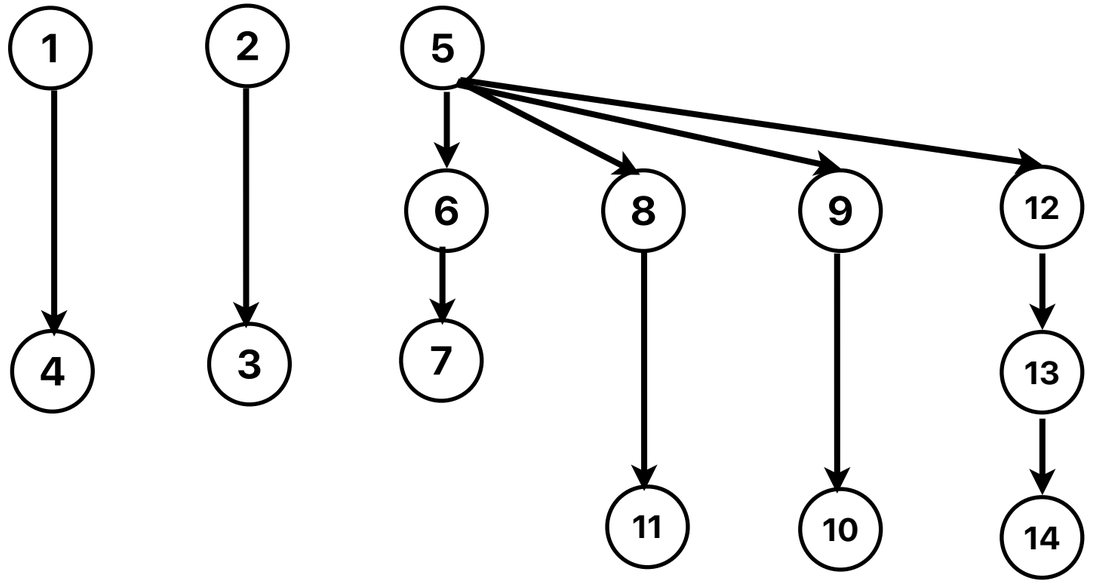


https://www.pollev.com/hungweitseng close in 1:30

If we can predict the future …

- Consider the following dynamic instructions:

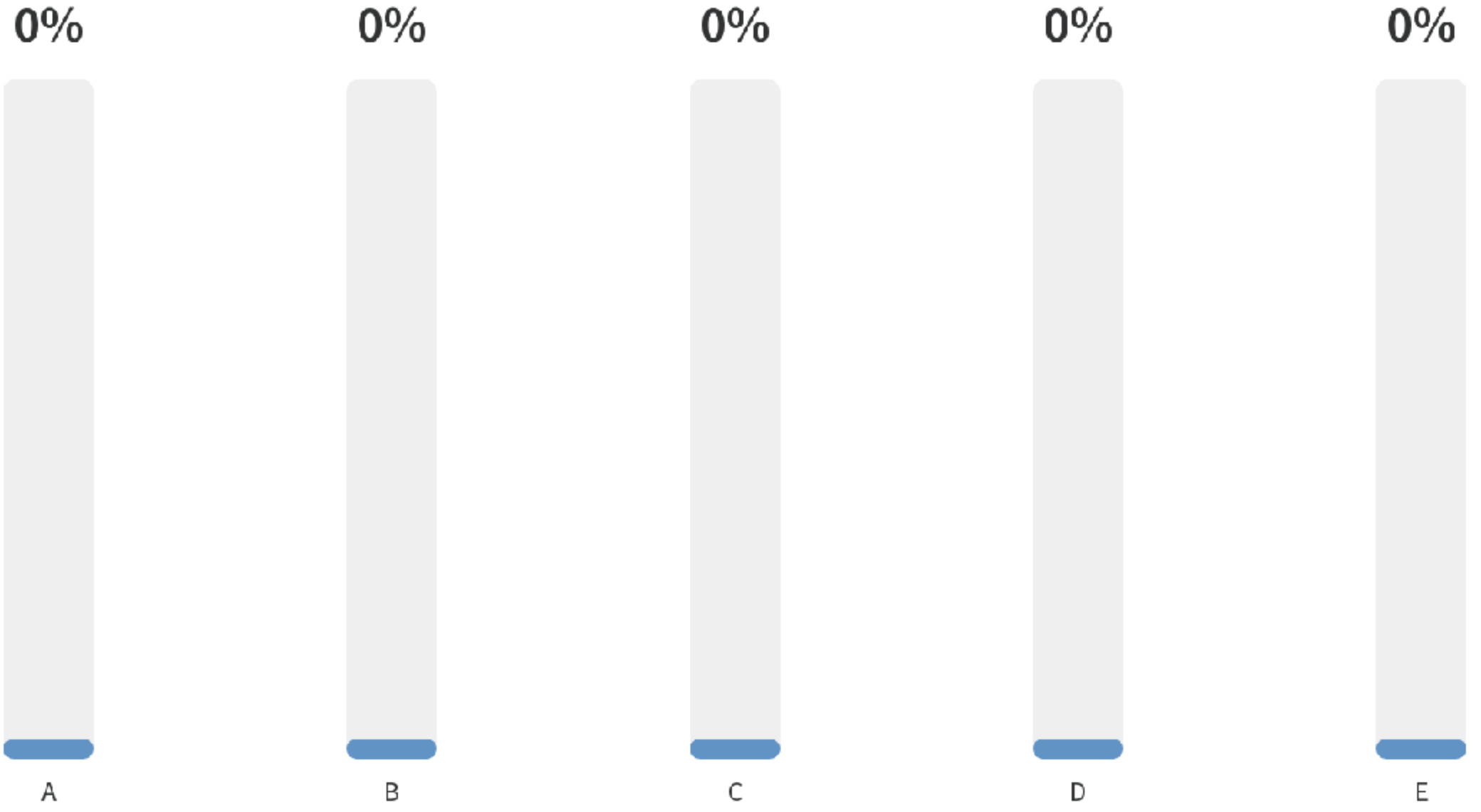
```
① movq    (%rdi,%rax), %rsi
② movq    (%rcx,%rax), %r8
③ movq    %r8, (%rdi,%rax)
④ movq    %rsi, (%rcx,%rax)
⑤ addq    $8, %rax
⑥ cmpq    %r9, %rax
⑦ jne     .L9
⑧ movq    (%rdi,%rax), %rsi
⑨ movq    (%rcx,%rax), %r8
⑩ movq    %r8, (%rdi,%rax)
⑪ movq    %rsi, (%rcx,%rax)
⑫ addq    $8, %rax
⑬ cmpq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



Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

A. (1) and (2)
B. (3) and (5)
C. (4) and (6)
D. (6) and (8)
E. (3) and (8)

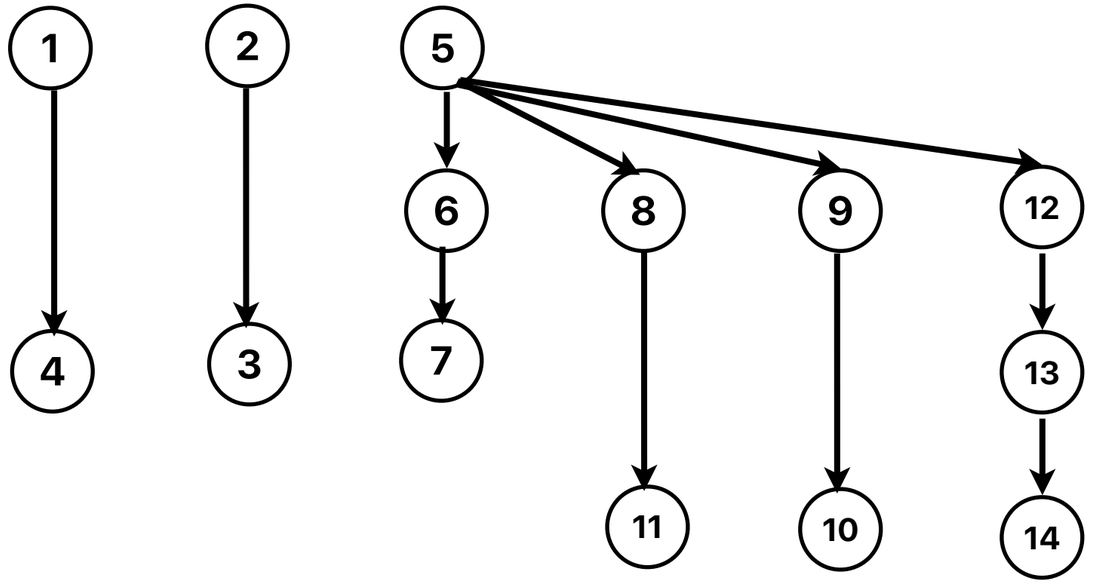
0



If we can predict the future …

- Consider the following dynamic instructions:

```
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
```



Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

A. (1) and (2)

B. (3) and (5)

C. (4) and (6)

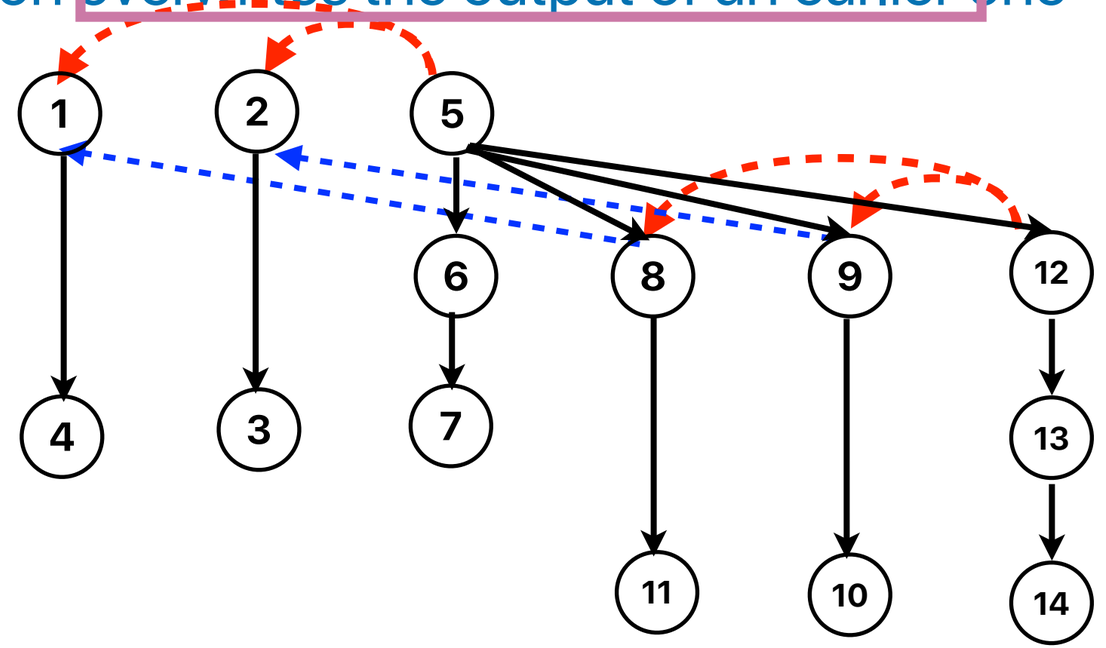
D. (6) and (8)

E. (3) and (8)

False dependencies

- We are still limited by **false dependencies**
- They are not "true" dependencies because they don't have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 5 and 1, 5 and 2, 12 and 8, 12 and 9
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 8 and 1
      9 and 2

```
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
```

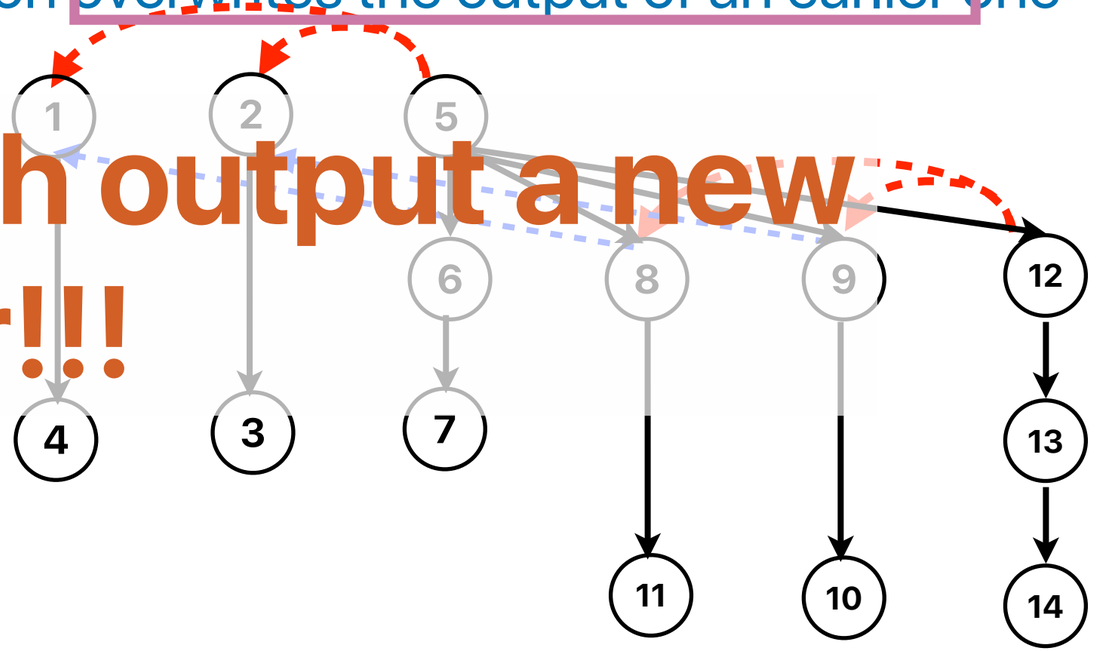


False dependencies

- We are still limited by **false dependencies**
- They are not "true" dependencies because they don't have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 5 and 1, 5 and 2, 12 and 8, 12 and 9
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 8 and 1
    - 9 and 2

We need to give each output a new register!!!

```
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
```



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
- False dependencies limits the freedom of out-of-order execution

What if we can use more registers...

```
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
```

```
①  movq    (%rdi,%rax), %t0
②  movq    (%rcx,%rax), %t1
③  movq    %t1, (%rdi,%rax)
④  movq    %t0, (%rcx,%rax)
⑤  addq    $8, %rax, %t2
⑥  cmpq    %r9, %t2
⑦  jne     .L9
⑧  movq    (%rdi, %t2), %t3
⑨  movq    (%rcx, %t2), %t4
⑩  movq    %t4, (%rdi,%t2)
⑪  movq    %t3, (%rcx,%t2)
⑫  addq    $8, %t2, %t5
⑬  cmpq    %r9, %t5
⑭  jne     .L9
```

All false dependencies are gone!!!

The mechanism of OoO: Register renaming + speculative execution

- K. C. Yeager, “The MIPS R10000 superscalar microprocessor,” in IEEE Micro, vol. 16, no. 2, pp. 28-41, April 1996.

Register renaming + OoO

- Redirecting the output of an instruction instance to a **physical register**
- Redirecting inputs of an instruction instance from **architectural registers** to correct **physical registers**
  - You need a mapping table between architectural and physical registers
  - You may also need reference counters to reclaim physical registers
- OoO: Executing an instruction all operands are ready (the values of depending physical registers are generated)
  - You will need an **issue logic** to **issue** an instruction to the target functional unit

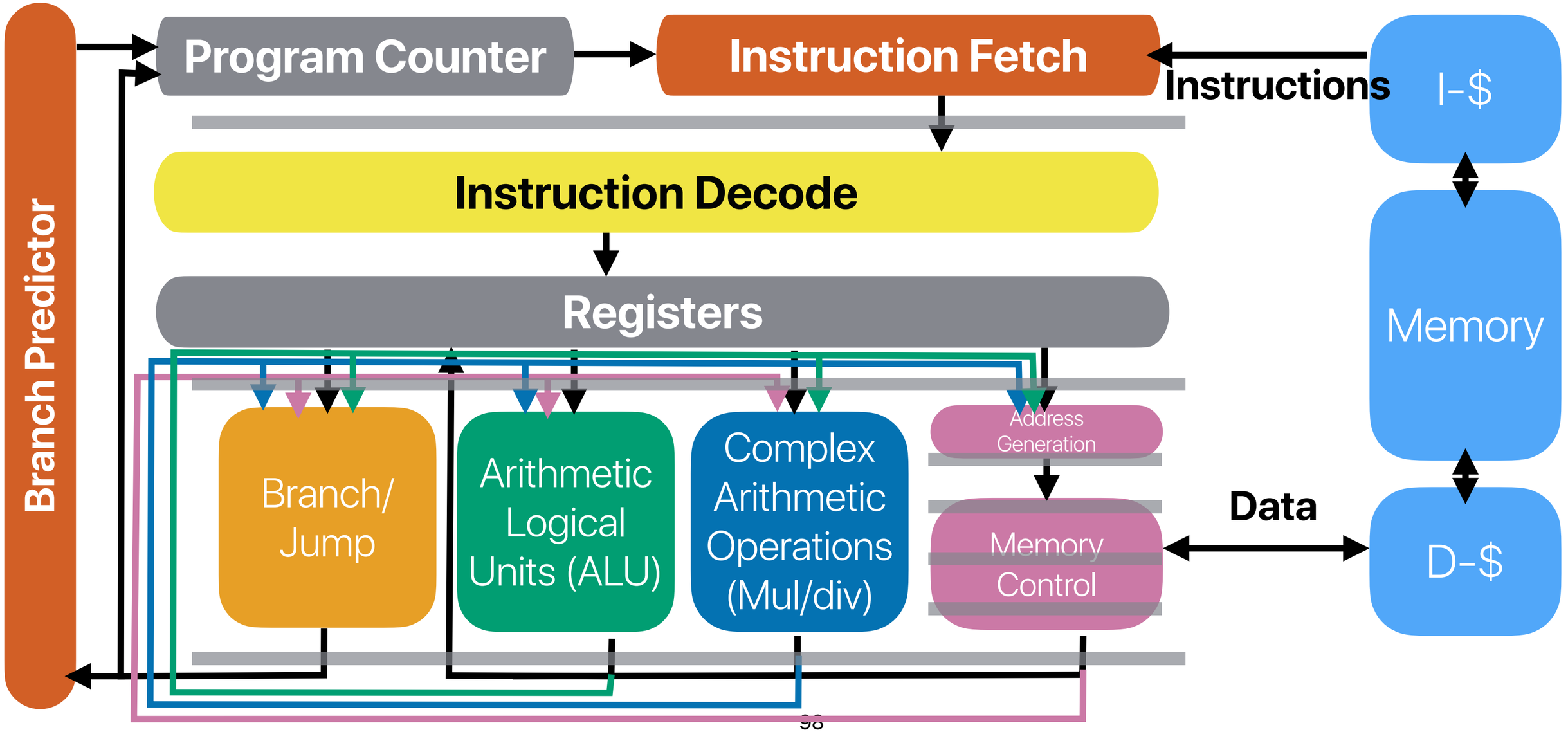
Can we really execute instructions OoO?

- Exceptions may occur anytime — divided by 0, page fault
  - A later instruction cannot write back its own result otherwise the architectural states won't be correct
  - Instructions after the one causes the exception should not be executed
- Hardware can schedule instruction across branch instructions with the help of branch prediction
  - Fetch instructions according to the branch prediction
  - However, branch predictor can never be perfect

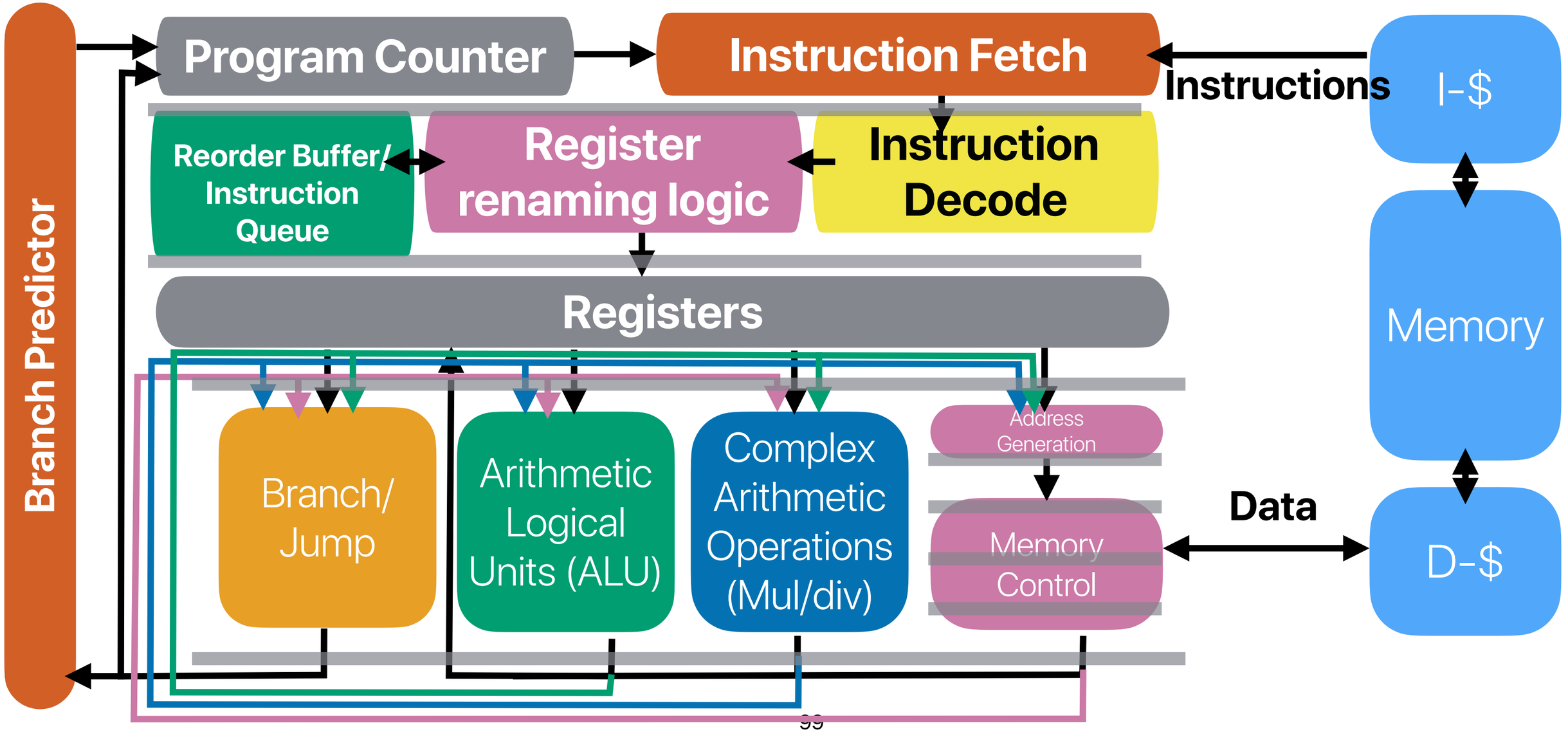
Speculative Execution

- **Speculative** execution mode: an executing instruction is considered as **speculative** before the processor hasn't determined if the instruction should be executed or not
- Reorder buffer (ROB)
  - The processor allocates an entry for each instruction in a reorder buffer
  - Store results in **reorder buffer and physical registers** when the instruction is still speculative
  - If an earlier instruction failed to commit due to an exception or mis-prediction, the physical registers and all ROB entries after the failed-to-commit instruction are flushed
- Commit/Retire
  - Present the execution result to the running program and in architectural registers when all prior instructions are non-speculative
  - Release the ROB entry

Data "forwarding"



Register renaming + OoO + RoB



Register renaming

```
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	
rcx	
rdi	
rsi	
r8	

	Valid	Value	In use		Valid	Value	In use
P1				P6			
P2				P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	
rcx	
rdi	
rsi	P1
r8	

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2				P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	0		1	P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	0		1	P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8												
9												
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	0		1	P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8												
9												
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	P3
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	0		1	P7			
P3	0		1	P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9												
10												
11												
12												
13												
14												
15												
16												

Instruction (5) is running ahead of (3)

	Physical Register
rax	P3
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	0		1	P7			
P3	0		1	P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
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
```

Only 1 of them can have a instruction at the same cycle

Instruction (4) is running ahead of (3)

Instruction (5) is running ahead of (3)

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				(1)(5)
10												
11												
12												
13												
14												
15												
16												

	Physical Register
rax	P3
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	0		1	P7			
P3	1		1	P8			
P4				P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10												
11												
12												
13												
14												
15												
16												

Retire/Commit (1)

	Physical Register
rax	P3
rcx	
rdi	
rsi	P1
r8	P2

	Valid	Value	In use
P1	1		1
P2	0		1
P3	1		1
P4			
P5			

	Valid	Value	In use
P6			
P7			
P8			
P9			
P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
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
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							(2)(5)
11												
12												
13												
14												
15												
16												

	Physical Register
rax	P3
rcx	
rdi	
rsi	P4
r8	P2

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	1		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5				P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax
⑬  cmpq %r9, %rax
⑭  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							~~(2)~~(5)
11	(11)	(10)	(7)(8)(9)		(3)	(4)			(6)			
12												
13												
14												
15												
16												

	Physical Register
rax	P3
rcx	
rdi	
rsi	P4
r8	P5

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	1		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax
⑬  cmpq %r9, %rax
⑭  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							~~(2)~~(5)
11	(11)	(10)	(7)(8)(9)		(3)	(4)			(6)			
12	(12)	(11)	(8)(9)(10)			(3)	(4)				(7)	(5)(6)
13												
14												
15												
16												

	Physical Register
rax	P3
rcx	
rdi	
rsi	P4
r8	P5

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	1		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax
⑬  cmpq %r9, %rax
⑭  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							~~(2)~~(5)
11	(11)	(10)	(7)(8)(9)		(3)	(4)			(6)			
12	(12)	(11)	(8)(9)(10)			(3)	(4)				(7)	(5)(6)
13	(13)	(12)	(9)(10)(11)	(8)			(3)	(4)				(5)(6)(7)
14												
15												
16												

	Physical Register
rax	P3
rcx	
rdi	
rsi	P4
r8	P5

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	1		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							~~(2)~~(5)
11	(11)	(10)	(7)(8)(9)		(3)	(4)			(6)			
12	(12)	(11)	(8)(9)(10)			(3)	(4)				(7)	(5)(6)
13	(13)	(12)	(9)(10)(11)	(8)			(3)	(4)				(5)(6)(7)
14	(14)	(13)	(10)(11)(12)	(9)	(8)			(3)				(4)(5)(6)(7)
15												
16												

	Physical Register
rax	P6
rcx	
rdi	
rsi	P4
r8	P5

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6	0		1
P2	1		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							~~(2)~~(5)
11	(11)	(10)	(7)(8)(9)		(3)	(4)			(6)			
12	(12)	(11)	(8)(9)(10)			(3)	(4)				(7)	(5)(6)
13	(13)	(12)	(9)(10)(11)	(8)			(3)	(4)				(5)(6)(7)
14	(14)	(13)	(10)(11)(12)	(9)	(8)			(3)				(4)(5)(6)(7)
15	(15)	(14)	(10)(11)(13)		(9)	(8)			(12)			(3)(4)(5)(6)(7)
16												

	Physical Register
rax	P6
rcx	
rdi	
rsi	P4
r8	P5

	Valid	Value	In use
P1	1		1
P2	1		1
P3	1		1
P4	0		1
P5	0		1

	Valid	Value	In use
P6	0		1
P7			
P8			
P9			
P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)											
2	(2)	(1)										
3	(3)	(2)	(1)									
4	(4)	(3)	(2)	(1)								
5	(5)	(4)	(3)	(2)	(1)							
6	(6)	(5)	(3)(4)		(2)	(1)						
7	(7)	(6)	(3)(4)(5)			(2)	(1)					
8	(8)	(7)	(3)(4)(6)				(2)	(1)	(5)			
9	(9)	(8)	(3)(6)(7)	(4)				(2)				~~(1)~~(5)
10	(10)	(9)	(6)(7)(8)	(3)	(4)							~~(2)~~(5)
11	(11)	(10)	(7)(8)(9)		(3)	(4)			(6)			
12	(12)	(11)	(8)(9)(10)			(3)	(4)				(7)	(5)(6)
13	(13)	(12)	(9)(10)(11)	(8)			(3)	(4)				(5)(6)(7)
14	(14)	(13)	(10)(11)(12)	(9)	(8)			(3)				(4)(5)(6)(7)
15	(15)	(14)	(10)(11)(13)		(9)	(8)			(12)			~~(3)(4)(5)(6)(7)~~
16	(16)	(15)	(10)(11)(14)			(9)	(8)		(13)			(12)

	Physical Register
rax	P6
rcx	
rdi	
rsi	P4
r8	P5

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6	1		1
P2	1		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

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

Register renaming

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

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

Register renaming

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

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

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

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

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

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

Register renaming

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

Only 1 of them can have a instruction at the same cycle

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

Register renaming

Only 1 of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

Register renaming

Only 1 of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi → P1
②  movq (%rcx,%rax), %r8  → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax          → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi → P4
⑨  movq (%rcx,%rax), %r8  → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax          → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

Register renaming

Only 1 of them can have a instruction at the same cycle

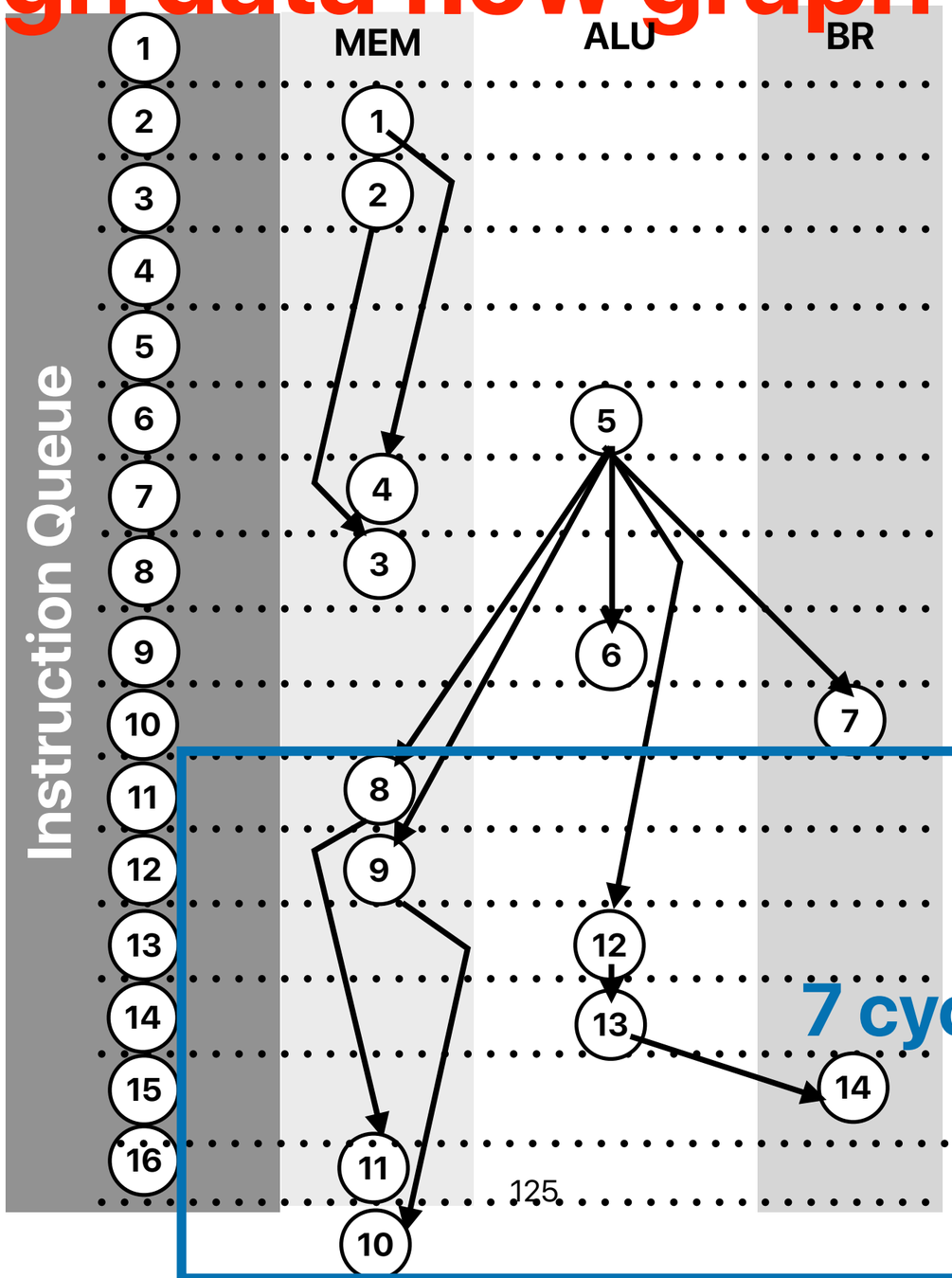
```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

7 cycles for 7 instructions
CPI = 1

Through data flow graph analysis

```
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
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```



7 cycles every iteration

CPI = $\frac{7}{7} = 1$!

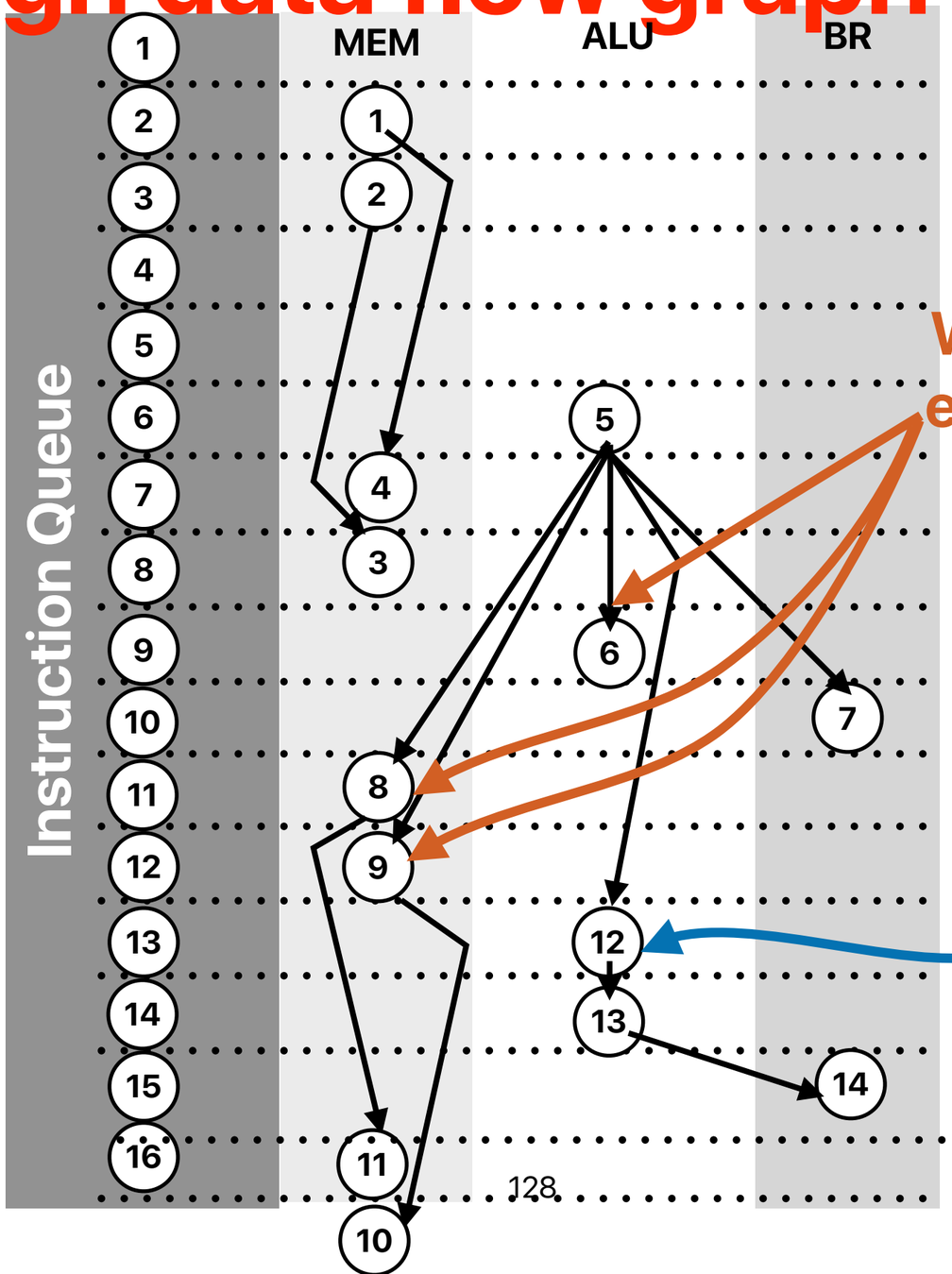
Takeaways: data hazards

- 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
- False dependencies limits the freedom of out-of-order execution
- Register renaming + Speculative execution enables more efficient execution by dynamically scheduling instructions whenever their data dependencies are resolved

If CPI==1 the limitation?

Through data flow graph analysis

```
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
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```



We cannot issue them earlier simply because structural hazards!

We could have this executed earlier if it's in the queue earlier

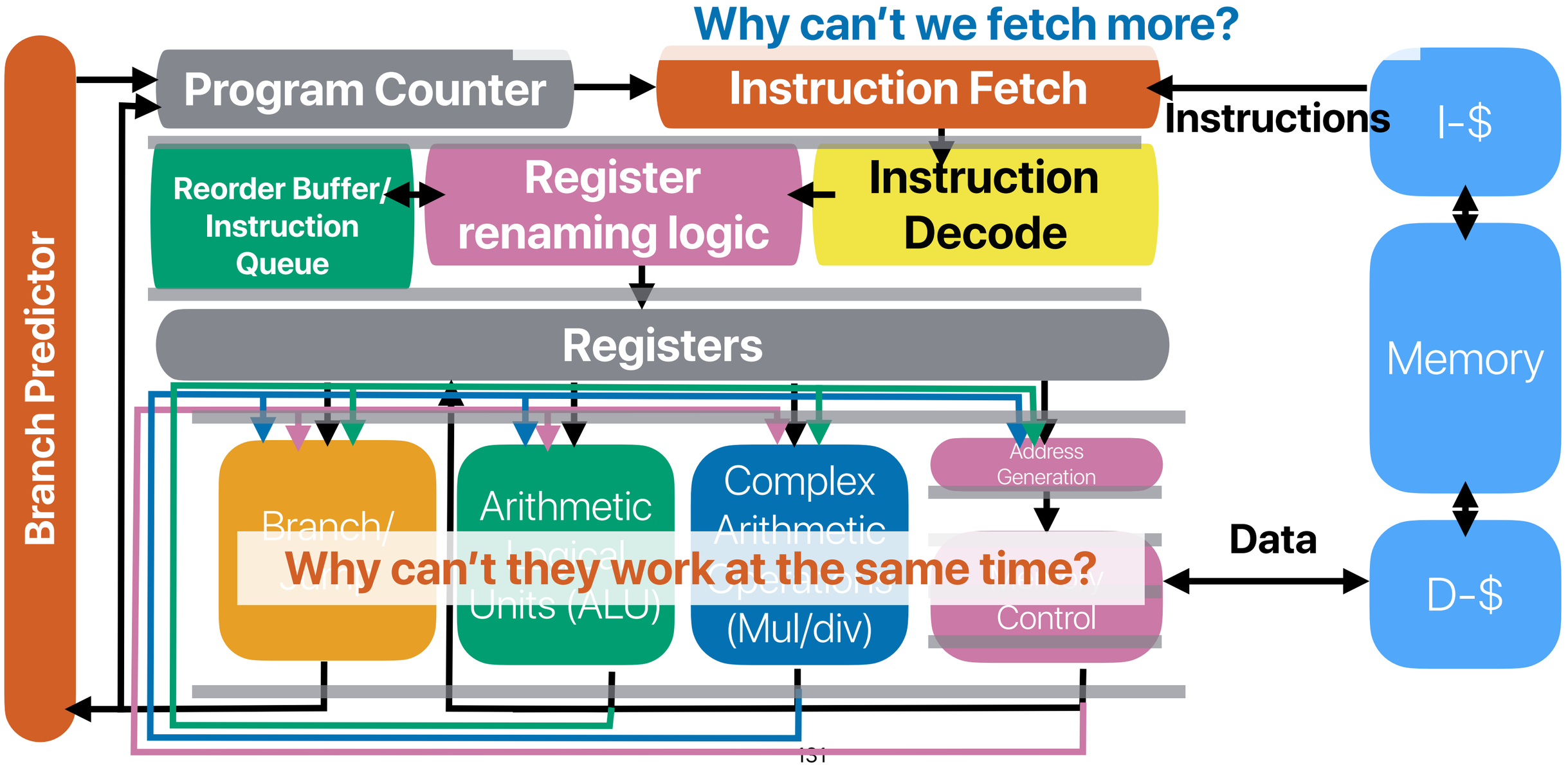
Super Scalar

Superscalar

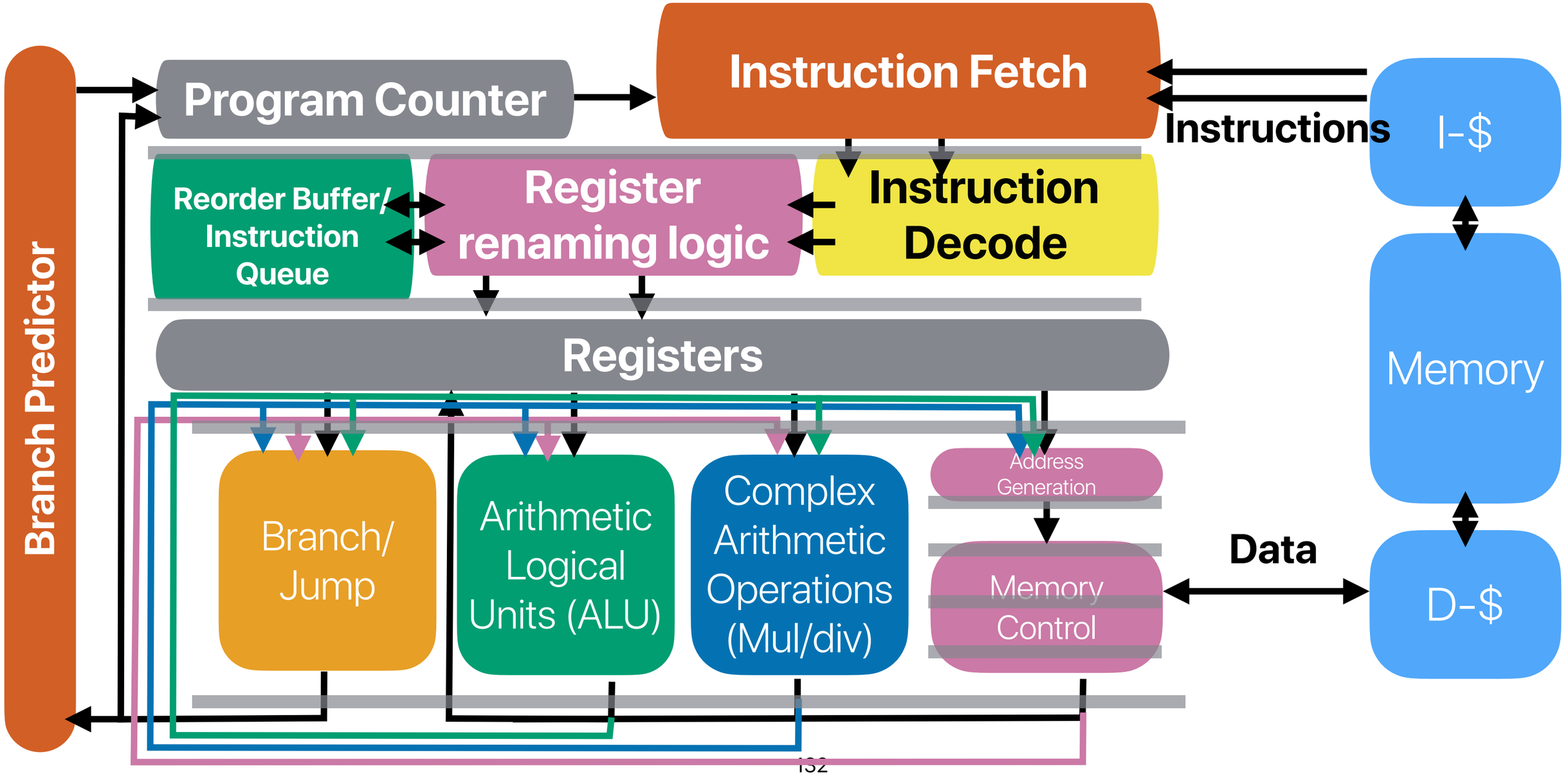
- Since we have many functional units now, we should fetch/decode more instructions each cycle so that we can have more instructions to issue!
- Super-scalar: fetch/decode/issue more than one instruction each cycle
  - **Fetch width:** how many instructions can the processor fetch/decode each cycle
  - **Issue width**: how many instructions can the processor issue each cycle
- The theoretical CPI should now be

$$\frac{1}{min(issue\ \ width, fetch\ \ width, decode\ \ width)}$$

Register renaming + OoO + RoB



Register renaming + SuperScalar



2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6	(11)(12)	(9)(10)	(3)(4)((6)(7)(8)		(2)	(1)						
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6	(11)(12)	(9)(10)	(3)(4)((6)(7)(8)		(2)	(1)						
7	(13)(14)	(11)(12)	(3)(4)((6)(7)(9)(10)	(8)		(2)	(1)		(5)			(5)
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6	(11)(12)	(9)(10)	(3)(4)((6)(7)(8)		(2)	(1)						
7	(13)(14)	(11)(12)	(3)(4)((6)(7)(9)(10)	(8)		(2)	(1)		(5)			(5)
8	(15)(16)	(13)(14)	(3)(4)(7)(10)(11)(12)	(9)	(8)		(2)	(1)	(6)			(5)(6)
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6	(11)(12)	(9)(10)	(3)(4)((6)(7)(8)		(2)	(1)						
7	(13)(14)	(11)(12)	(3)(4)((6)(7)(9)(10)	(8)		(2)	(1)		(5)			(5)
8	(15)(16)	(13)(14)	(3)(4)(7)(10)(11)(12)	(9)	(8)		(2)	(1)	(6)			(5)(6)
9	(17)(18)	(15)(16)	(3)(10)(11)(12)(13)(14)	(4)	(9)	(8)		(2)			(7)	(1)(5)(6)
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6	(11)(12)	(9)(10)	(3)(4)((6)(7)(8)		(2)	(1)						
7	(13)(14)	(11)(12)	(3)(4)((6)(7)(9)(10)	(8)		(2)	(1)		(5)			(5)
8	(15)(16)	(13)(14)	(3)(4)(7)(10)(11)(12)	(9)	(8)		(2)	(1)	(6)			(5)(6)
9	(17)(18)	(15)(16)	(3)(10)(11)(12)(13)(14)	(4)	(9)	(8)		(2)			(7)	(1)(5)(6)
10	(19)(20)	(17)(18)	(10)(11)(13)(14)(15)(16)	(3)	(4)	(9)	(8)		(12)			(2)(5)(6)(7)
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

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

2-issue SS + Register renaming +OoO

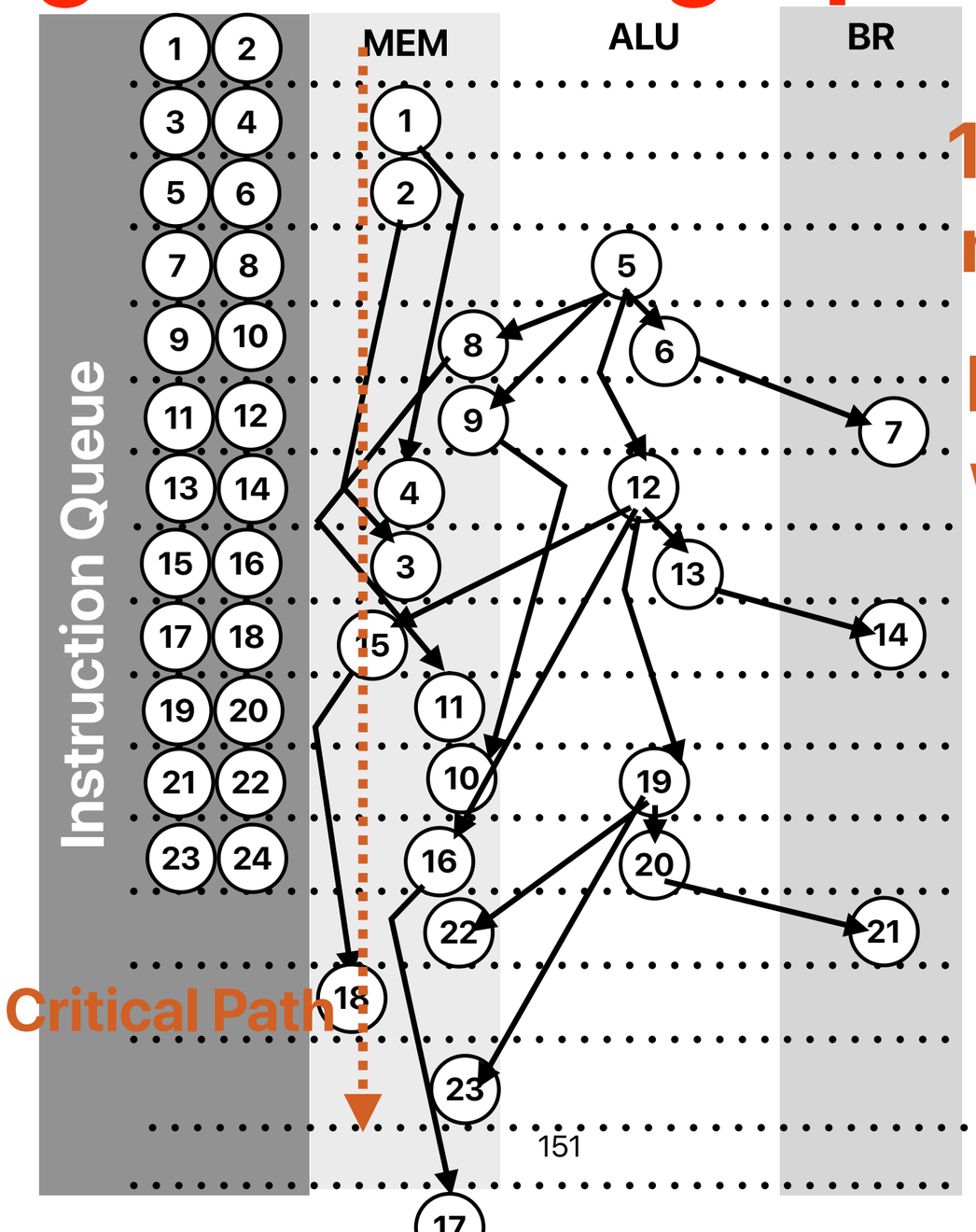
2 issue: "2" of them can have a instruction at the same cycle

```
①  movq (%rdi,%rax), %rsi   → P1
②  movq (%rcx,%rax), %r8    → P2
③  movq %r8, (%rdi,%rax)
④  movq %rsi, (%rcx,%rax)
⑤  addq $8, %rax            → P3
⑥  cmpq %r9, %rax
⑦  jne  .L9
⑧  movq (%rdi,%rax), %rsi   → P4
⑨  movq (%rcx,%rax), %r8    → P5
⑩  movq %r8, (%rdi,%rax)
⑪  movq %rsi, (%rcx,%rax)
⑫  addq $8, %rax            → P6
⑬  cmpq %r9, %rax
⑭  jne  .L9
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
```

	IF	ID	REN	AG	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1)(2)											
2	(3)(4)	(1)(2)										
3	(5)(6)	(3)(4)	(1)(2)									
4	(7)(8)	(5)(6)	(2)(3)(4)	(1)								
5	(9)(10)	(7)(8)	(3)(4)(5)(6)	(2)	(1)							
6	(11)(12)	(9)(10)	(3)(4)((6)(7)(8)		(2)	(1)						
7	(13)(14)	(11)(12)	(3)(4)((6)(7)(9)(10)	(8)		(2)	(1)		(5)			(5)
8	(15)(16)	(13)(14)	(3)(4)(7)(10)(11)(12)	(9)	(8)		(2)	(1)	(6)			(5)(6)
9	(17)(18)	(15)(16)	(3)(10)(11)(12)(13)(14)	(4)	(9)	(8)		(2)			(7)	~~(1)~~(5)(6)
10	(19)(20)	(17)(18)	(10)(11)(13)(14)(15)(16)	(3)	(4)	(9)	(8)		(12)			~~(2)~~(5)(6)(7)
11	(21)(22)	(19)(20)	(10)(11)(14)(16)(17)(18)	(15)	(3)	(4)	(9)	(8)	(13)			(5)(6)(7)(12)
12		(21)(22)	(16)(17)(18)(19)(20	(11)	(15)	(3)	(4)	(9)			(14)	(5)(6)(7)(8)(12)(13)
13			(16)(17)(18)(20)(21)(22)	(10)	(11)	(15)	(3)	(4)	(19)			(5)(6)(7)(8)(9)(12)(13)(14)
14				(16)	(10)	(11)	(15)	(3)	(20)			(4)(5)(6)(7)(8)(9)(12)(13)(14)(19)
15					(16)	(10)	(11)	(15)			(21)	~~(3)(4)(5)(6)(7)(8)~~(9)(12)(13)(14)(19)(20)
16				(17)		(16)	(10)	(11)				(12)(13)(14)(15)(19)(20(21)
17					(17)		(16)	(10)				(11)(12)(13)(14)(15)(19)(20(21)
18						(17)		(16)				~~(10)(11)(12)(13)(14)(15)~~(19)(20)(21)
19				(18)			(17)					~~(16)~~(19)(20)(21)
20					(18)			(17)				(19)(20)(21)

Through data flow graph analysis

```
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
⑮  movq (%rdi,%rax), %rsi
⑯  movq (%rcx,%rax), %r8
⑰  movq %r8, (%rdi,%rax)
⑱  movq %rsi, (%rcx,%rax)
⑲  addq $8, %rax
⑳  cmpq %r9, %rax
㉑  jne  .L9
㉒  movq (%rdi,%rax), %rsi
㉓  movq (%rcx,%rax), %r8
㉔  movq %r8, (%rdi,%rax)
㉕  movq %rsi, (%rcx,%rax)
㉖  addq $8, %rax
㉗  cmpq %r9, %rax
```



12 cycles for every 11 memory instructions

If we have 11 loops, it will have 44 memory instructions, 77 instructions in total and take 48 cycles

CPI:

$$\frac{48}{77} = 0.62$$

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
- False dependencies limits the freedom of out-of-order execution
- Register renaming + Speculative execution enables more efficient execution by dynamically scheduling instructions whenever their data dependencies are resolved
- Super scalar further improves the utilization of hardware and throughput

The pipelines of Modern Processors

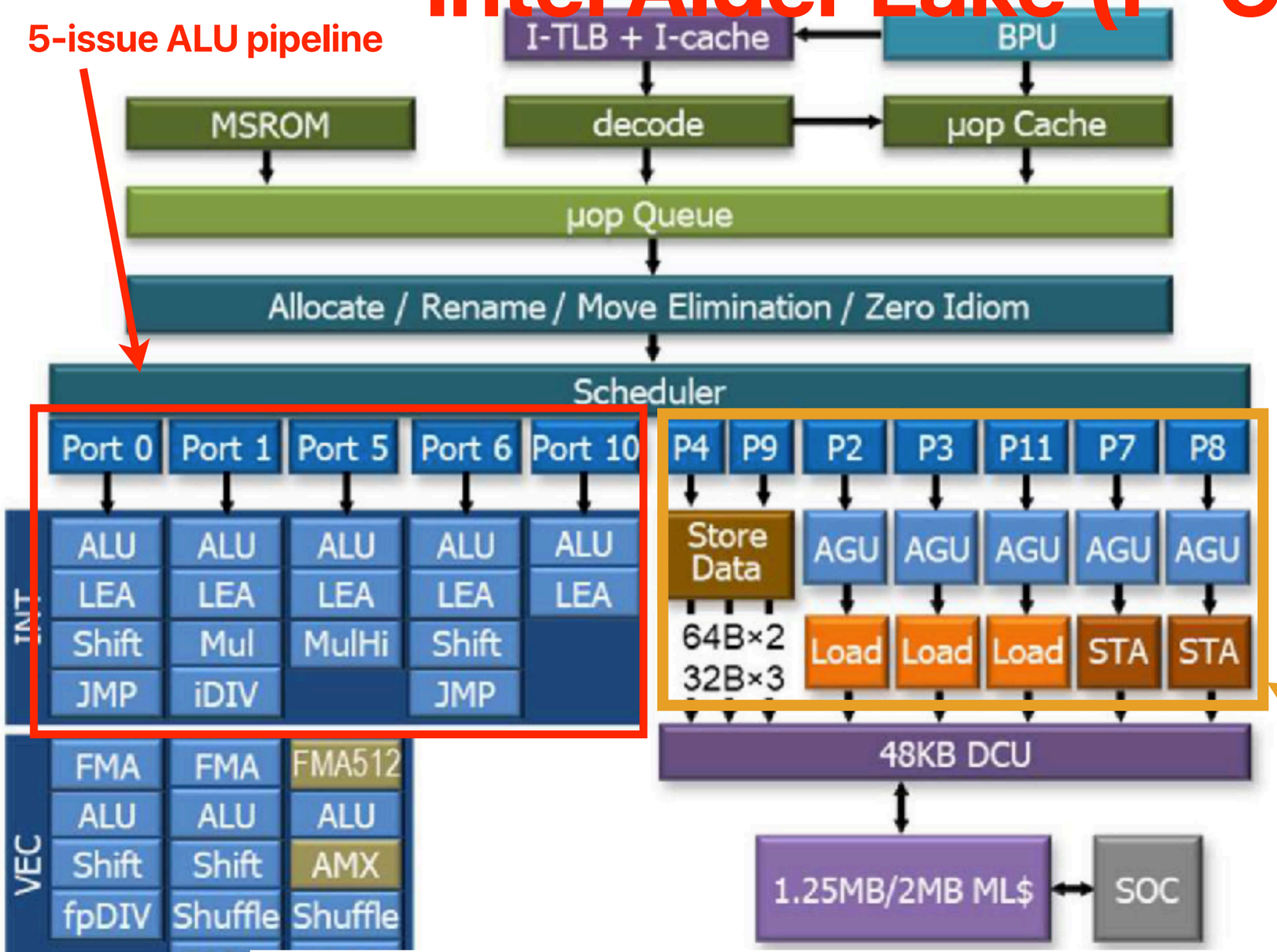
Intel Alder Lake (P-Core)

$$MinCPI = \frac{1}{12}$$

$$MinINTInst.CPI = \frac{1}{5}$$

$$MinMEMInst.CPI = \frac{1}{7}$$

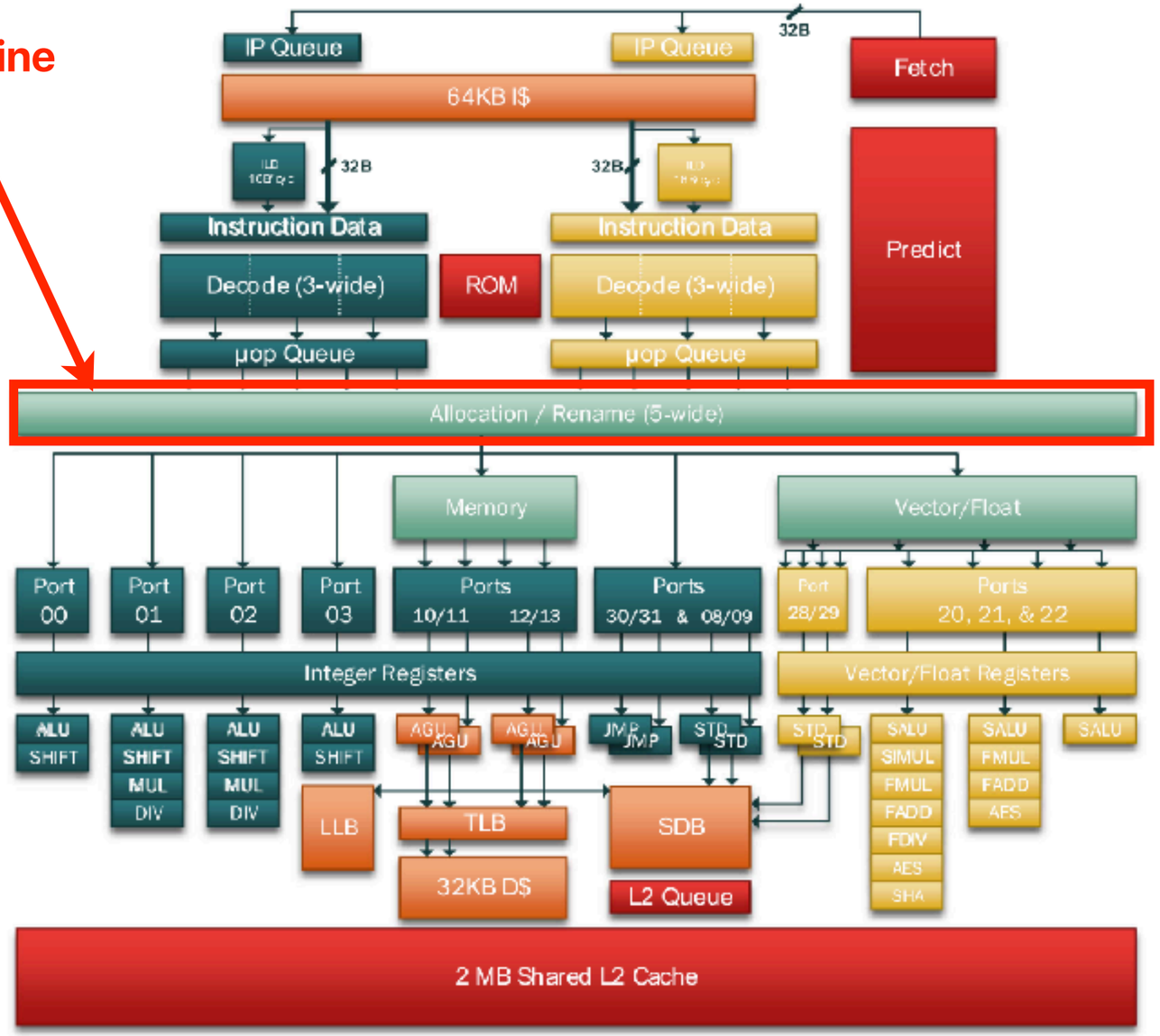
$$MinBRInst.CPI = \frac{1}{2}$$



https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Intel Alder Lake (E-Core)

5-issue pipeline



AMD Zen 3 (RyZen 5000 Series)

3-issue memory pipeline

4-issue integer pipeline + 1 additional branch

$$MinCPI = \frac{1}{8}$$

$$MinINTInst\,.\,CPI = \frac{1}{4}$$

$$MinMEMInst\,.\,CPI = \frac{1}{3}$$

$$MinBRInst\,.\,CPI = \frac{1}{2}$$

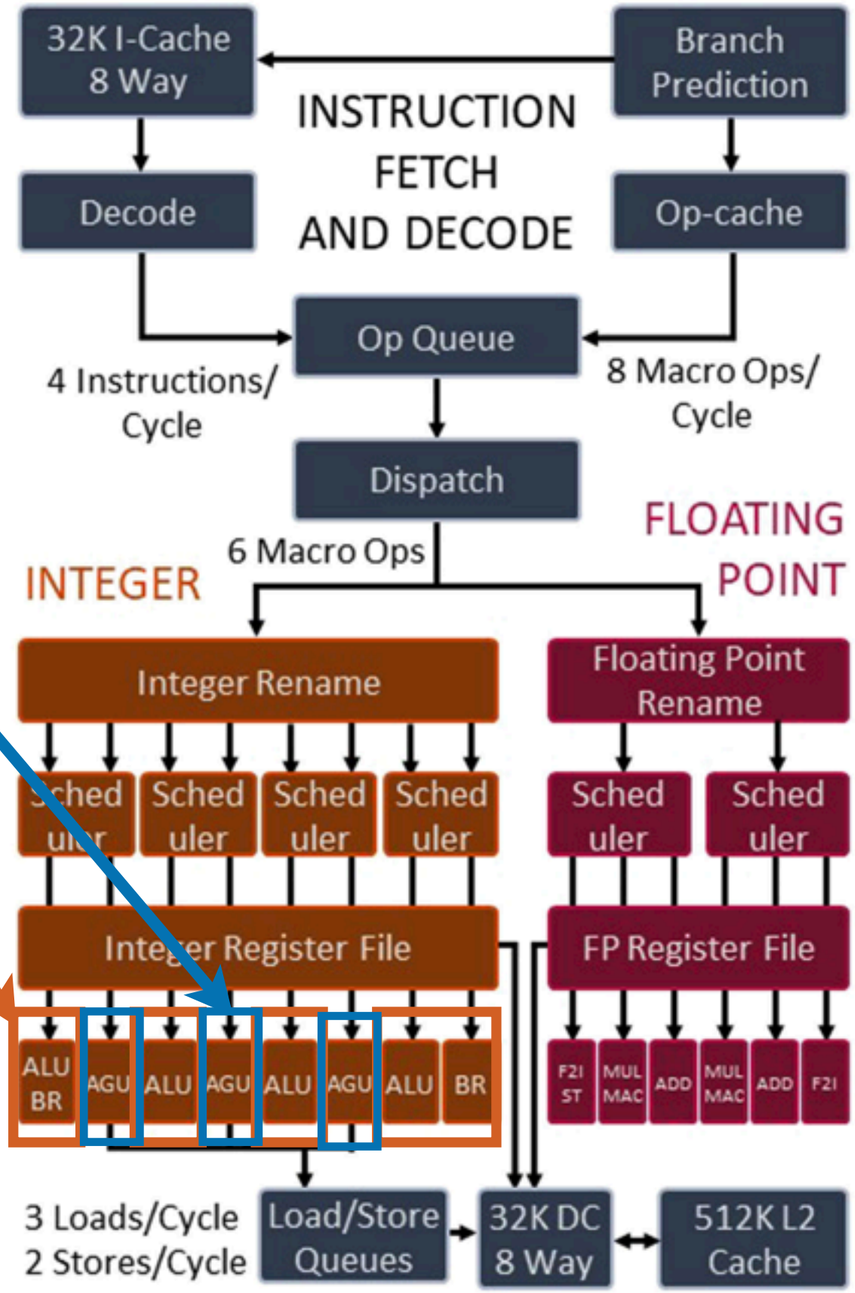


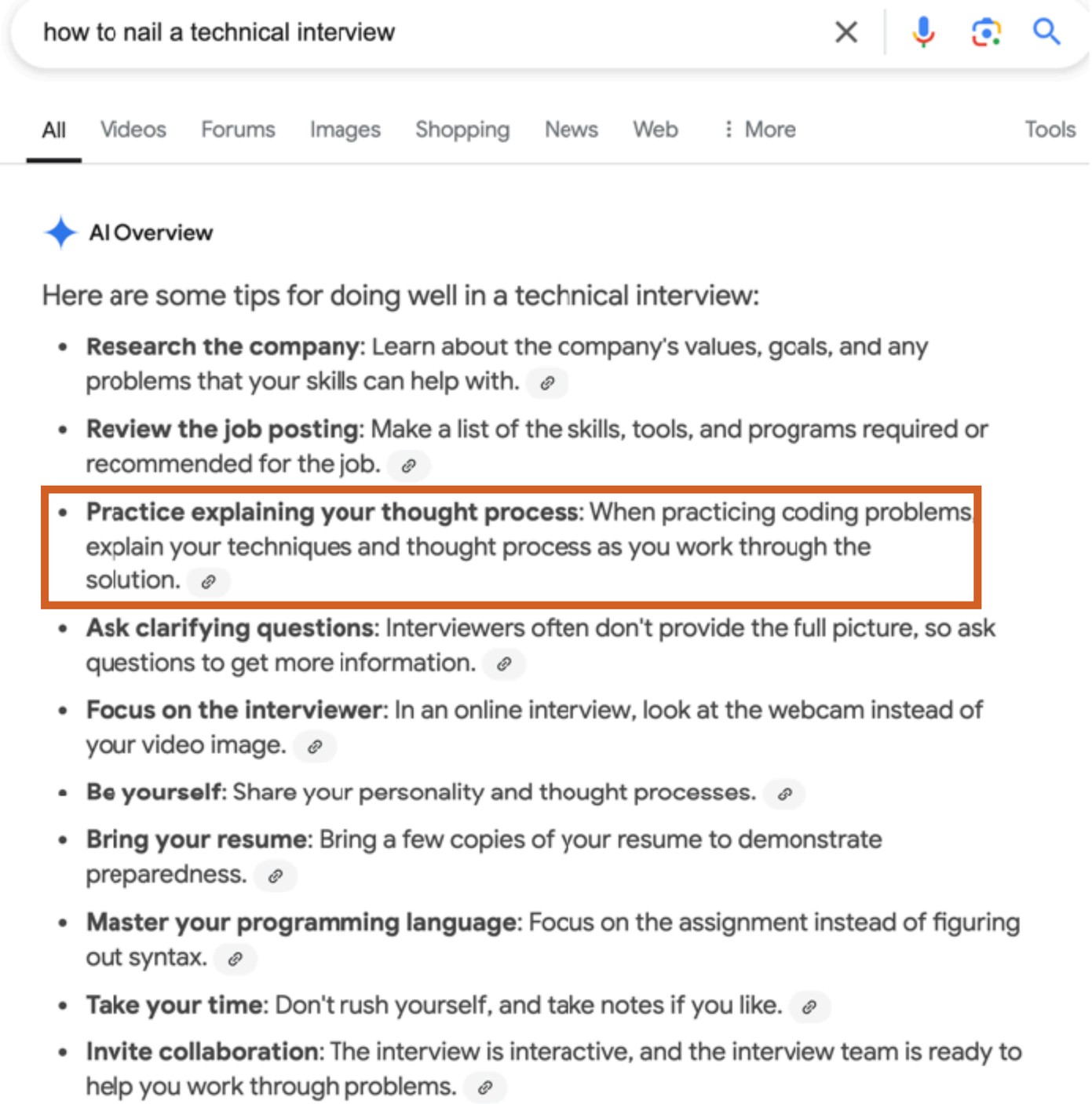
FIGURE 1. "Zen 3" block diagram.

Summary: Characteristics of modern processor architectures

- Multiple-issue pipelines with multiple functional units available
  - Multiple ALUs
  - Multiple Load/store units
  - Dynamic OoO scheduling to reorder instructions whenever possible
- Cache — very high hit rate **if your code has good locality**
  - Very matured data/instruction prefetcher
- Branch predictors — very high accuracy **if your code is predictable**
  - Perceptron
  - TAGE

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
- False dependencies limits the freedom of out-of-order execution
- Register renaming + Speculative execution enables more efficient execution by dynamically scheduling instructions whenever their data dependencies are resolved
- Super scalar further improves the utilization of hardware and throughput
- Modern processors are all very wide-issue super scalar processors with OoO capabilities



Practice in your assignments and examines!

- Why do I pick/support this solution?
  - Prove it delivers the right result
  - Justify it's better "something" compared against other "alternatives"
- What's the solution?
  - Describe the high-level architecture/idea of your solution
- How does it work out in the problem?

Announcements

- **Assignment 3 & Programming assignment 2** due this evening
- **Reading Quiz 7** due next **Tuesday** before the lecture

Computer
Science &
Engineering

203

つづく