Instruction Scheduling & Programming Modern Processors (I)

Hung-Wei Tseng

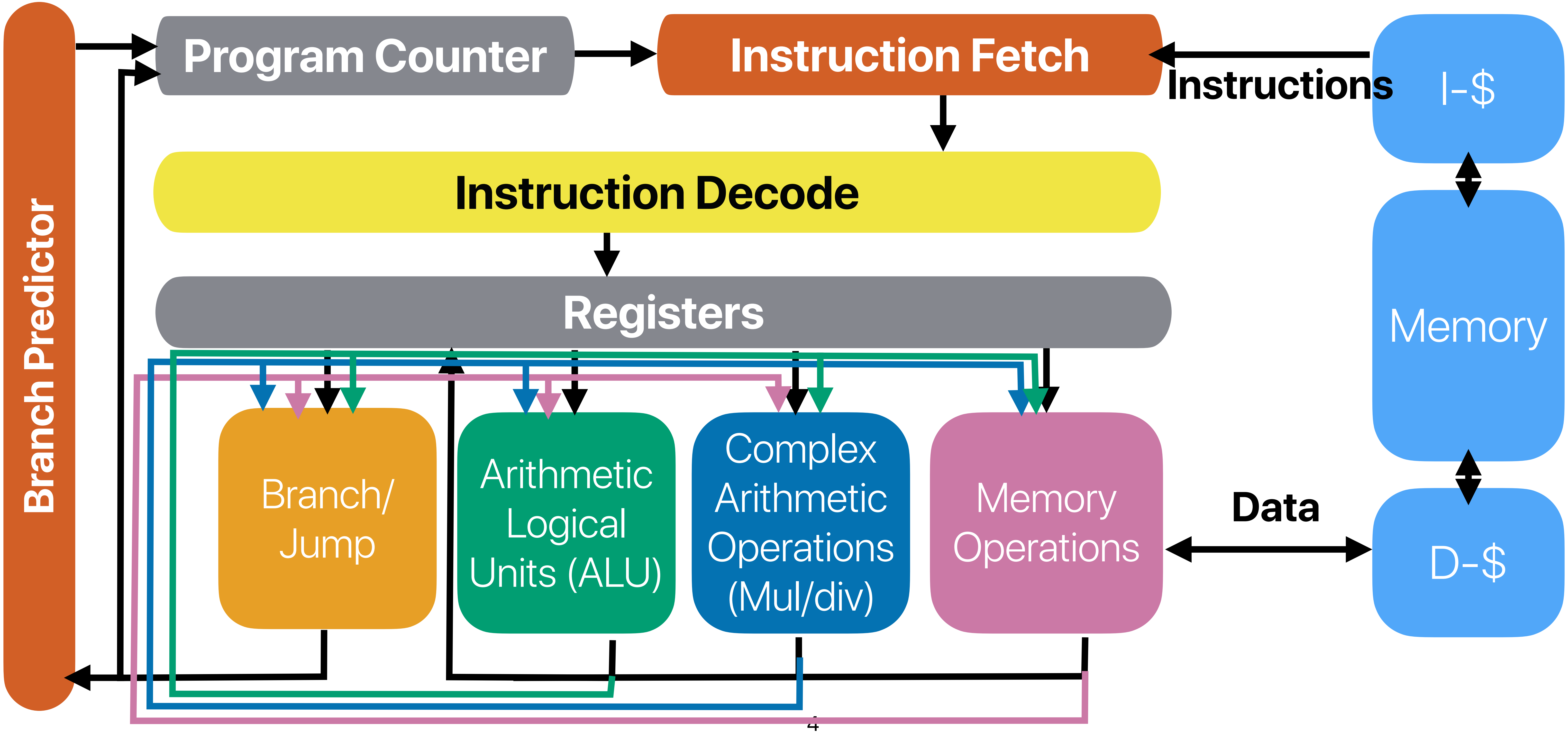
Recap: Three pipeline hazards

- 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

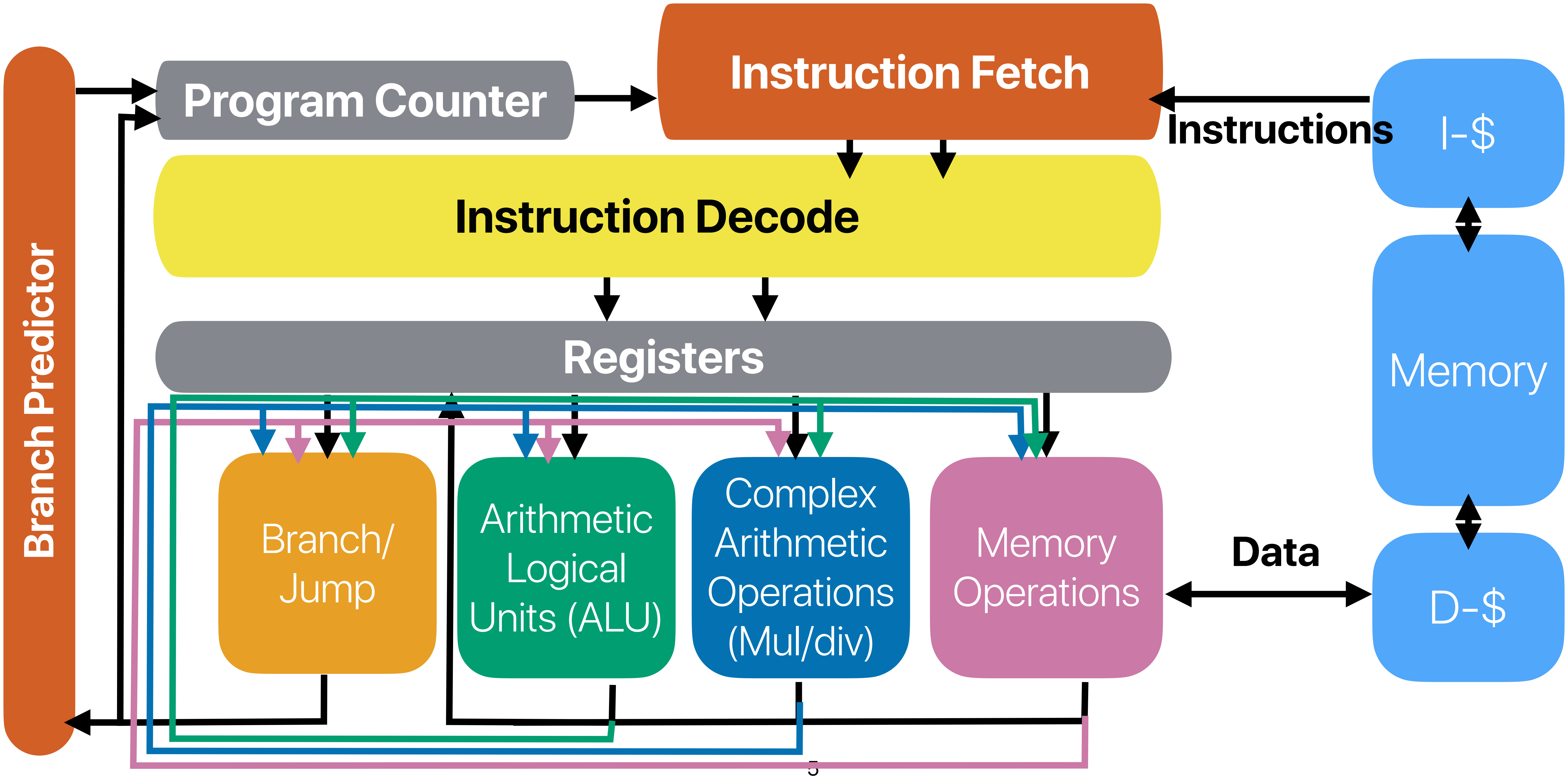
Recap: addressing hazards

- Structural hazards
  - Stall
  - Modify hardware design
- Control hazards
  - Stall
  - Static prediction
  - Dynamic prediction
- Data Hazards
  - Stall
  - Data forwarding
  - Dynamic instruction scheduling

Recap: Data “forwarding”



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

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

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)
13	(11)(12)	(10)	(9)	(8)			(7)
14		(11)(12)	(10)	(9)	(8)		
15			(11)(12)	(10)	(9)	(8)	
16				(11)(12)	(10)	(9)	(8)
17					(11)	(10)	(9)

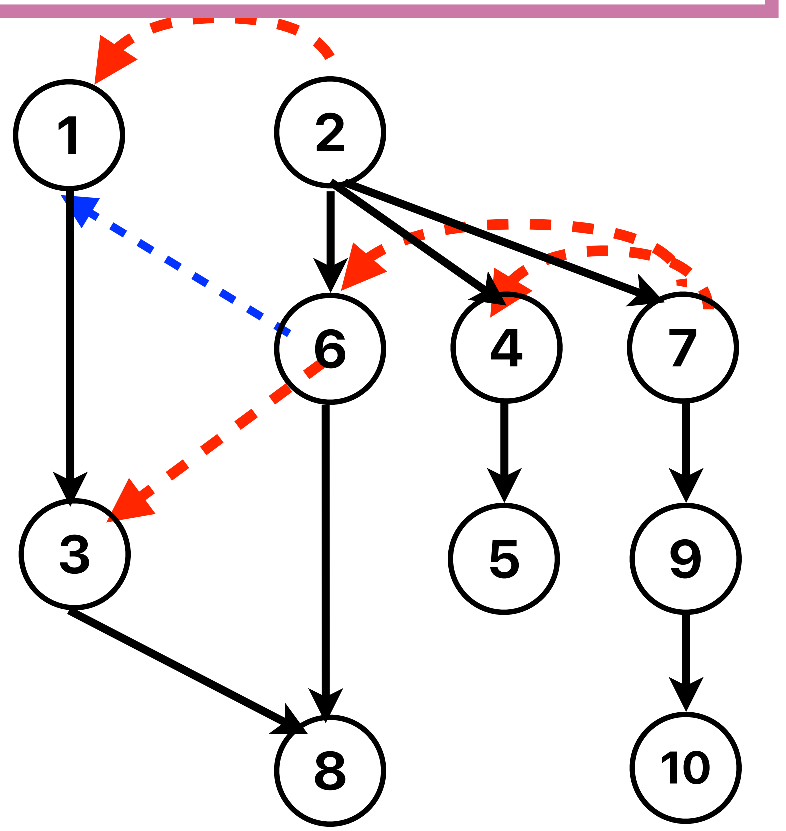
Everything we need for (4) is ready here! Why can't we execute it?

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

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
```



Recap: 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

Outline

- Out-of-order, Dynamic instruction scheduling
- Programming on Modern Processor

Register renaming + speculative execution

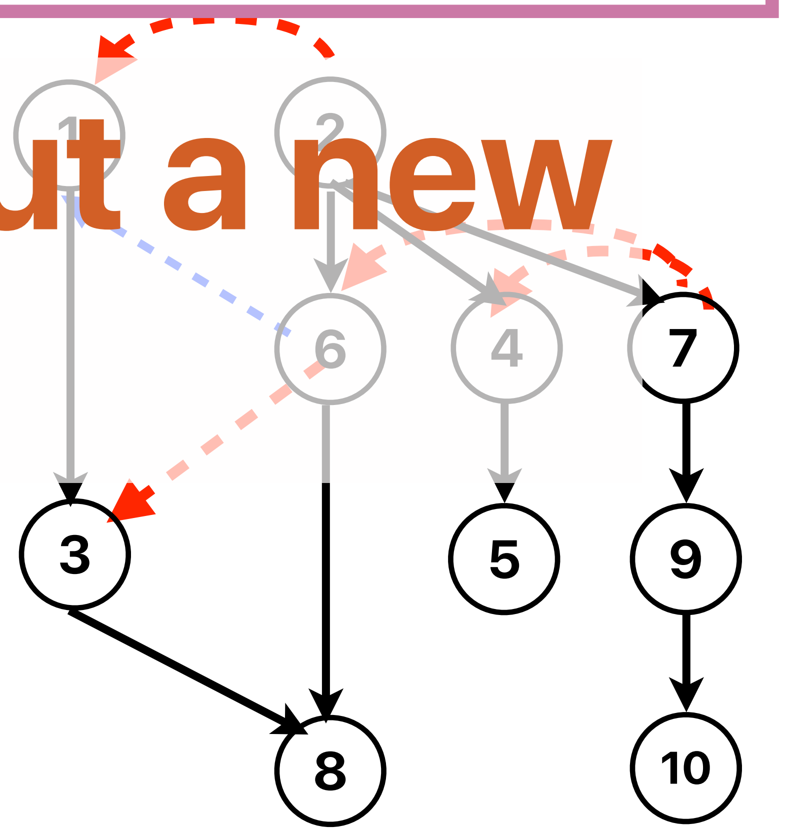
- K. C. Yeager, “The Mips R10000 superscalar microprocessor,” in IEEE Micro, vol. 16, no. 2, pp. 28–41, April 1996.

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
```

→

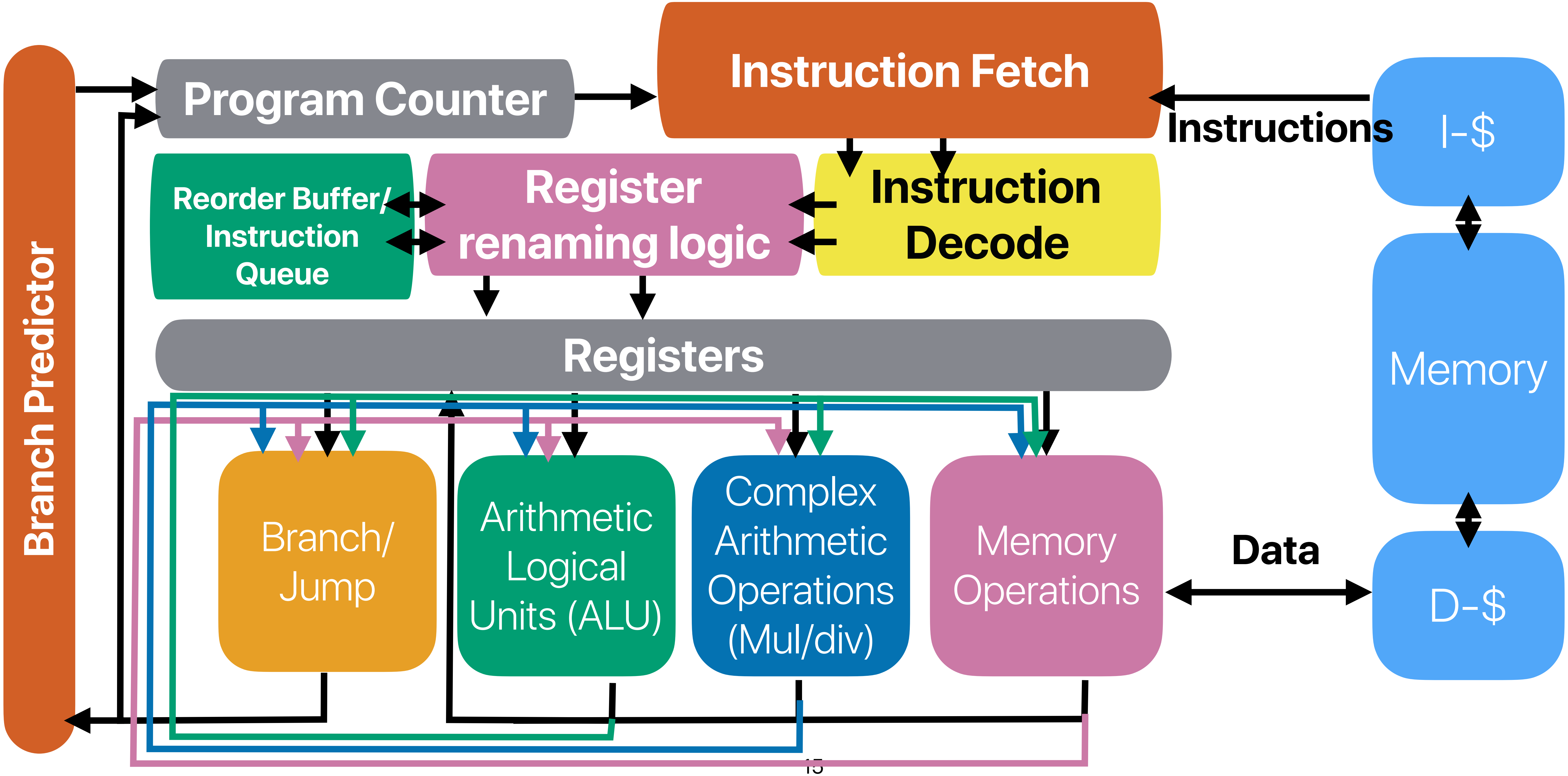
```
①  movl   (%rdi), %ecx
②  addq   $4, %rdi, %t0
③  addl   %ecx, %eax, %t1
④  cmpq   %rdx, %t0
⑤  jne    .L3
⑥  movl   (%t0), %t2
⑦  addq   $4, %t0, %t3
⑧  addl   %t1, %t2, %t4
⑨  cmpq   %rdx, %t3
⑩  jne    .L3
```

All false dependencies are gone!!!

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

```
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
```

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

	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

```
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
```

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

	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

```
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
```

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

	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

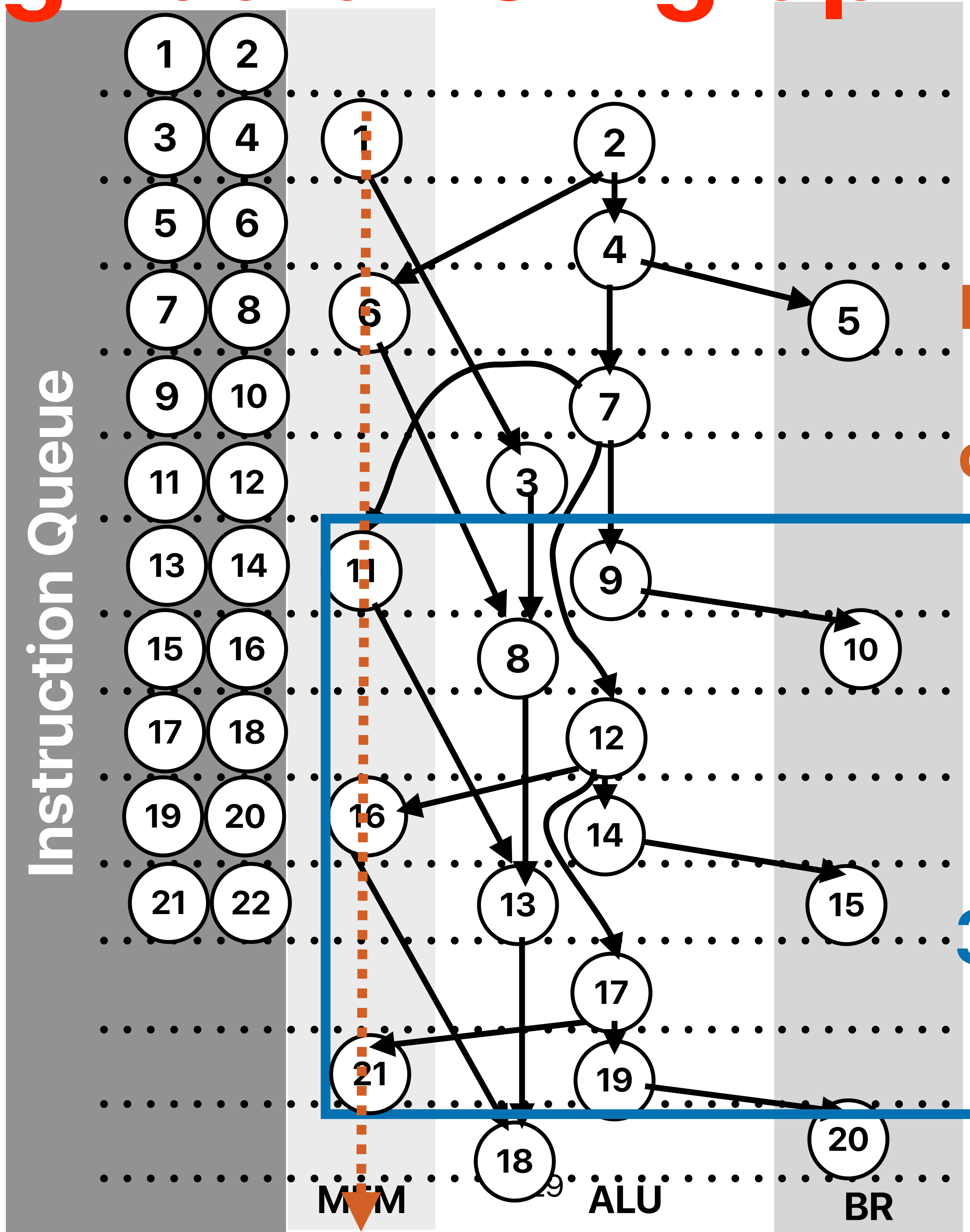
```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx → P7
⑫  addq    $4, %rdi → P8
⑬  addl 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

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

	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

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

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

What about "linked list"

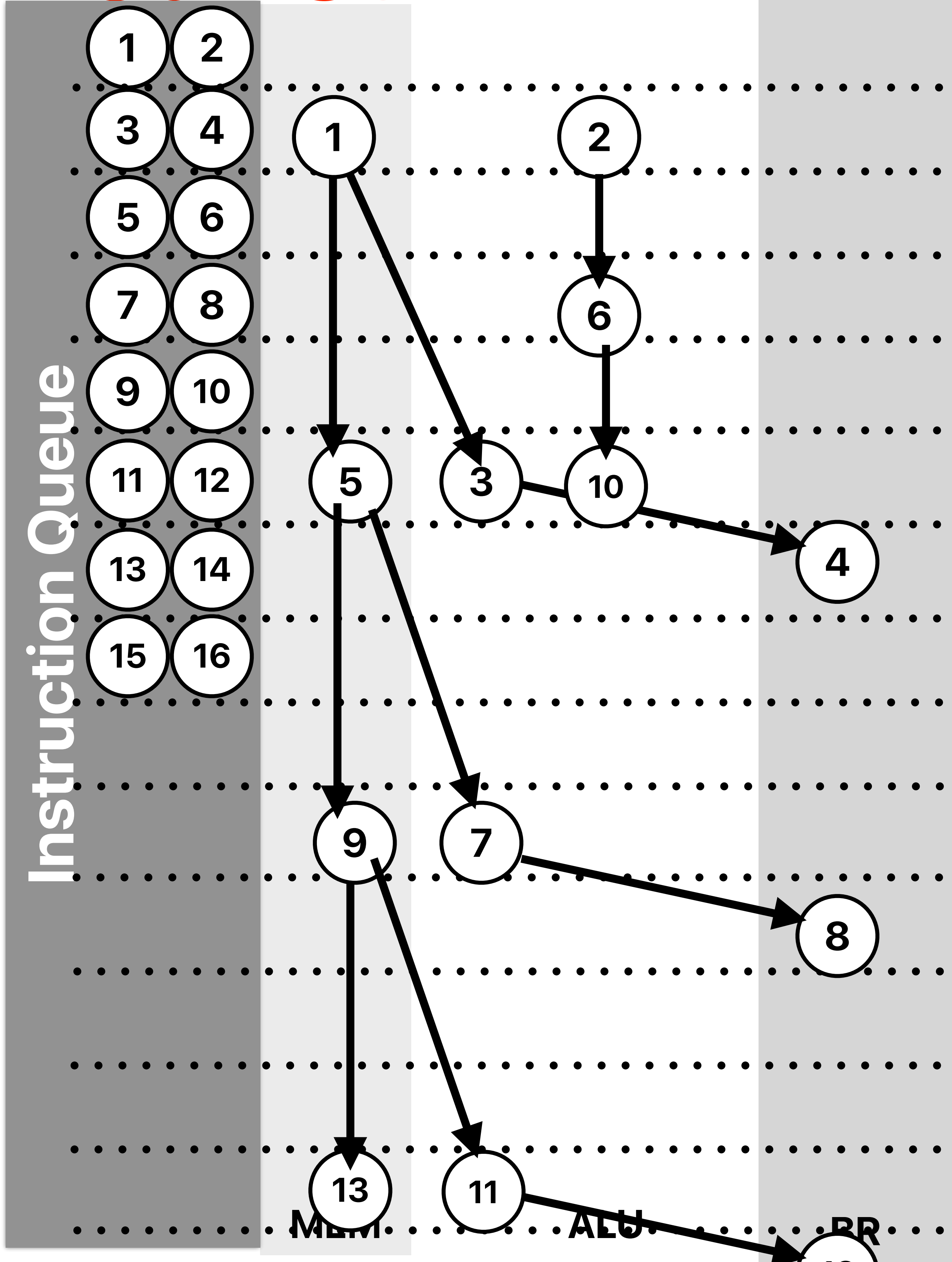
- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.
  Which of the following C state of the code snippet determines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



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

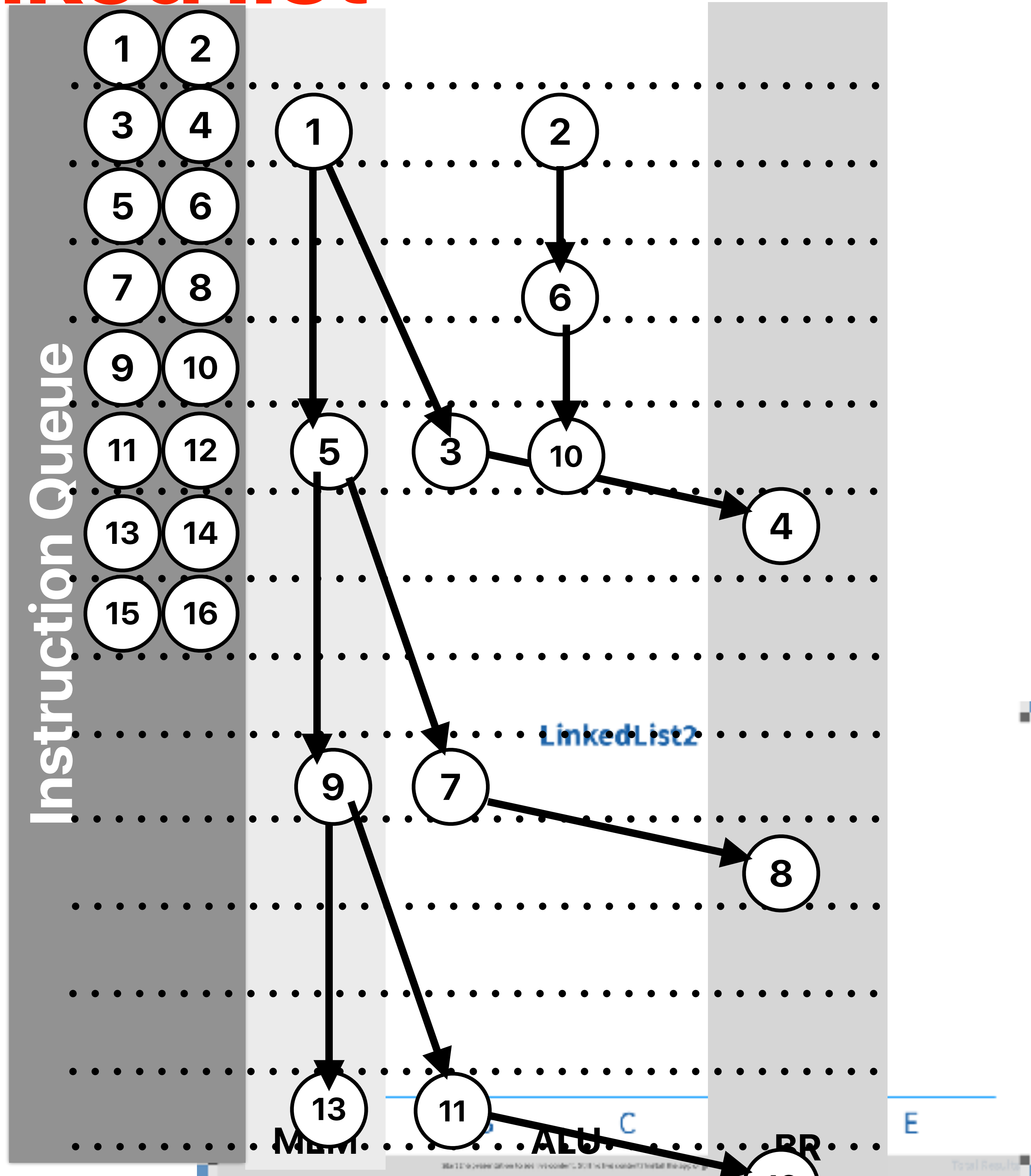
What about "linked list"

- 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

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



What about “linked list”

Performance determined by the critical path

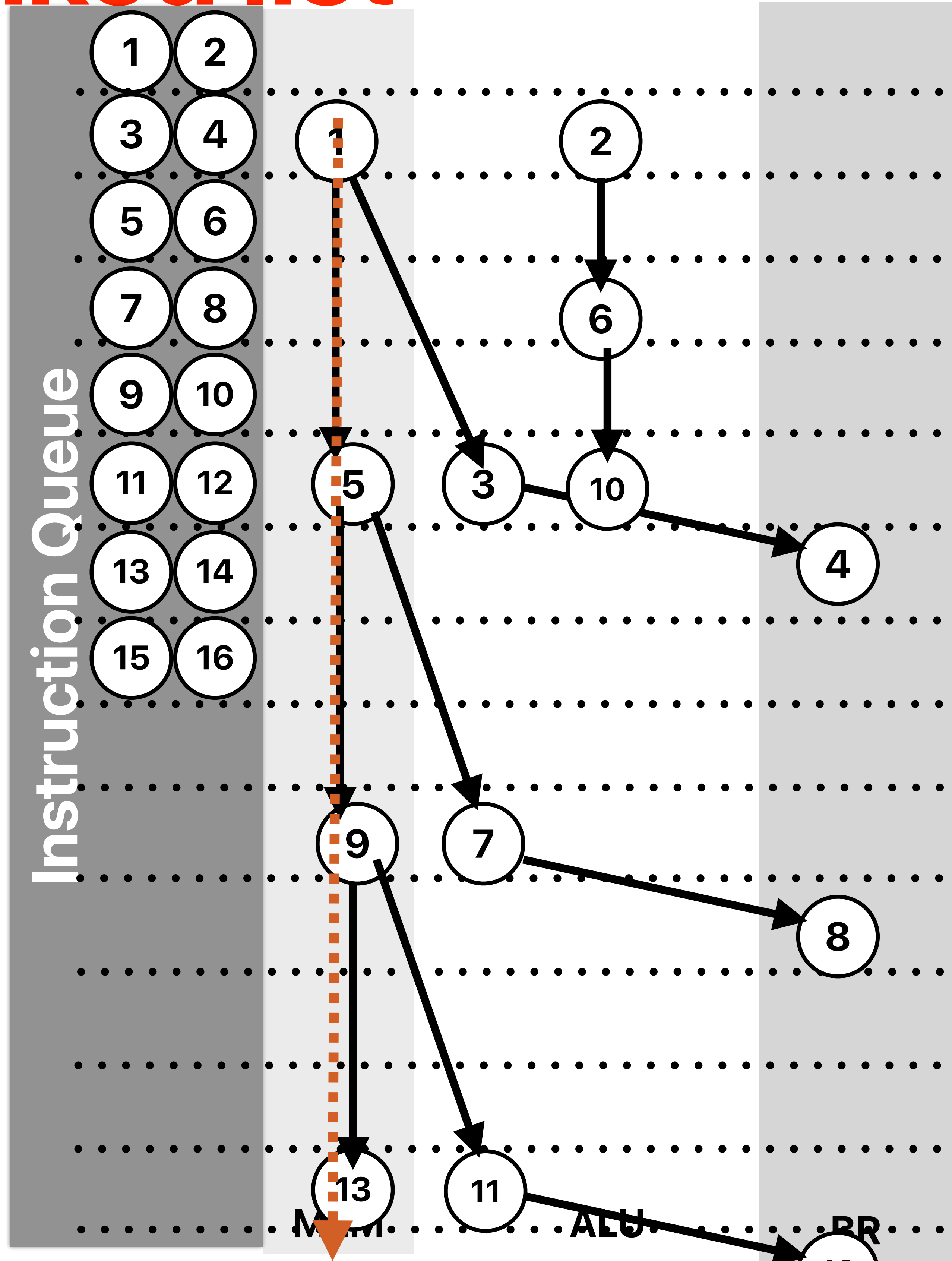
4 cycles each iteration

4 instructions per iteration

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

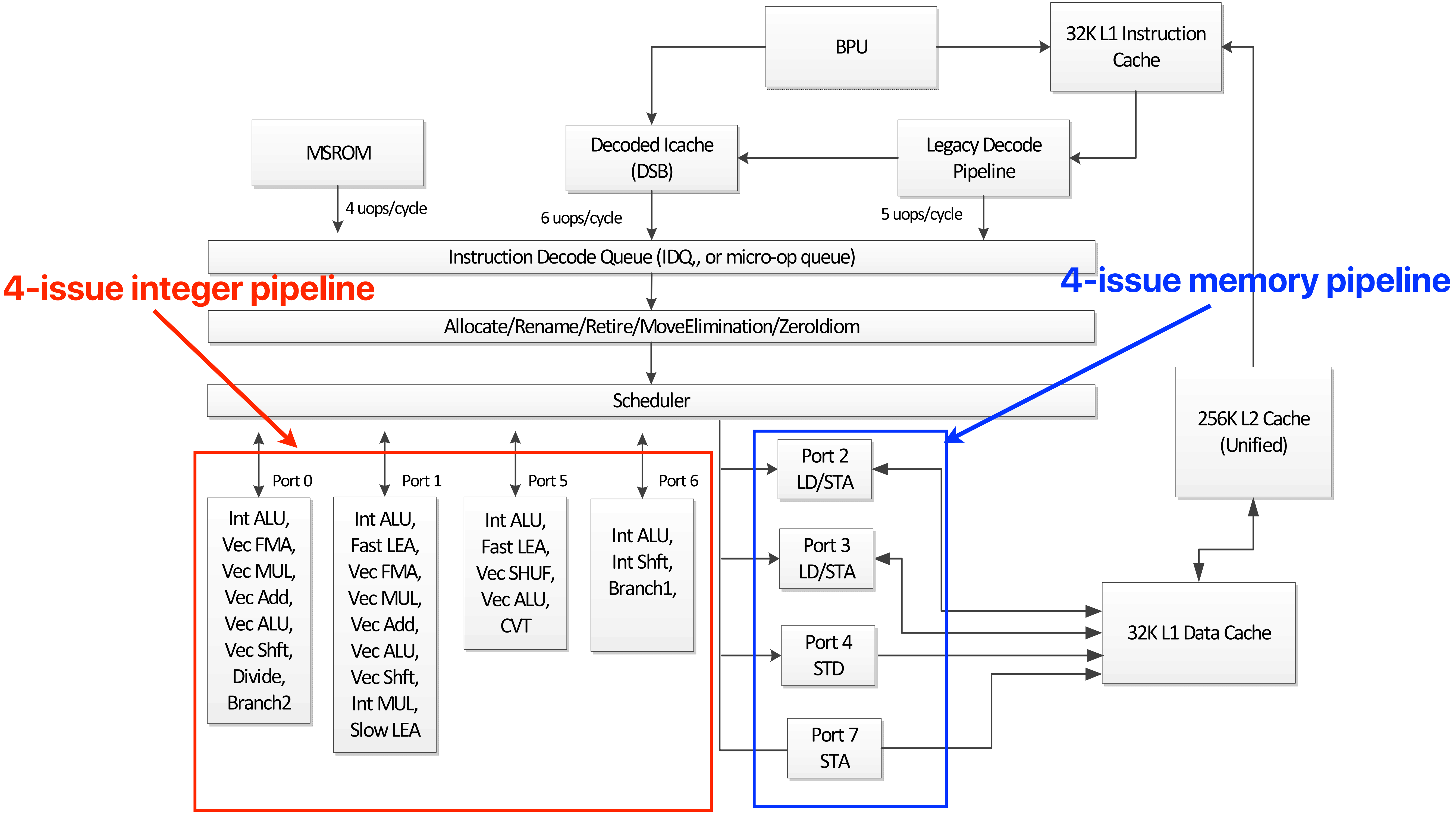
```
do {
     number_of_nodes++;
     current = current->next;
} while ( current != NULL );
```

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



The pipelines of Modern Processors

Intel Skylake



BPU
32K L1 Instruction Cache
MSROM
Decoded Icache (DSB)
Legacy Decode Pipeline
4 uops/cycle
6 uops/cycle
5 uops/cycle
Instruction Decode Queue (IDQ,, or micro-op queue)
4-issue integer pipeline
4-issue memory pipeline
Allocate/Rename/Retire/MoveElimination/ZeroIdiom
Scheduler
256K L2 Cache (Unified)
Port 0
Port 1
Port 5
Port 6
Int ALU, Vec FMA, Vec MUL, Vec Add, Vec ALU, Vec Shft, Divide, Branch2
Int ALU, Fast LEA, Vec FMA, Vec MUL, Vec Add, Vec ALU, Vec Shft, Int MUL, Slow LEA
Int ALU, Fast LEA, Vec SHUF, Vec ALU, CVT
Int ALU, Int Shft, Branch1,
Port 2 LD/STA
Port 3 LD/STA
Port 4 STD
Port 7 STA
32K L1 Data Cache

Recap: Intel Skylake

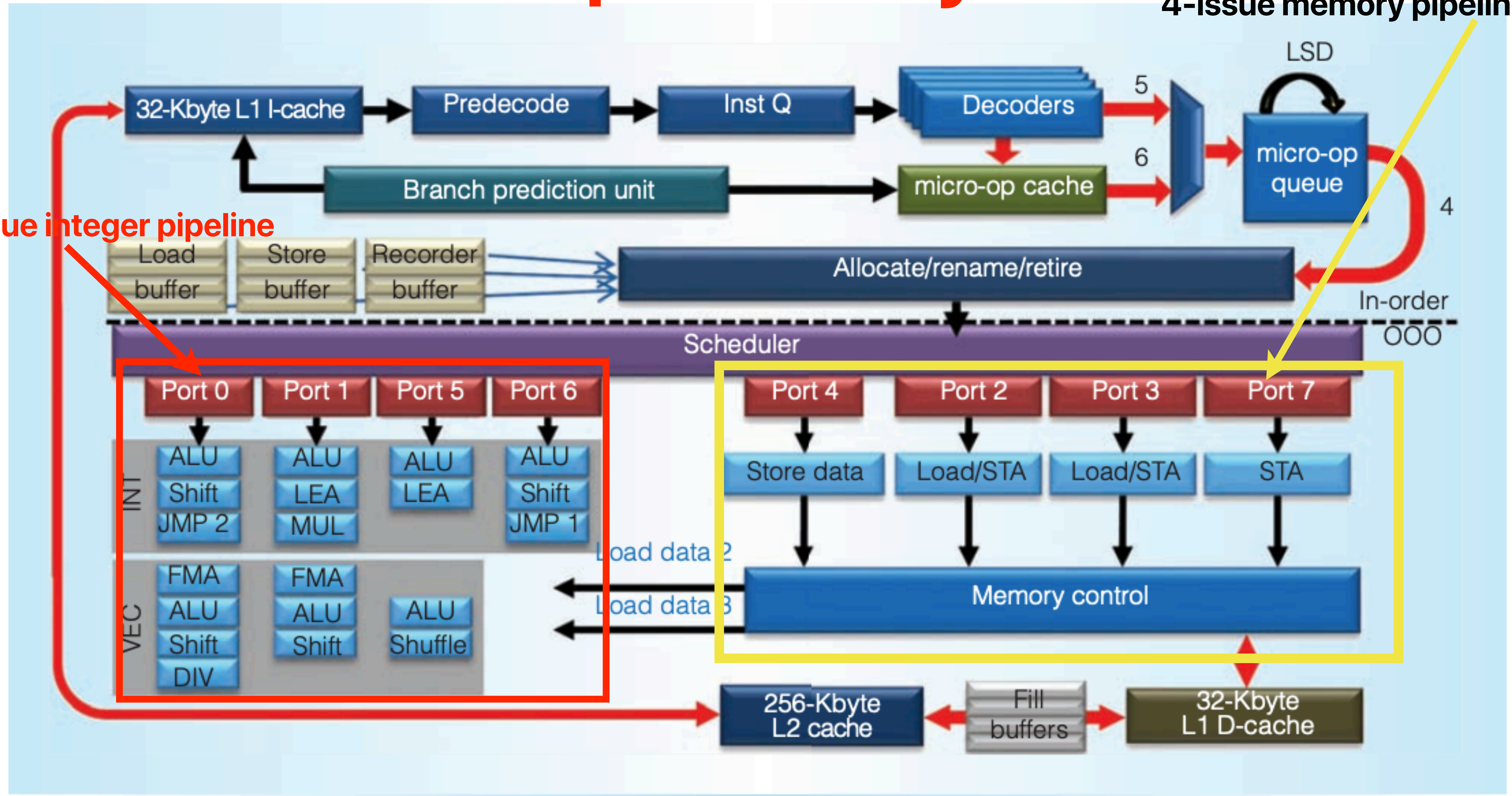
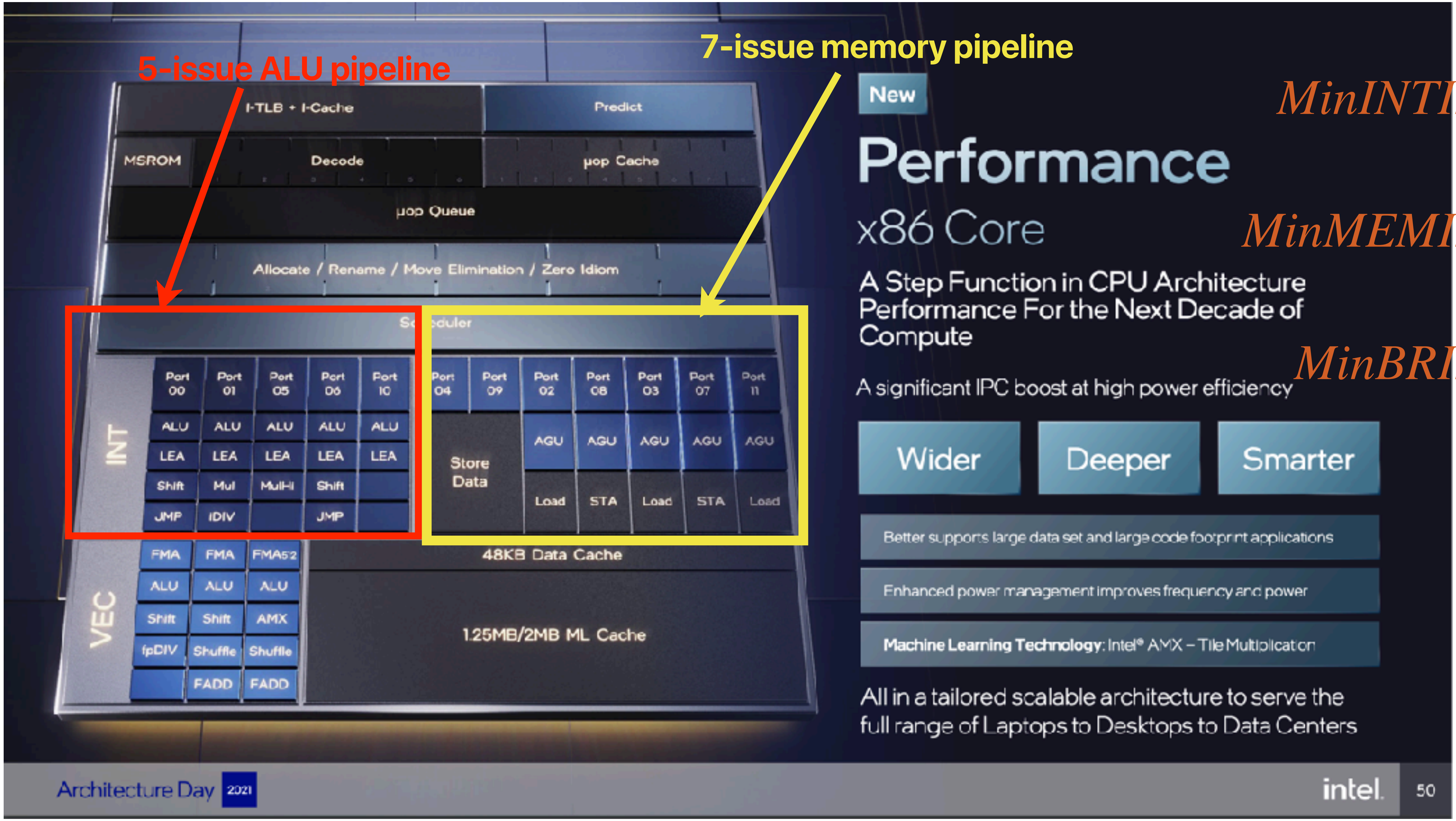


Figure 4. Skylake core block diagram.

Intel Alder Lake



New

Performance

x86 Core

A Step Function in CPU Architecture Performance For the Next Decade of Compute

A significant IPC boost at high power efficiency

Wider | Deeper | Smarter

Better supports large data set and large code footprint applications

Enhanced power management improves frequency and power

Machine Learning Technology: Intel® AMX – Tile Multiplication

All in a tailored scalable architecture to serve the full range of Laptops to Desktops to Data Centers

Architecture Day 2021

intel.

50

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

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

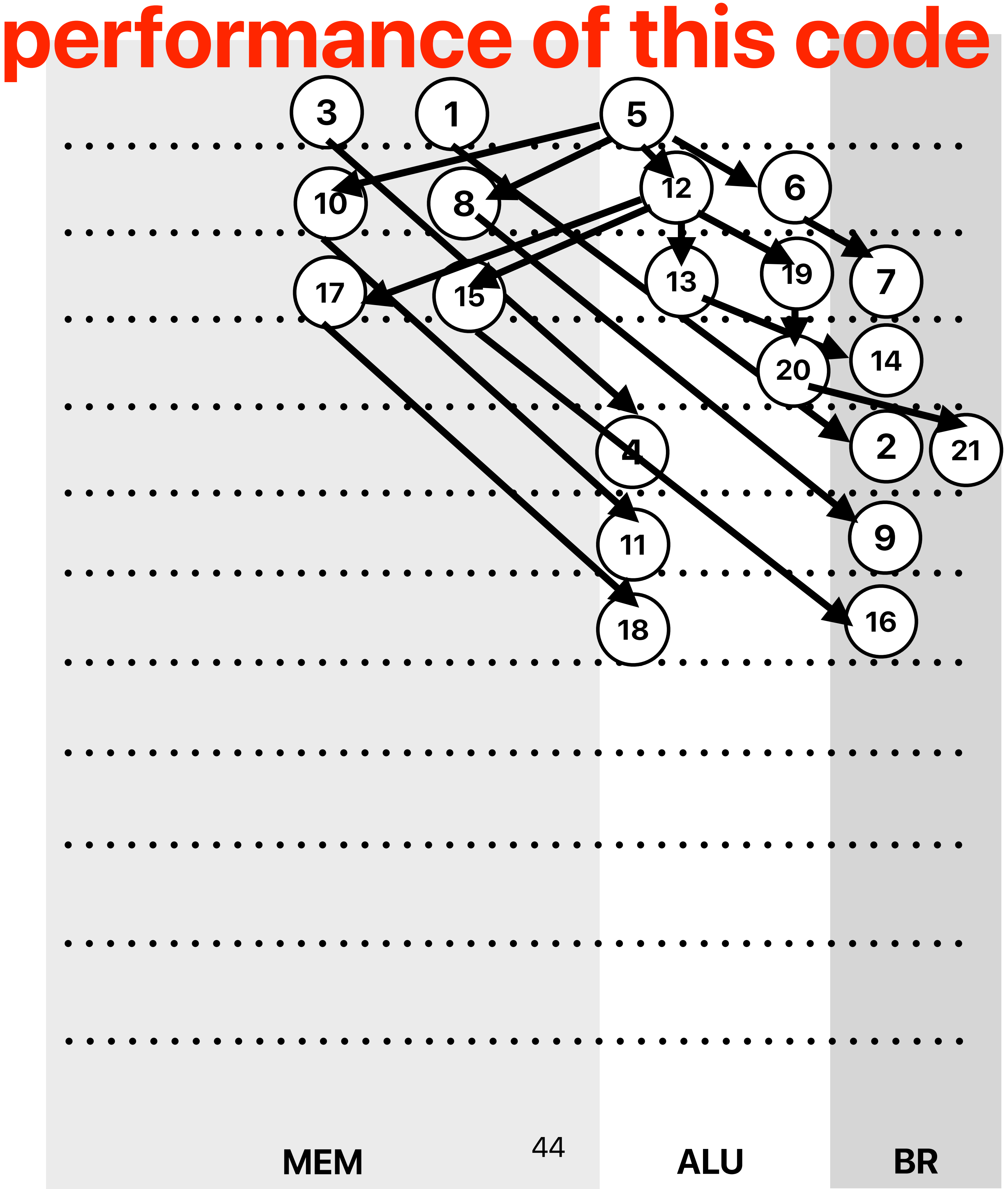
$$MinMEMInst.CPI = \frac{1}{7}$$

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

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

Project the performance of this code on Alder Lake

```
①  .L10:
    cmpq   $0, 8(%rdi)
②  je  .L9
③  movslq (%rdi), %rdx
④  addq   %rdx, %rax
⑤  .L9:
    addq   $16, %rdi
⑥  cmpq   %rdi, %rcx
⑦  jne .L10
⑧  .L10:
    cmpq   $0, 8(%rdi)
⑨  je  .L9
⑩  movslq (%rdi), %rdx
⑪  addq   %rdx, %rax
⑫  .L9:
    addq   $16, %rdi
⑬  cmpq   %rdi, %rcx
⑭  jne .L10
⑮  .L10:
    cmpq   $0, 8(%rdi)
⑯  je  .L9
⑰  movslq (%rdi), %rdx
⑱  addq   %rdx, %rax
⑲  .L9:
    addq   $16, %rdi
⑳  cmpq   %rdi, %rcx
㉑  jne .L10
```



AMD Zen 3 (RyZen 5000 Series)

3-issue memory pipeline

4-issue integer pipeline + 1 additional branch

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

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

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

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

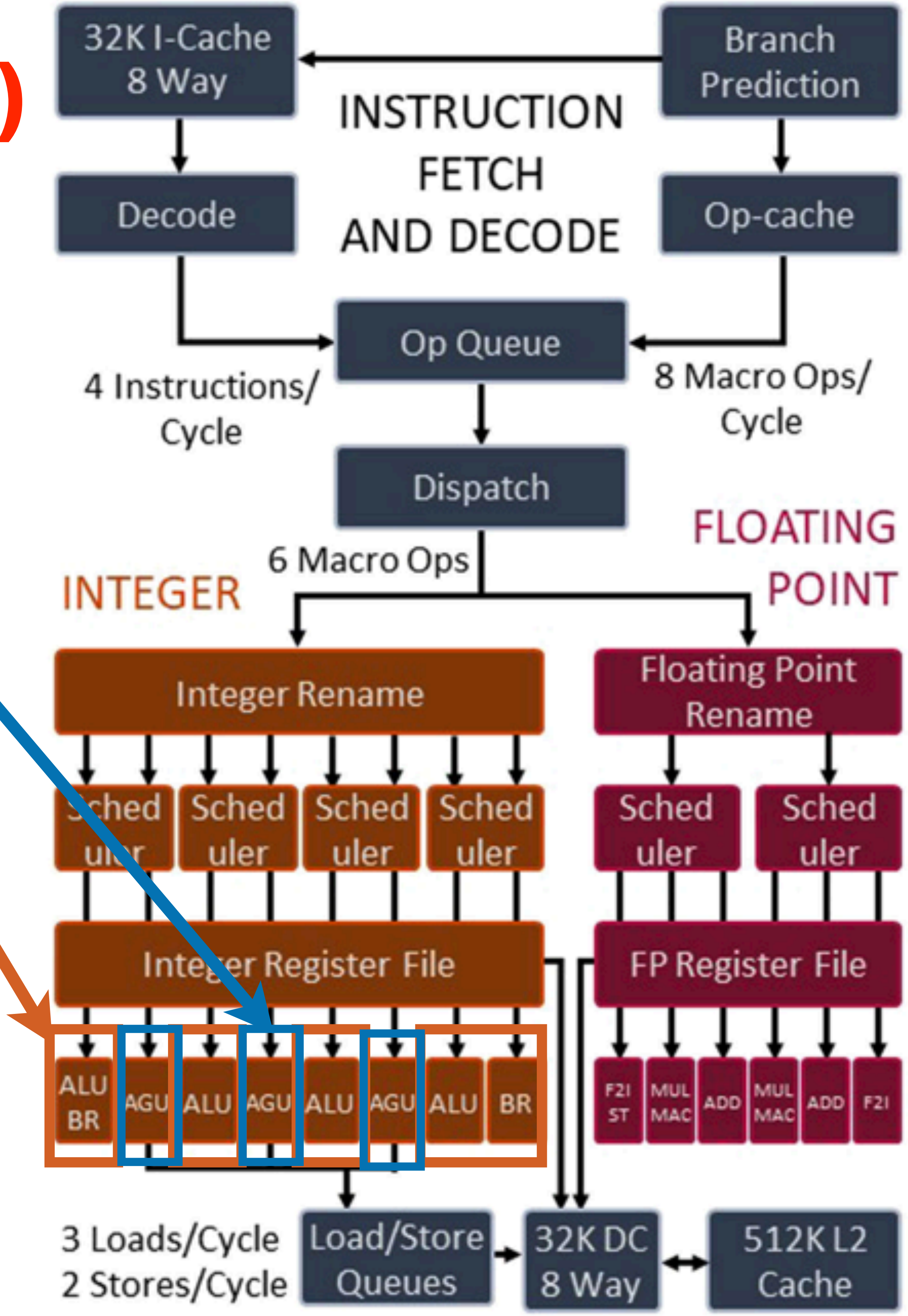


FIGURE 1. "Zen 3" block diagram.

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
- Branch predictors

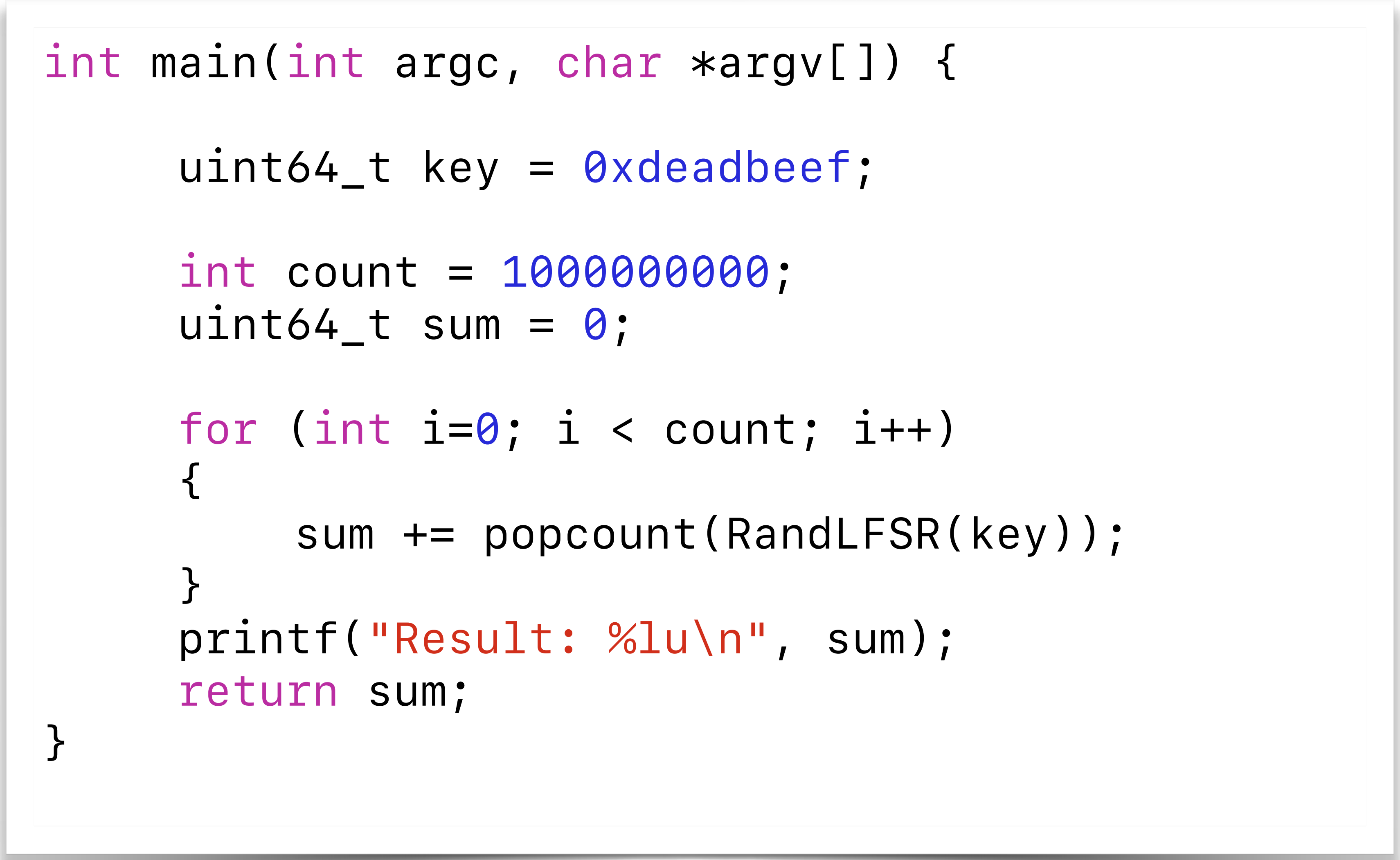
Performance Programming on Modern Processors

Demo: Popcount

- The population count (or popcount) of a specific value is the number of set bits (i.e., bits in 1s) in that value.
- Applications
  - Parity bits in error correction/detection code
  - Cryptography
  - Sparse matrix
  - Molecular Fingerprinting
  - Implementation of some succinct data structures like bit vectors and wavelet trees.

Demo: pop count

- Given a 64-bit integer number, find the number of 1s in its binary representation.
- Example 1:
  Input: 9487
  Output: 7
  Explanation: 9487's binary representation is
  0b10010100001111

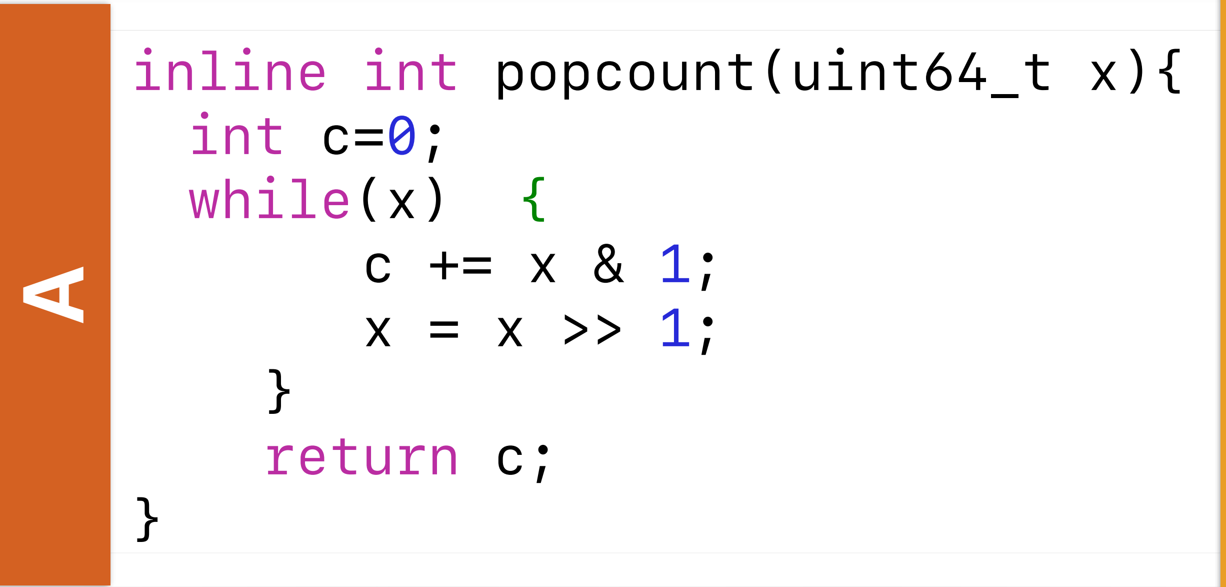


```
int main(int argc, char *argv[]) {

	uint64_t key = 0xdeadbeef;

	int count = 1000000000;
	uint64_t sum = 0;

	for (int i=0; i < count; i++)
	{
		sum += popcount(RandLFSR(key));
	}
	printf("Result: %lu\n", sum);
	return sum;
}
```

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

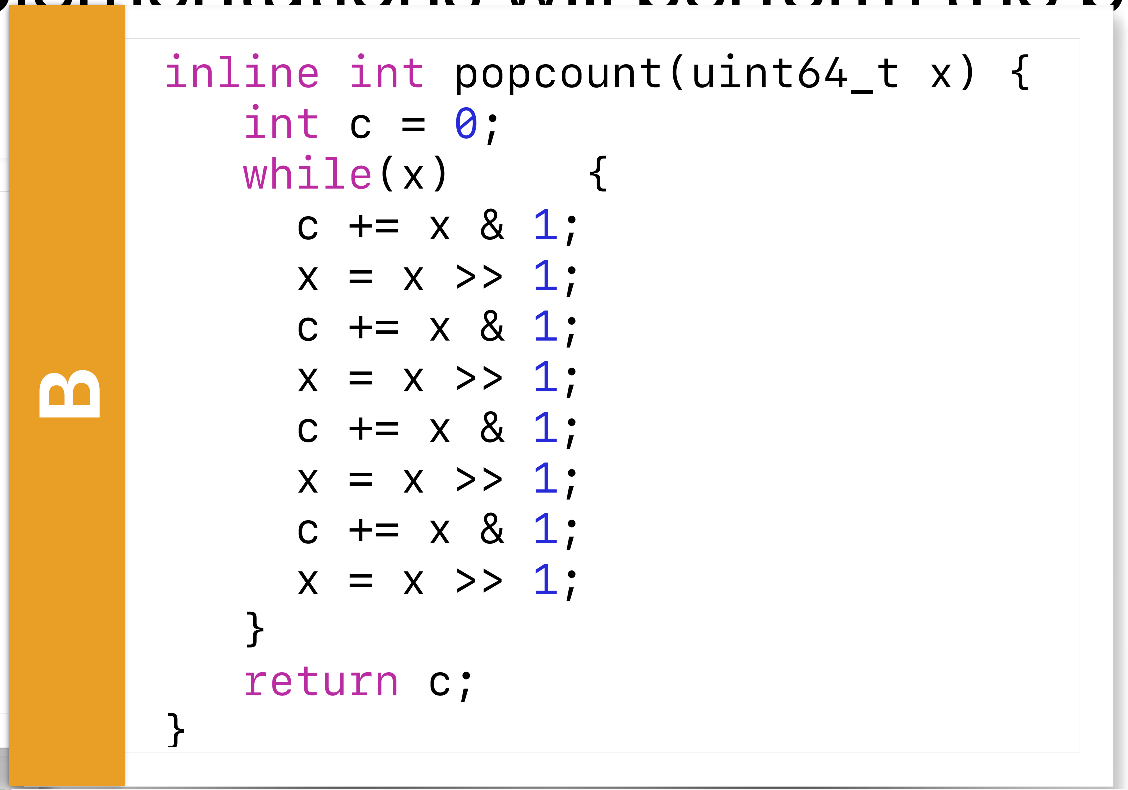
Five implementations

- Which of the following implementations will perform the best on modern pipeline processors?



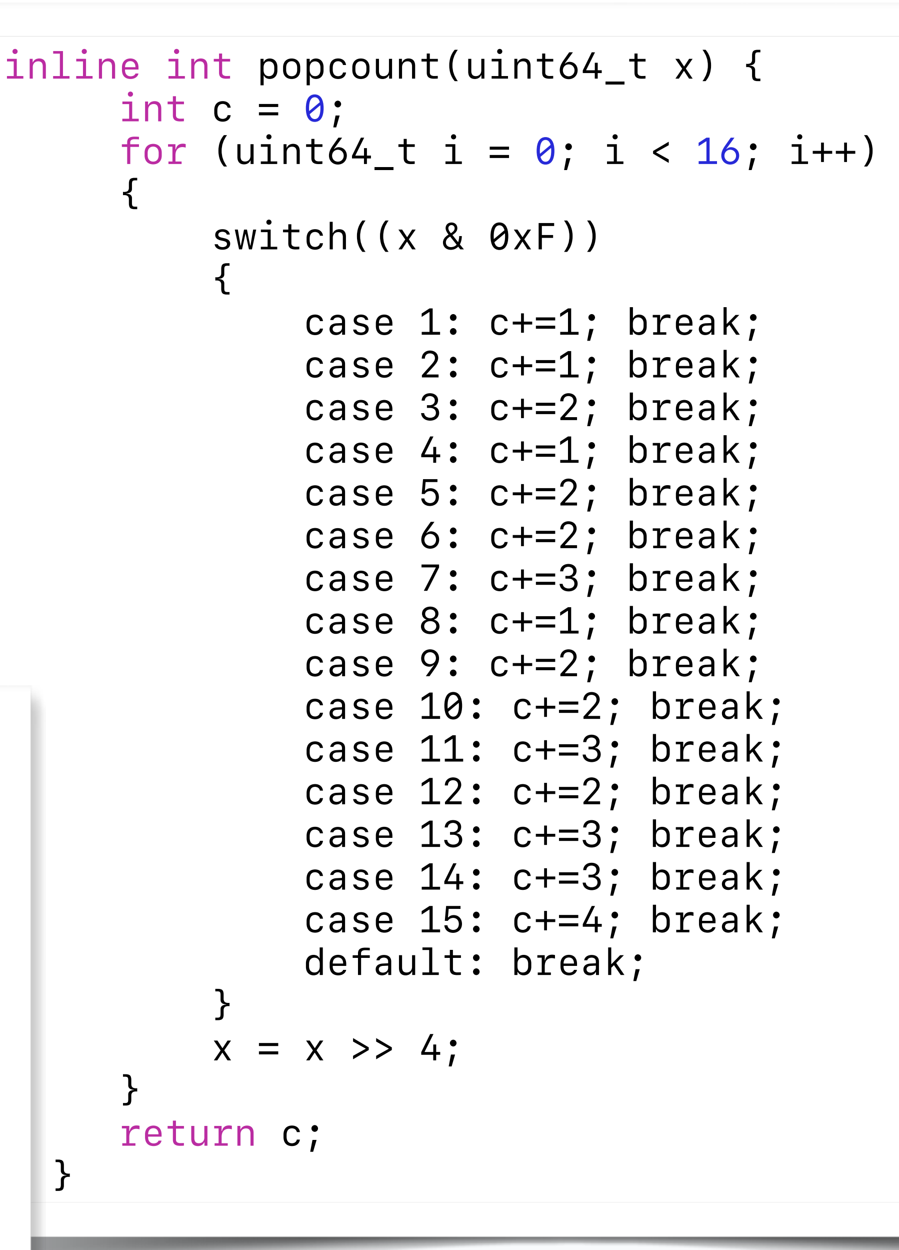
A

```
inline int popcount(uint64_t x){
  int c=0;
  while(x)  {
        c += x & 1;
        x = x >> 1;
    }
    return c;
}
```



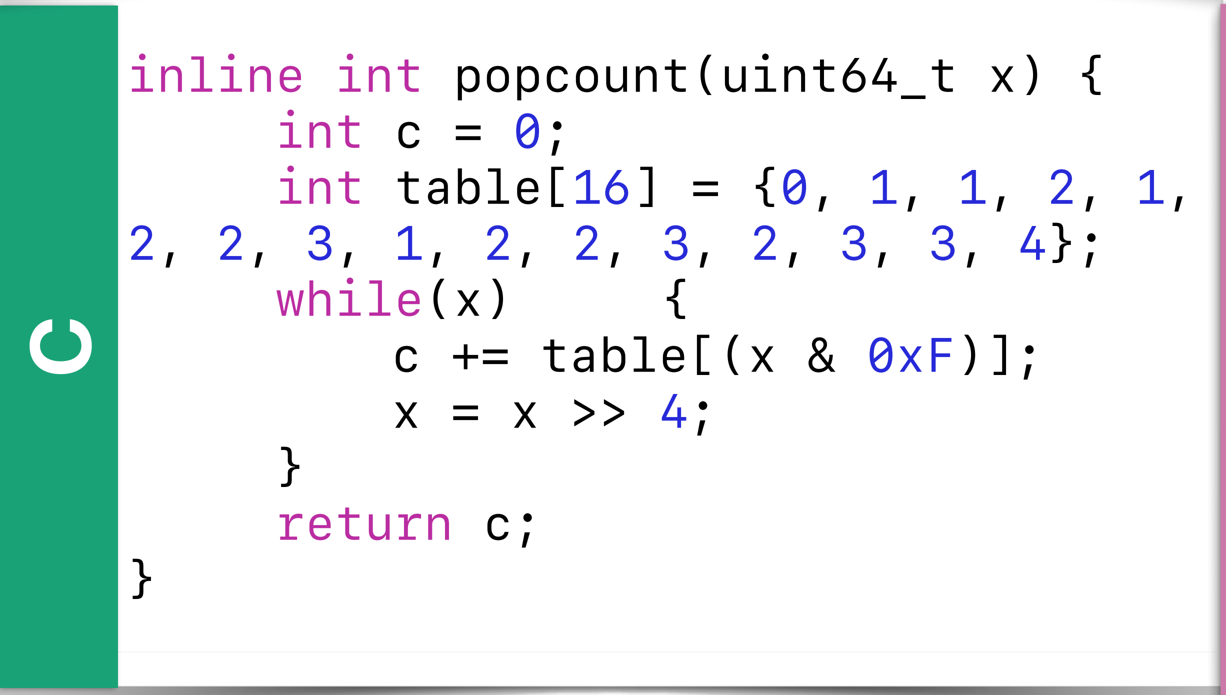
B

```
inline int popcount(uint64_t x) {
   int c = 0;
   while(x)     {
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
   }
   return c;
}
```



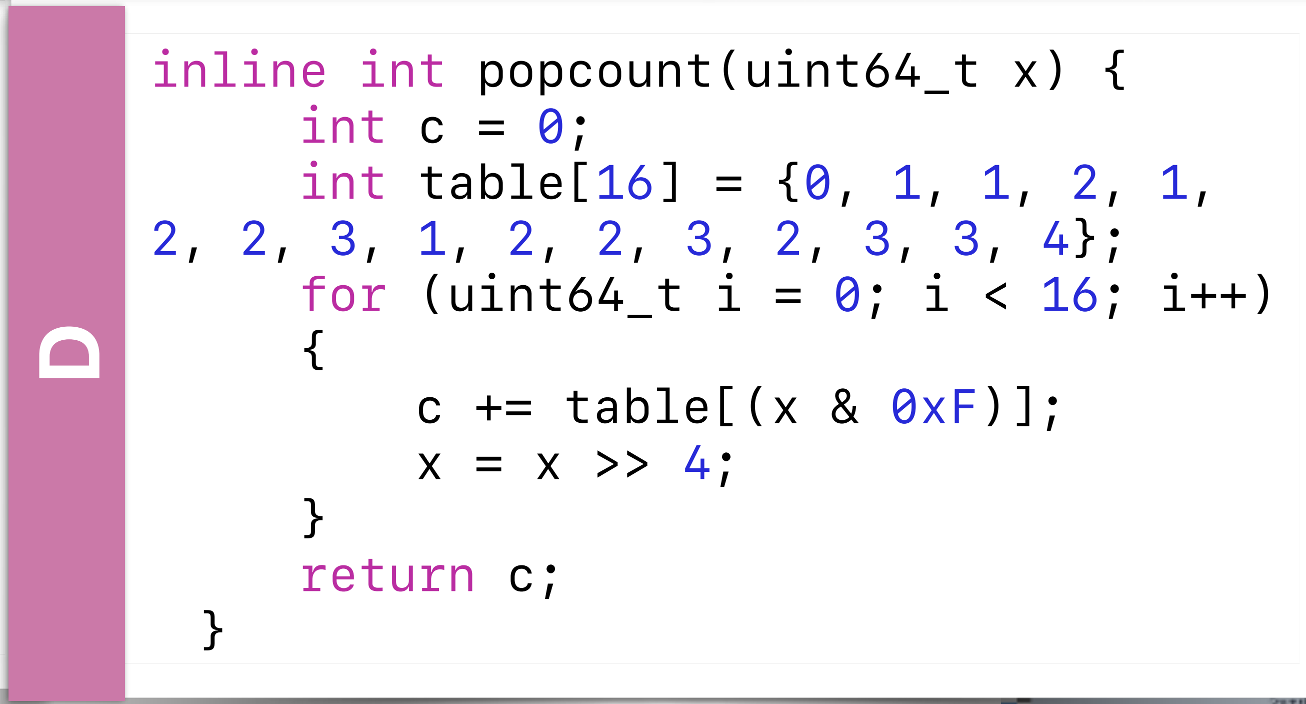
E

```
inline int popcount(uint64_t x) {
     int c = 0;
     for (uint64_t i = 0; i < 16; i++)
     {
         switch((x & 0xF))
         {
             case 1: c+=1; break;
             case 2: c+=1; break;
             case 3: c+=2; break;
             case 4: c+=1; break;
             case 5: c+=2; break;
             case 6: c+=2; break;
             case 7: c+=3; break;
             case 8: c+=1; break;
             case 9: c+=2; break;
             case 10: c+=2; break;
             case 11: c+=3; break;
             case 12: c+=2; break;
             case 13: c+=3; break;
             case 14: c+=3; break;
             case 15: c+=4; break;
             default: break;
         }
         x = x >> 4;
     }
     return c;
}
```



C

```
inline int popcount(uint64_t x) {
     int c = 0;
     int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
     while(x)      {
          c += table[(x & 0xF)];
          x = x >> 4;
     }
     return c;
}
```



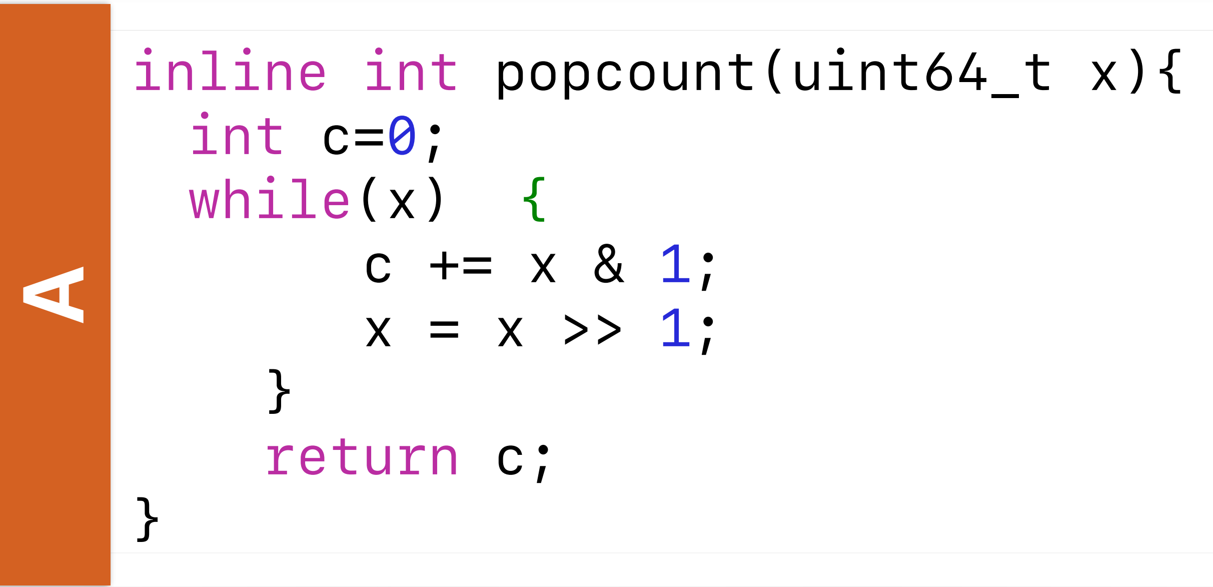
D

```
inline int popcount(uint64_t x) {
     int c = 0;
     int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
     for (uint64_t i = 0; i < 16; i++)
     {
          c += table[(x & 0xF)];
          x = x >> 4;
     }
     return c;
 }
```

Five implementations

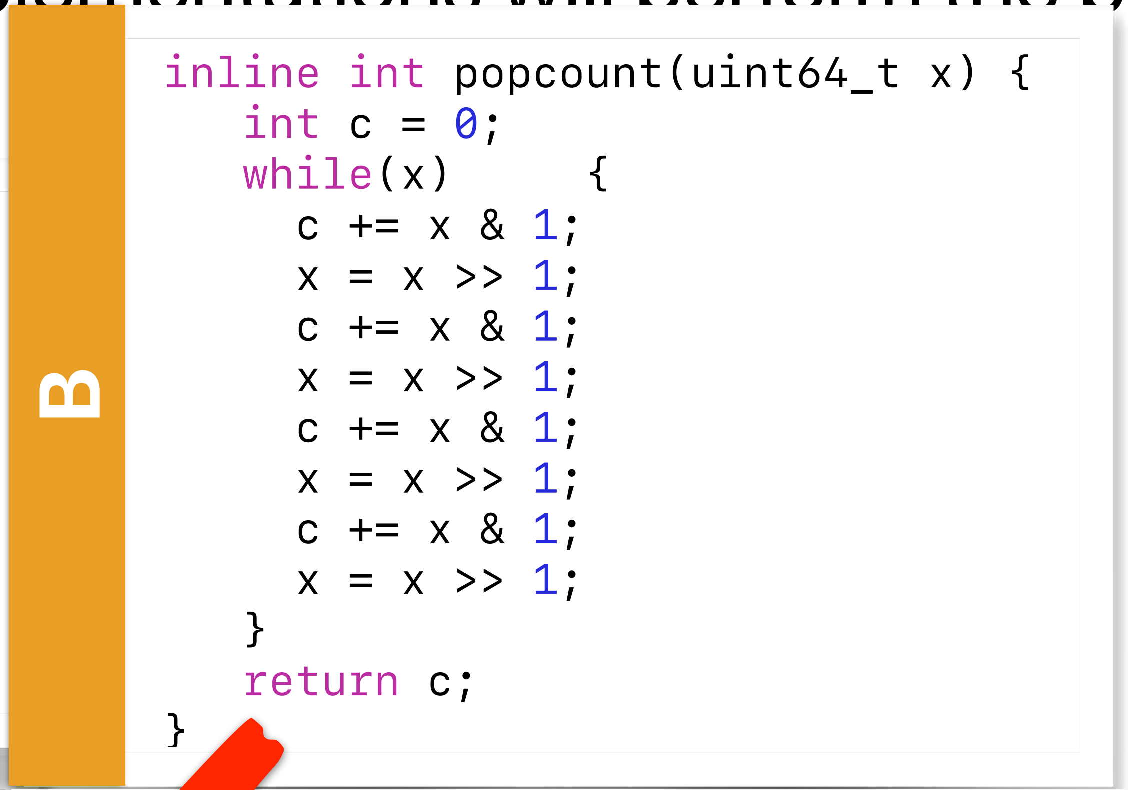
- Which of the following implementations will perform the best on modern pipeline processors?

A



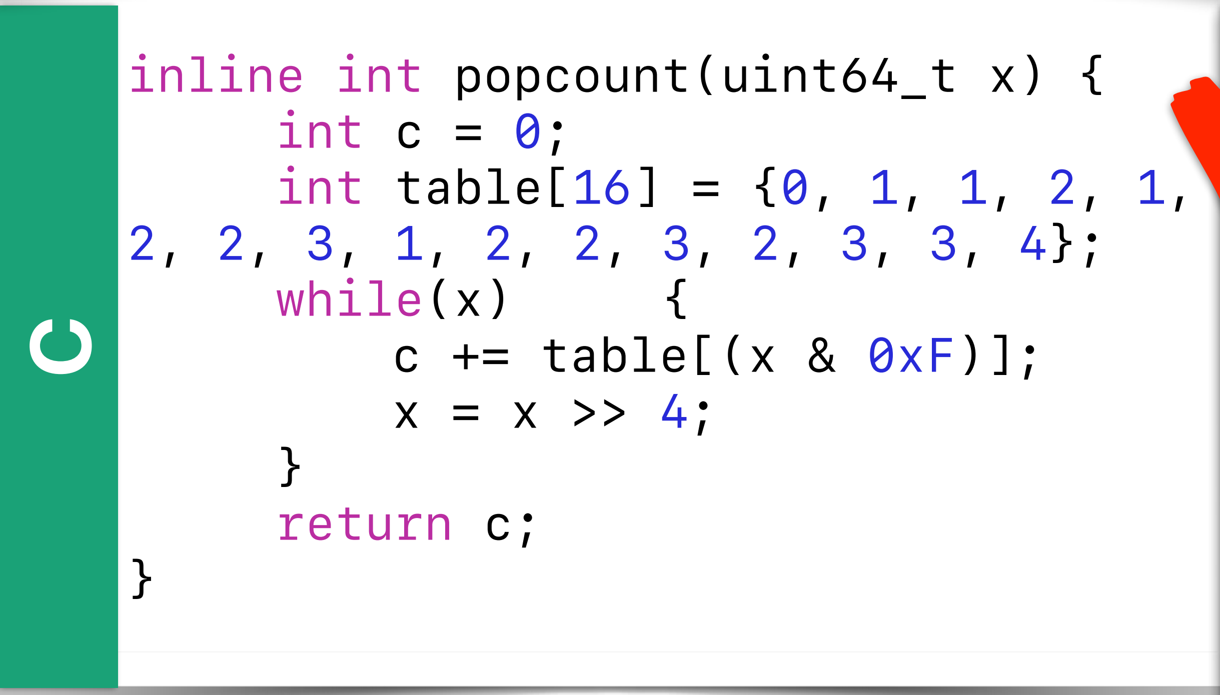
```
inline int popcount(uint64_t x){
  int c=0;
  while(x)  {
        c += x & 1;
        x = x >> 1;
    }
    return c;
}
```

B



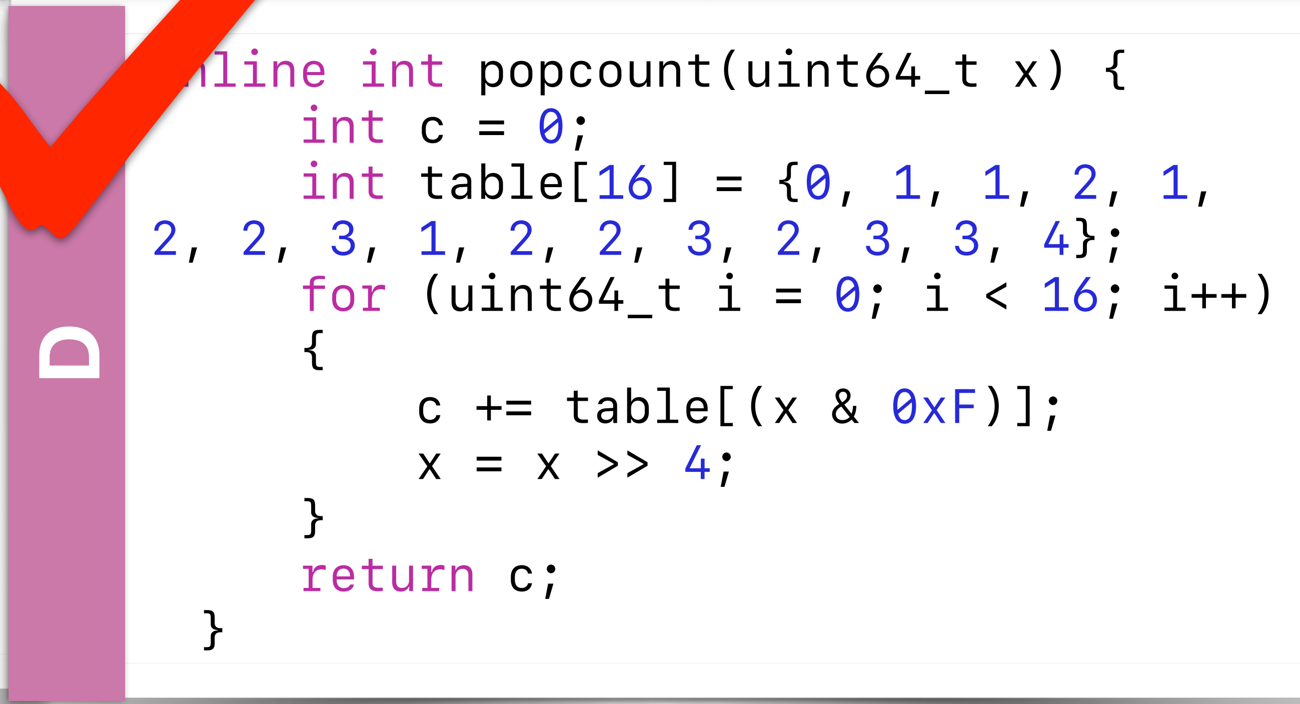
```
inline int popcount(uint64_t x) {
   int c = 0;
   while(x)     {
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
   }
   return c;
}
```

C



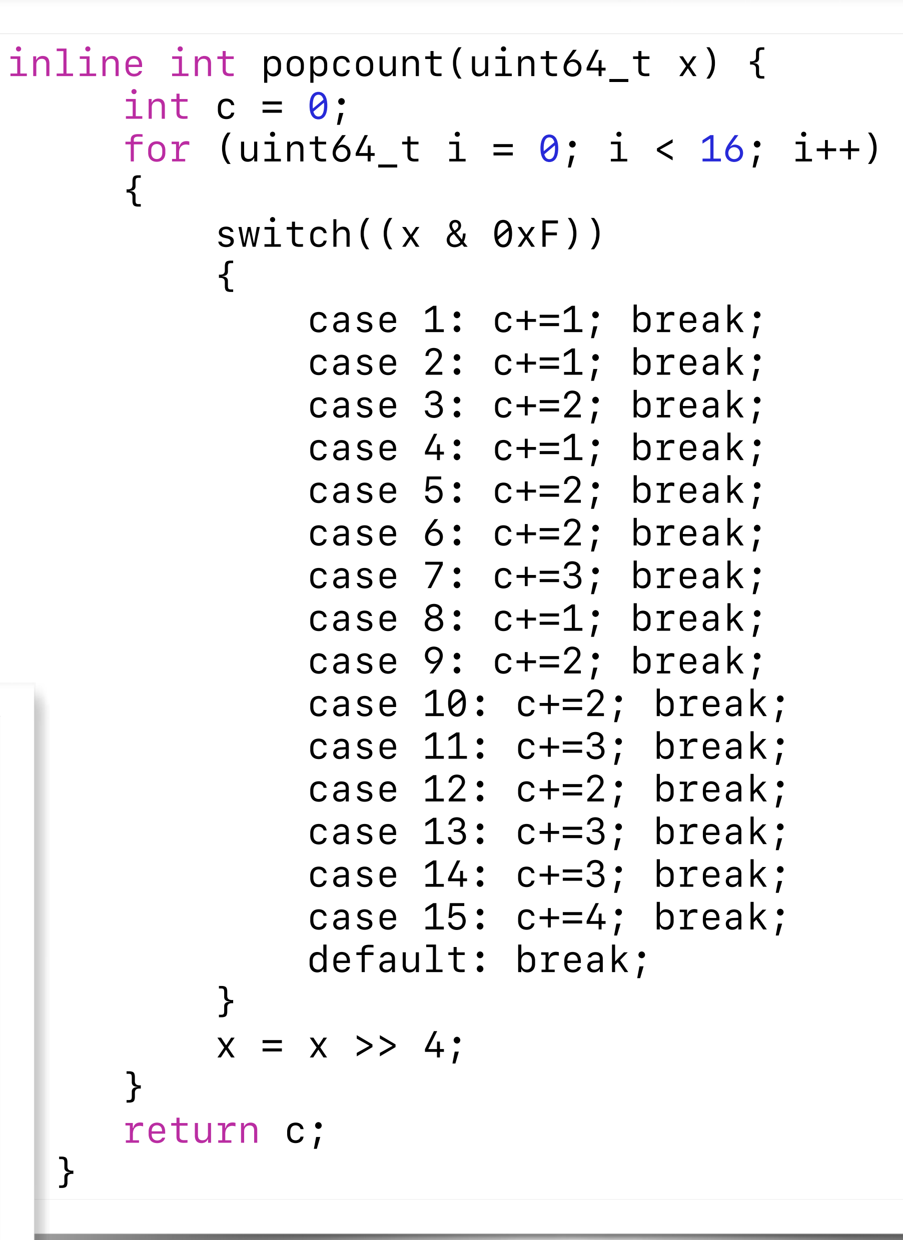
```
inline int popcount(uint64_t x) {
     int c = 0;
     int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
     while(x)     {
          c += table[(x & 0xF)];
          x = x >> 4;
     }
     return c;
}
```

D ✓



```
inline int popcount(uint64_t x) {
     int c = 0;
     int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
     for (uint64_t i = 0; i < 16; i++)
     {
          c += table[(x & 0xF)];
          x = x >> 4;
     }
     return c;
 }
```

E



```
inline int popcount(uint64_t x) {
     int c = 0;
     for (uint64_t i = 0; i < 16; i++)
     {
          switch((x & 0xF))
          {
               case 1: c+=1; break;
               case 2: c+=1; break;
               case 3: c+=2; break;
               case 4: c+=1; break;
               case 5: c+=2; break;
               case 6: c+=2; break;
               case 7: c+=3; break;
               case 8: c+=1; break;
               case 9: c+=2; break;
               case 10: c+=2; break;
               case 11: c+=3; break;
               case 12: c+=2; break;
               case 13: c+=3; break;
               case 14: c+=3; break;
               case 15: c+=4; break;
               default: break;
          }
          x = x >> 4;
     }
     return c;
}
```

Announcements

- Assignment #3 due **tonight**
- Reading Quiz due next **Tuesday**
- iEval
  - Submit by 6/9 and take a screenshot of your submission
  - Submit your screen screenshot in gradescope — it counts as a “full-credit” notebook assignment (technically help to drop two of your lowest notebook assignments)

Computer
Science &
Engineering

203

つづく