Speculative Execution & Multithreaded Processor Architectures

Hung-Wei Tseng

Recap: addressing hazards

- Structural hazards
  - Stall
  - Modify hardware design
- Control hazards
  - Stall
  - Static prediction
  - Dynamic prediction
- Data hazards
  - Stall
  - Data forwarding
  - Dynamic Scheduling

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

- This instruction has completed its own work in the current stage
- No other instruction is occupying the next stage
- The next stage has all its inputs ready

Tomasulo in motion

①	ld	X6,0(X10)	D	AQ	AR	I	MEM	WB									
②	add	X7,X6,X12		D	I	I	I	I	INT	WB							
③	sd	X7,0(X10)			D	AQ	AR	I	I	I	MEM	WB					
④	addi	X10,X10,8				D	I	INT	WB								
⑤	bne	X10,X5,LOOP					D	I	I	BR	WB						
⑥	ld	X6,0(X10)						D	AQ	AR	I	MEM	WB				
⑦	add	X7,X6,X12							D	D	I	I	I	INT	WB		
⑧	sd	X7,0(X10)									D	AQ	AR	I	I	MEM	
⑨	addi	X10,X10,8										D	I	I	INT	WB	
⑩	bne	X10,X5,LOOP											D	I	I	I	BF

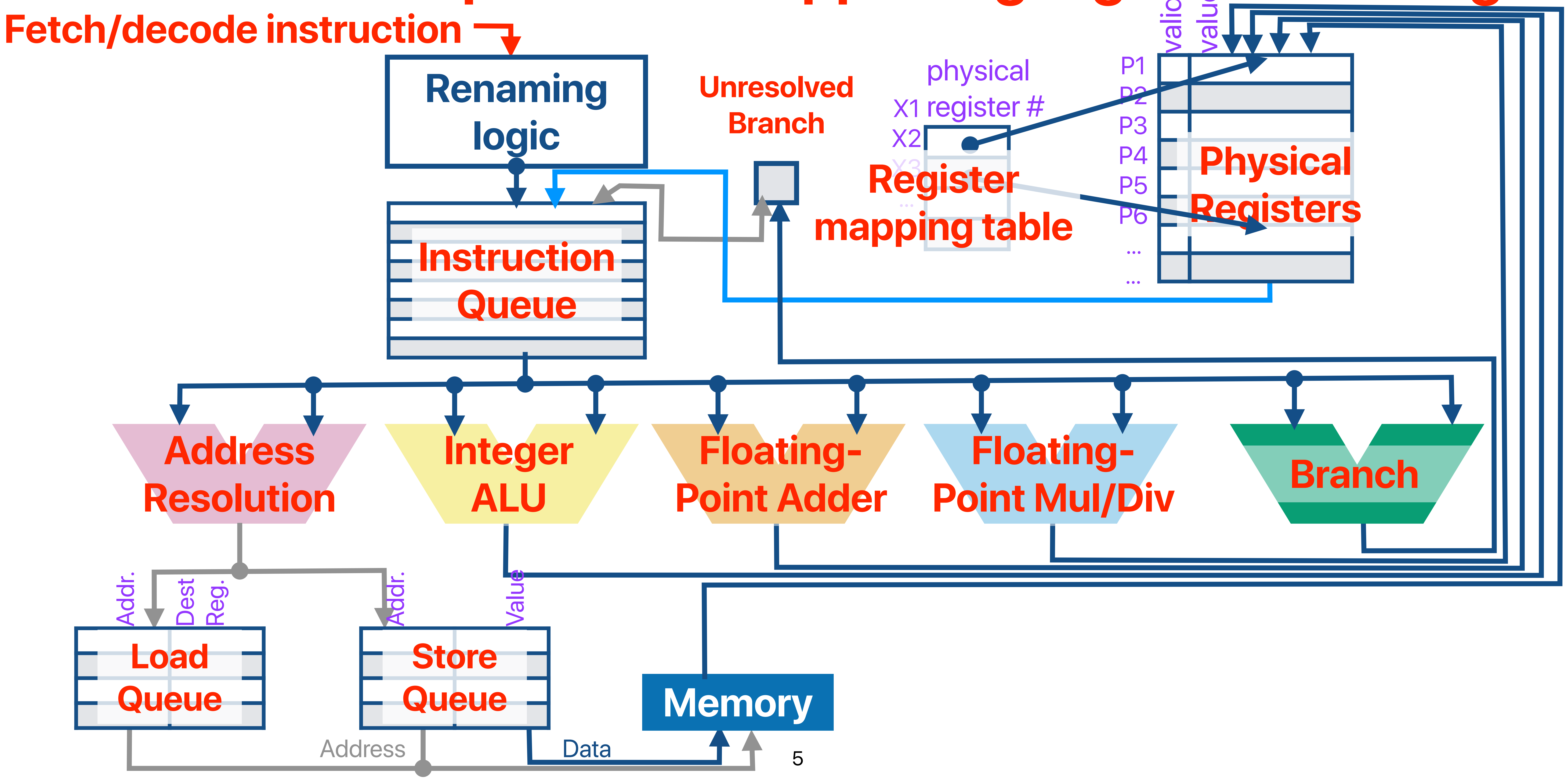
no reservation station for add!

Takes 13 cycles to issue all instructions

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	~~ld~~	~~0~~	~~INT2~~						~~6~~
LD3									
ST1	~~sd~~	~~0~~	~~[X10]~~	~~INT1~~					~~3~~
ST2	~~sd~~	~~0~~	~~[X10]~~	~~INT2~~					~~8~~
ST3									
INT1	add	8	INT2						9
INT2	~~add~~		~~[X12]~~		~~[LD2]~~				~~7~~
MUL1									
MUL2									
BR	br		[X5]	INT1					10

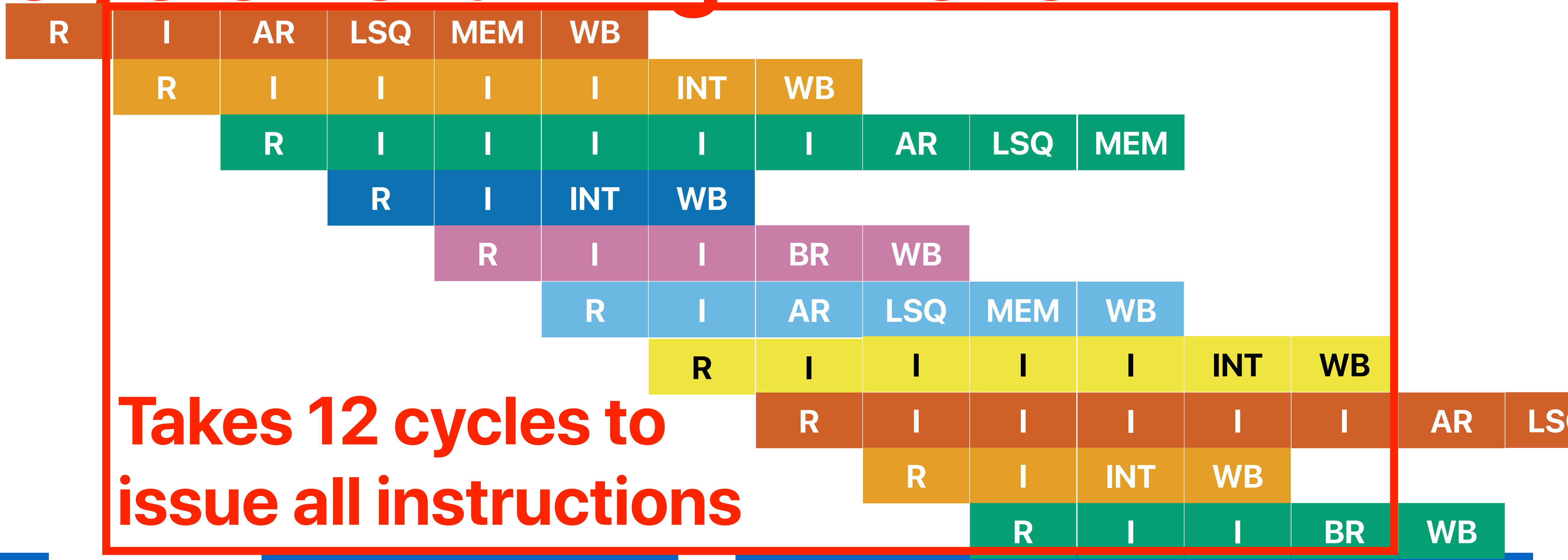
X5	
X6	LD2
X7	INT2
X10	INT1
X12	

Overview of a processor supporting register renaming



Register renaming in motion

```
① ld   X6,0(X10)
② add  X7,X6,X12
③ sd   X7,0(X10)
④ addi X10,X10,8
⑤ bne  X10,X5,LOOP
⑥ ld   X6,0(X10)
⑦ add  X7,X6,X12
⑧ sd   X7,0(X10)
⑨ addi X10,X10,8
⑩ bne  X10,X5,LOOP
```



Takes 12 cycles to issue all instructions

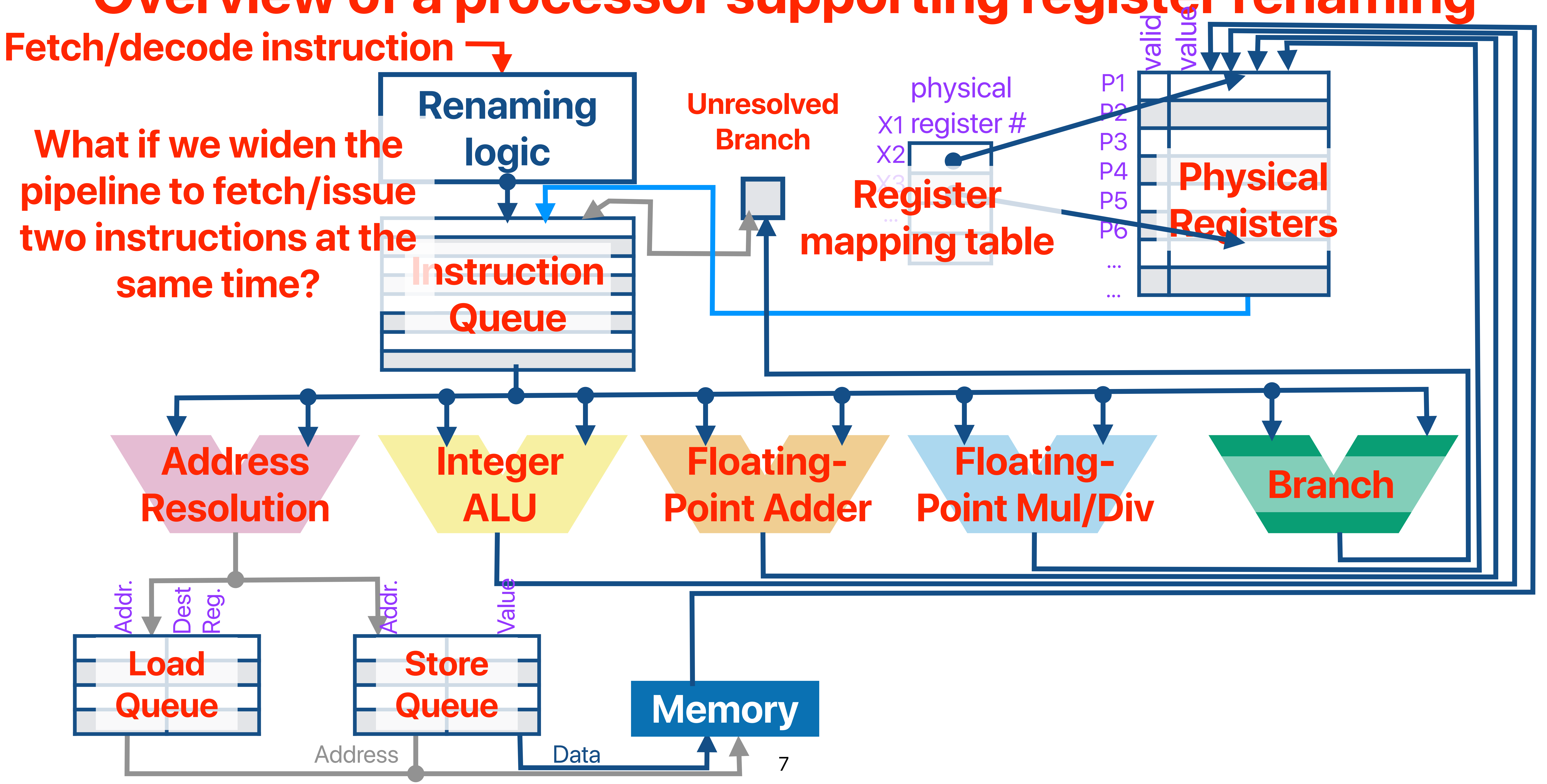
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	~~add P2, P1, X12~~
3	~~sd P2, 0(X10)~~
4	~~addi P3, X10, 8~~
5	~~bne P3, X5, LOOP~~
6	~~ld P4, 0(P3)~~
7	~~add P5, P1, X12~~
8	sd P5, 0(P3)
9	~~addi P6, P3, 8~~
10	~~bne P6, 0(X10)~~

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

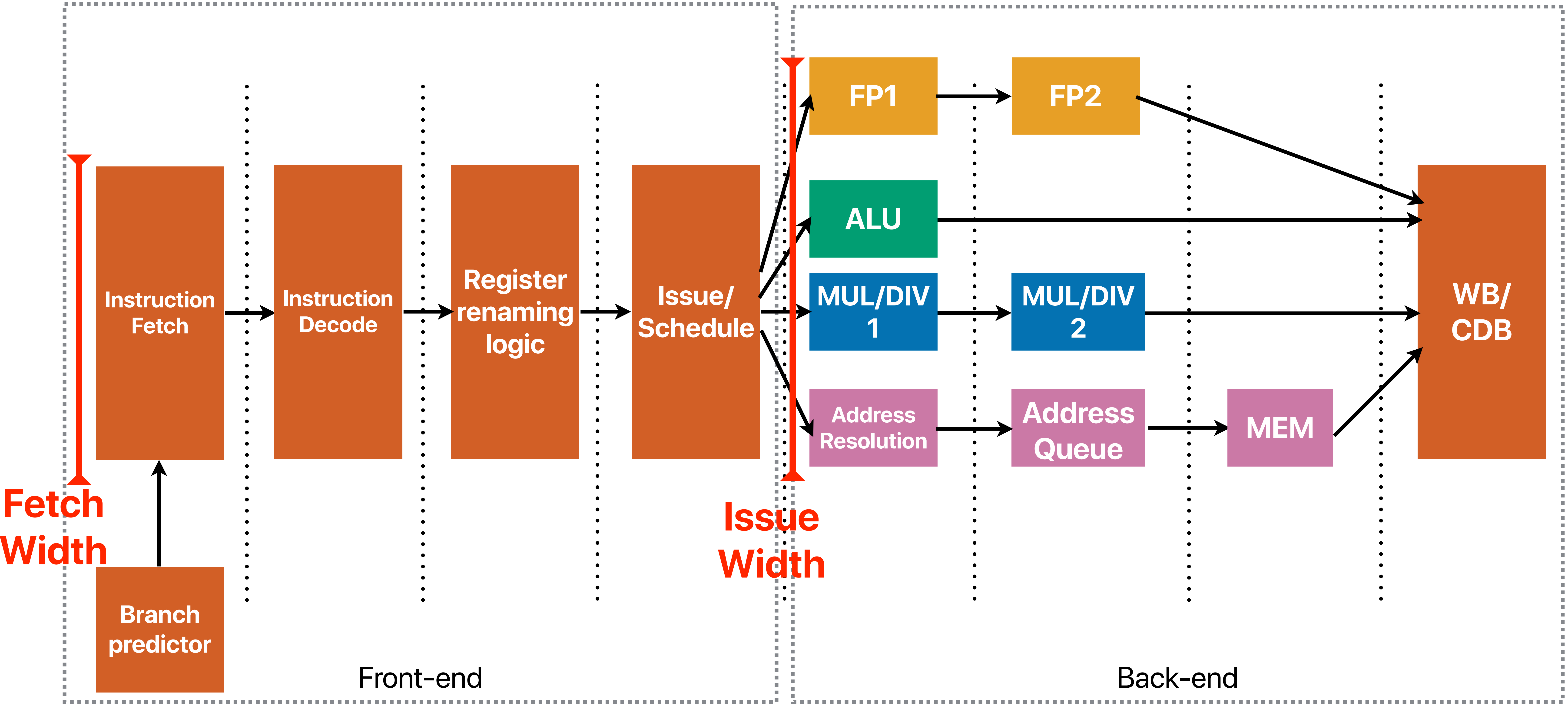
	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6	1		1
P2	1		1	P7			
P3	1		1	P8			
P4	1		1	P9			
P5	1		1	P10			

Overview of a processor supporting register renaming

What if we widen the pipeline to fetch/issue two instructions at the same time?



Recap: Super Scalar Pipeline



Superscalar

- Since we have more 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

What about "linked list"

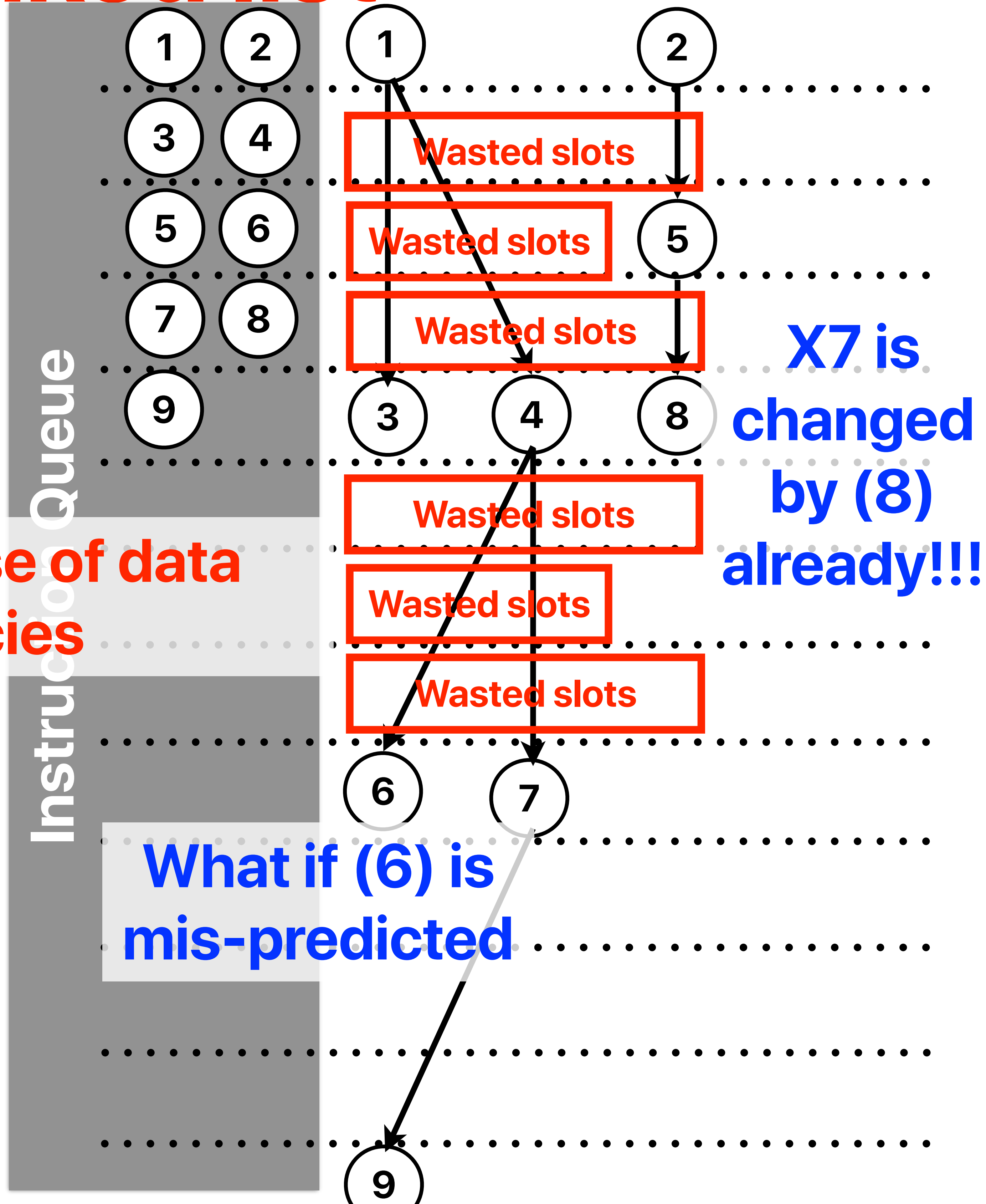
Static instructions

```
LOOP: ld    X10, 8(X10)
      addi   X7, X7, 1
      bne   X10, X0, LOOP
```

Dynamic instructions

```
① ld    X10, 8(X10)
② addi   X7, X7, 1
③ bne   X10, X0, LOOP
④ ld    X10, 8(X10)
⑤ addi   X7, X7, 1
⑥ bne   X10, X0, LOOP
⑦ ld    X10, 8(X10)
⑧ addi   X7, X7, 1
⑨ bne   X10, X0, LOOP
```

ILP is low because of data dependencies



Team scores

8	15.5	11	9

Outline

- The Concept of Speculative Execution and Reorder Buffer
- Simultaneous Multithreading
- Chip Multiprocessor

Poll close in 1:30

In which pipeline stage can we change PCs?

- How many of the following pipeline stages can an instruction change the program counter?
  - ① IF
  - ② ID
  - ③ EXE
  - ④ MEM
  - ⑤ WB

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

Poll close in 1:30

In which pipeline stage can we change P

1 — Instinct

- How many of the following pipeline stages can an instruction change the program counter?
  ① IF
  ② ID
  ③ EXE
  ④ MEM
  ⑤ WB
  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

In which pipeline stage can we change PCs?

- How many of the following pipeline stages can an instruction change the program counter?
  - ① IF — page fault, illegal address
  - ② ID — unknown instruction
  - ③ EXE — divide by zero, overflow, underflow, branch mis-prediction
  - ④ MEM — page fault, illegal address
  - ⑤ WB
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5

If you have no idea what's an "exception" and why it's changing the PC — you need to take CS202!

2-issue RR processor in motion

	Instruction														
①	`ld X6,0(X10)`	R	I	AR	AQ	MEM	WB							
②	`add X7,X6,X12`	R	I	I	I	I	I	INT	WB					
③	`sd X7,0(X10)`		R	I	AR	AQ	AQ	AQ	AQ	MEM	WB			
④	`addi X10,X10,8`		R	I	INT	WB								
⑤	`bne X10,X5,LOOP`			R	I	I	BR	WB						
⑥	`ld X6,0(X10)`			R	I	I	AR	AQ	MEM	WB				
⑦	`add X7,X6,X12`				R	I	I	I	I	I	INT	WB		
⑧	`sd X7,0(X10)`				R	I	I	AR	AQ	AQ	AQ	AQ	MEM	WB
⑨	`addi X10,X10,8`					R	I	I	INT	WB				
⑩	`bne X10,X5,LOOP`					R	I	I	I	I	BR	WB		

X10 is already changed!

What if exception occurs here?

	Renamed instruction	
1	~~ld~~	~~P1, 0(X10)~~
2	~~add~~	~~P2, P1, X12~~
3	~~sd~~	~~P2, 0(X10)~~
4	addi	P3, X10, 8
5	~~bne~~	~~P3, X5, LOOP~~
6	~~ld~~	~~P4, 0(P3)~~
7	~~add~~	~~P5, P1, X12~~
8	sd	P5, 0(P3)
9	~~addi~~	~~P6, P3, 8~~
10	bne	P6, 0(X10)

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

Speculative Execution

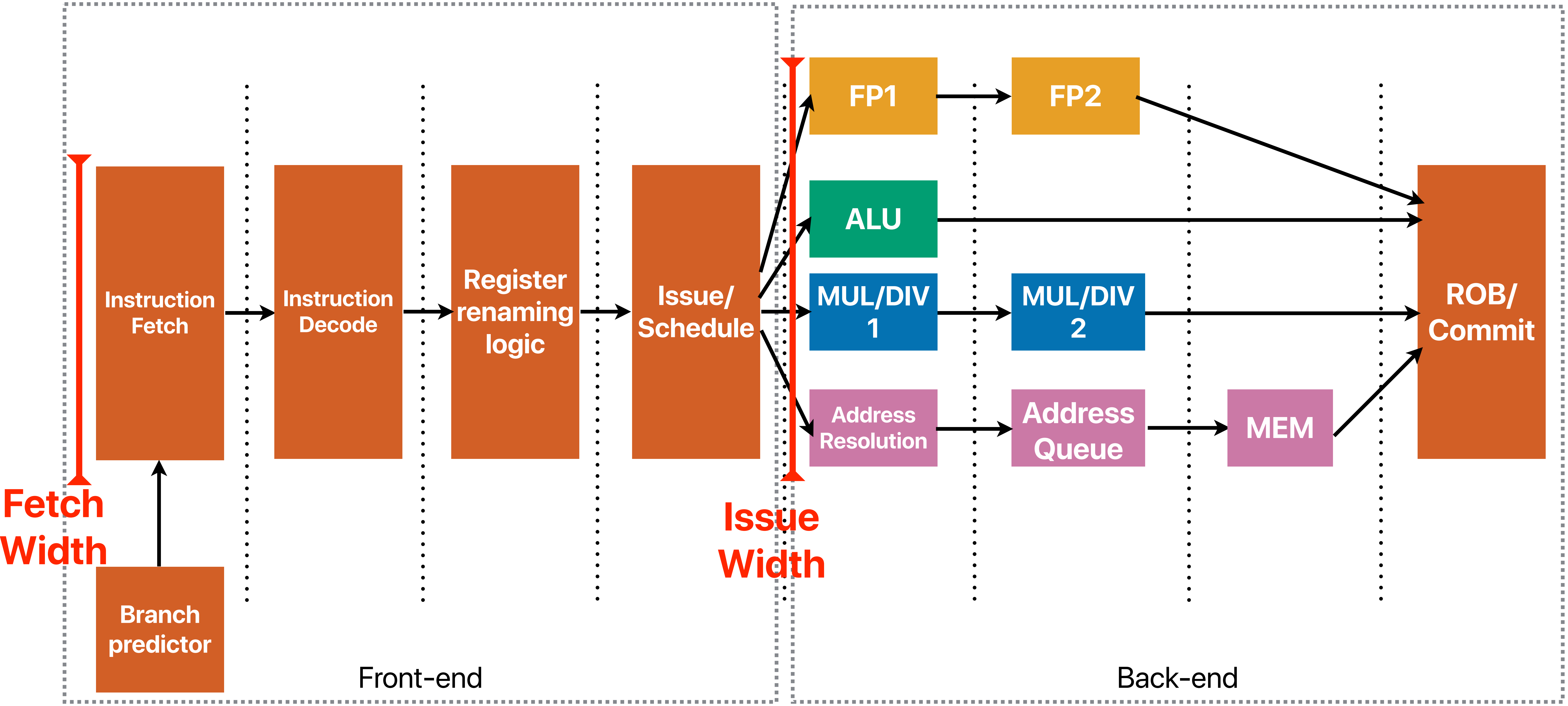
- The PC can potentially change any time during execution
  - Exceptions
  - Branches
- Any execution of an instruction before a prior instruction finishes is considered as **speculative execution**
- Because it's speculative, we need to preserve the capability to restore to the states before it's executed
  - Flush incorrectly fetched instructions
  - Restore updated register values
  - Fetch the right instructions (correct branch target, exception handler)

Reorder Buffer (ROB)

Reorder buffer/Commit stage

- Reorder buffer — a buffer keep track of the program order of instructions
  - Can be combined with IQ or physical registers — make either as a circular queue
- Commit stage — should the outcome of an instruction be realized
  - An instruction can only leave the pipeline if all it’s previous are committed
  - If any prior instruction failed to commit, the instruction should yield it’s ROB entry, restore all it’s architectural changes

Pipeline SuperScalar/OoO/ROB



2-issue RR processor in motion

```
①   ld   X6,0(X10)    R
②   add  X7,X6,X12    R
③   sd   X7,0(X10)
④   addi X10,X10,8
⑤   bne  X10,X5,LOOP
⑥   ld   X6,0(X10)
⑦   add  X7,X6,X12
⑧   sd   X7,0(X10)
⑨   addi X10,X10,8
⑩   bne  X10,X5,LOOP
```

	Renamed instruction	
1	ld P1, 0(X10)	← head
2	add P2, P1, X12	← tail
3		
4		
5		
6		
7		
8		
9		
10		

	Physical Register
X5	
X6	P1
X7	P2
X10	
X12	

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

2-issue RR processor in motion

#	Instruction			
①	ld	X6,0(X10)	R	I
②	add	X7,X6,X12	R	I
③	sd	X7,0(X10)		R
④	addi	X10,X10,8		R
⑤	bne	X10,X5,LOOP		
⑥	ld	X6,0(X10)		
⑦	add	X7,X6,X12		
⑧	sd	X7,0(X10)		
⑨	addi	X10,X10,8		
⑩	bne	X10,X5,LOOP		

	Renamed instruction		
1	ld	P1, 0(X10)	← head
2	add	P2, P1, X12	
3	sd	P2, 0(X10)	
4	addi	P3, X10, 8	← tail
5			
6			
7			
8			
9			
10			

	Physical Register
X5	
X6	P1
X7	P2
X10	P3
X12	

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

2-issue RR processor in motion

```
① ld   X6,0(X10)
② add  X7,X6,X12
③ sd   X7,0(X10)
④ addi X10,X10,8
⑤ bne  X10,X5,LOOP
⑥ ld   X6,0(X10)
⑦ add  X7,X6,X12
⑧ sd   X7,0(X10)
⑨ addi X10,X10,8
⑩ bne  X10,X5,LOOP
```

Instruction			
①	R	I	AR
②	R	I	I
③		R	I
④		R	I
⑤			R
⑥			R

	Renamed instruction	
1	ld P1, 0(X10)	← head
2	add P2, P1, X12	
3	sd P2, 0(X10)	
4	addi P3, X10, 8	
5	bne P3, X5, LOOP	
6	ld P4, 0(P3)	← tail
7		
8		
9		
10		

	Physical Register
X5	
X6	P1
X7	P2
X10	P3
X12	

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

2-issue RR processor in motion

	Instruction				
①	ld X6,0(X10)	R	I	AR	AQ
②	add X7,X6,X12	R	I	I	I
③	sd X7,0(X10)		R	I	I
④	addi X10,X10,8		R	I	INT
⑤	bne X10,X5,LOOP			R	I
⑥	ld X6,0(X10)			R	I
⑦	add X7,X6,X12				R
⑧	sd X7,0(X10)				R
⑨	addi X10,X10,8				
⑩	bne X10,X5,LOOP				

	Renamed instruction	
1	ld P1, 0(X10)	← head
2	add P2, P1, X12	
3	sd P2, 0(X10)	
4	addi P3, X10, 8	
5	bne P3, X5, LOOP	
6	ld P4, 0(P3)	
7	add P5, P1, X12	
8	sd P5, 0(P3)	← tail
9		
10		

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

	Instruction		Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
①	ld	X6,0(X10)	R	I	AR	AQ	MEM
②	add	X7,X6,X12	R	I	I	I	I
③	sd	X7,0(X10)		R	I	I	I
④	addi	X10,X10,8		R	I	INT	C
⑤	bne	X10,X5,LOOP			R	I	I
⑥	ld	X6,0(X10)			R	I	I
⑦	add	X7,X6,X12				R	I
⑧	sd	X7,0(X10)				R	I
⑨	addi	X10,X10,8					R
⑩	bne	X10,X5,LOOP					R

	Renamed instruction		
1	ld	P1, 0(X10)	← head
2	add	P2, P1, X12	
3	sd	P2, 0(X10)	
4	addi	P3, X10, 8	
5	bne	P3, X5, LOOP	
6	ld	P4, 0(P3)	
7	add	P5, P1, X12	
8	sd	P5, 0(P3)	
9	addi	P6, P3, 8	
10	bne	P6, 0(X10)	← tail

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

①	ld X6,0(X10)	R	I	AR	AQ	MEM	C
②	add X7,X6,X12	R	I	I	I	I	I
③	sd X7,0(X10)		R	I	I	I	I
④	addi X10,X10,8		R	I	INT	C	C
⑤	bne X10,X5,LOOP			R	I	I	BR
⑥	ld X6,0(X10)			R	I	I	AR
⑦	add X7,X6,X12				R	I	I
⑧	sd X7,0(X10)				R	I	I
⑨	addi X10,X10,8					R	I
⑩	bne X10,X5,LOOP					R	I

	Renamed instruction	
1	ld	P1, 0(X10)
2	add	P2, P1, X12
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
5	bne	P3, X5, LOOP
6	ld	P4, 0(P3)
7	add	P5, P1, X12
8	sd	P5, 0(P3)
9	addi	P6, P3, 8
10	bne	P6, 0(X10)

head (at entry 1)

tail (at entry 10)

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

①	ld X6,0(X10)	R	I	AR	AQ	MEM	C	
②	add X7,X6,X12	R	I	I	I	I	I	INT
③	sd X7,0(X10)		R	I	I	I	I	I
④	addi X10,X10,8		R	I	INT	C	C	C
⑤	bne X10,X5,LOOP			R	I	I	BR	C
⑥	ld X6,0(X10)			R	I	I	AR	AQ
⑦	add X7,X6,X12				R	I	I	I
⑧	sd X7,0(X10)				R	I	I	I
⑨	addi X10,X10,8					R	I	I
⑩	bne X10,X5,LOOP					R	I	I

	Renamed instruction		
1	~~ld~~	~~P1, 0(X10)~~	
2	add	P2, P1, X12	← head
3	sd	P2, 0(X10)	
4	addi	P3, X10, 8	
5	bne	P3, X5, LOOP	
6	ld	P4, 0(P3)	
7	add	P5, P1, X12	
8	sd	P5, 0(P3)	
9	addi	P6, P3, 8	
10	bne	P6, 0(X10)	← tail

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

	Instruction	1	2	3	4	5	6	7	8
①	ld X6,0(X10)	R	I	AR	AQ	MEM	C		
②	add X7,X6,X12	R	I	I	I	I	I	INT	C
③	sd X7,0(X10)		R	I	I	I	I	I	I
④	addi X10,X10,8		R	I	INT	C	C	C	C
⑤	bne X10,X5,LOOP			R	I	I	BR	C	C
⑥	ld X6,0(X10)			R	I	I	AR	AQ	MEM
⑦	add X7,X6,X12				R	I	I	I	I
⑧	sd X7,0(X10)				R	I	I	I	I
⑨	addi X10,X10,8					R	I	I	INT
⑩	bne X10,X5,LOOP					R	I	I	I

	Renamed instruction	
1	~~ld P1, 0(X10)~~	
2	add P2, P1, X12	← head
3	sd P2, 0(X10)	
4	addi P3, X10, 8	
5	bne P3, X5, LOOP	
6	ld P4, 0(P3)	
7	add P5, P1, X12	
8	sd P5, 0(P3)	
9	addi P6, P3, 8	
10	bne P6, 0(X10)	← tail

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

	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			

2-issue RR processor in motion

①	ld X6,0(X10)	R	I	AR	AQ	MEM	C			
②	add X7,X6,X12	R	I	I	I	I	I	INT	C	
③	sd X7,0(X10)		R	I	I	I	I	I	I	AR
④	addi X10,X10,8		R	I	INT	C	C	C	C	C
⑤	bne X10,X5,LOOP			R	I	I	BR	C	C	C
⑥	ld X6,0(X10)			R	I	I	AR	AQ	MEM	C
⑦	add X7,X6,X12				R	I	I	I	I	I
⑧	sd X7,0(X10)				R	I	I	I	I	I
⑨	addi X10,X10,8					R	I	I	INT	C
⑩	bne X10,X5,LOOP					R	I	I	I	I

	Renamed instruction	
1	~~ld~~	~~P1, 0(X10)~~
2	~~add~~	~~P2, P1, X12~~
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
5	bne	P3, X5, LOOP
6	ld	P4, 0(P3)
7	add	P5, P1, X12
8	sd	P5, 0(P3)
9	addi	P6, P3, 8
10	bne	P6, 0(X10)

head (at 3)

tail (at 10)

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

#	Instruction										
①	ld X6,0(X10)	R	I	AR	AQ	MEM	C				
②	add X7,X6,X12	R	I	I	I	I	I	INT	C		
③	sd X7,0(X10)		R	I	I	I	I	I	I	AR	AQ
④	addi X10,X10,8		R	I	INT	C	C	C	C	C	C
⑤	bne X10,X5,LOOP			R	I	I	BR	C	C	C	C
⑥	ld X6,0(X10)			R	I	I	AR	AQ	MEM	C	C
⑦	add X7,X6,X12				R	I	I	I	I	I	INT
⑧	sd X7,0(X10)				R	I	I	I	I	I	I
⑨	addi X10,X10,8					R	I	I	INT	C	C
⑩	bne X10,X5,LOOP					R	I	I	I	I	BR

	Renamed instruction	
1	~~ld~~	~~P1, 0(X10)~~
2	~~add~~	~~P2, P1, X12~~
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
5	bne	P3, X5, LOOP
6	ld	P4, 0(P3)
7	add	P5, P1, X12
8	sd	P5, 0(P3)
9	addi	P6, P3, 8
10	bne	P6, 0(X10)

head → 3

tail → 10

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

	Instruction											
①	ld X6,0(X10)	R	I	AR	AQ	MEM	C					
②	add X7,X6,X12	R	I	I	I	I	I	INT	C			
③	sd X7,0(X10)		R	I	I	I	I	I	I	AR	AQ	MEM
④	addi X10,X10,8		R	I	INT	C	C	C	C	C	C	C
⑤	bne X10,X5,LOOP			R	I	I	BR	C	C	C	C	C
⑥	ld X6,0(X10)			R	I	I	AR	AQ	MEM	C	C	C
⑦	add X7,X6,X12				R	I	I	I	I	I	INT	C
⑧	sd X7,0(X10)				R	I	I	I	I	I	I	I
⑨	addi X10,X10,8					R	I	I	INT	C	C	C
⑩	bne X10,X5,LOOP					R	I	I	I	I	BR	C

	Renamed instruction	
1	~~ld~~	~~P1, 0(X10)~~
2	~~add~~	~~P2, P1, X12~~
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
5	bne	P3, X5, LOOP
6	ld	P4, 0(P3)
7	add	P5, P1, X12
8	sd	P5, 0(P3)
9	addi	P6, P3, 8
10	bne	P6, 0(X10)

head (→ 3)

tail (→ 10)

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

#	Instruction	1	2	3	4	5	6	7	8	9	10	11	12
①	`ld X6,0(X10)`	R	I	AR	AQ	MEM	C						
②	`add X7,X6,X12`	R	I	I	I	I	I	INT	C				
③	`sd X7,0(X10)`		R	I	I	I	I	I	I	AR	AQ	MEM	C
④	`addi X10,X10,8`		R	I	INT	C	C	C	C	C	C	C	C
⑤	`bne X10,X5,LOOP`			R	I	I	BR	C	C	C	C	C	C
⑥	`ld X6,0(X10)`			R	I	I	AR	AQ	MEM	C	C	C	C
⑦	`add X7,X6,X12`				R	I	I	I	I	I	INT	C	C
⑧	`sd X7,0(X10)`				R	I	I	I	I	I	I	I	AR
⑨	`addi X10,X10,8`					R	I	I	INT	C	C	C	C
⑩	`bne X10,X5,LOOP`					R	I	I	I	I	BR	C	C

	Renamed instruction
1	~~ld P1, 0(X10)~~
2	~~add P2, P1, X12~~
3	~~sd P2, 0(X10)~~
4	~~addi P3, X10, 8~~
5	~~bne P3, X5, LOOP~~
6	~~ld P4, 0(P3)~~
7	~~add P5, P1, X12~~
8	sd P5, 0(P3) ← head
9	addi P6, P3, 8
10	bne P6, 0(X10) ← tail

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

①	ld	X6,0(X10)	R	I	AR	AQ	MEM	C						
②	add	X7,X6,X12	R	I	I	I	I	I	INT	C				
③	sd	X7,0(X10)		R	I	I	I	I	I	I	AR	AQ	MEM	C	
④	addi	X10,X10,8		R	I	INT	C	C	C	C	C	C	C	C	
⑤	bne	X10,X5,LOOP			R	I	I	BR	C	C	C	C	C	C	
⑥	ld	X6,0(X10)			R	I	I	AR	AQ	MEM	C	C	C	C	
⑦	add	X7,X6,X12				R	I	I	I	I	I	INT	C	C	
⑧	sd	X7,0(X10)				R	I	I	I	I	I	I	I	AR	AQ
⑨	addi	X10,X10,8					R	I	I	INT	C	C	C	C	C
⑩	bne	X10,X5,LOOP					R	I	I	I	I	BR	C	C	C

	Renamed instruction	
1	~~ld~~	~~P1, 0(X10)~~
2	~~add~~	~~P2, P1, X12~~
3	~~sd~~	~~P2, 0(X10)~~
4	~~addi~~	~~P3, X10, 8~~
5	~~bne~~	~~P3, X5, LOOP~~
6	~~ld~~	~~P4, 0(P3)~~
7	~~add~~	~~P5, P1, X12~~
8	sd	P5, 0(P3)
9	addi	P6, P3, 8
10	bne	P6, 0(X10)

head (at 8)

tail (at 10)

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

```
① ld   X6,0(X10)
② add  X7,X6,X12
③ sd   X7,0(X10)
④ addi X10,X10,8
⑤ bne  X10,X5,LOOP
⑥ ld   X6,0(X10)
⑦ add  X7,X6,X12
⑧ sd   X7,0(X10)
⑨ addi X10,X10,8
⑩ bne  X10,X5,LOOP
```

Instruction														
①	R	I	AR	AQ	MEM	C								
②	R	I	I	I	I	I	INT	C						
③		R	I	I	I	I	I	I	AR	AQ	MEM	C		
④		R	I	INT	C	C	C	C	C	C	C	C		
⑤			R	I	I	BR	C	C	C	C	C	C		
⑥			R	I	I	AR	AQ	MEM	C	C	C	C		
⑦				R	I	I	I	I	I	INT	C	C		
⑧				R	I	I	I	I	I	I	I	AR	AQ	MEM
⑨					R	I	I	INT	C	C	C	C	C	C
⑩					R	I	I	I	I	BR	C	C	C	C

	Renamed instruction			
1	~~ld~~	~~P1, 0(X10)~~		
2	~~add~~	~~P2, P1, X12~~		
3	~~sd~~	~~P2, 0(X10)~~		
4	~~addi~~	~~P3, X10, 8~~		
5	~~bne~~	~~P3, X5, LOOP~~		
6	~~ld~~	~~P4, 0(P3)~~		
7	~~add~~	~~P5, P1, X12~~		
8	sd	P5, 0(P3)	←	head
9	addi	P6, P3, 8		
10	bne	P6, 0(X10)	←	tail

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

2-issue RR processor in motion

#	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
①	ld X6,0(X10)	R	I	AR	AQ	MEM	C									
②	add X7,X6,X12	R	I	I	I	I	I	INT	C							
③	sd X7,0(X10)		R	I	I	I	I	I	I	AR	AQ	MEM	C			
④	addi X10,X10,8		R	I	INT	C	C	C	C	C	C	C	C			
⑤	bne X10,X5,LOOP			R	I	I	BR	C	C	C	C	C	C			
⑥	ld X6,0(X10)			R	I	I	AR	AQ	MEM	WB	C	C	C			
⑦	add X7,X6,X12				R	I	I	I	I	I	INT	C	C			
⑧	sd X7,0(X10)				R	I	I	I	I	I	I	I	AR	AQ	MEM	C
⑨	addi X10,X10,8					R	I	I	INT	C	C	C	C	C	C	C
⑩	bne X10,X5,LOOP					R	I	I	I	I	BR	C	C	C	C	C

	Renamed instruction	
1	~~ld~~	~~P1, 0(X10)~~
2	~~add~~	~~P2, P1, X12~~
3	~~sd~~	~~P2, 0(X10)~~
4	~~addi~~	~~P3, X10, 8~~
5	~~bne~~	~~P3, X5, LOOP~~
6	~~ld~~	~~P4, 0(P3)~~
7	~~add~~	~~P5, P1, X12~~
8	~~sd~~	~~P5, 0(P3)~~
9	~~addi~~	~~P6, P3, 8~~
10	~~bne~~	~~P6, 0(X10)~~

tail head

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

Poll close in 1:30

How good is SS/OoO/ROB with this code?

- Consider the following dynamic instructions

```
① ld   X1, 0(X10)
② addi X10, X10, 8
③ add  X20, X20, X1
④ bne  X10, X2, LOOP
⑤ ld   X1, 0(X10)
⑥ addi X10, X10, 8
⑦ add  X20, X20, X1
⑧ bne  X10, X2, LOOP
```

Assume a superscalar processor with **issue width as 2** & unlimited physical registers that can fetch up to 2 instructions per cycle, 3 cycles to execute a memory instruction how many cycles it takes to issue all instructions?

A. 1
B. 3
C. 5
D. 7
E. 9

Poll close in 1:30

How good is SS/OoO/ROB with this co

1 — Instinct

- Consider the following dynamic instructions

```
① ld   X1, 0(X10)
② addi X10, X10, 8
③ add  X20, X20, X1
④ bne  X10, X2, LOOP
⑤ ld   X1, 0(X10)
⑥ addi X10, X10, 8
⑦ add  X20, X20, X1
⑧ bne  X10, X2, LOOP
```

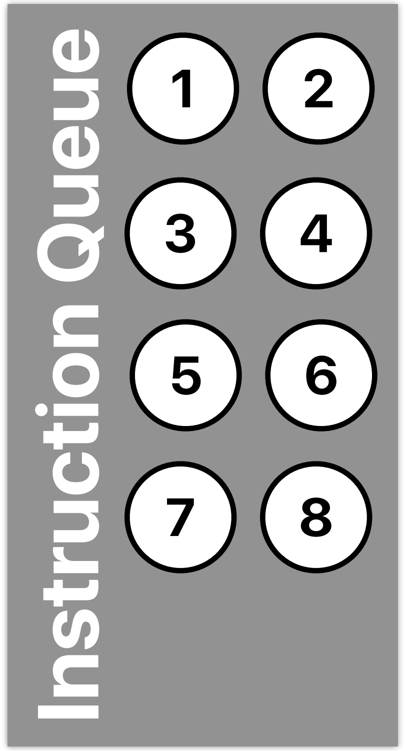
Assume a superscalar processor with **issue width as 2** & unlimited physical registers that can fetch up to 2 instructions per cycle, 3 cycles to execute a memory instruction how many cycles it takes to issue all instructions?

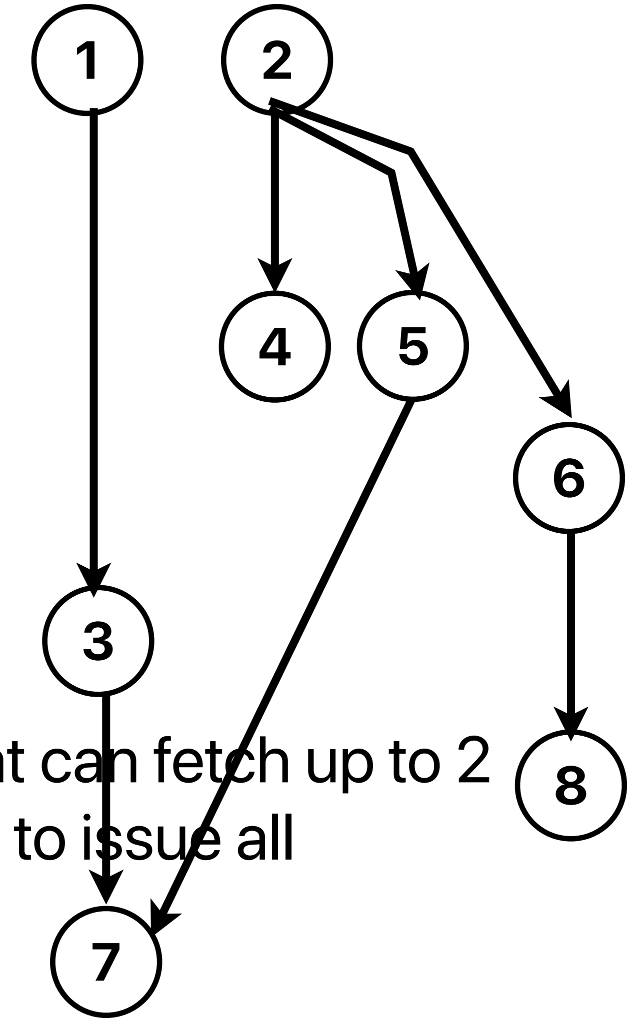
A. 1
B. 3
C. 5
D. 7
E. 9

How good is SS/OoO/ROB with this code?

- Consider the following dynamic instructions

```
①  ld   X1, 0(X10)
②  addi X10, X10, 8
③  add  X20, X20, X1
④  bne  X10, X2, LOOP
⑤  ld   X1, 0(X10)
⑥  addi X10, X10, 8
⑦  add  X20, X20, X1
⑧  bne  X10, X2, LOOP
```





Assume a superscalar processor with **issue width as 2** & unlimited physical registers that can fetch up to 2 instructions per cycle, 3 cycles to execute a memory instruction how many cycles it takes to issue all instructions?

A. 1
B. 3
C. 5
D. 7
E. 9

A feature of speculative execution

Poll close in 1:30

Putting it all together

- How many of the following would happen given the modern processor microarchitecture?
  ① The branch predictor will predict not taken for branch A
  ② The cache may contain the content of `array2[array1[16] * 512];`
  ③ temp can potentially become the value of `array2[array1[16] * 512];`
  ④ The program will raise an exception

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

```
unsigned int array1_size = 16;

uint8_t array1[160] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 260};
uint8_t array2[256 * 512];

void bar(size_t x) {
  if (x < array1_size) { // Branch A: Taken if the statement is not going to be executed.
    temp &= array2[array1[x] * 512];
  }
}

void foo(size_t x) {
    int i = 0, j=0;
    for(j=0;j<10000;j++)
            bar(rand()%17);
}
```

Poll close in 1:30

Putting it all together

1 — Instinct

- How many of the following would happen given the modern processor microarchitecture?
  1. The branch predictor will predict not taken for branch A
  2. The cache may contain the content of `array2[array1[16] * 512];`
  3. temp can potentially become the value of `array2[array1[16] * 512];`
  4. The program will raise an exception

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

```
unsigned int array1_size = 16;

uint8_t array1[160] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 260};
uint8_t array2[256 * 512];

void bar(size_t x) {
  if (x < array1_size) { // Branch A: Taken if the statement is not going to be executed.
    temp &= array2[array1[x] * 512];
  }
}

void foo(size_t x) {
    int i = 0, j=0;
    for(j=0;j<10000;j++)
            bar(rand()%17);
}
```

Putting it all together

- How many of the following would happen given the modern processor microarchitecture?
  - ✓ ① The branch predictor will predict not taken for branch A — very likely
  - ✓ ② The cache may contain the content of `array2[array1[16] * 512];` — possibly — where the security issues come from
  - ③ temp can potentially become the value of `array2[array1[16] * 512];` — not really, as x < array1_size
  - ✓ ④ The program will raise an exception — maybe?

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

```
unsigned int array1_size = 16;

uint8_t array1[160] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 260};
uint8_t array2[256 * 512];

void bar(size_t x) {
  if (x < array1_size) { // Branch A: Taken if the statement is not going to be executed.
    temp &= array2[array1[x] * 512];
  }
}

void foo(size_t x) {
    int i = 0, j=0;
    for(j=0;j<10000;j++)
            bar(rand()%17);
}
```

Spectre and meltdown

What happen when mis-speculation detected

- Exceptions and incorrect branch prediction can cause "rollback" of transient instructions
- Old register states are preserved, can be restored
- Memory writes are buffered, can be discarded
- Cache modifications are not restored!

Speculative execution on the following code

```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

- Execution without speculation is safe
  - CPU will never read array1[x] for any x ≥ array1_size
- Execution with speculation can be exploited
  - Attacker sets up some conditions
  - train branch predictor to assume 'if' is likely true
  - make array1_size and array2[] uncached
  - Invokes code with out-of-bounds x such that array1[x] is a secret
  - Processor recognizes its error when array1_size arrives, restores its architectural state, and proceeds with 'if' false
  - Attacker detects cache change (e.g. basic FLUSH+RELOAD or EVICT+RELOAD)
  - E.g. next read to array2[i*256] will be fast i=array[x] since this got cached

Poll close in 1:30

How good is SS/OoO/ROB with this code?

- Consider the following dynamic instructions

```
① ld   X1, 0(X10)
② addi X10, X10, 8
③ add  X20, X20, X1
④ bne  X10, X2, LOOP
```

Assume a superscalar processor with issue width as 2 & unlimited physical registers that can fetch up to 4 instructions per cycle, 3 cycles to execute a memory instruction and the loop will execute for 10,000 times, what's the average CPI?

A. 0.5
B. 0.75
C. 1
D. 1.25
E. 1.5

Poll close in 1:30

How good is SS/OoO/ROB with this co

1 — Instinct

- Consider the following dynamic instructions

```
①    ld   X1, 0(X10)
②    addi X10, X10, 8
③    add  X20, X20, X1
④    bne  X10, X2, LOOP
```

Assume a superscalar processor with issue width as 2 & unlimited physical registers that can fetch up to 4 instructions per cycle, 3 cycles to execute a memory instruction and the loop will execute for 10,000 times, what's the average CPI?

A. 0.5
B. 0.75
C. 1
D. 1.25
E. 1.5

How good is SS/OoO/ROB wi s code?

- Consider the following dynamic instructions

```
①  ld   X1, 0(X10)
②  addi X10, X10, 8
③  add  X20, X20, X1
④  bne  X10, X2, LOOP
```

Assume a superscalar processor with issue width as 2 & ur sical registers that can fetch up to 4 instructions per cycle, 3 cycles to exe ory instruction and the loop will execute for 10,000 times, what's the avera

A. 0.5

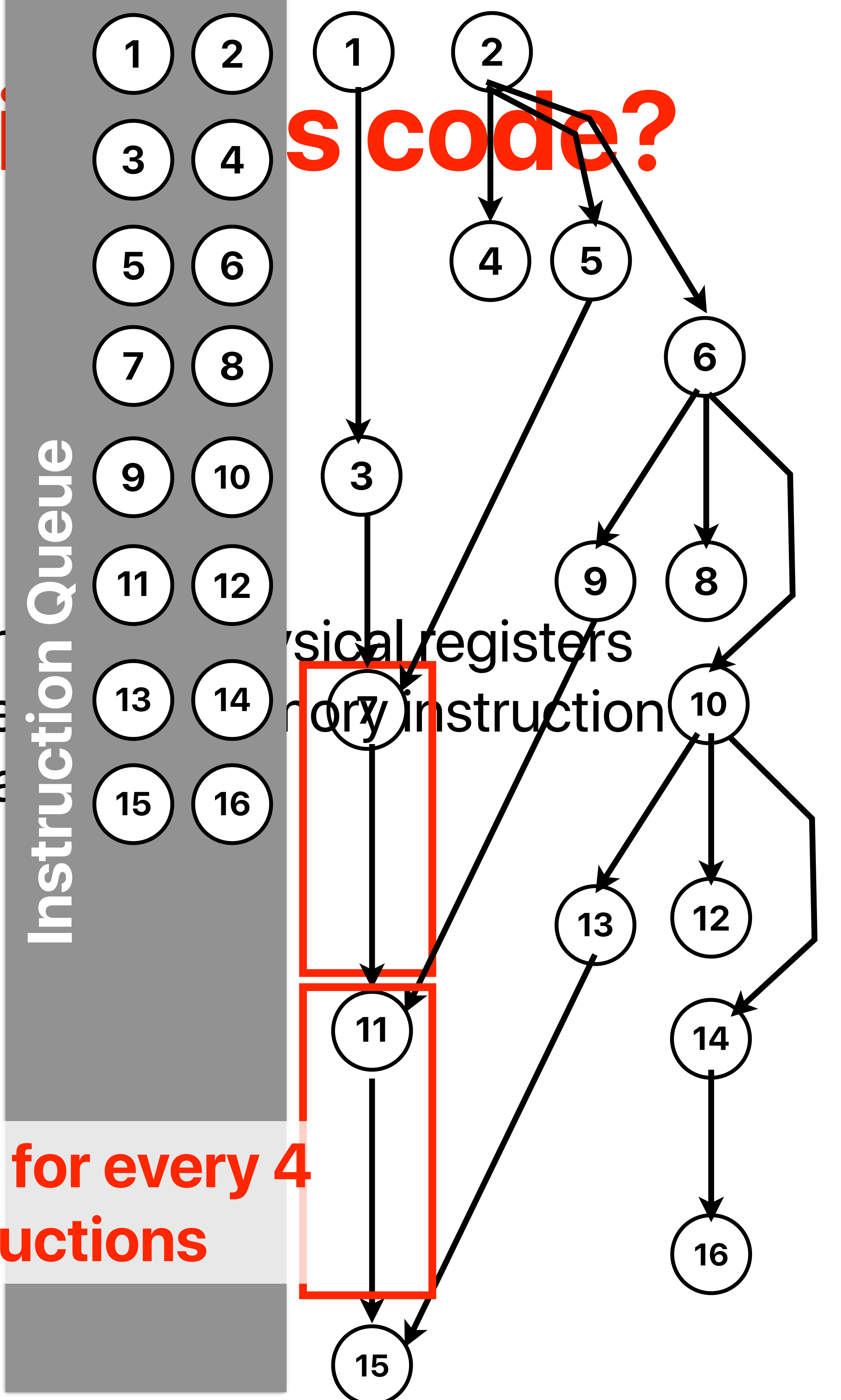
B. 0.75

C. 1

D. 1.25

E. 1.5

```
①  ld   X1, 0(X10)
②  addi X10, X10, 8
③  add  X20, X20, X1
④  bne  X10, X2, LOOP
⑤  ld   X1, 0(X10)
⑥  addi X10, X10, 8
⑦  add  X20, X20, X1
⑧  bne  X10, X2, LOOP
⑨  ld   X1, 0(X10)
⑩  addi X10, X10, 8
⑪  add  X20, X20, X1
⑫  bne  X10, X2, LOOP
```



3 cycles for every 4 instructions

What about "linked list"

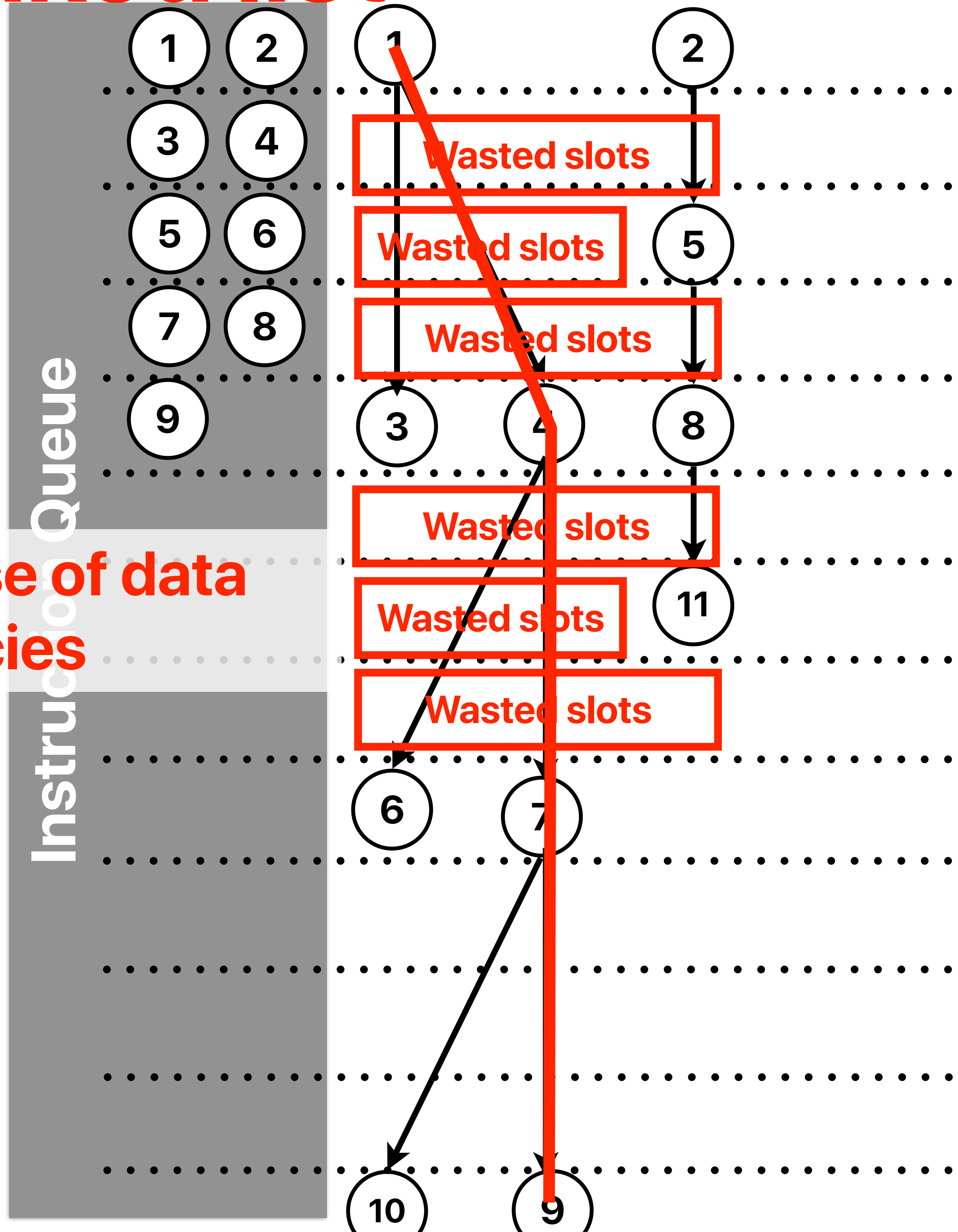
Static instructions

```
LOOP: ld    X10, 8(X10)
      addi   X7, X7, 1
      bne   X10, X0, LOOP
```

Dynamic instructions

```
① ld    X10, 8(X10)
② addi   X7, X7, 1
③ bne   X10, X0, LOOP
④ ld    X10, 8(X10)
⑤ addi   X7, X7, 1
⑥ bne   X10, X0, LOOP
⑦ ld    X10, 8(X10)
⑧ addi   X7, X7, 1
⑨ bne   X10, X0, LOOP
```

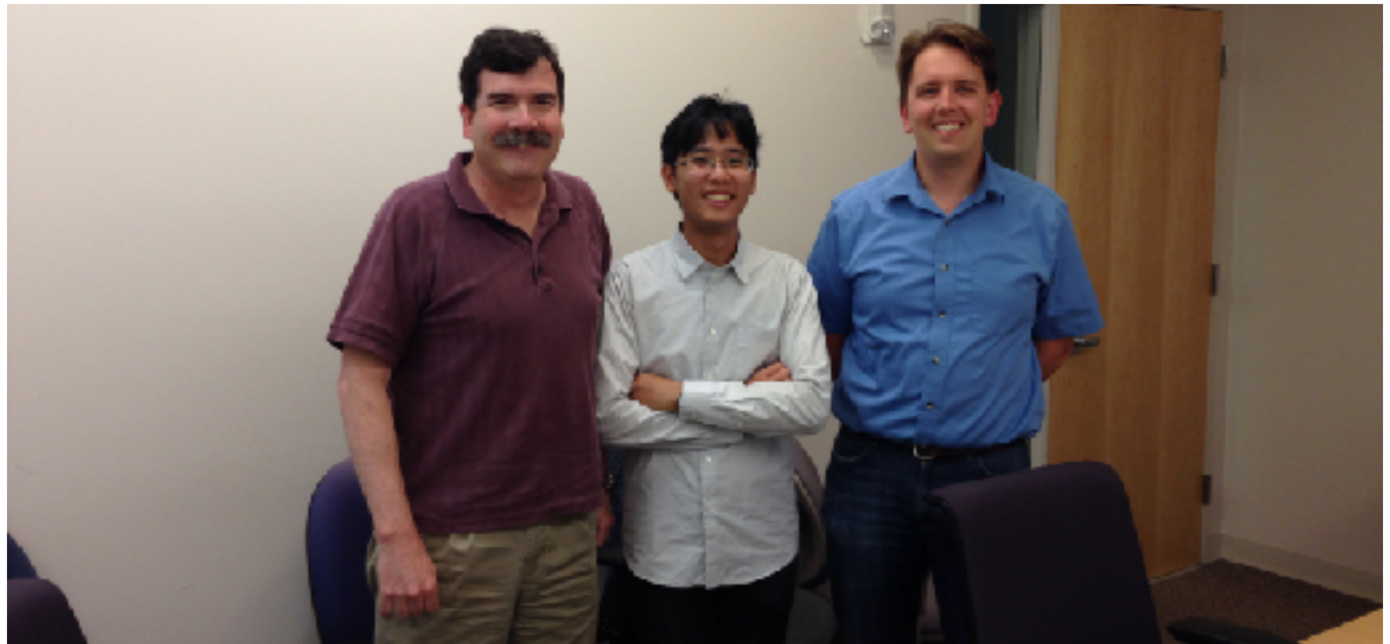
ILP is low because of data dependencies



Demo: ILP within a program

- perf is a tool that captures performance counters of your processors and can generate results like branch mis-prediction rate, cache miss rates and ILP.

Simultaneous multithreading

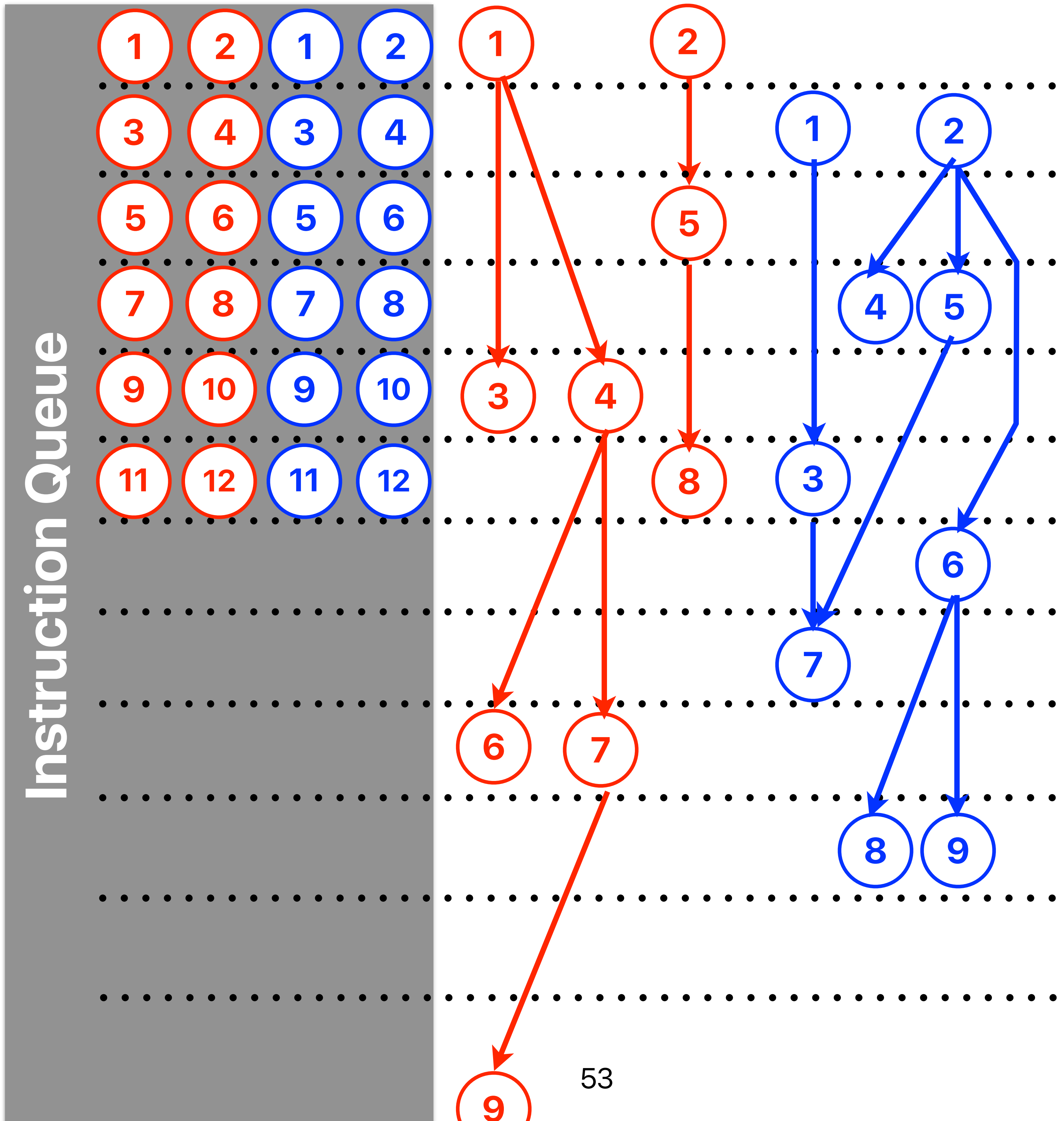


Simultaneous multithreading

- The processor can schedule instructions from different threads/processes/programs
- Fetch instructions from different threads/processes to fill the not utilized part of pipeline
  - Exploit "thread level parallelism" (TLP) to solve the problem of insufficient ILP in a single thread
  - You need to create an illusion of multiple processors for OSs

Simultaneous multithreading

```
① ld    X10, 8(X10)
② addi  X7, X7, 1
③ bne   X10, X0, LOOP
④ ld    X10, 8(X10)
⑤ addi  X7, X7, 1
⑥ bne   X10, X0, LOOP
⑦ ld    X10, 8(X10)
⑧ addi  X7, X7, 1
⑨ bne   X10, X0, LOOP
```




```
①  ld    X1, 0(X10)
②  addi  X10, X10, 8
③  add   X20, X20, X1
④  bne   X10, X2, LOOP
⑤  ld    X1, 0(X10)
⑥  addi  X10, X10, 8
⑦  add   X20, X20, X1
⑧  bne   X10, X2, LOOP
⑨  ld    X1, 0(X10)
⑩  addi  X10, X10, 8
⑪  add   X20, X20, X1
⑫  bne   X10, X2, LOOP
```

Poll close in 1:30

Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  ① Program counter
  ② Register mapping tables
  ③ Physical registers
  ④ ALUs
  ⑤ Data cache
  ⑥ Reorder buffer/Instruction Queue
  A. 2
  B. 3
  C. 4
  D. 5
  E. 6

Poll close in 1:30

Architectural support for simultaneous multithrea

1 — Instinct

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter
  - ② Register mapping tables
  - ③ Physical registers
  - ④ ALUs
  - ⑤ Data cache
  - ⑥ Reorder buffer/Instruction Queue

  A. 2

  B. 3

  C. 4

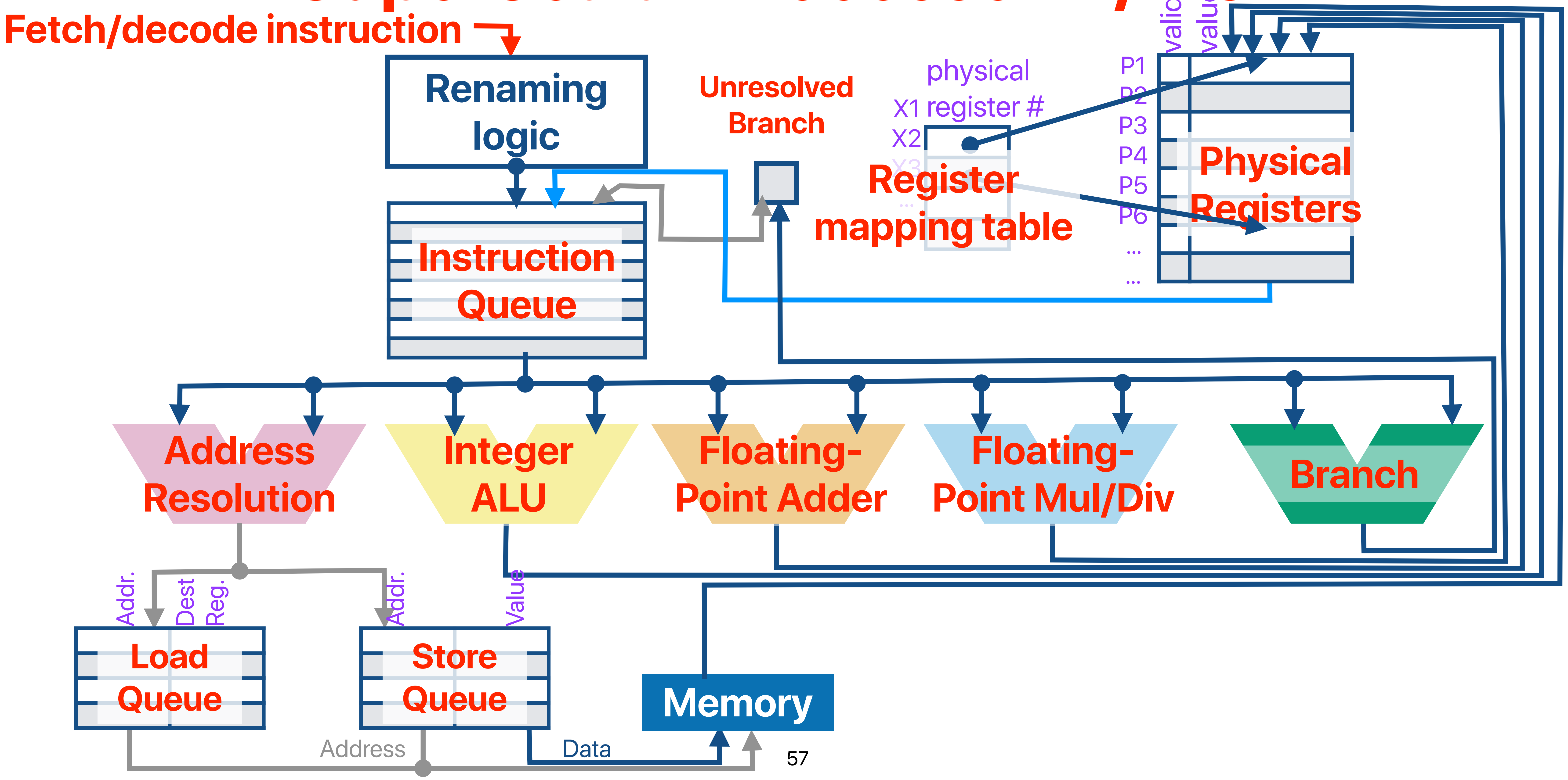
  D. 5

  E. 6

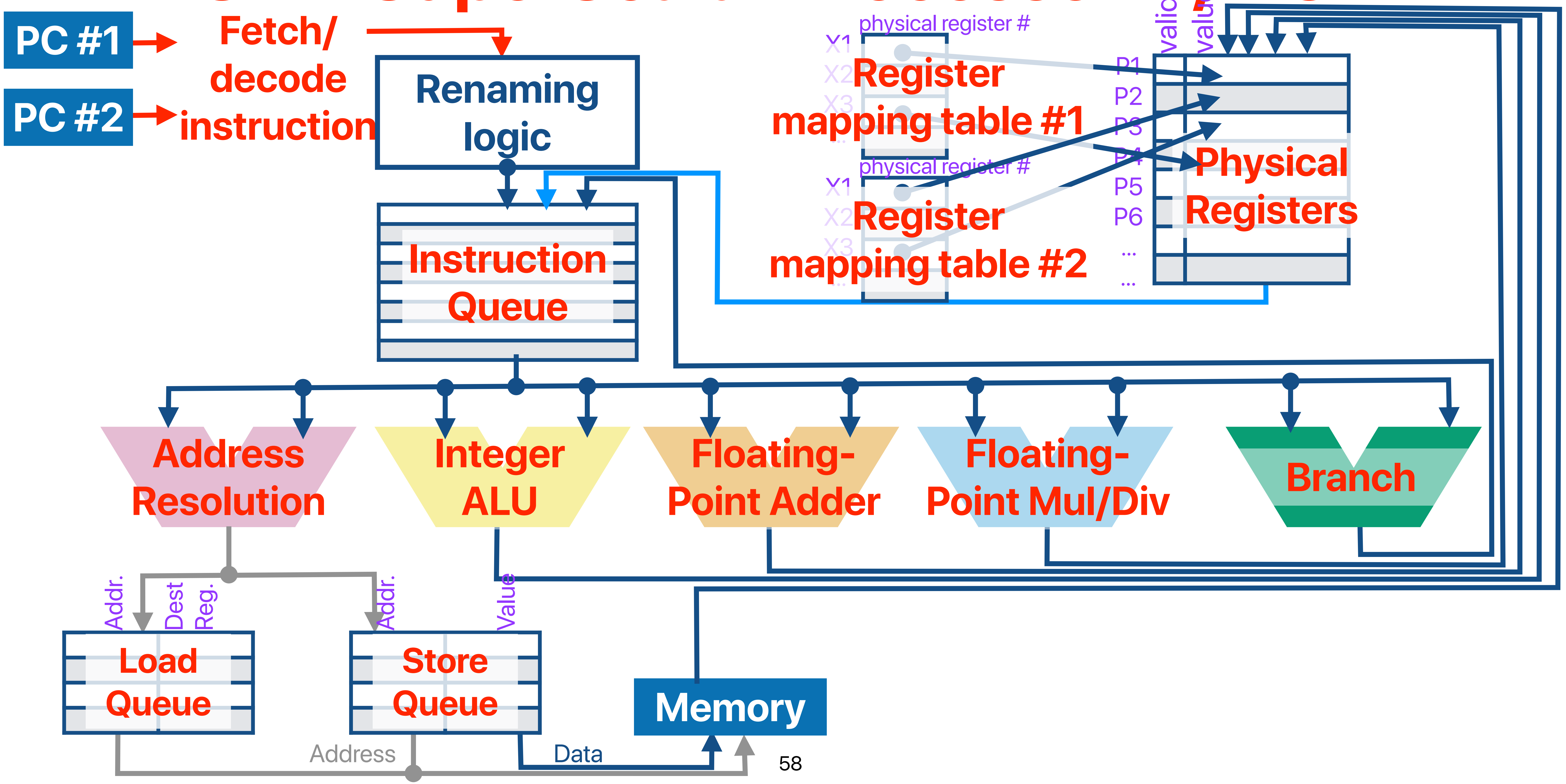
Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  ① Program counter — you need to have one for each context
  ② Register mapping tables — you need to have one for each context
  ③ Physical registers — you can share
  ④ ALUs — you can share
  ⑤ Data cache — you can share
  ⑥ Reorder buffer/Instruction Queue — you need to indicate which context the instruction is from
  A. 2
  B. 3
  C. 4
  D. 5
  E. 6

SuperScalar Processor w/ ROB



SMT SuperScalar Processor w/ ROB



Poll close in 1:30

SMT

- How many of the following about SMT are correct?
  ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  ② SMT can improve the throughput of a single-threaded application
  ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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

Poll close in 1:30

1 — Instinct

SMT

- How many of the following about SMT are correct?
  ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  ② SMT can improve the throughput of a single-threaded application
  ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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

SMT

- How many of the following about SMT are correct?
  1. SMT makes processors with deep pipelines more tolerable to mis-predicted branches **We can execute from other threads/contexts instead of the current one**
  2. **hurt, b/c you are sharing resource with other threads.** SMT can ~~improve~~ the throughput of a single-threaded application
  3. SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width **We can execute from other threads/ contexts instead of the current one**
  4. SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications. **b/c we're sharing the cache**

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

Announcement

- Project due next Monday
- Reading quiz due this Wednesday
- Assignment #5 will be up tomorrow — start EARLY!!!
- iEVAL, starting tomorrow until 12/11
  - Please fill the survey to let us know your opinion!
  - Don't forget to take a screenshot of your submission and submit through iLearn — it counts as a **full credit assignment**
  - **We will drop your lowest 2 assignment grades**
- Final Exam
  - Starting from 12/10 to 12/15 11:59pm (we won't provide any technical support after 12pm 12/15), any consecutive 180 minutes you pick
  - Similar to the midterm, but more time and about 1.5x longer
  - Will release a sample final at the end of the last lecture
- Office Hours on Zoom (the office hour link, not the lecture one)
  - Hung-Wei/Prof. Usagi: M 8p-9p, W 2p-3p
  - Quan Fan: F 1p-3p

Computer Science & Engineering

203

つづく