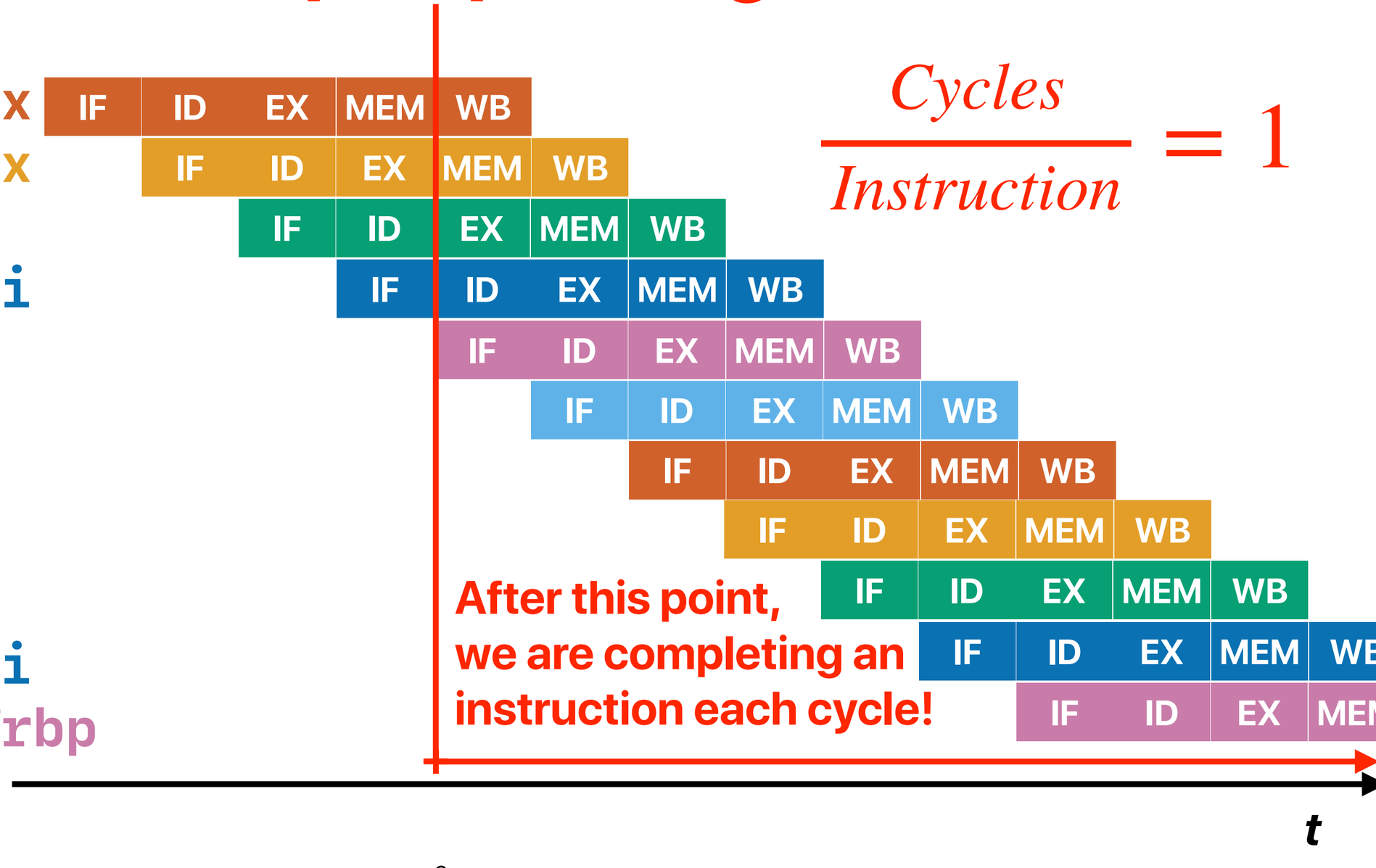
Modern Processor Pipeline: SuperScalar & Dynamic Instruction Scheduling

Hung-Wei Tseng

Recap: Pipelining

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



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

After this point, we are completing an instruction each cycle!

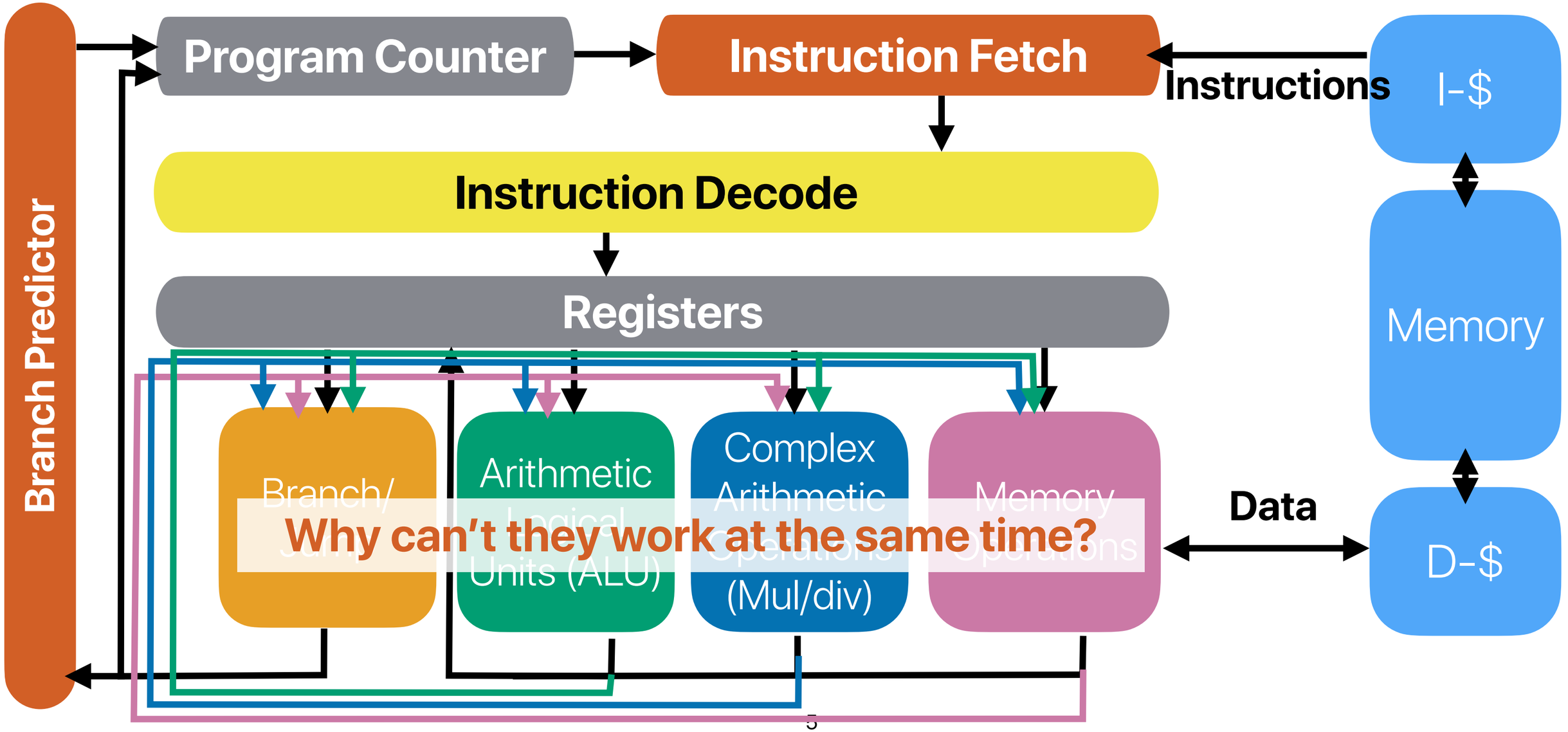
Recap: Three pipeline hazards

- 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

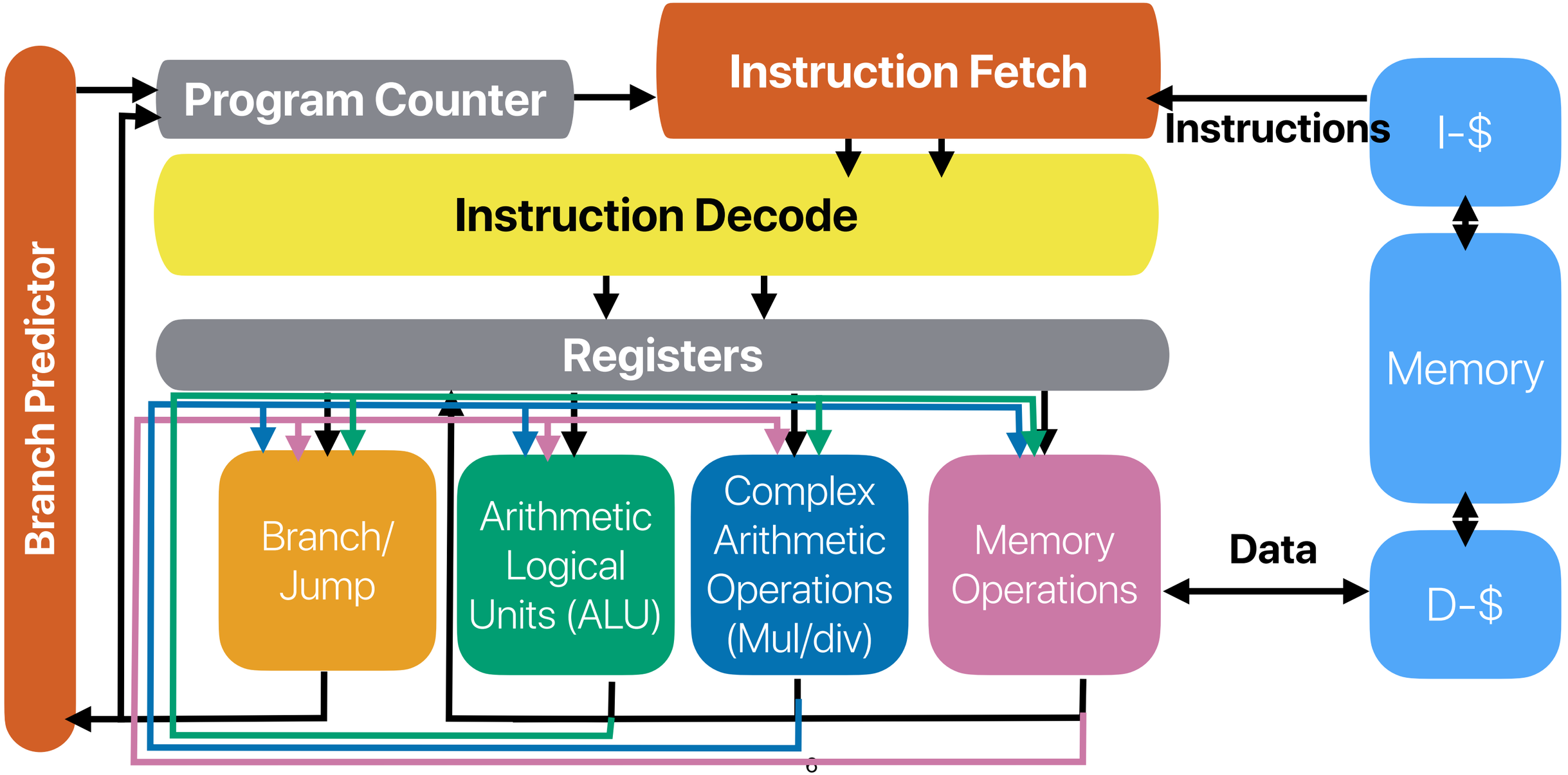
Recap: addressing hazards

- 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
- Data hazards
  - Stall
  - Data forwarding

Data "forwarding"



Super Scalar



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 ne
for (4) is ready he
Why can't we
execute it?

Outline

- SuperScalar
- Dynamic Instruction Scheduling

If we loop many times (assume perfect predictor)

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
⑪ movl    (%rdi), %ecx
⑫ addq    $4, %rdi
⑬ addl    %ecx, %eax
⑭ cmpq    %rdx, %rdi
⑮ jne     .L3
```

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

Everything we need for (4) is ready here! Why can't we execute it?

Why can't I start loading (6) & (11)?

What do you need to execution an instruction?

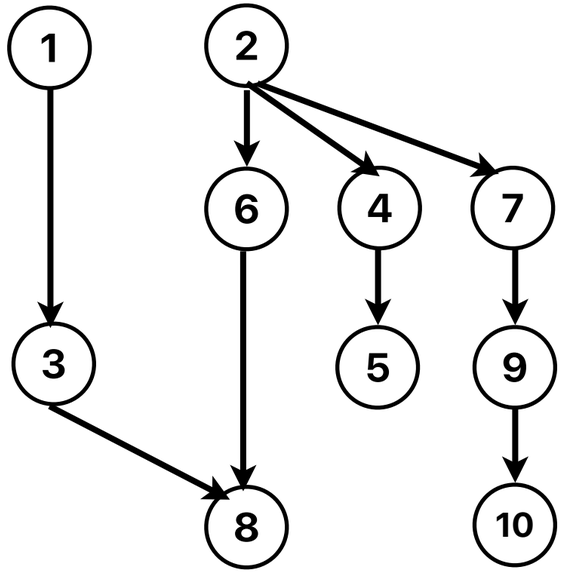
- Whenever the instruction is decoded — put decoded instruction somewhere
- Whenever the inputs are ready — **all data dependencies are resolved**
- Whenever the target functional unit is available

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

Scheduling instructions: based on data dependencies

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

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```



- **In theory**, instructions without dependencies can be executed in parallel or out-of-order
- Instructions with dependencies 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 movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

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 (4)
C. (3) and (6)
D. (4) and (7)
E. (6) and (7)

If we can predict the future …

- Consider the following dynamic instructions:

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

Can we use "branch prediction" to predict the future and reorder instructions across the branch?

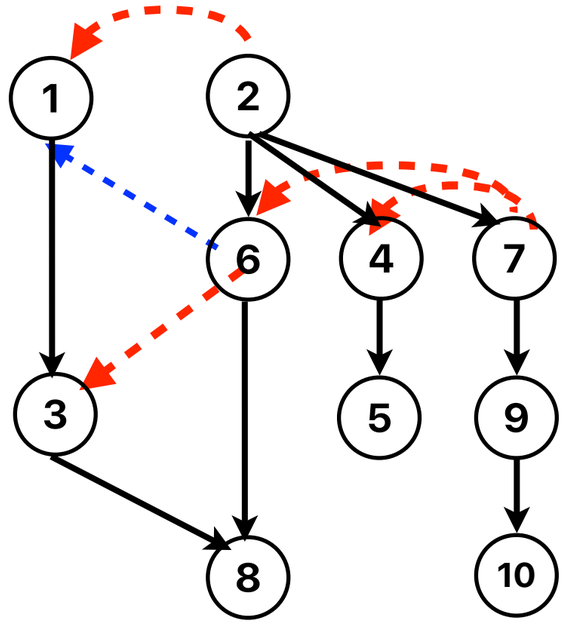
Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

A. (1) and (2)
B. (3) and (4)
C. (3) and (6)
D. (4) and (7)
E. (6) and (7)

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
    - 2 and 1, 6 and 3, 7 and 4, 7 and 6
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1

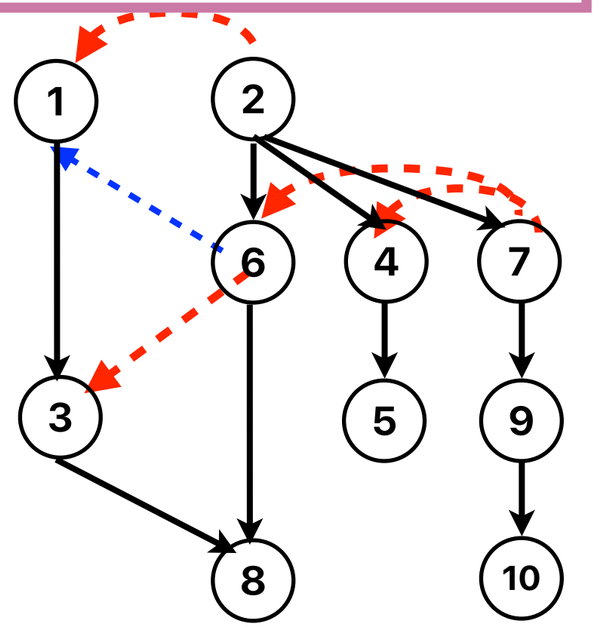
```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
```



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
    - 2 and 1, 6 and 3, 7 and 4, 7 and 6
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
```



Limitations of Compiler Optimizations

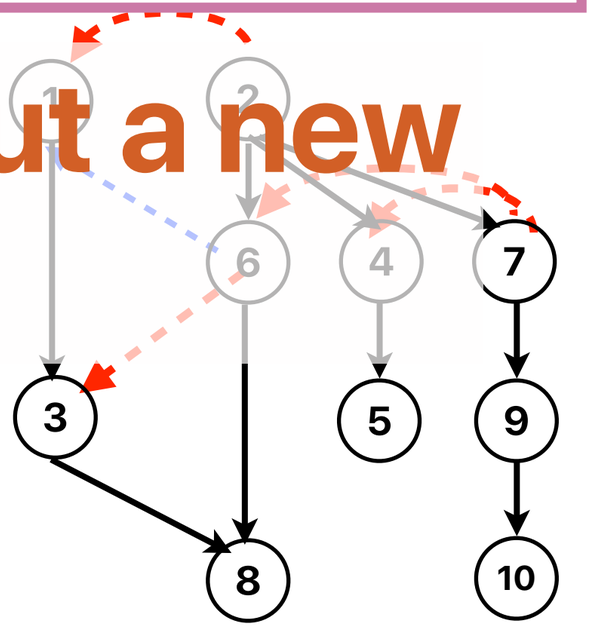
- If the hardware (e.g., pipeline changes), the same compiler optimization may not be that helpful
- The compiler can only optimize on static instructions, but cannot optimize dynamic instructions
- Compilers are limited by the registers an ISA provides

Recap: 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
    - 2 and 1, 6 and 3, 7 and 4, 7 and 6
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1

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

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
```



What if we can use more registers...

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

```
① movl    (%rdi), %ecx
② addq    $4, %rdi, %t0
③ addl    %ecx, %eax, %t1
④ cmpq    %rdx, %t0
⑤ jne     .L3
⑥ movl    (%t0), %t2
⑦ addq    $4, %t0, %t3
⑧ addl    %t1, %t2, %t4
⑨ cmpq    %rdx, %t3
⑩ jne     .L3
```

All false dependencies are gone!!!

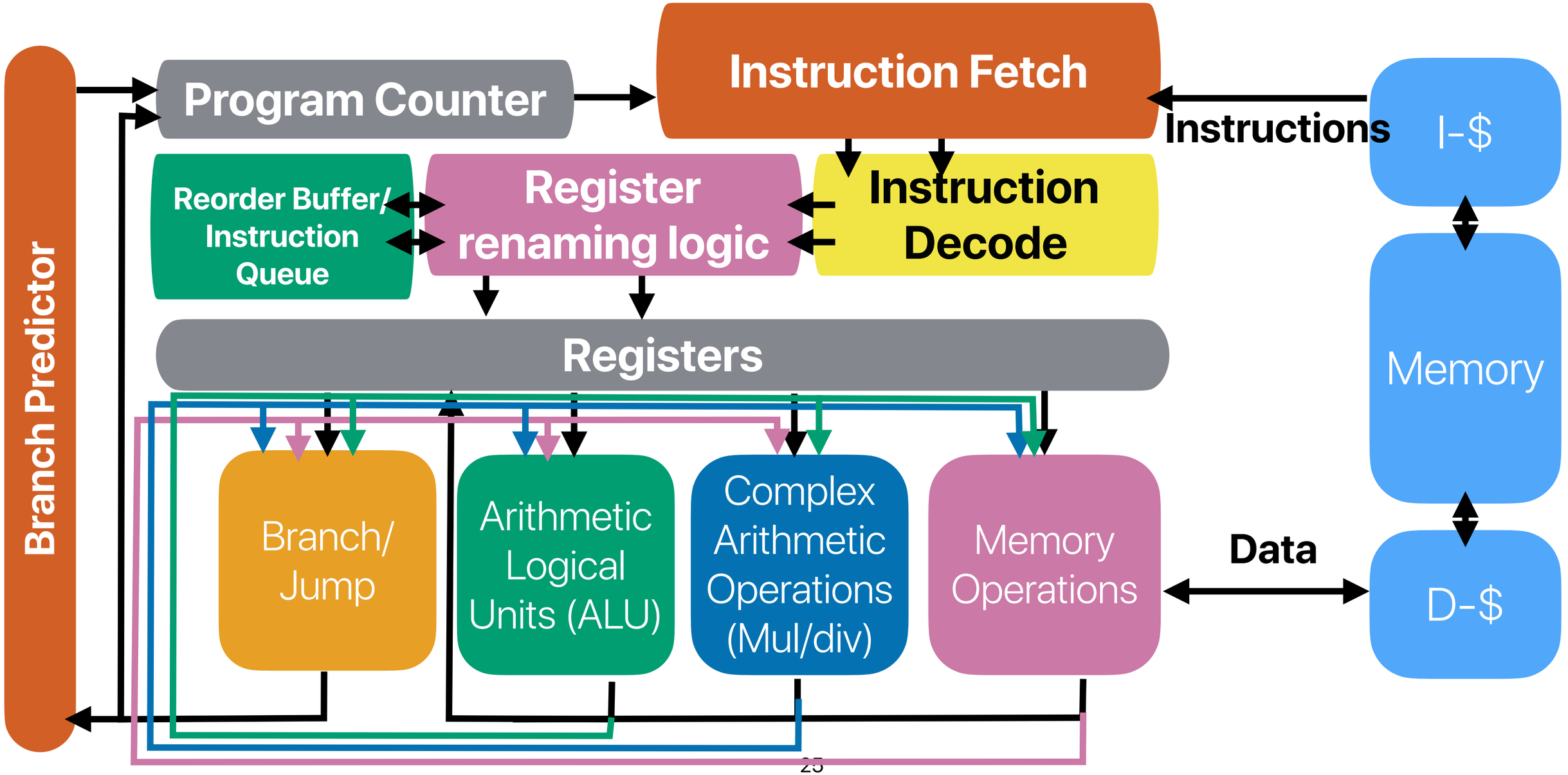
Register renaming + speculative execution

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

Speculative Execution

- Exceptions (e.g. divided by 0, page fault) may occur anytime
  - A later instruction cannot write back its own result otherwise the architectural states won't be correct
- 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
- Execute instructions across branches
  - Speculative execution: execute an instruction before the processor know if we need to execute or not
  - Execute an instruction all operands are ready (the values of depending physical registers are generated)
  - Store results in **reorder buffer** before the processor knows if the instruction is going to be executed or not.

Register renaming



Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	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											

	Physical Register
eax	
ecx	
rdi	
rdx	

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx
⑦ addq   $4, %rdi
⑧ addl   %ecx, %eax
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx
⑫ addq   $4, %rdi
⑬ addl   %ecx, %eax
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5											
6											
7											
8											
9											
10											
11											

	Physical Register
eax	
ecx	P1
rdi	P2
rdx	

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi → P2
③  addl    %ecx, %eax → P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6											
7											
8											
9											
10											
11											

(4) is now executing before (3)!

	Physical Register
eax	
ecx	P1
rdi	P2
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx
⑫ addq   $4, %rdi
⑬ addl   %ecx, %eax
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7											
8											
9											
10											
11											

	Physical Register
eax	P6
ecx	P1
rdi	P5
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx
⑫ addq   $4, %rdi
⑬ addl   %ecx, %eax
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8											
9											
10											
11											

	Physical Register
eax	P6
ecx	P1
rdi	P5
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			(1)(2)(4)(5)(7)
9											
10											
11											

	Physical Register
eax	P6
ecx	P7
rdi	P8
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			(1)(2)(4)(5)(7)
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			(3)(4)(5)(7)
10											
11											

	Physical Register
eax	P9
ecx	P7
rdi	P8
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)(2)~~(4)(5)(7)
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	(6)(7)(9)
11											

	Physical Register
eax	P9
ecx	P7
rdi	P8
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)~~(2)(4)(5)(7)
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15)(16)(17)(18)			(11)		(12)			(8)(9)(10)

	Physical Register
eax	P6
ecx	P1
rdi	P5
rdx	P4

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

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)(2)(4)(5)(7)~~
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15)(16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18)(19)(20)	(16)			(11)	(14)			(12)
13											
14											
15											

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)(2)(4)(5)(7)~~
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	~~(6)~~(7)(9)
11		(19)(20)	(13)(14)(15)(16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18)(19)(20)	(16)			(11)	(14)			(12)
13			(17)(18)(19)(20)		(16)			(13)		(15)	(11)(12)(14)
14											
15											

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)(2)~~(4)(5)(7)
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15)(16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18)(19)(20)	(16)			(11)	(14)			(12)
13			(17)(18)(19)(20)		(16)			(13)		(15)	~~(11)(12)~~(14)
14						(16)		(17)			(13)(14)(15)
15											

Register renaming

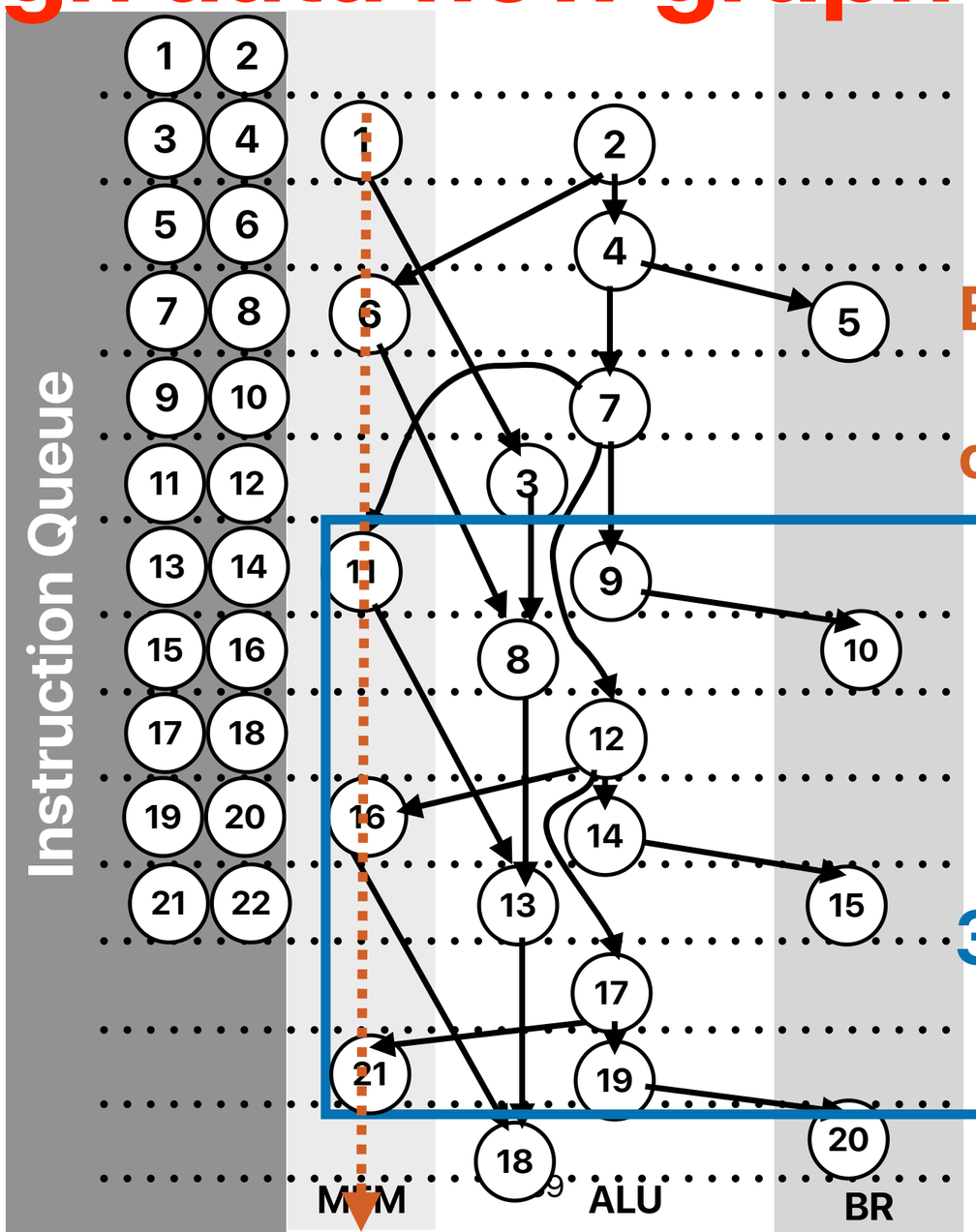
2-issue: Only 2 of them can have instructions at the same cycle

```
① movl   (%rdi), %ecx → P1
② addq   $4, %rdi → P2
③ addl   %ecx, %eax → P3
④ cmpq   %rdx, %rdi
⑤ jne    .L3
⑥ movl   (%rdi), %ecx → P4
⑦ addq   $4, %rdi → P5
⑧ addl   %ecx, %eax → P6
⑨ cmpq   %rdx, %rdi
⑩ jne    .L3
⑪ movl   (%rdi), %ecx → P7
⑫ addq   $4, %rdi → P8
⑬ addl   %ecx, %eax → P9
⑭ cmpq   %rdx, %rdi
⑮ jne    .L3
```

	IF	ID	REN	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)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)(2)(4)(5)(7)~~
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15)(16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18)(19)(20)	(16)			(11)	(14)			(12)
13			(17)(18)(19)(20)		(16)			(13)		(15)	~~(11)(12)~~(14)
14						(16)		(17)			~~(13)(14)(15)~~
15							(16)	(19)			(17)

Through data flow graph analysis

```
①  movl (%rdi), %ecx
②  addq $4, %rdi
③  addl %ecx, %eax
④  cmpq %rdx, %rdi
⑤  jne  .L3
⑥  movl (%rdi), %ecx
⑦  addq $4, %rdi
⑧  addl %ecx, %eax
⑨  cmpq %rdx, %rdi
⑩  jne  .L3
⑪  movl (%rdi), %ecx
⑫  addq $4, %rdi
⑬  addl %ecx, %eax
⑭  cmpq %rdx, %rdi
⑮  jne  .L3
⑯  movl (%rdi), %ecx
⑰  addq $4, %rdi
⑱  addl %ecx, %eax
⑲  cmpq %rdx, %rdi
⑳  jne  .L3
㉑  movl (%rdi), %ecx
```



Execution time is determined by the "critical path" composed by 1, 6, 11, ..., 1+5n

3 cycles every iteration

CPI = $\frac{3}{5} = 0.6$!

What about “linked list”

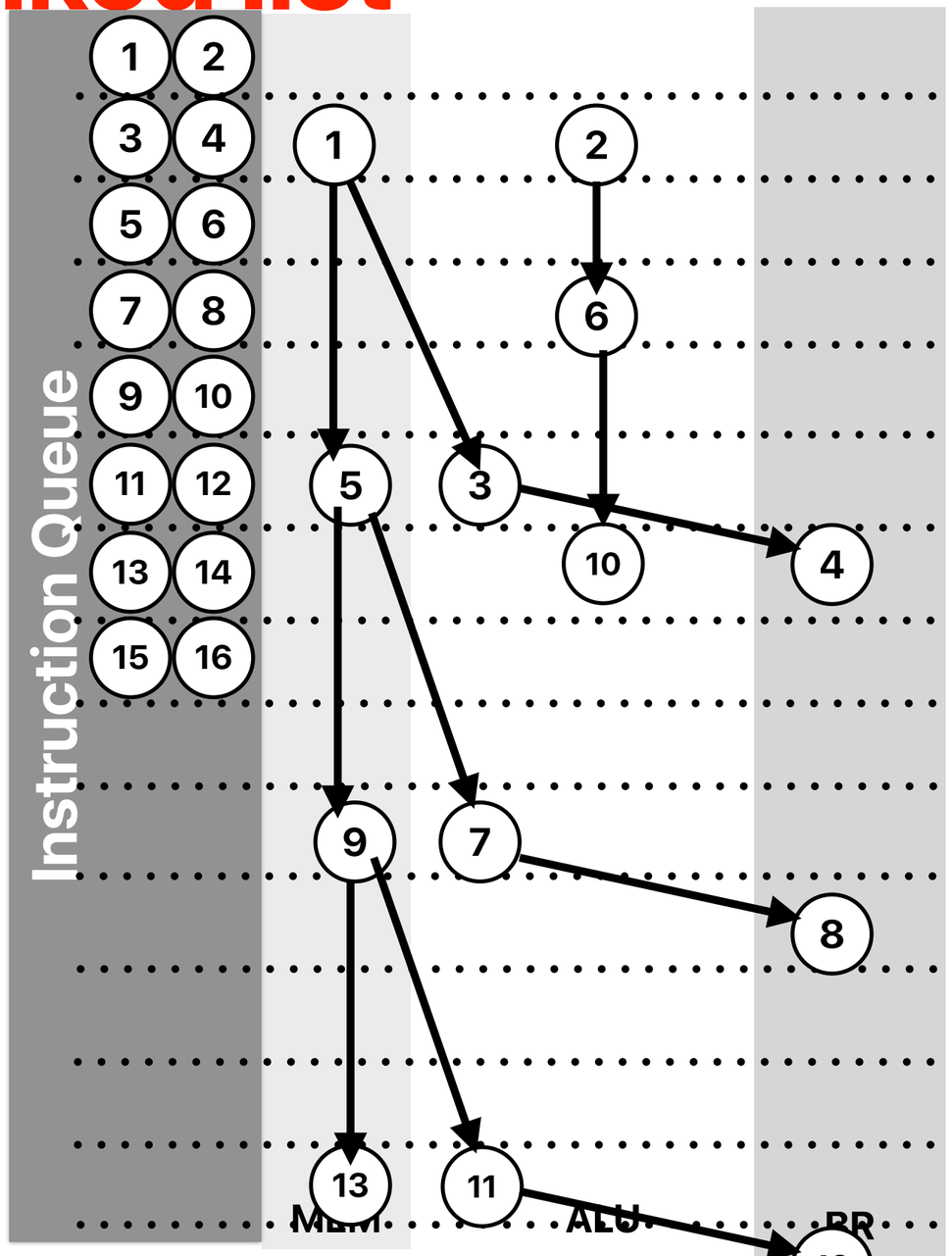
- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.
  Which of the following C state of the code snippet determines the performance?

```
A.do {
B.     number_of_nodes++;
C.     current = current->next;
D.} while ( current != NULL );
```

What about “linked list”

Dynamic instructions

```
①  .L3:    movq    8(%rdi), %rdi
②          addl    $1, %eax
③          testq   %rdi, %rdi
④          jne     .L3
⑤  .L3:    movq    8(%rdi), %rdi
⑥          addl    $1, %eax
⑦          testq   %rdi, %rdi
⑧          jne     .L3
⑨  .L3:    movq    8(%rdi), %rdi
⑩          addl    $1, %eax
⑪          testq   %rdi, %rdi
⑫          jne     .L3
⑬  .L3:    movq    8(%rdi), %rdi
⑭          addl    $1, %eax
⑮          testq   %rdi, %rdi
⑯          jne     .L3
```

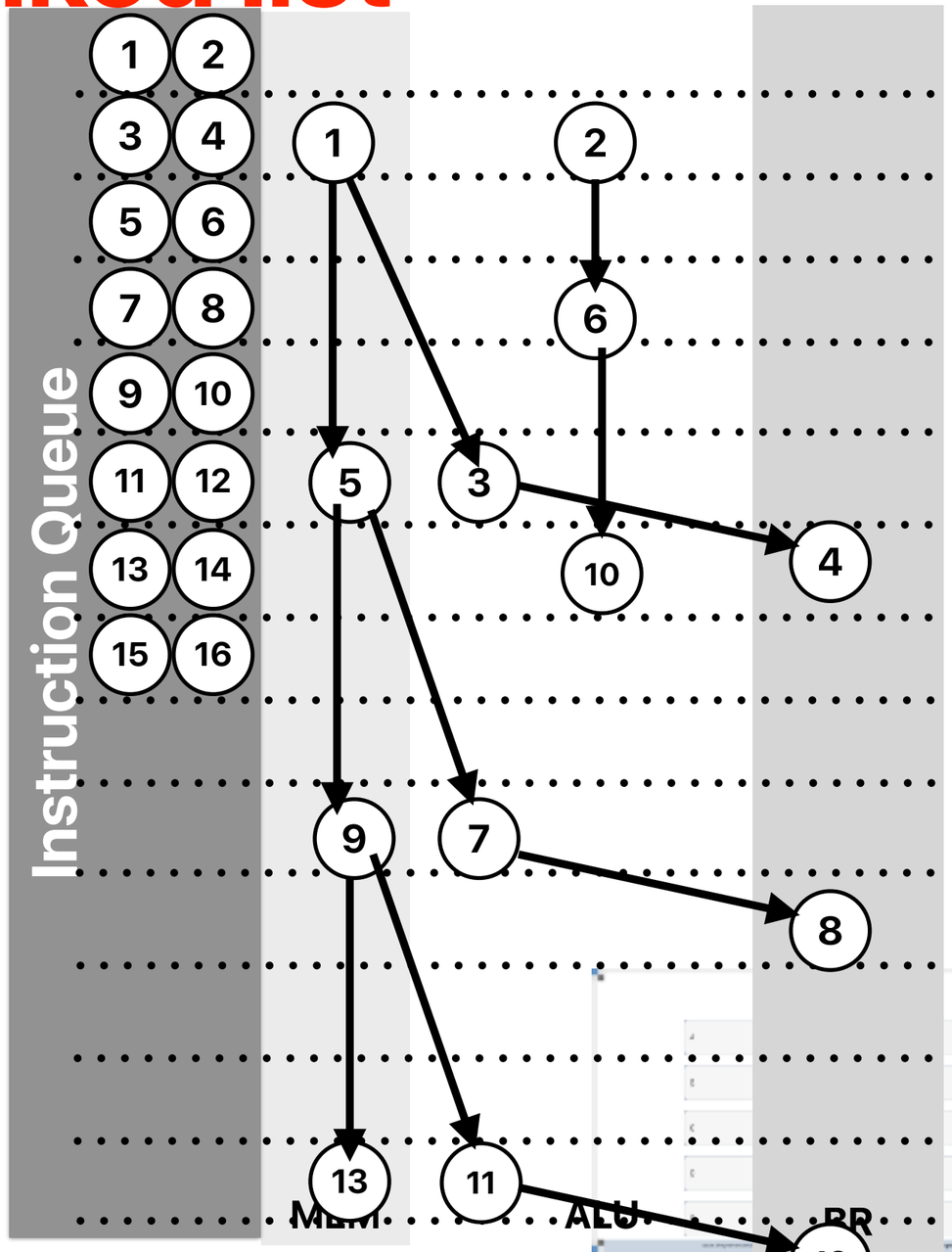


What about "linked list"

- For the following C code and it's translation in x86, **what's average CPI?** Assume the current PC is already at instruction (1) and this linked list has thousands of nodes. This processor can fetch and issue **2** instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )
```

A. 0.5
B. 0.8
C. 1.0
D. 1.2
E. 1.5



What about “linked list”

Performance determined by the critical path

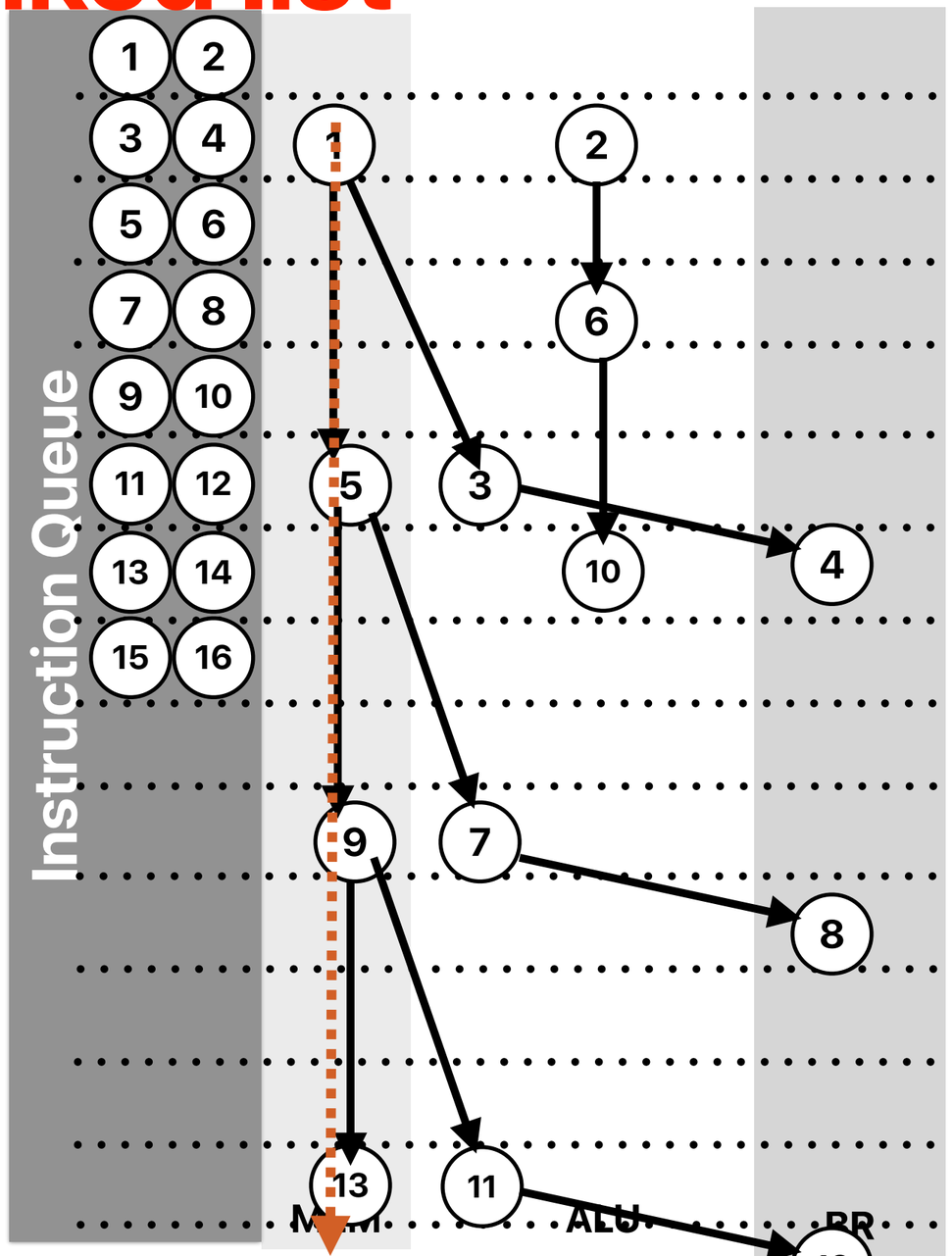
4 cycles each iteration

4 instructions per iteration

$$CPI = \frac{4}{4} = 1$$

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



The pipelines of Modern Processors

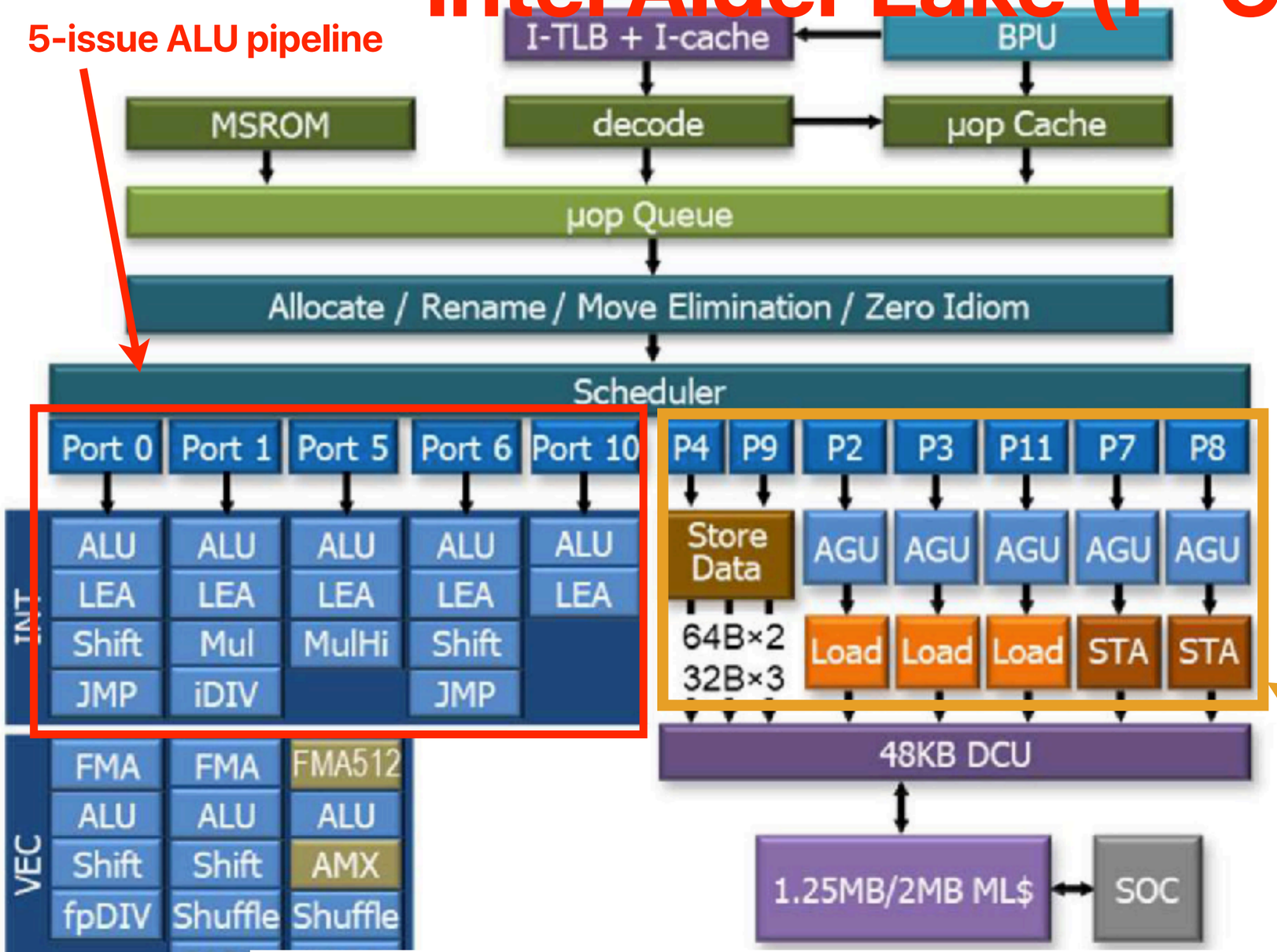
Intel Alder Lake (P-Core)

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

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

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

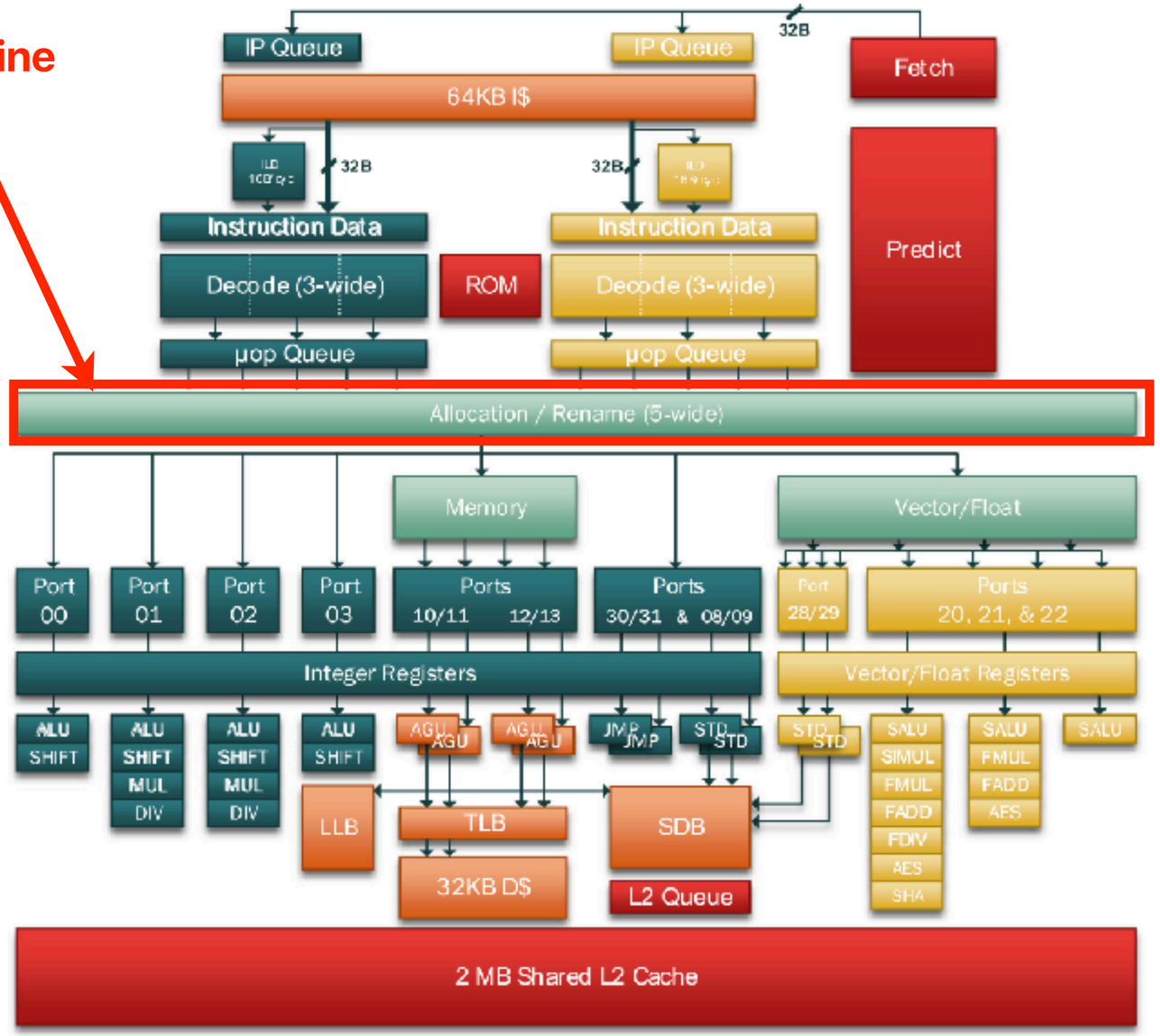
$$MinBRInst.CPI = \frac{1}{2}$$



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

Intel Alder Lake (E-Core)

5-issue pipeline



Announcements

- **Last reading quiz** due this Wednesday before the lecture
- Assignment #4 is up due this Sunday

Computer
Science &
Engineering

142

つづく