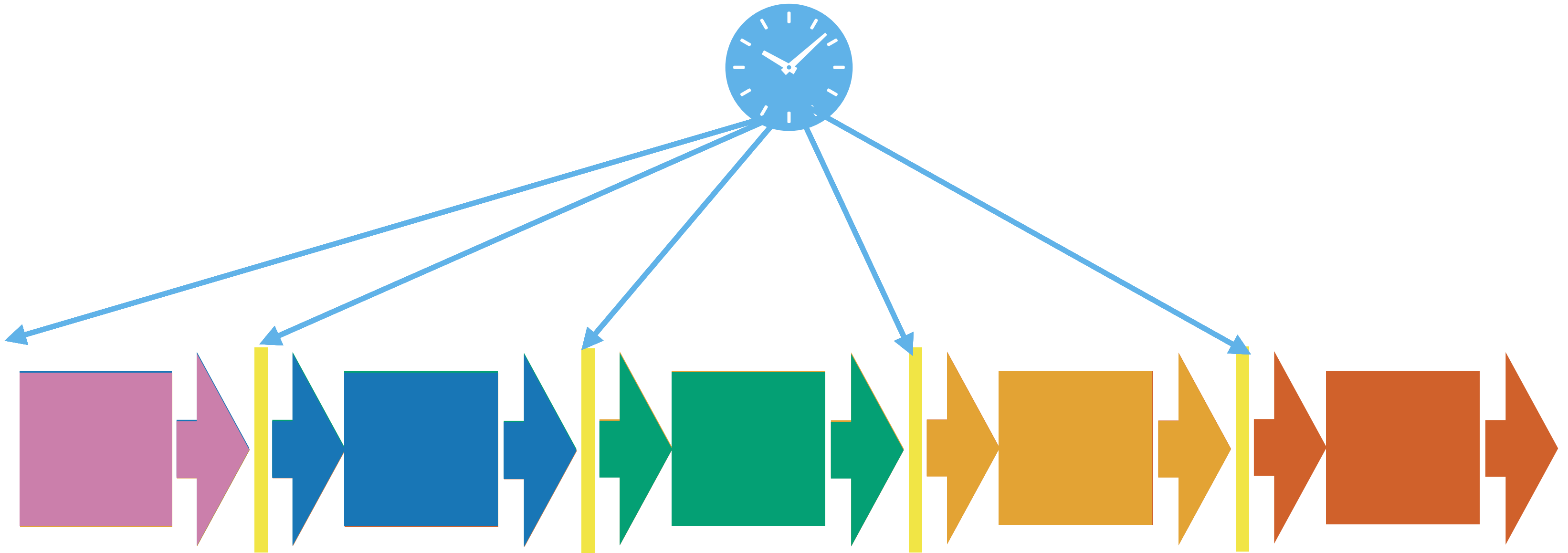


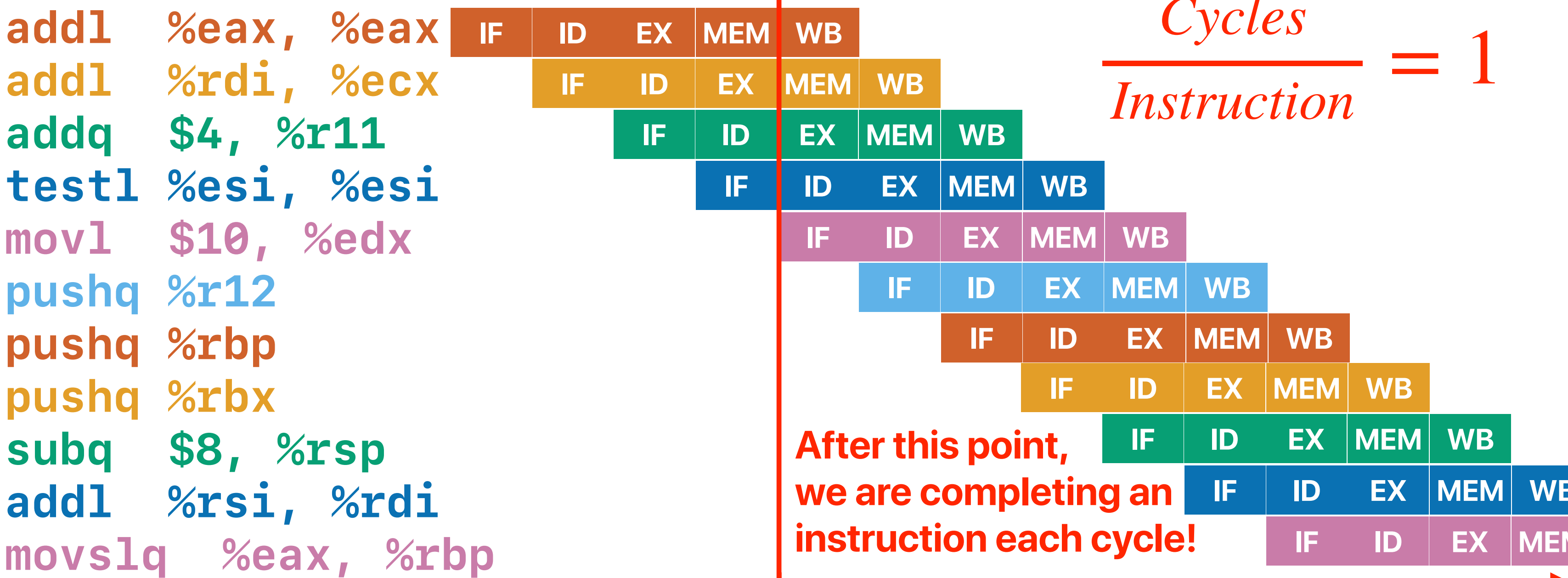
Data Hazards, Data forwarding & SuperScalar

Hung-Wei Tseng

Recap: Pipelining



Recap: Pipelining



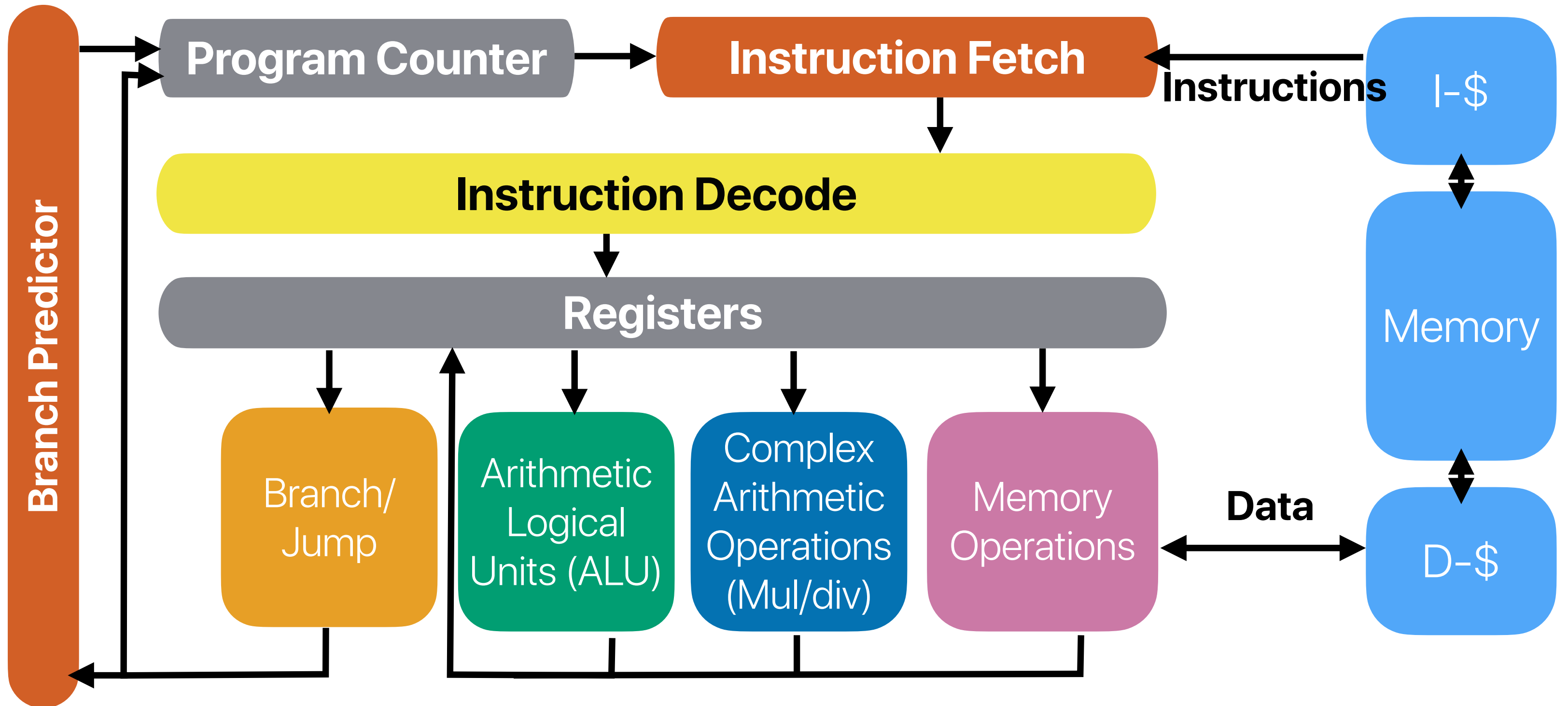
Recap: Three pipeline hazards

- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that's not yet generated or propagated when the instruction needs that

Recap: addressing hazards

- Structural hazards
 - Stall
 - Modify hardware design
- Control hazards
 - Stall
 - Static prediction
 - Dynamic prediction — all “high-performance” processors nowadays have pretty decent branch predictors
 - Local bimodal
 - Global 2-level
 - Perceptron

The "current" pipeline



3:00

**What's the most inefficient place you ever had for
dealing with your business?
What will you do to improve that if you're the
manager?**

Outline

- Data hazards
- Data forwarding
- SuperScalar

Branch and programming

Demo revisited

```
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!!!

SELECT count(*) FROM TABLE WHERE val < A and val >= B;



Demo revisited

- Why the performance is better when option is not "0"
 - ① The amount of dynamic instructions needs to execute is a lot smaller
 - ② The amount of branch instructions to execute is smaller
 - ③ The amount of branch mis-predictions is smaller
 - ④ The amount of data accesses is smaller

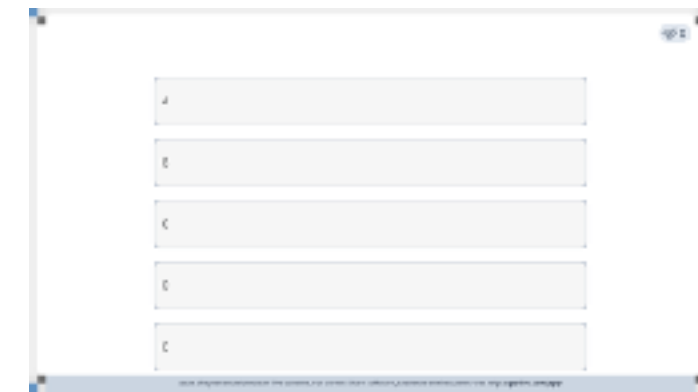
A. 0 `if(option)`
 `std::sort(data, data + arraySize);`

B. 1

C. 2 `for (unsigned i = 0; i < 100000; ++i) {`
 `int threshold = std::rand();`

D. 3 `for (unsigned i = 0; i < arraySize; ++i) {`
 `if (data[i] >= threshold)`
 `sum ++;`

E. 4 `}`
 `}`
 `}`



Demo revisited

- Why the performance is better when option is not "0"
 - ① The amount of dynamic instructions needs to execute is a lot smaller
 - ② The amount of branch instructions to execute is smaller
 - ✓③ The amount of branch mis-predictions is smaller
 - ④ The amount of data accesses is smaller

A. 0

B. 1

C. 2

D. 3

E. 4

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

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

branch X

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

Demo revisited: 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!!!

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 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)
```

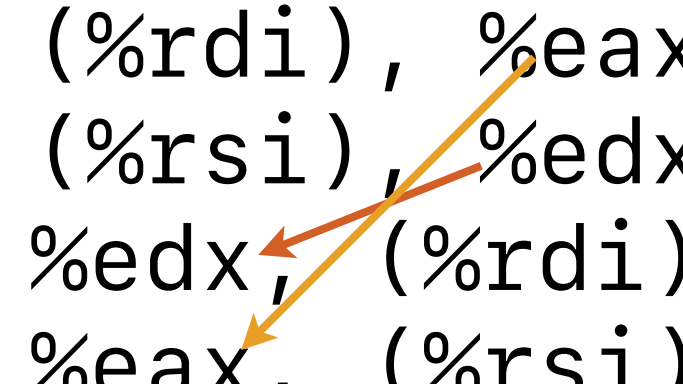
```
int temp = *a;
*a = *b;
*b = temp;
```

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

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



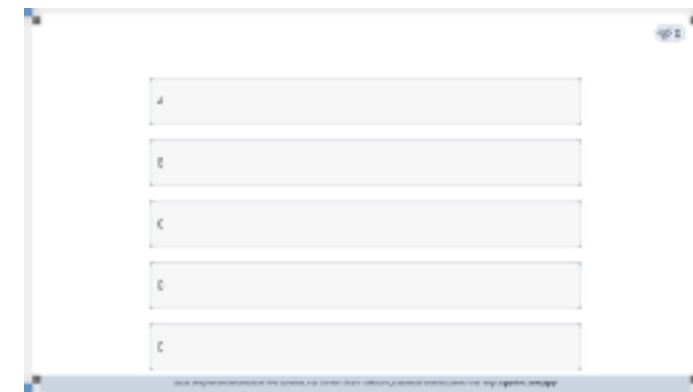
How many data dependencies do we have?

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

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

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

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



How many dependencies do we have?

- How many pairs of data dependencies 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

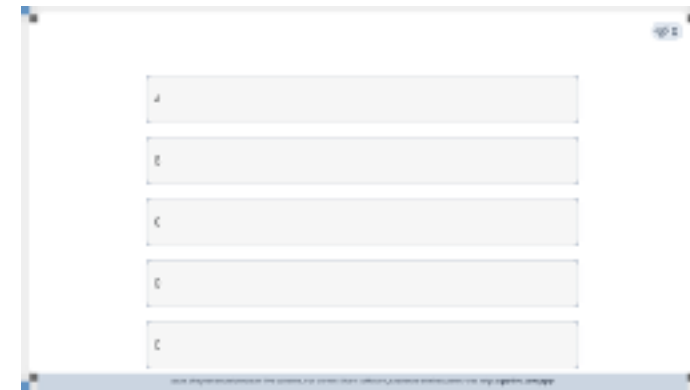


Data hazards?

- How many pairs of data dependences in the following x86 instructions will result in data hazards if a memory operation (assume 100% cache hit rate) takes 4 cycles?

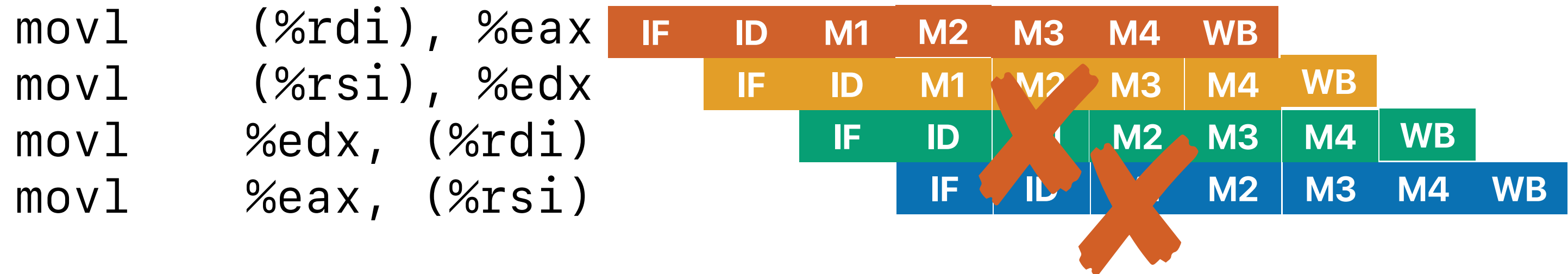
```
movl    (%rdi), %eax  
movl    (%rsi), %edx  
movl    %edx, (%rdi)  
movl    %eax, (%rsi)
```

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



Data hazards?

- How many pairs of data dependences in the following x86 instructions will result in data hazards if a memory operation (assume 100% cache hit rate) takes 4 cycles?



A. 0

B. 1

C. 2

D. 3

E. 4

Data hazards

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

%edx does not have our desired value

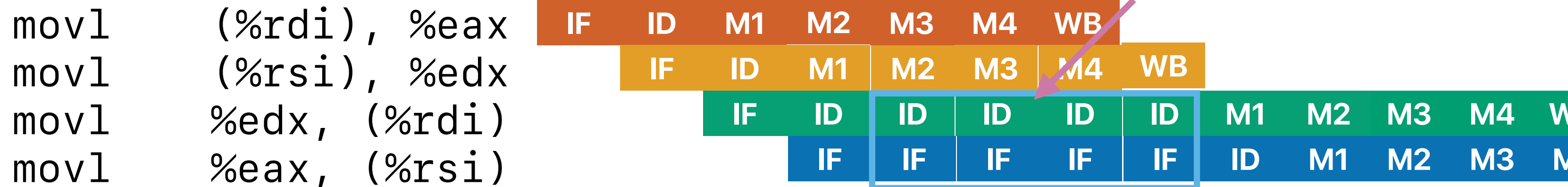
	IF	ID	ALU/BR/M1	M2	M3	M4	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(4)	(3)	(2)	(1)			
5		(4)	(3)	(2)	(1)		
6			(4)	(3)	(2)	(1)	
7				(4)	(3)	(2)	(1)
8					(4)	(3)	(2)
9						(4)	(3)
10							(4)
11							
12							
13							
14							

%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.

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



4 additional cycles

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.

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

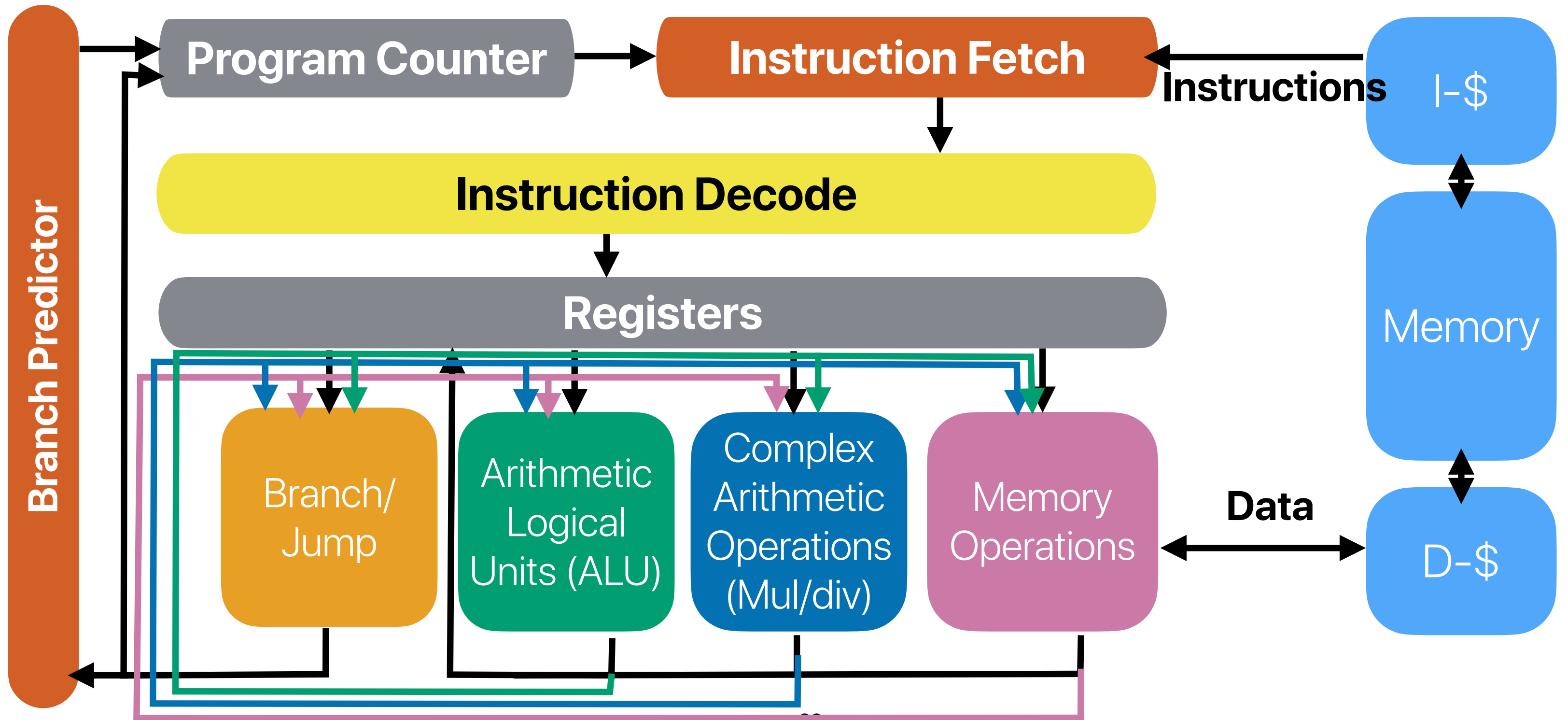
we have the value for %edx already!
Why another cycle?

	IF	ID	ALU/BR/M1	M2	M3	M4	WB
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)	(3)			
11				(4)	(3)		
12					(4)	(3)	
13						(4)	(3)
14							(4)

Solution 2: Data forwarding

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

Data "forwarding"





How many data dependencies are still problematic?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if a memory operation (assume 100% cache hit rate) takes 4 cycles?

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

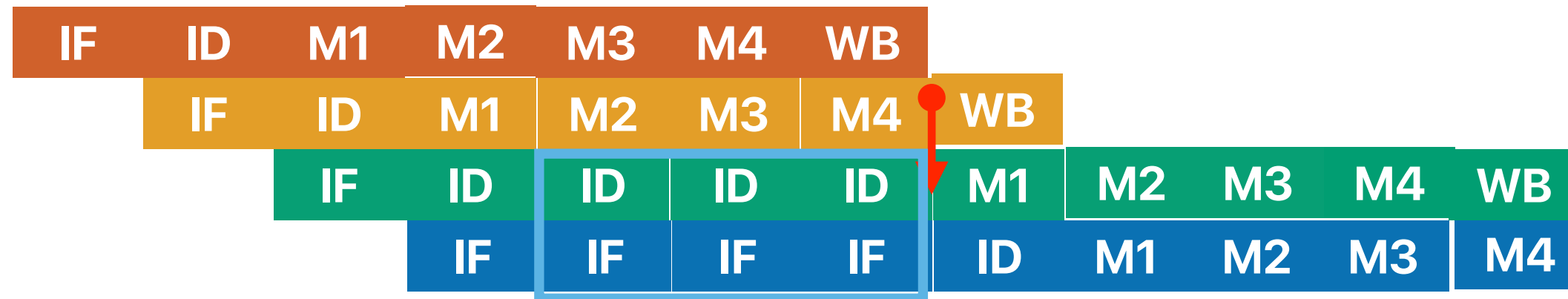
- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

A screenshot of a poll interface. It shows five empty input boxes, each preceded by a letter (A, B, C, D, E) in a small font. The boxes are arranged vertically. At the bottom, there is a small text line that reads "Don't forget to click the 'Submit' button at the bottom right of the page."

How many data dependencies are still problematic?

- How many pairs of data dependencies in the following x86 instructions are still problematic with data forwarding if a memory operation (assume 100% cache hit rate) takes 4 cycles?

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



3 additional cycles

A. 0

B. 1

C. 2

D. 3

E. 4

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

Solution 2: Data forwarding

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

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

	IF	ID	ALU/BR/M1	M2	M3	M4	WB
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)	data forwarding			(2)	(1)
8		(4)					(2)
9			(4)	(3)			
10				(4)	(3)		
11					(4)	(3)	
12						(4)	(3)
13							(4)
14							



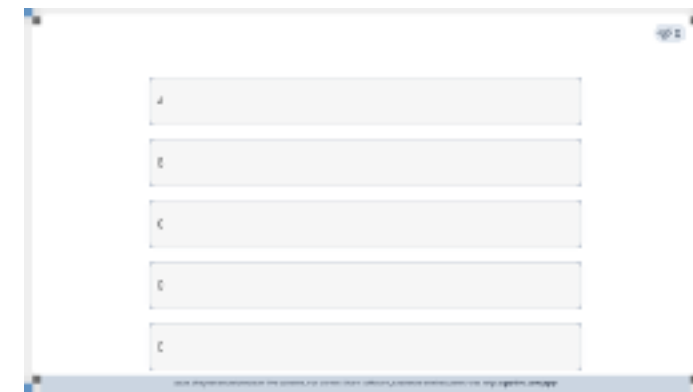
How many of data hazards w/ Data Forwarding?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if both a memory operation and an xorl take 4 cycles?

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

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

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



How many of data hazards w/ Data Forwarding?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if both a memory operation and an xorl take 4 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

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

Another code example

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

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

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



The effect of code optimization

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

① `movl (%rdi), %ecx`

② `addl %ecx, %eax`

③ `addq $4, %rdi`

④ `cmpq %rdx, %rdi`

⑤ `jne .L3`

⑥ `ret`

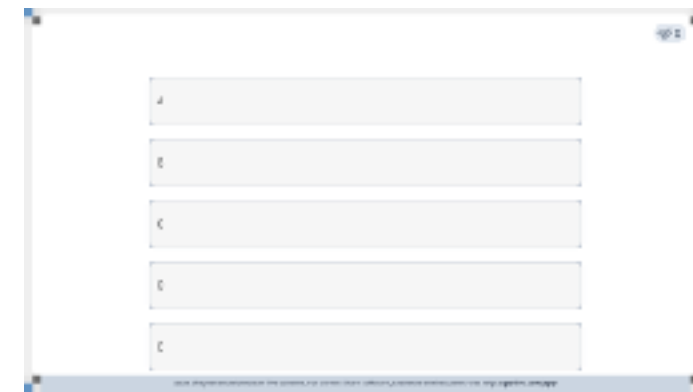
A. (1) & (2)

B. (2) & (3)

C. (3) & (4)

D. (4) & (5)

E. None of the pairs can be reordered



The effect of code optimization

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

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

A. (1) & (2)

B. (2) & (3)

C. (3) & (4)

D. (4) & (5)

E. None of the pairs can be reordered

Compiler optimization

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

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

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

Compiler optimization

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

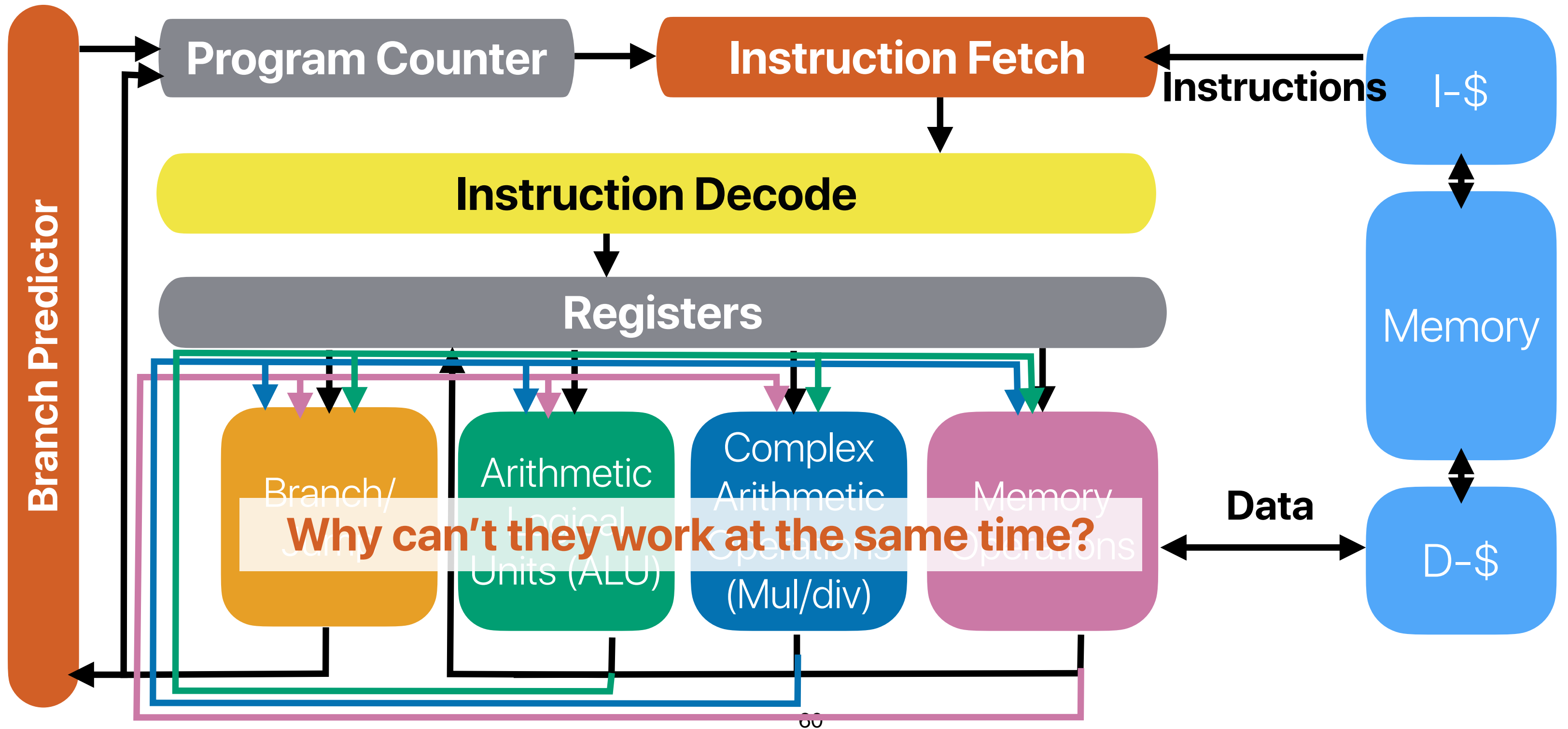
```
.L3:  
①    movl    (%rdi), %ecx  
②    addq    $4, %rdi  
③    addl    %ecx, %eax  
④    cmpq    %rdx, %rdi  
⑤    jne     .L3  
⑥    ret
```

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

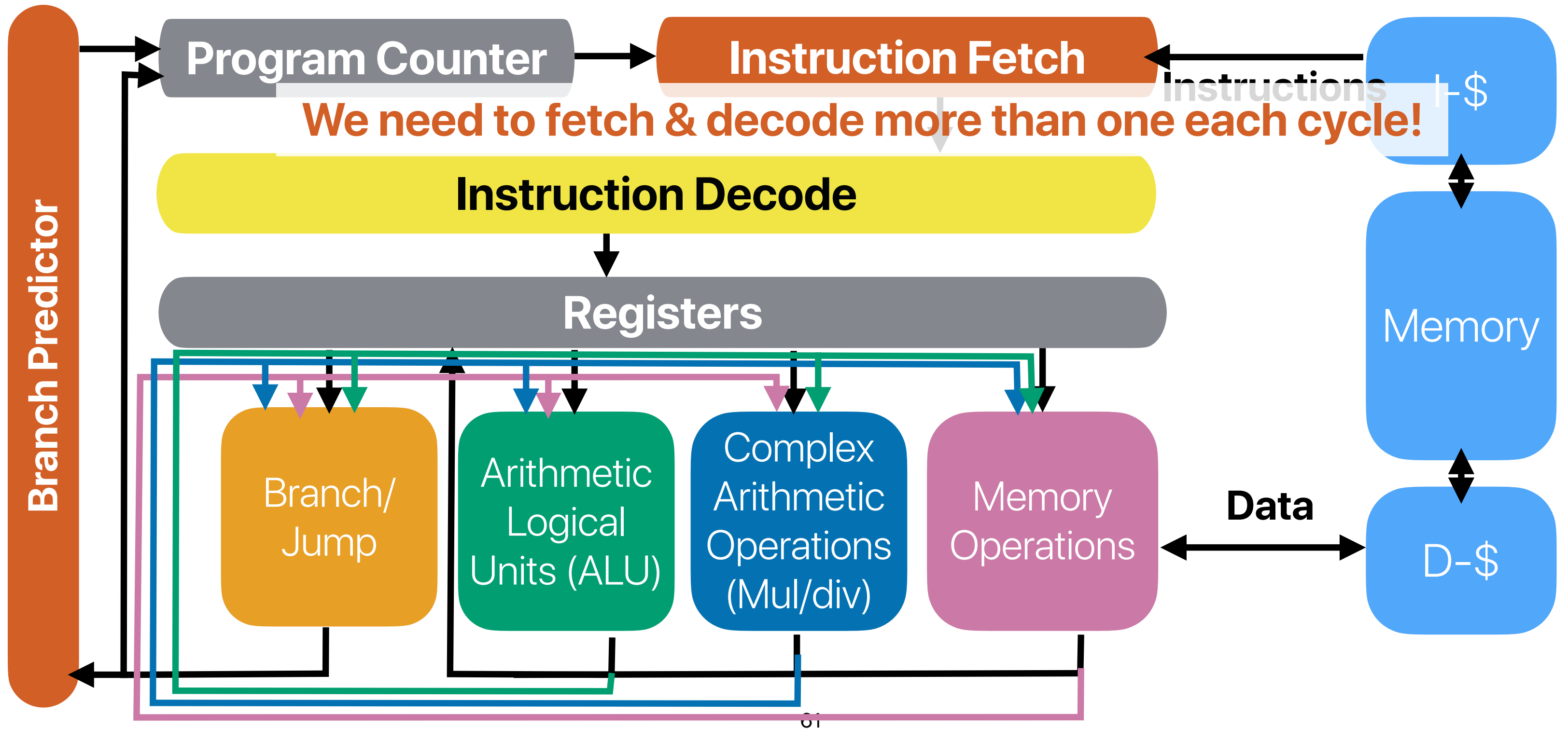
addq is not depending on movl and
ALU is free! can we execute them
together?

If $CPI == 1$ the limitation?

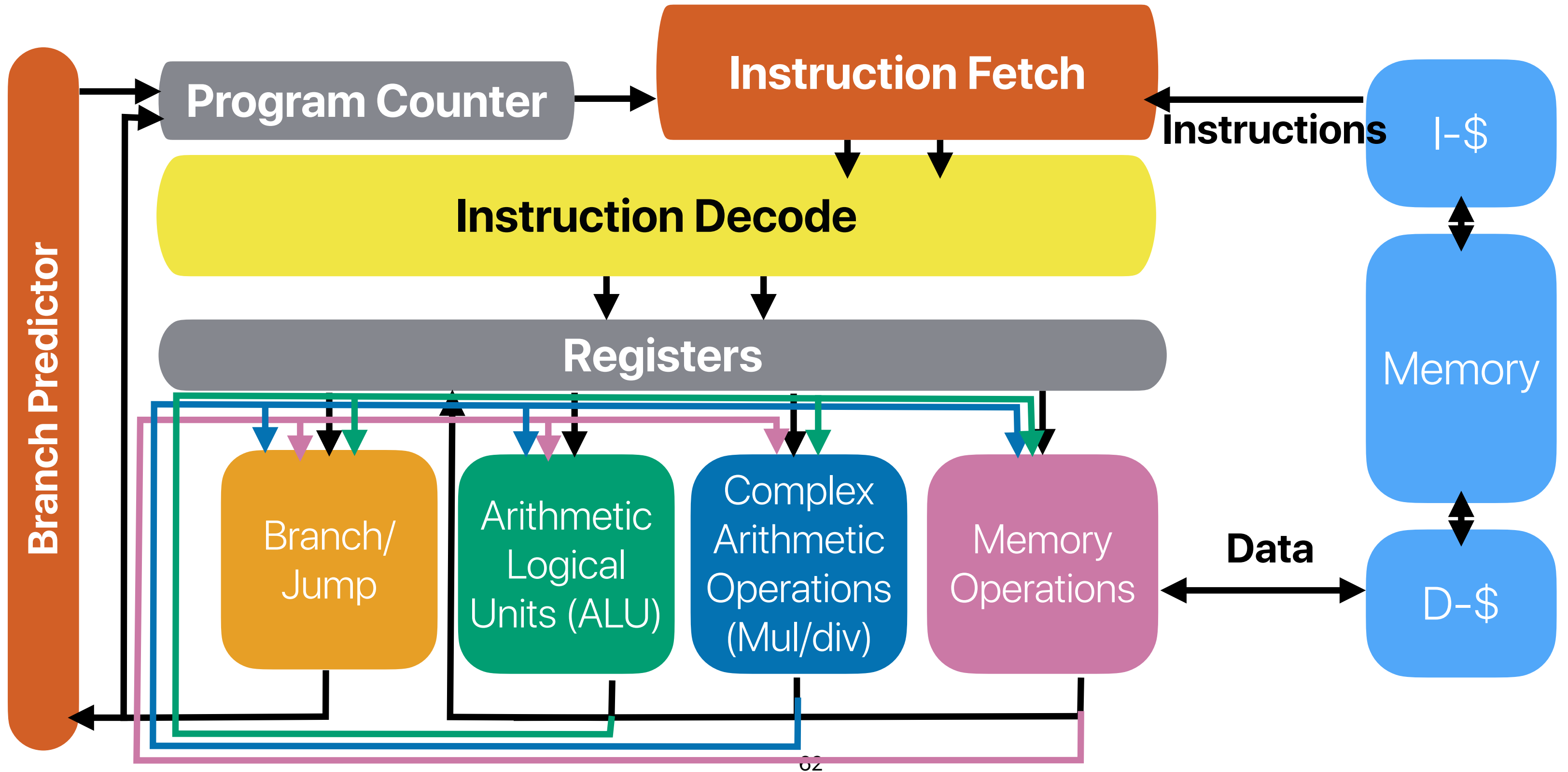
Data "forwarding"



Data "forwarding"



Super Scalar



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

1

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

Superscalar: fetch/issue width == 2, theoretical CPI = 0.5

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

.L3:

①

movl

(%rdi), %ecx

②

addq

\$4, %rdi

③

addl

%ecx, %eax

④

cmpq

%rdx, %rdi

⑤

jne

.L3

⑥

ret

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

Everything we need for (4) is ready here

Why can't we execute it?

Announcements

- Tips for answering examine question (or more importantly, technical interviews)
 - Be precise — pulling everything you know simply shows you don't really understanding what the question is asking
 - Important things go first — interviewers and graders lose patience very quick, you need to give a summary or conclusion before going into detail
 - We cannot read your mind. Explaining your thoughts after the examine (e.g., regrading requests) or interview (e.g., through follow-up) **does not** help win anything back
 - Show your work — simply tell the interviewer we can use DP to solve this problem won't give you positive review results
- **Last reading quiz** due this Wednesday before the lecture

Computer Science & Engineering

142

つづく

