

Data Hazards & Dynamic Instruction Scheduling (III)

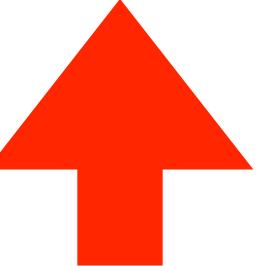
Hung-Wei Tseng

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

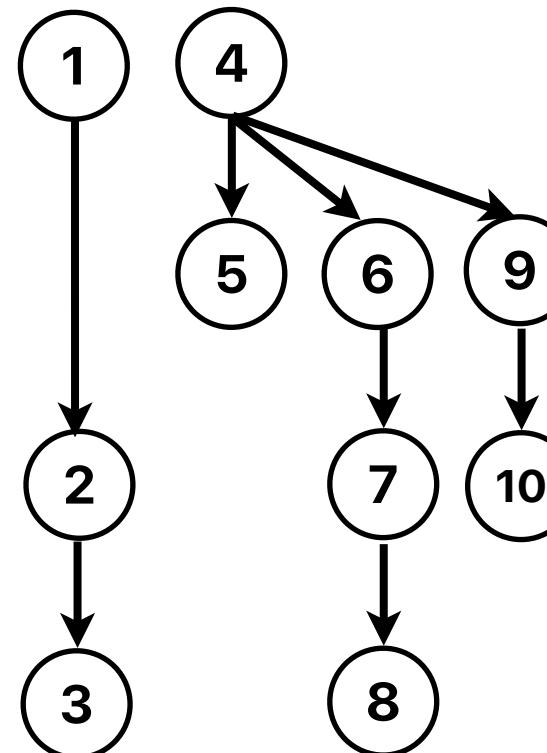


- This instruction has completed its own work in the current stage
- No other instruction is occupying the next stage
- The next stage has all its inputs ready

Recap: Scheduling instructions: based on data dependencies

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

- 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

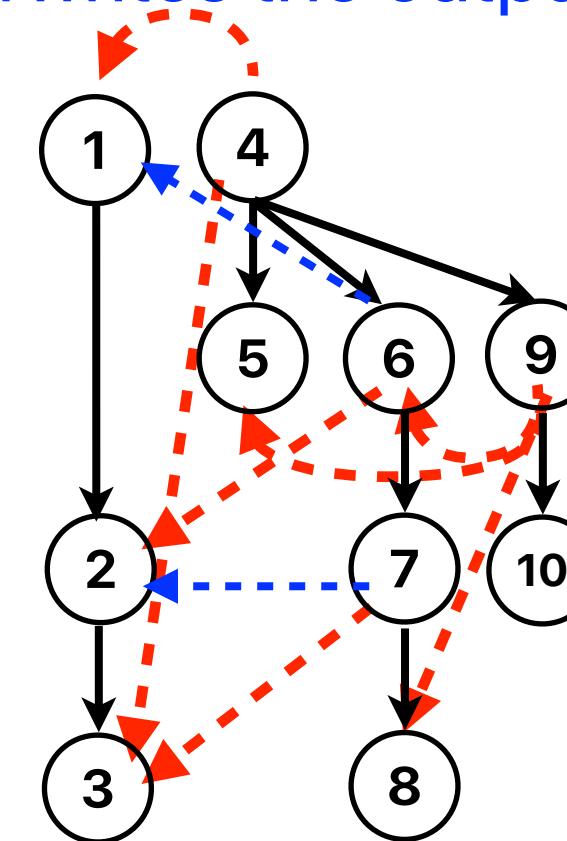


- **In theory**, instructions without dependencies can be executed in parallel or out-of-order
- Instructions with dependencies can never be reordered

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
 - 4 and 1 4 and 3, 6 and 2, 7 and 3, 9 and 5, 9 and 6, 9 and 8
 - WAW (Write After Write): a later instruction overwrites the output of an earlier one
 - 6 and 1, 7 and 2

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



Why OoO instead of compilers

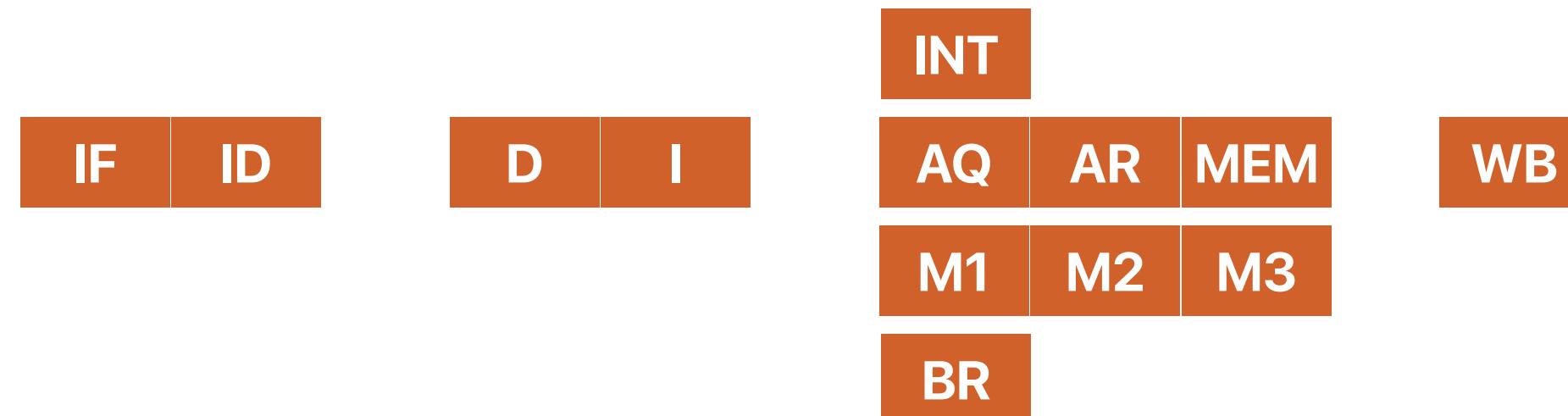
- Compiler can only see/optimize **static instructions**, instructions in the compiled binary
- Compiler cannot optimize **dynamic instructions**, the real instruction sequence when executing the program
 - Compiler cannot re-order 3, 5 or 4, 5
 - Compiler cannot predict cache misses
- Compiler optimization is constrained by **false dependencies** due to limited number of registers (even worse for x86)
 - Instructions
`ld X1, 0(X0)` and
`addi X0, X0, 4`
do not have a data dependency
- Compiler optimizations do not work for all architectures
 - The code optimization in the previous example works for single pipeline, but not for superscalar

Static instructions
LOOP: <code>ld X1, 0(X0)</code>
<code>addi X0, X0, 4</code>
<code>add X2, X2, X1</code>
<code>bne X0, X3, LOOP</code>
<code>ld X0, 0(\$sp)</code>
<code>ld X1, 4(\$sp)</code>

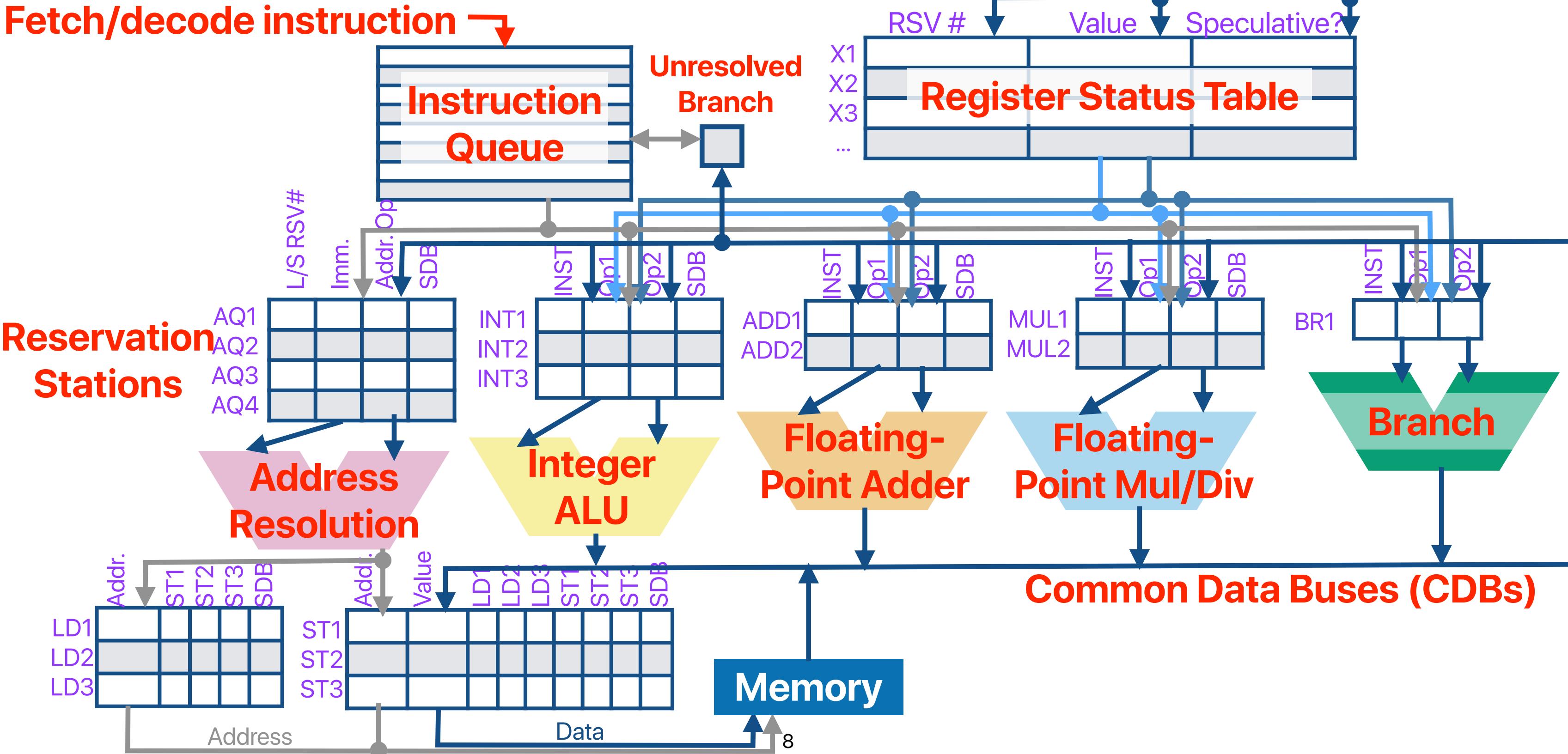
Dynamic instructions
1: <code>ld X1, 0(X0)</code>
2: <code>addi X0, X0, 4</code>
3: <code>add X2, X2, X1</code>
4: <code>bne X0, X3, LOOP</code>
5: <code>ld X1, 0(X0)</code>
6: <code>addi X0, X0, 4</code>
7: <code>add X2, X2, X1</code>
8: <code>bne X0, X3, LOOP</code>

Pipeline in Tomasulo

- Dispatch (D) — allocate a “reservation station” for a decoded instruction
- Issue (I) — collect pending values/branch outcome from common data bus
- Execute (INT, AQ/AQ/MEM, M1/M2/M3, BR) — send the instruction to its corresponding pipeline if no structural hazards
- Write Back (WB) — broadcast the result through CDB



Overview of a processor supporting Tomasulo's algorithm



Team scores



8

14.5

11

8

Outline

- Tomasulo's algorithm
- Register Renaming
- SuperScalar
- Simultaneous Multithreading

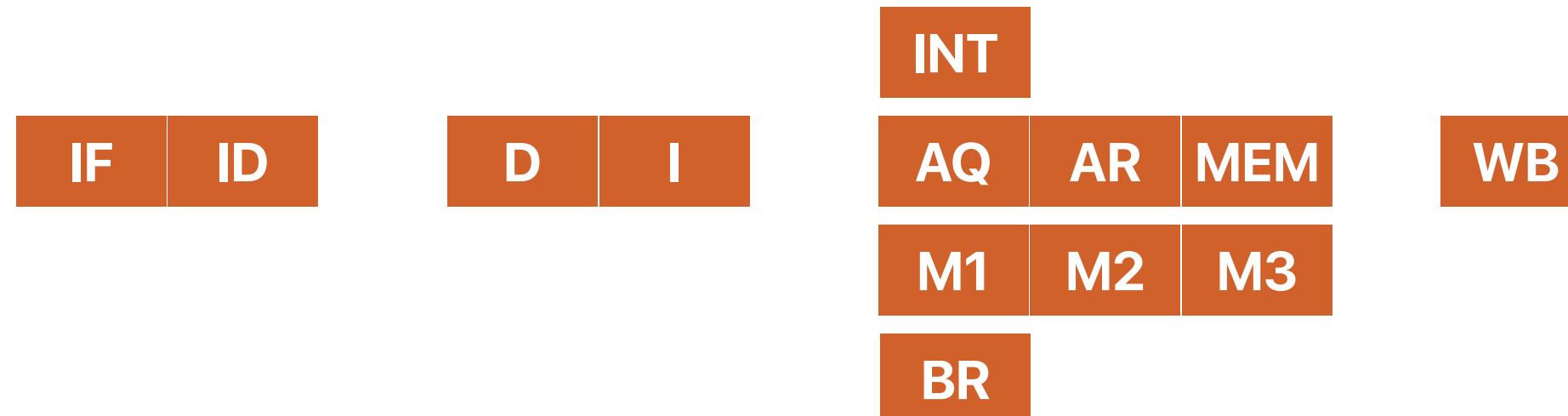
Dynamic instruction scheduling/ Out-of-order (OoO) execution



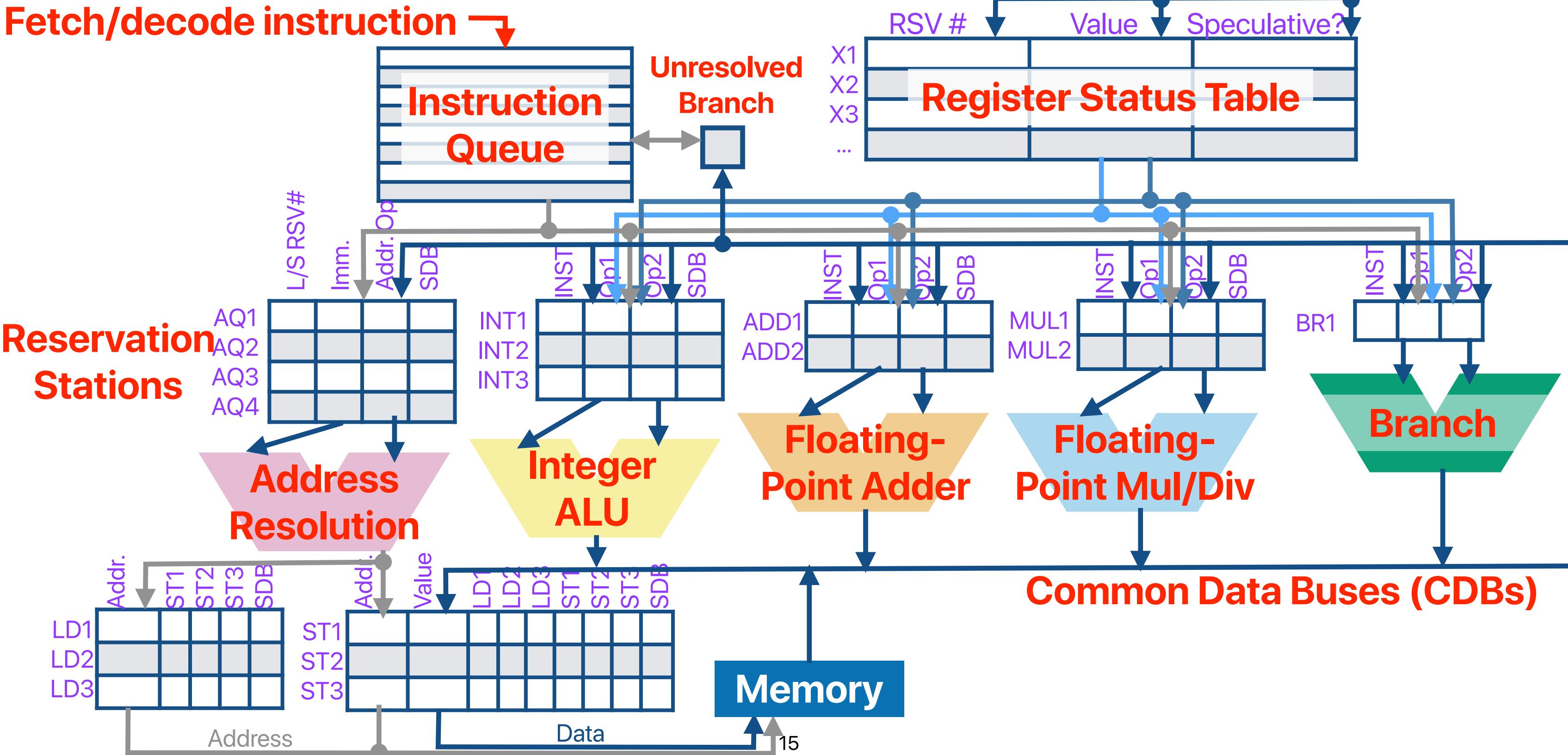
Tomasulo's Algorithm

Pipeline in Tomasulo

- Dispatch (D) — allocate a “reservation station” for a decoded instruction
- Issue (I) — collect pending values/branch outcome from common data bus
- Execute (INT, AQ/AQ/MEM, M1/M2/M3, BR) — send the instruction to its corresponding pipeline if no structural hazards
- Write Back (WB) — broadcast the result through CDB



Overview of a processor supporting Tomasulo's algorithm



Tomasulo 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

Tomasulo 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

Tomasulo in motion

①	ld	X6, 0(X10)	D	AQ
②	add	X7, X6, X12		D

- 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

Tomasulo in motion

①	ld	X6, 0(X10)	D	AQ	AR
②	add	X7, X6, X12		D	I
③	sd	X7, 0(X10)			D

- 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

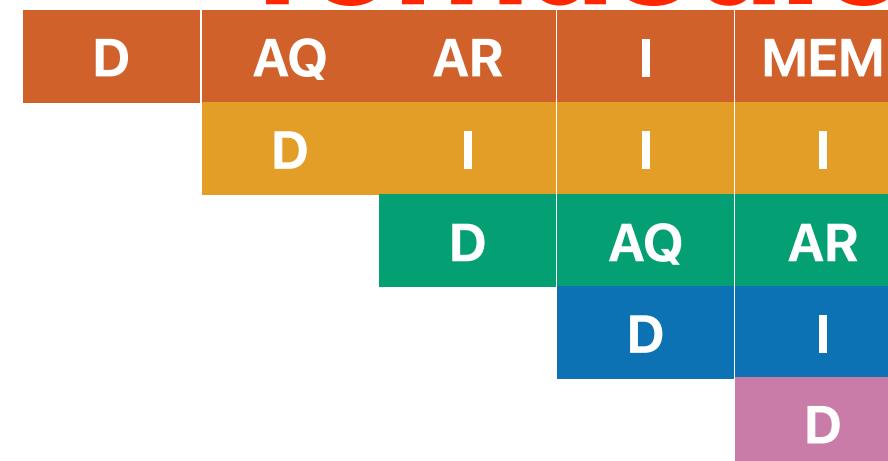
Tomasulo in motion

			D	AQ	AR	I
②	add	X7,X6,X12		D	I	I
③	sd	X7,0(X10)		D		AQ
④	addi	X10,X10,8				D

- 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

Tomasulo in motion

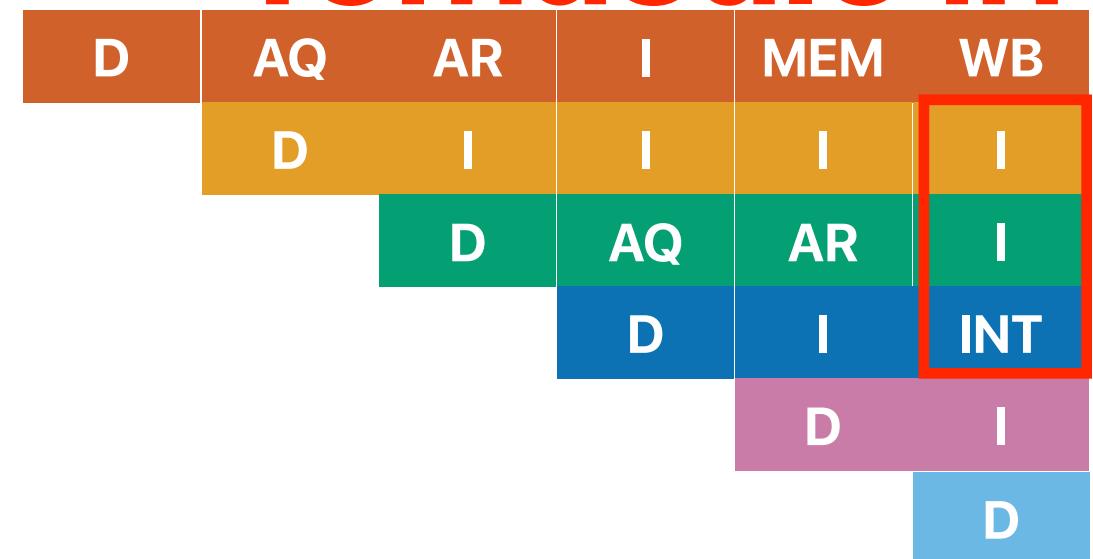
- ① ld X6, 0(X10)
- ② add X7, X6, X12
- ③ sd X7, 0(X10)
- ④ addi X10, X10, 8
- ⑤ bne X10, X5, LOOP
- ⑥ ld X6, 0(X10)
- ⑦ add X7, X6, X12
- ⑧ sd X7, 0(X10)
- ⑨ addi X10, X10, 8
- ⑩ bne X10, X5, LOOP



	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2										X6	LD1	
LD3										X7	INT1	
ST1	sd	0	[X10]				INT1		3	X10	INT2	
ST2										X12		
ST3												
INT1	add		[X12]		LD1				2			
INT2	addi	8	[X10]						4			
MUL1												
MUL2												
BR	bne		[X5]		INT2				5			

Tomasulo 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



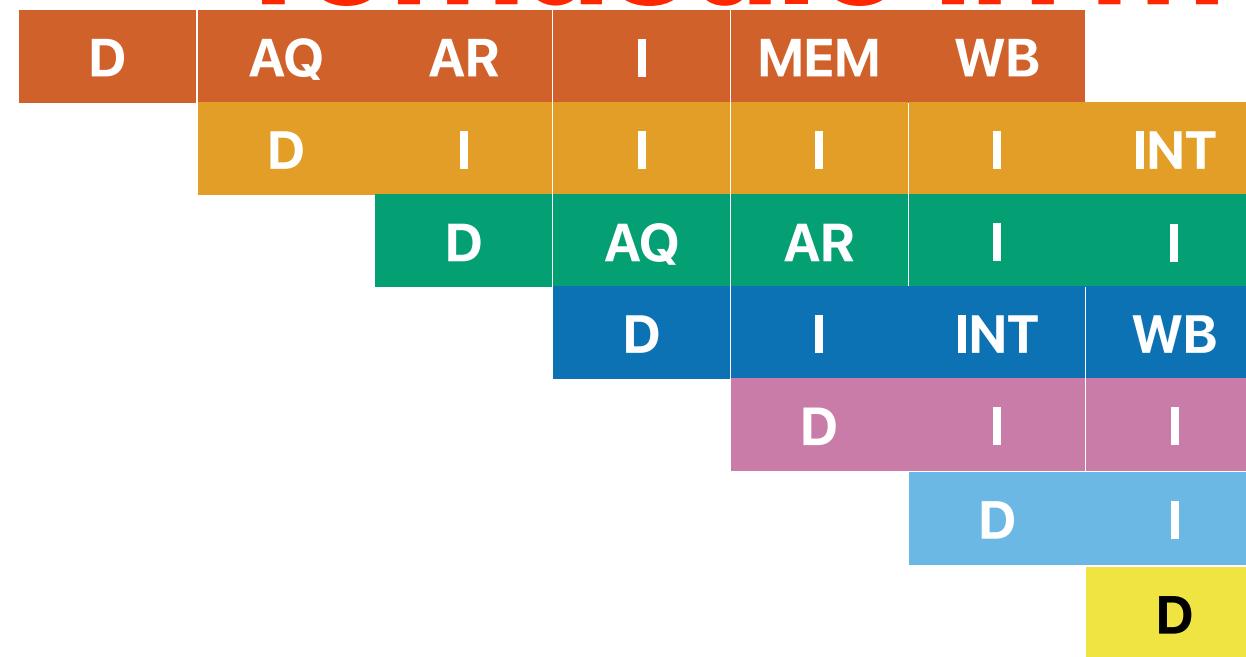
addi is now ahead of add!

	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									
ST3									
INT1	add	LD1	[X12]						2
INT2	addi	8	[X10]						4
MUL1									
MUL2									
BR	bne		[X5]		INT2				5

RSV #	Value	Spec?
X5		
X6	LD2	LD1 1
X7	INT1	
X10	INT2	
X12		

Tomasulo 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

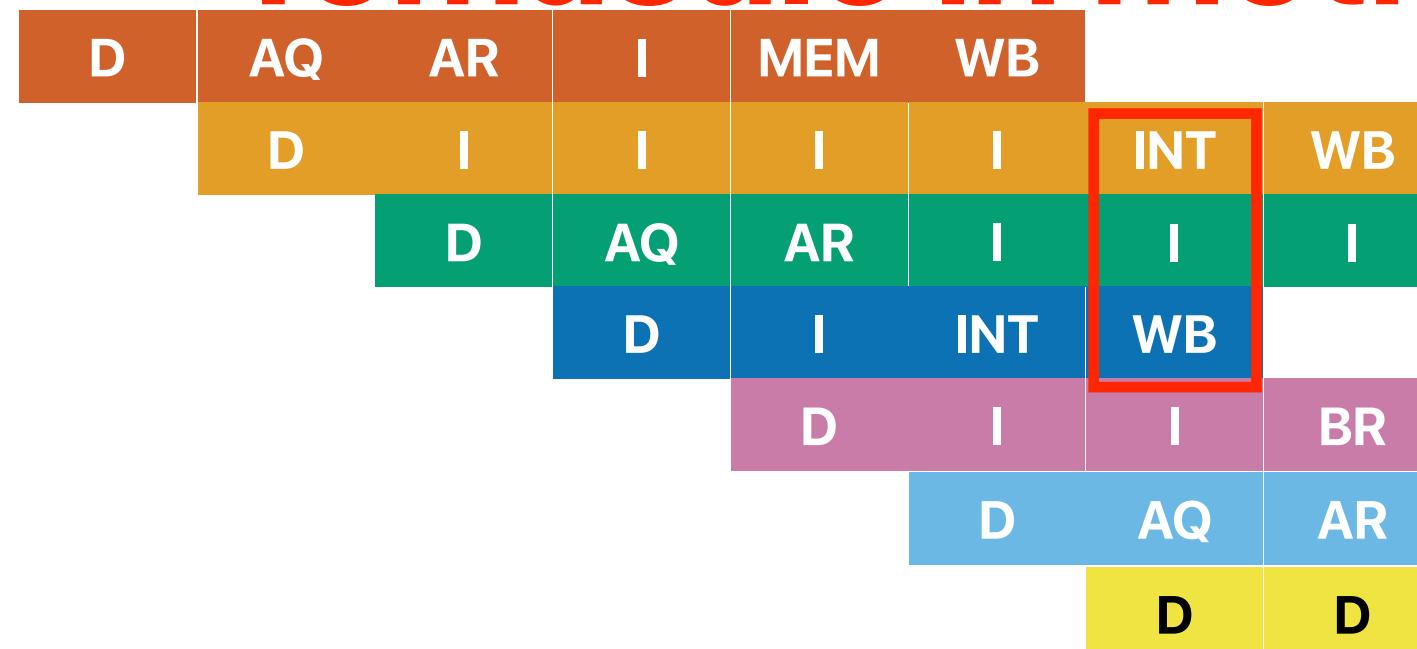


no reservation station for add!

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2	ld	0	INT2						6	X6	LD2	1
LD3										X7	INT1	
ST1	sd	0	[X10]				INT1		3	X10		INT2
ST2										X12		
ST3												
INT1	add	LD1	[X12]						2			
INT2	addi	8	[X10]						4			
MUL1												
MUL2												
BR	bne	INT2	[X5]						5			

Tomasulo 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



addi is finished
ahead of **add** and **sd**!

	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									
ST3									
INT1	add	LD1	[X12]					2	
INT2	add		[X12]		[LD2]			7	
MUL1									
MUL2									
BR	bne	INT2	[X5]						

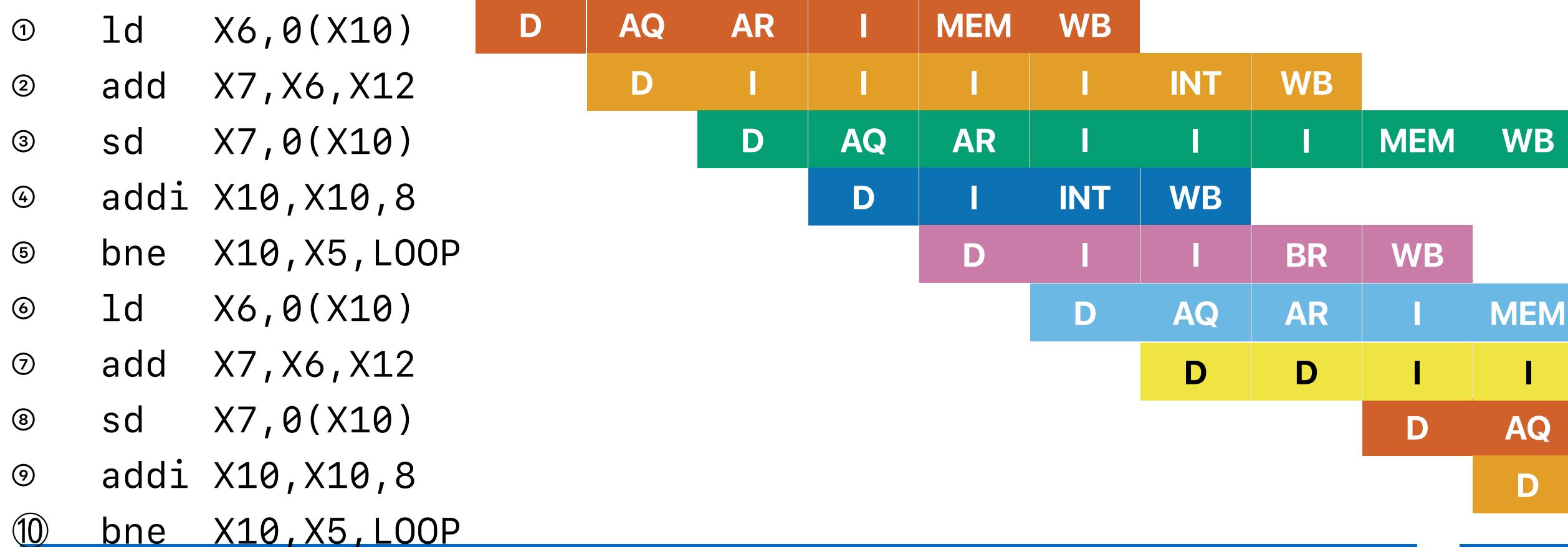
	RSV #	Value	Spec?
X5			
X6	LD2	1	
X7	INT2	INT1	1
X10		INT2	
X12			

Tomasulo in motion

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

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2	ld	0	INT2						6	X6	LD2	
LD3										X7	INT2	
ST1	sd	0	[X10]	INT1					3	X10		INT2
ST2	sd	0	[X10]				INT2		8	X12		
ST3												
INT1	add	LD1	[X12]						2			
INT2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	bne	INT2	[X5]						5			

Tomasulo in motion



	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2	ld	0	INT2						6	X6	LD2	
LD3										X7	INT2	
ST1	sd	0	[X10]	INT1					3	X10	INT1	
ST2	sd	0	[X10]				INT2		8	X12		
ST3												
INT1	add	8	INT2						9			
INT2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	bne	INT2	[X5]						5			

Tomasulo in motion

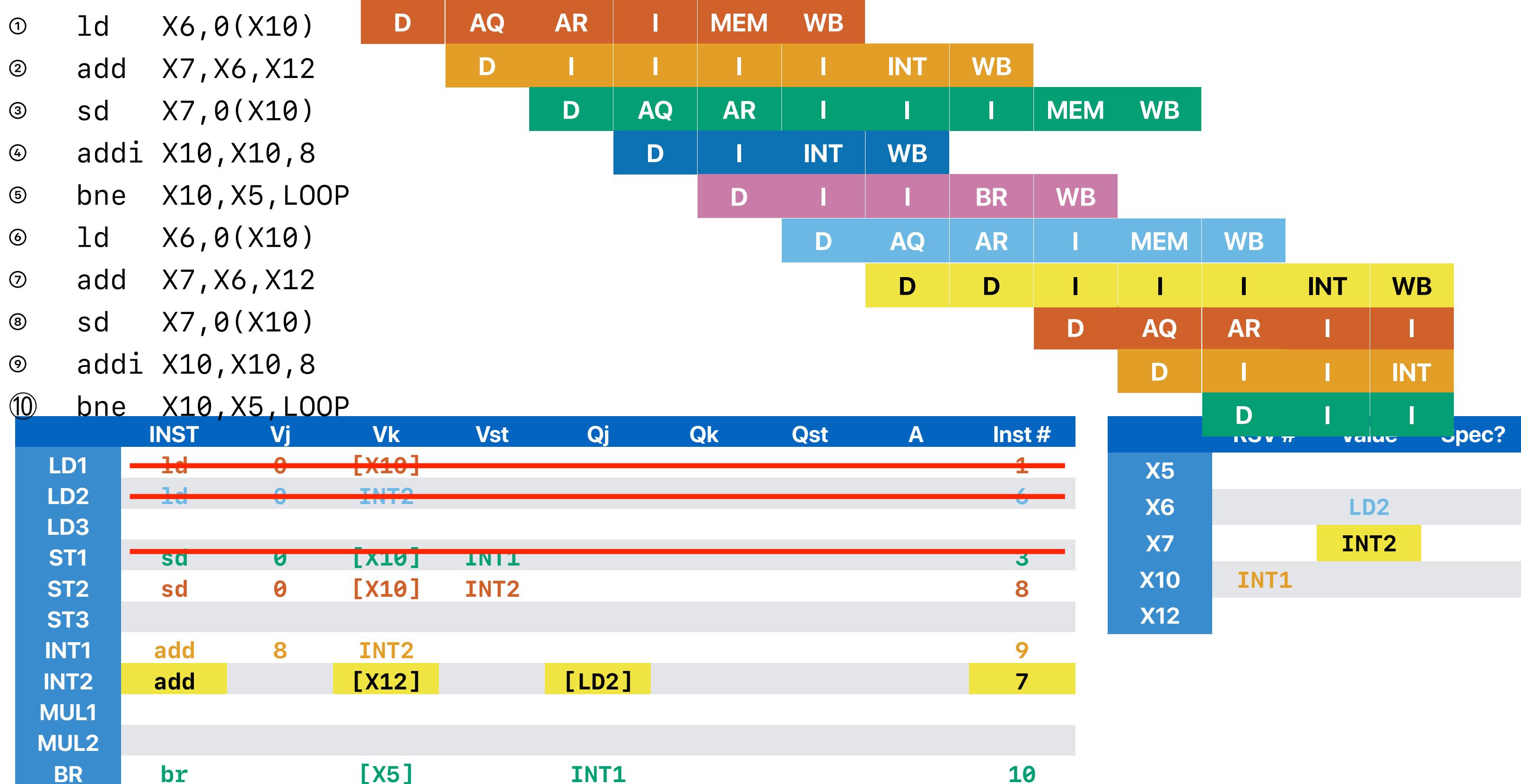
			D	AQ	AR	I	MEM	WB				
①	ld	X6, 0(X10)	D	AQ	AR	I	MEM	WB				
②	add	X7, X6, X12		D	I	I	I	I	INT	WB		
③	sd	X7, 0(X10)		D	AQ	AR	I	I	I	MEM	WB	
④	addi	X10, X10, 8			D	I	INT	WB				
⑤	bne	X10, X5, LOOP				D	I	I	BR	WB		
⑥	ld	X6, 0(X10)				D	AQ	AR	I	MEM	WB	
⑦	add	X7, X6, X12					D	D	I	I	I	
⑧	sd	X7, 0(X10)						D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

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	Rsv #	Value	Spec?
LD1	ld	0	[X10]						1		X5	
LD2	ld	0	INT2						6		X6	LD2
LD3											X7	INT2
ST1	sd	0	[X10]	INT1					3		X10	INT1
ST2	sd	0	[X10]				INT2		8		X12	
ST3												
INT1	add	8	INT2						9			
INT2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	br		[X5]		INT1				10			

Tomasulo in motion

		D	AQ	AR	I	MEM	WB					
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	I	MEM	WB	
④	addi	X10, X10, 8	D	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	I	INT		
⑧	sd	X7, 0(X10)			D	AQ	AR	I				
⑨	addi	X10, X10, 8		D	I	I						
⑩	bne	X10, X5, LOOP			D	I						
	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	Rcv.	value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2	ld	0	INT2						6	X6		LD2
LD3										X7	INT2	
ST1	sd	0	[X10]	INT1					3	X10		INT1
ST2	sd	0	[X10]				INT2		8	X12		
ST3												
INT1	add	8	INT2						9			
INT2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	br		[X5]		INT1				10			

Tomasulo in motion



Tomasulo in motion

The timing diagram illustrates the execution of 10 instructions over 10 clock cycles. Each row represents an instruction, and each column represents a stage in the pipeline or a specific control signal.

Instruction Details:

INST	V _j	V _k	V _{st}	Q _j	Q _k	Q _{st}	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				

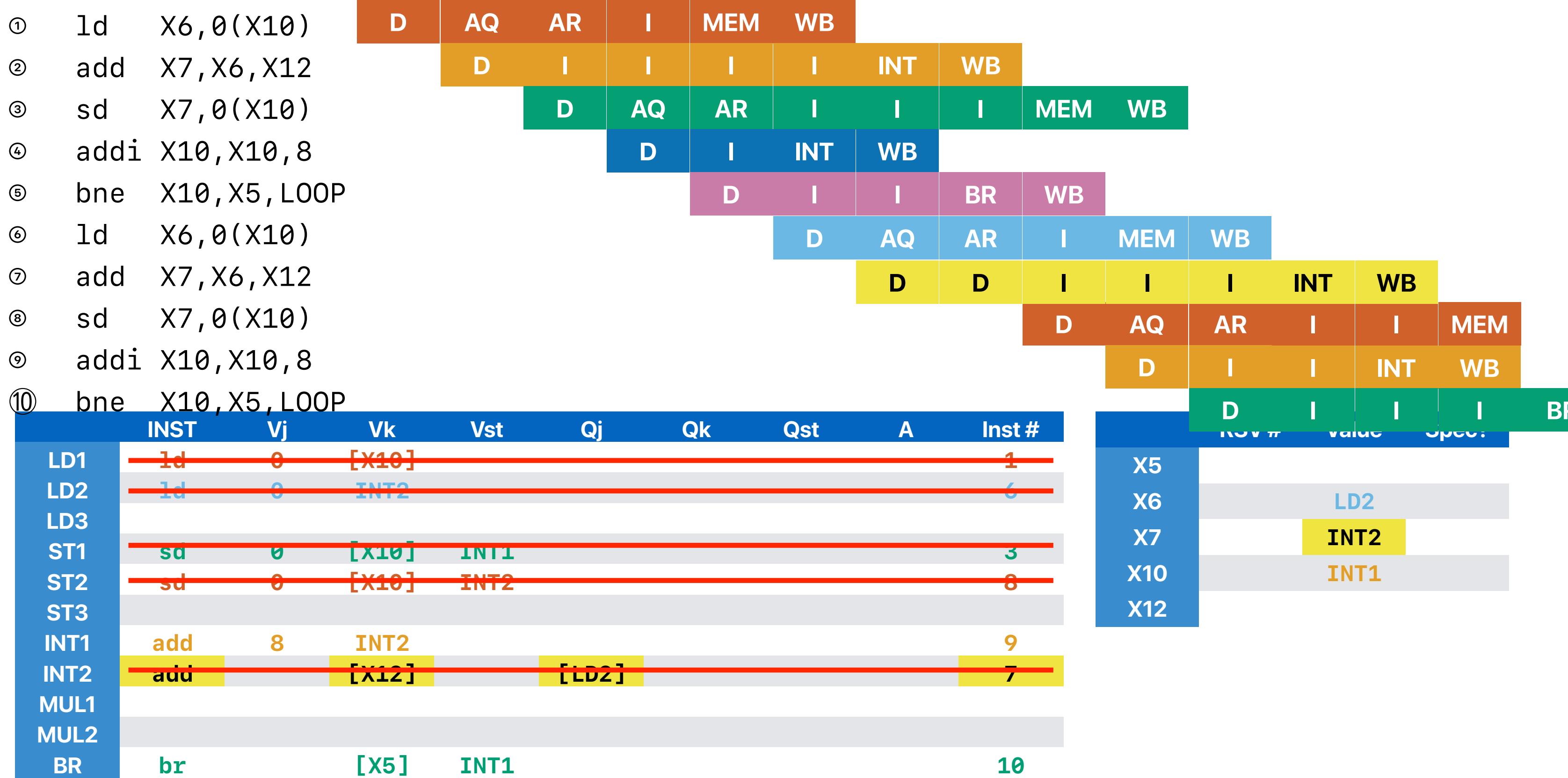
Control Signals:

- D:** Data path control.
- AQ:** Address Queue control.
- AR:** Register Read control.
- I:** Instruction control.
- MEM:** Memory control.
- WB:** Write Back control.
- INT:** Interrupt control.
- BR:** Branch control.

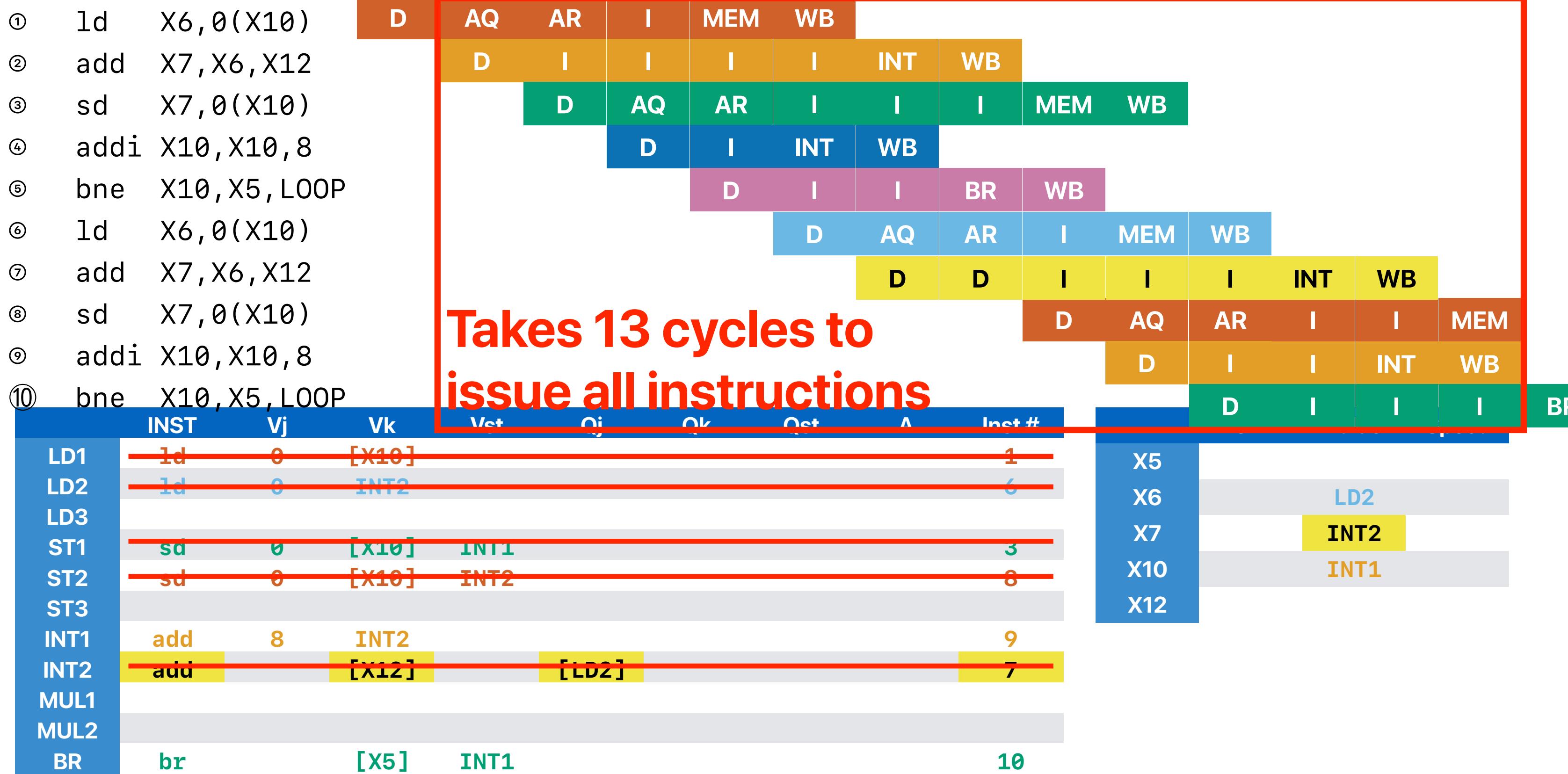
Memory Access:

- LD1:** Load from X10 into Vj.
- LD2:** Load from X12 into Vj.
- ST1:** Store value INT1 to X10.
- ST2:** Store value INT2 to X10.
- INT1:** Add X10 + 8 into Vj.
- INT2:** Add X12 + LD2 into Vj.
- BR:** Branch to X5 if INT1 is zero.

Tomasulo in motion



Tomasulo in motion



Register renaming

- K. C. Yeager, "The Mips R10000 superscalar microprocessor," in IEEE Micro, vol. 16, no. 2, pp. 28-41, April 1996.
- R. E. Kessler, "The Alpha 21264 microprocessor," in IEEE Micro, vol. 19, no. 2, pp. 24-36, March-April 1999.

Tomasulo in motion

① ld X6, 0(X10) D AQ AR I MEM WB

② add X7, X6, X12 D 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 I

no reservation station for add!

Takes 13 cycles to issue all instructions

	INST	V _j	V _k	V _{st}	Q _j	Q _k	Q _{st}	A	Inst #	
LD1	ld	0	[X10]						1	X5
LD2	ld	0	INT2						6	X6
LD3										LD2
ST1	sd	0	[X10]	INT1					3	X7
ST2	sd	0	[X10]	INT2					8	X10
ST3										INT1
INT1	add	8	INT2						9	X12
INT2	add		[X12]	[LD2]					7	
MUL1										
MUL2										
BR	br		[X5]	INT1					10	

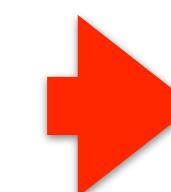
Recap: Why is B better than A?

A

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

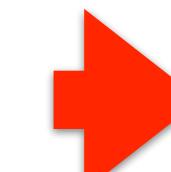
B

```
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;  
}
```

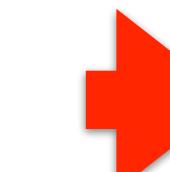


```
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
bne x1, x0, LOOP
```

4*n instructions



```
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
bne x1, x0, LOOP
```



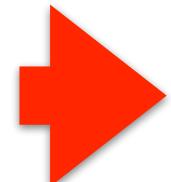
```
and x2, x1, 1  
shr x4, x1, 1  
shr x5, x1, 2  
shr x6, x1, 3  
shr x1, x1, 4  
and x7, x4, 1  
and x8, x5, 1  
and x9, x6, 1  
add x3, x3, x2  
add x3, x3, x7  
add x3, x3, x8  
add x3, x3, x9  
bne x1, x0, LOOP
```

13*(n/4) = 3.25*n instructions

35 Only one branch for four iterations in A

Recap: Why is B better than A?

```
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
bne x1, x0, LOOP
```

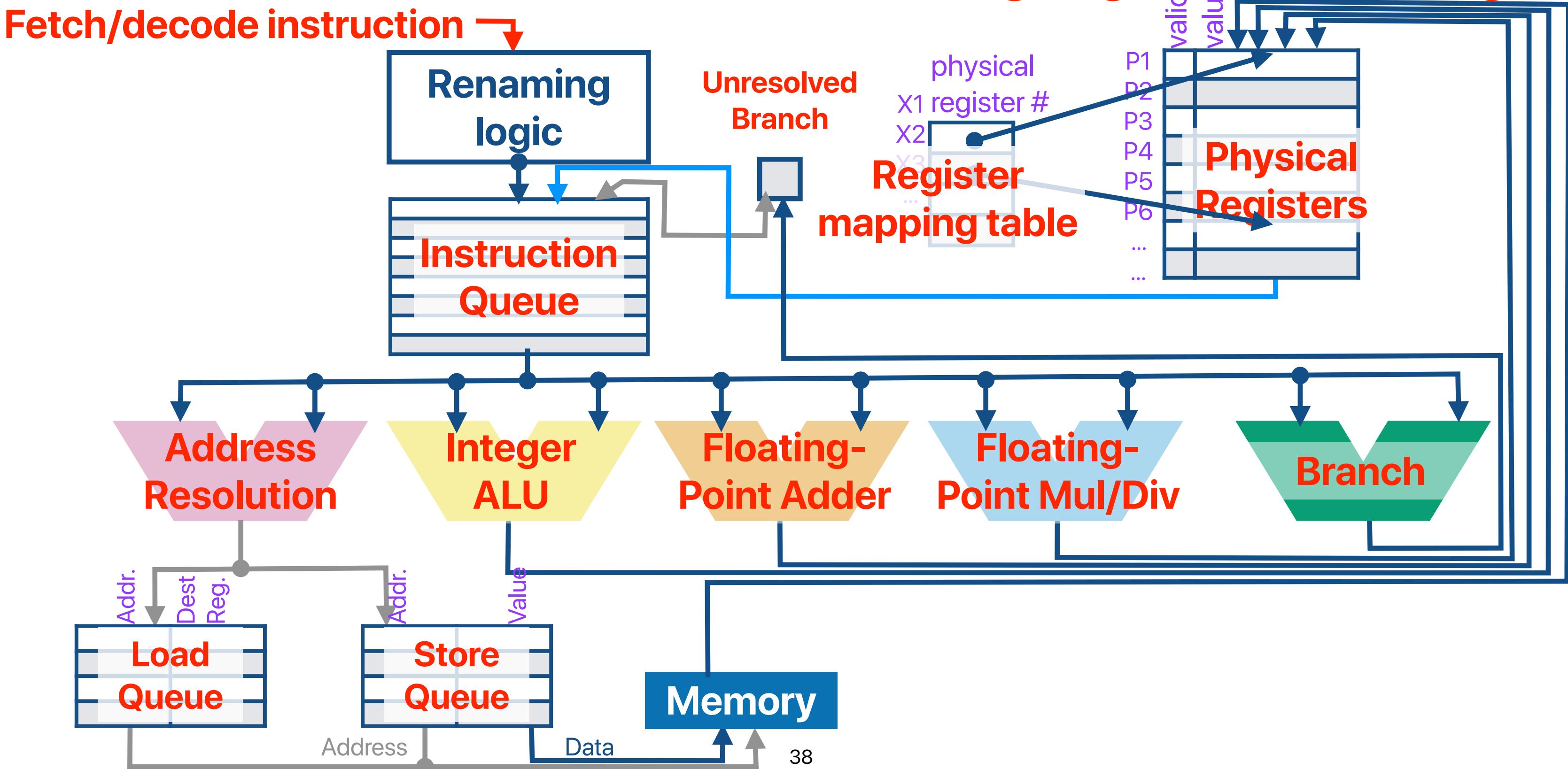


```
and x2, x1, 1  
shr x4, x1, 1  
shr x5, x1, 2  
shr x6, x1, 3  
shr x1, x1, 4  
and x7, x4, 1  
and x8, x5, 1  
and x9, x6, 1  
add x3, x3, x2  
add x3, x3, x7  
add x3, x3, x8  
add x3, x3, x9  
bne x1, x0, LOOP
```

Register renaming

- Decouple “reservation stations” from functional units
- Provide a set of “physical registers” and a mapping table mapping “architectural registers” to “physical registers”
- Allocate a physical register for a new output
- Stages
 - Dispatch/Rename (R) — allocate a “physical register” for the output of a decoded instruction
 - Issue (I) — collect pending values/branch outcome from common data bus
 - Execute (INT, AQ/AQ/MEM, M1/M2/M3, BR) — send the instruction to its corresponding pipeline if no structural hazards
 - Write Back (WB) — broadcast the result through CDB

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

R

Renamed instruction	
1	ld P1, 0(X10)
2	
3	
4	
5	
6	
7	
8	
9	
10	

Physical Register	
X5	
X6	P1
X7	
X10	
X12	

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

Register renaming in motion

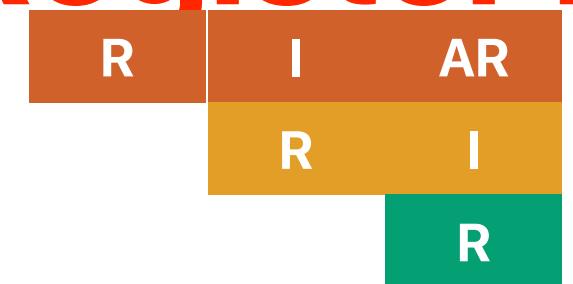
- ① ld X6, 0(X10)
- ② add X7, X6, X12
- ③ sd X7, 0(X10)
- ④ addi X10, X10, 8
- ⑤ bne X10, X5, LOOP
- ⑥ ld X6, 0(X10)
- ⑦ add X7, X6, X12
- ⑧ sd X7, 0(X10)
- ⑨ addi X10, X10, 8
- ⑩ bne X10, X5, LOOP



Renamed instruction		Physical Register			Valid Value In use			Valid Value In use			
		X5		X6	P1	X7	P2	X10	P3	X12	P4
1	ld P1, 0(X10)								P1	0	1
2	add P2, P1, X12								P2	0	1
3									P3		
4									P4		
5									P5		
6									P6		
7									P7		
8									P8		
9									P9		
10									P10		

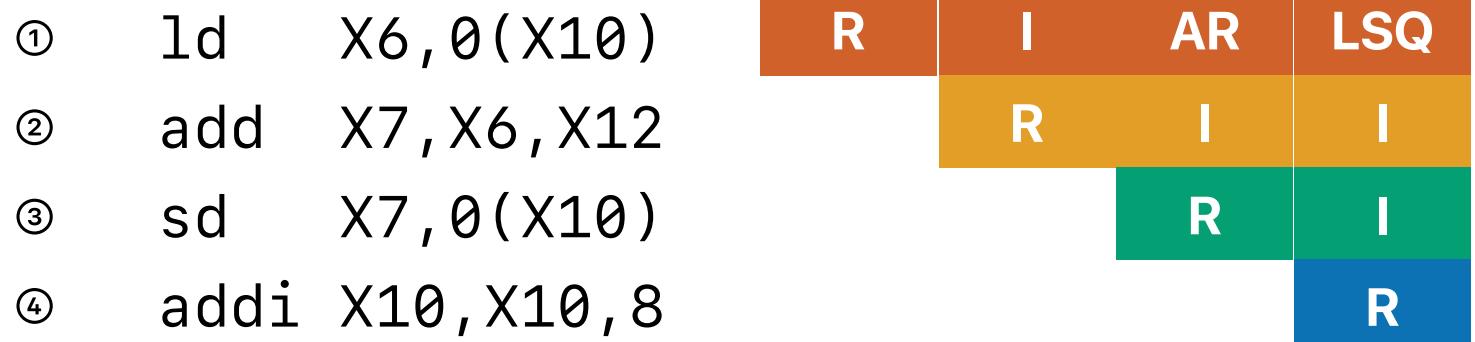
Register renaming in motion

- 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



Renamed instruction		Physical Register			Valid Value In use			Valid Value In use			
		X5		X6	P1	X7	P2	X10	P3	X12	P4
1	ld P1, 0(X10)			X5					P1	0	1
2	add P2, P1, X12			X6	P1				P2	0	1
3	sd P2, 0(X10)			X7	P2				P3		
4				X10					P4		
5				X12					P5		
6									P6		
7									P7		
8									P8		
9									P9		
10									P10		

Register renaming in motion



Renamed instruction		
1	ld	P1, 0(X10)
2	add	P2, P1, X12
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
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		

Register renaming in motion

- | | | R | I | AR | LSQ | MEM |
|---|-------------------|---|---|----|-----|-----|
| ① | ld X6, 0(X10) | R | I | AR | LSQ | MEM |
| ② | add X7, X6, X12 | R | I | I | I | |
| ③ | sd X7, 0(X10) | | R | I | I | |
| ④ | addi X10, X10, 8 | | | R | I | |
| ⑤ | bne X10, X5, LOOP | | | | | R |
| 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)	
2	add P2, P1, X12	
3	sd P2, 0(X10)	
4	addi P3, X10, 8	
5	bne P3, X5, LOOP	
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			

Register renaming in motion

			R	I	AR	LSQ	MEM	WB
①	ld	X6, 0(X10)	R	I	AR	LSQ	MEM	WB
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	
⑤	bne	X10, X5, LOOP				R	I	
⑥	ld	X6, 0(X10)					R	
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)
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	
8	
9	
10	

	Physical Register
X5	
X6	P1
X7	P2
X10	P3
X12	

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

Register renaming in motion

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

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		
9		
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			

Register renaming in motion

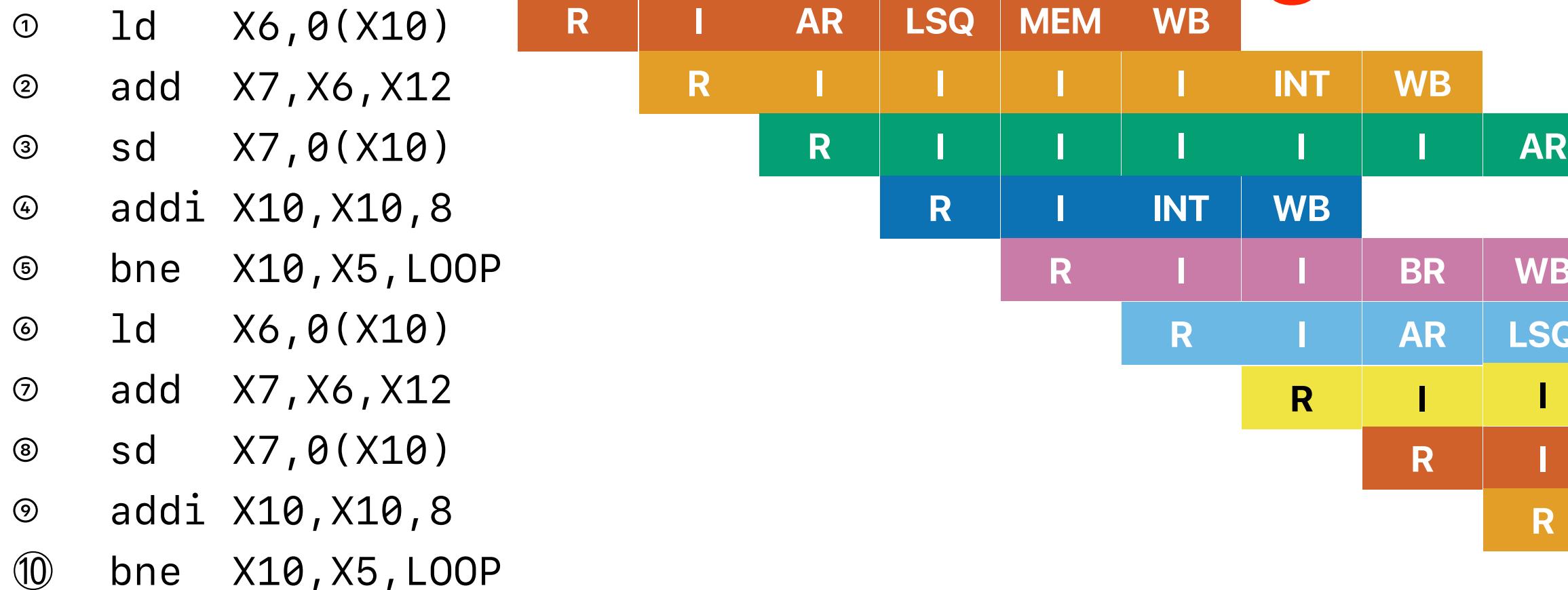
			R	I	AR	LSQ	MEM	WB	
①	ld	X6, 0(X10)	R	I	AR	LSQ	MEM	WB	
②	add	X7, X6, X12	R	I	I	I	I	INT	WB
③	sd	X7, 0(X10)	R	I	I	I	I	I	I
④	addi	X10, X10, 8		R	I	INT	WB		
⑤	bne	X10, X5, LOOP			R	I	I	BR	
⑥	ld	X6, 0(X10)			R	I	AR		
⑦	add	X7, X6, X12			R	I			
⑧	sd	X7, 0(X10)				R			
⑨	addi	X10, X10, 8							
⑩	bne	X10, X5,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

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		
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	0		1	P9			
P5	0		1	P10			

Register renaming in motion

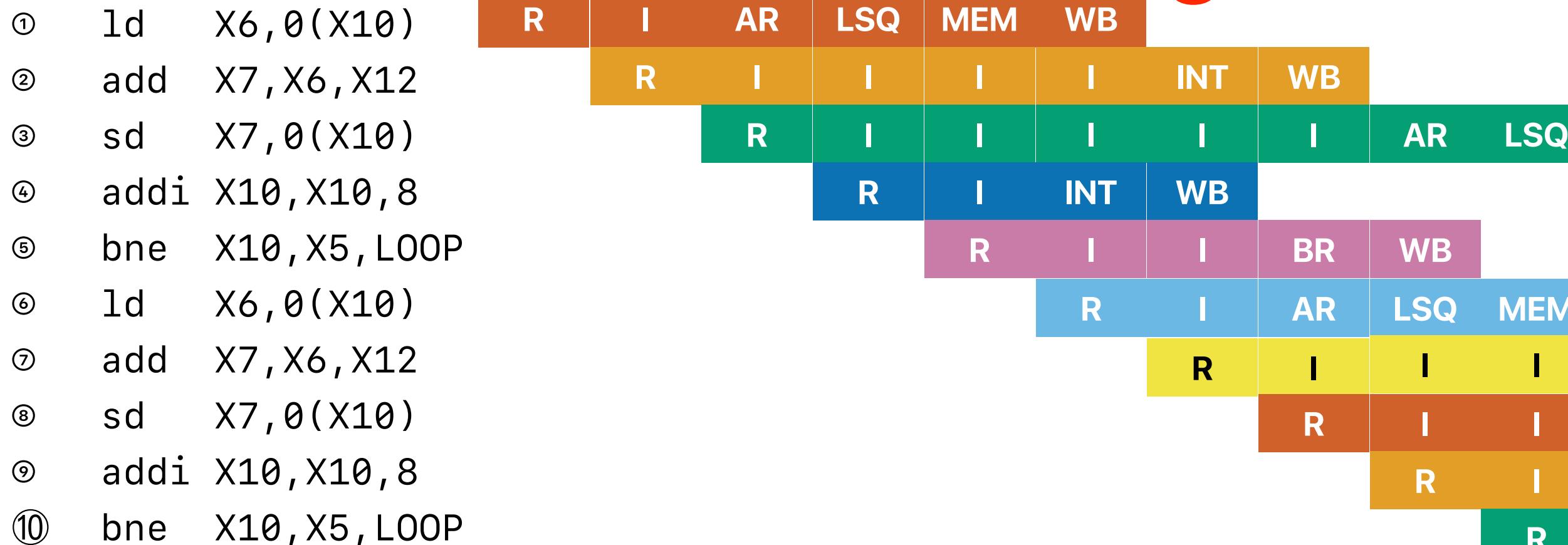


	Renamed instruction
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

Register renaming in motion

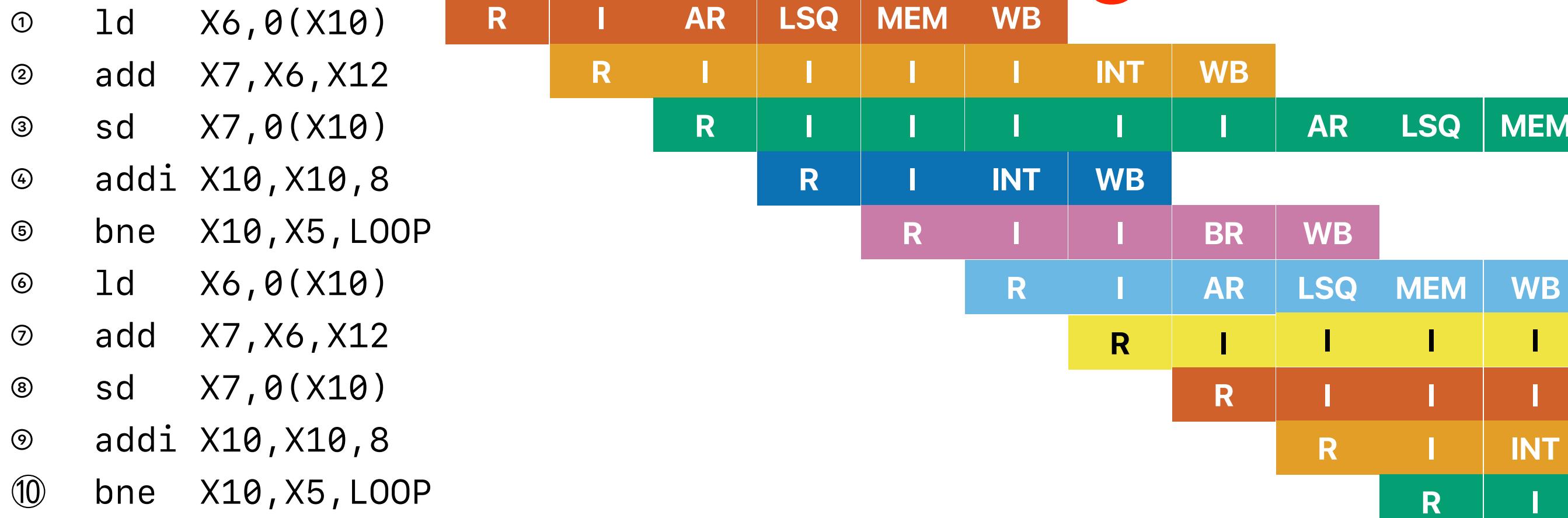


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	
X6	P1
X7	P5
X10	P3
X12	

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

Register renaming in motion

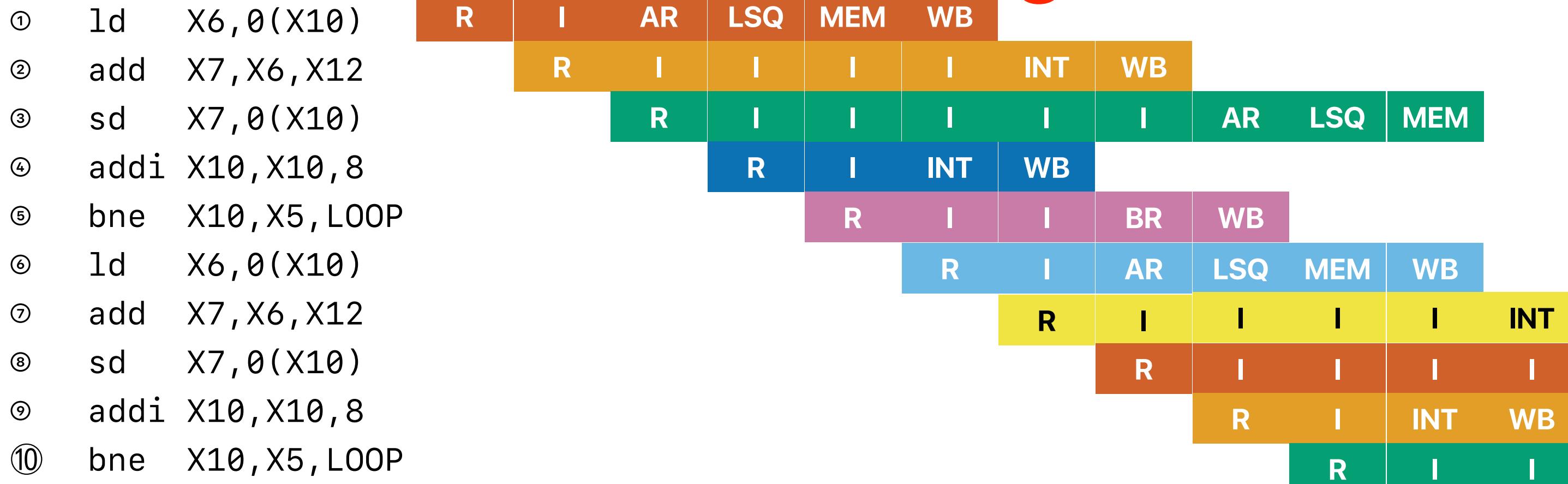


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	
X6	P1
X7	P5
X10	P3
X12	

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

Register renaming in motion

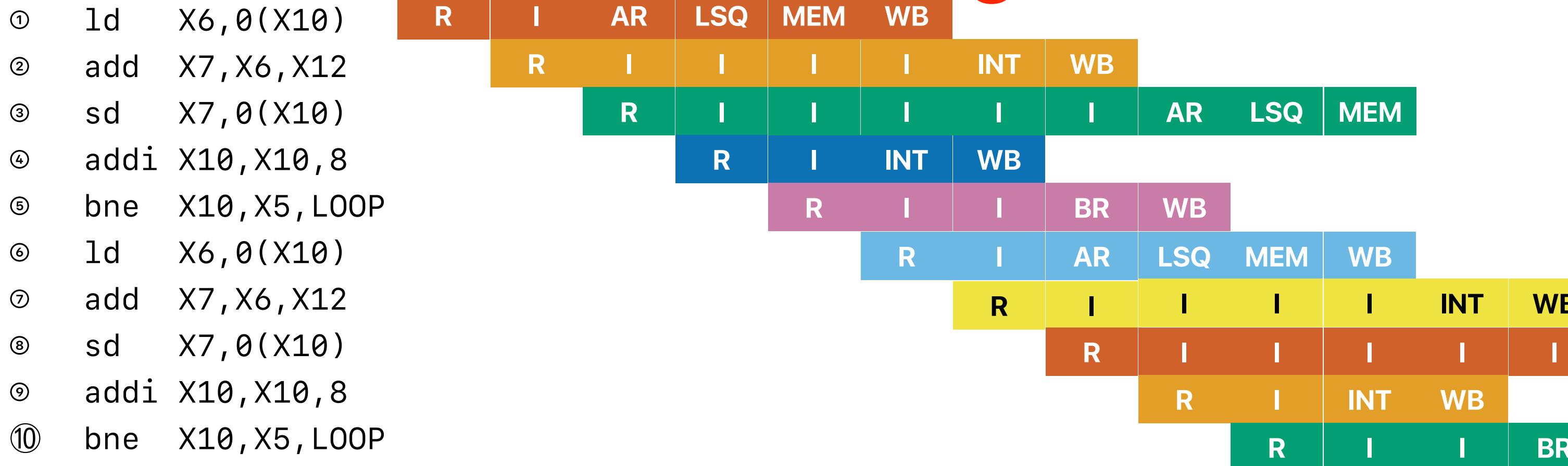


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	P1
X6	P1
X7	P5
X10	P3
X12	

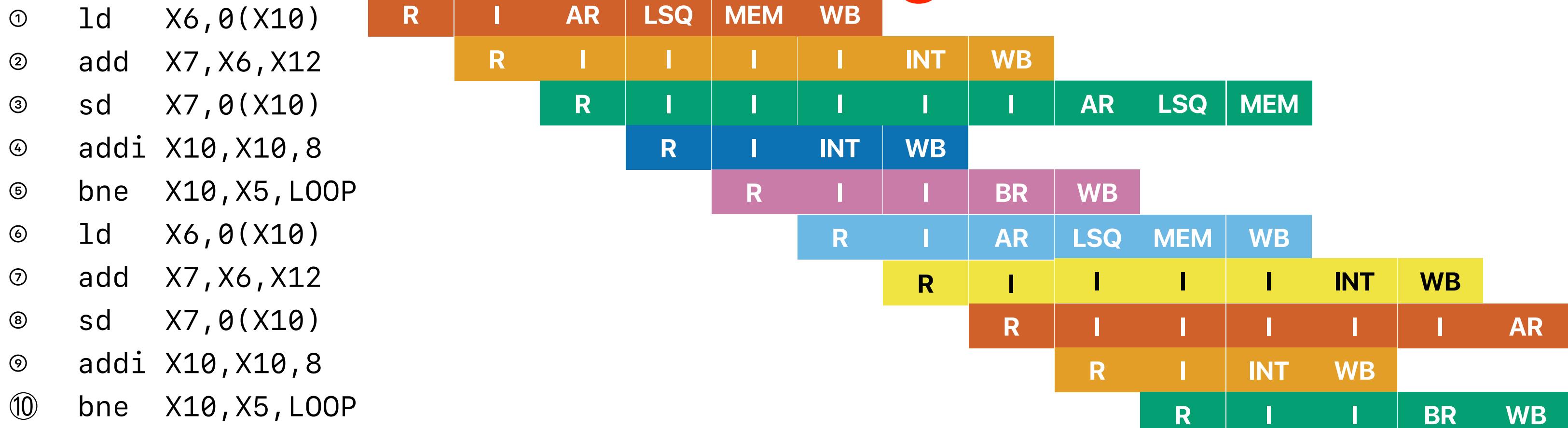
Valid			Valid		
Value			Value		
In use			In use		
P1	1	1	P6	1	1
P2	1	1	P7		
P3	1	1	P8		
P4	1	1	P9		
P5	0	1	P10		

Register renaming in motion



Renamed instruction	Physical Register			Valid Value In use			Valid Value In use		
	X5	X6	X7	P1	P2	P3	P6	P7	P8
1 ld P1, 0(X10)				1		1	1		1
2 add P2, P1, X12				1		1		1	
3 sd P2, 0(X10)				1		1		1	
4 addi P3, X10, 8				1		1		1	
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)									

Register renaming in motion

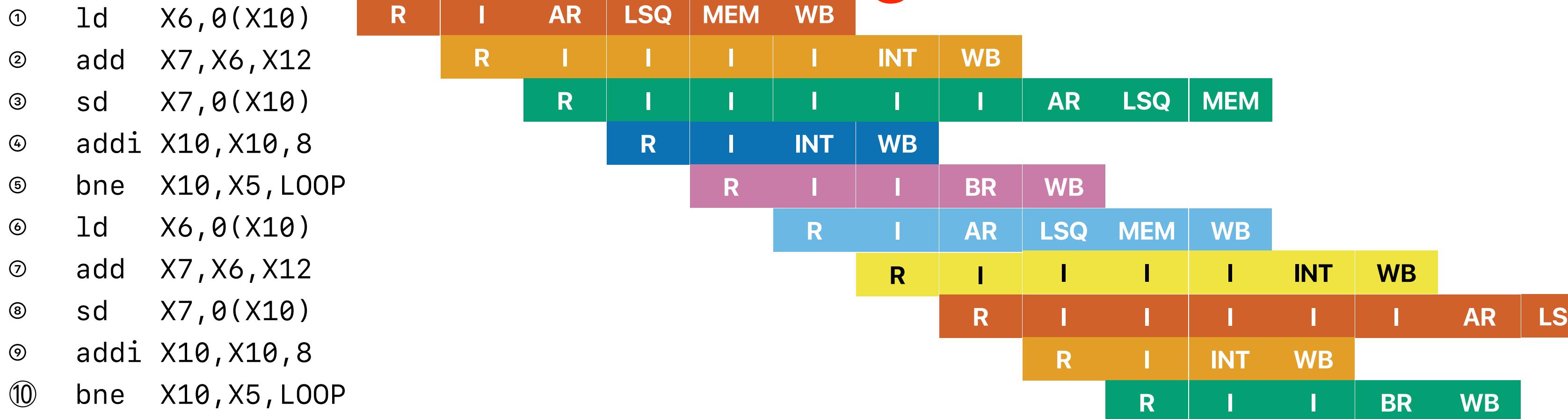


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	P1
X6	P1
X7	P5
X10	P3
X12	

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

Register renaming in motion



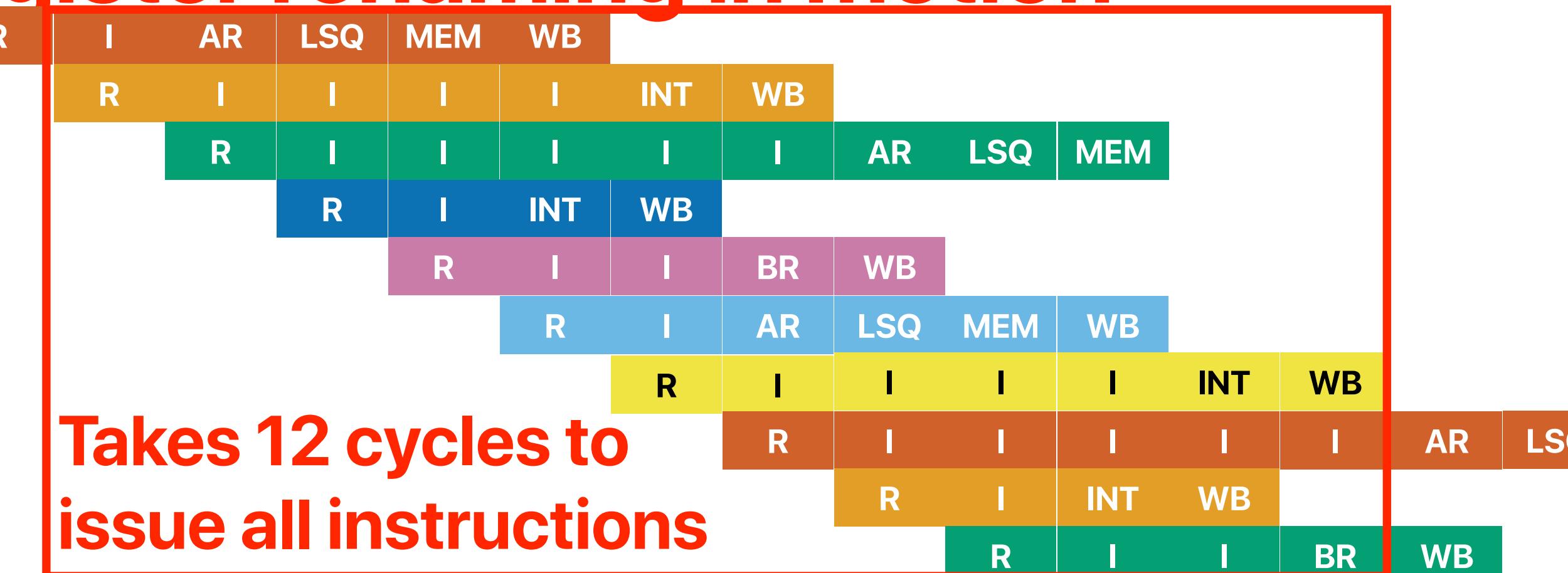
Renamed instruction		
1 ld P1, 0(X10)		
2 add P2, P1, X12		
3 sd P2, 0(X10)		
4 addi P3, X10, 8		
5 bne P3, X5, LOOP		
6 ld P4, 0(P3)		
7 add P5, P1, X12		
8 sd P5, 0(P3)		
9 addi P6, P3, 8		
10 bne P6, 0(X10)		

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		

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



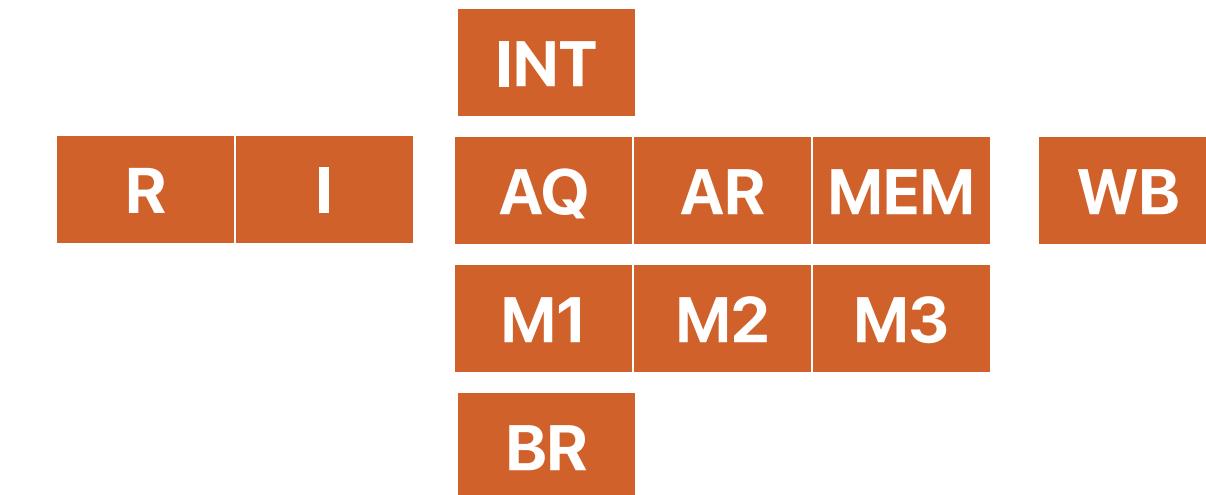
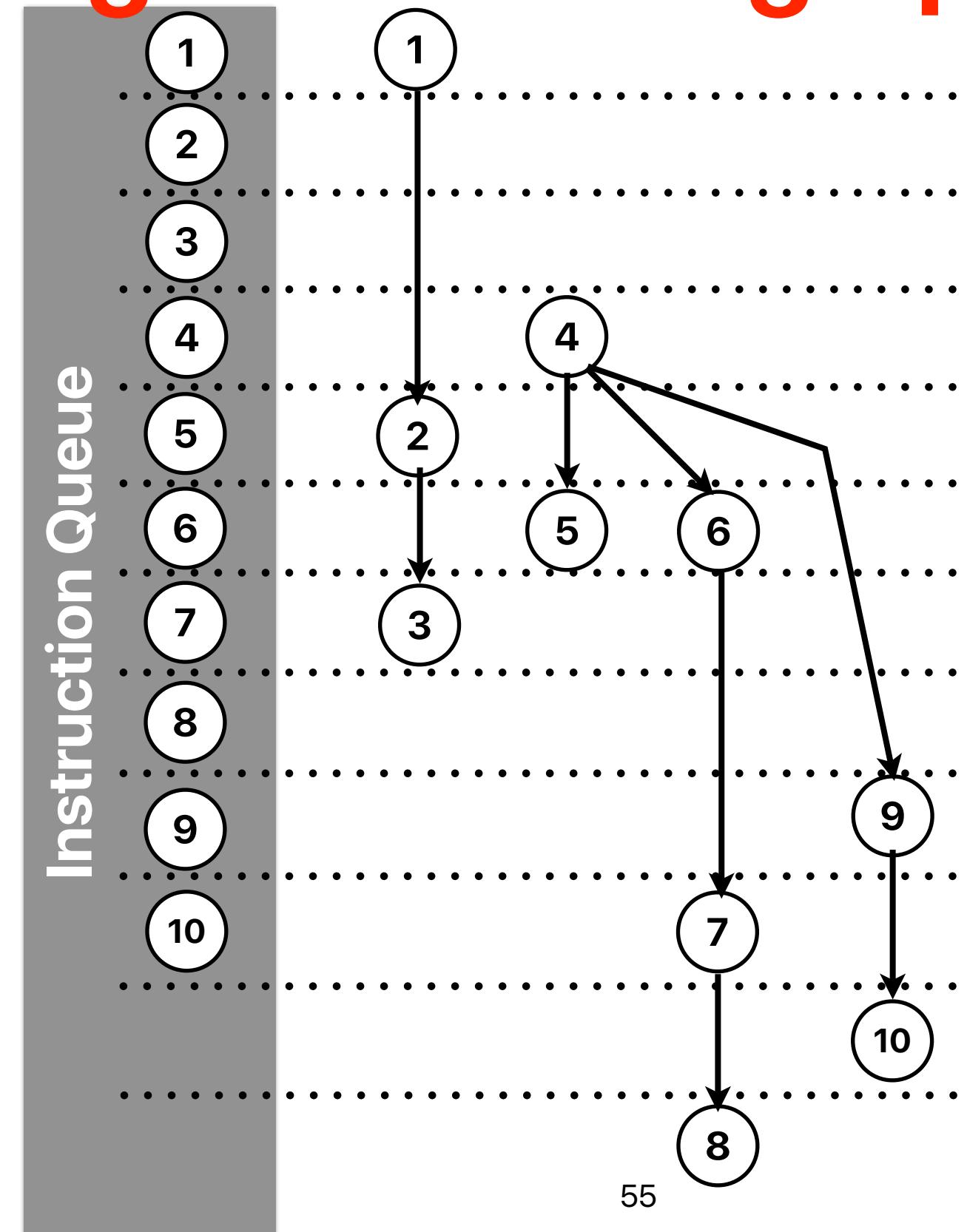
Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	P1
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			

Through data flow graph analysis

- 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



INT — 2 cycles for depending instruction to start
 MEM — 4 cycles for the depending instruction to start
 MUL/DIV — 4 cycles for the depending instruction to start
 BR — 2 cycles to resolve

What about “linked list”

- For the following C code and its translation in RISC-V, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor only fetches 1 instruction 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 )
```

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

- A. 9
- B. 10
- C. 11
- D. 12
- E. 13

What about “linked list”

- For the following C code and its translation in RISC-V, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor only fetches 1 instruction 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 )
```

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

- A. 9
- B. 10
- C. 11
- D. 12
- E. 13

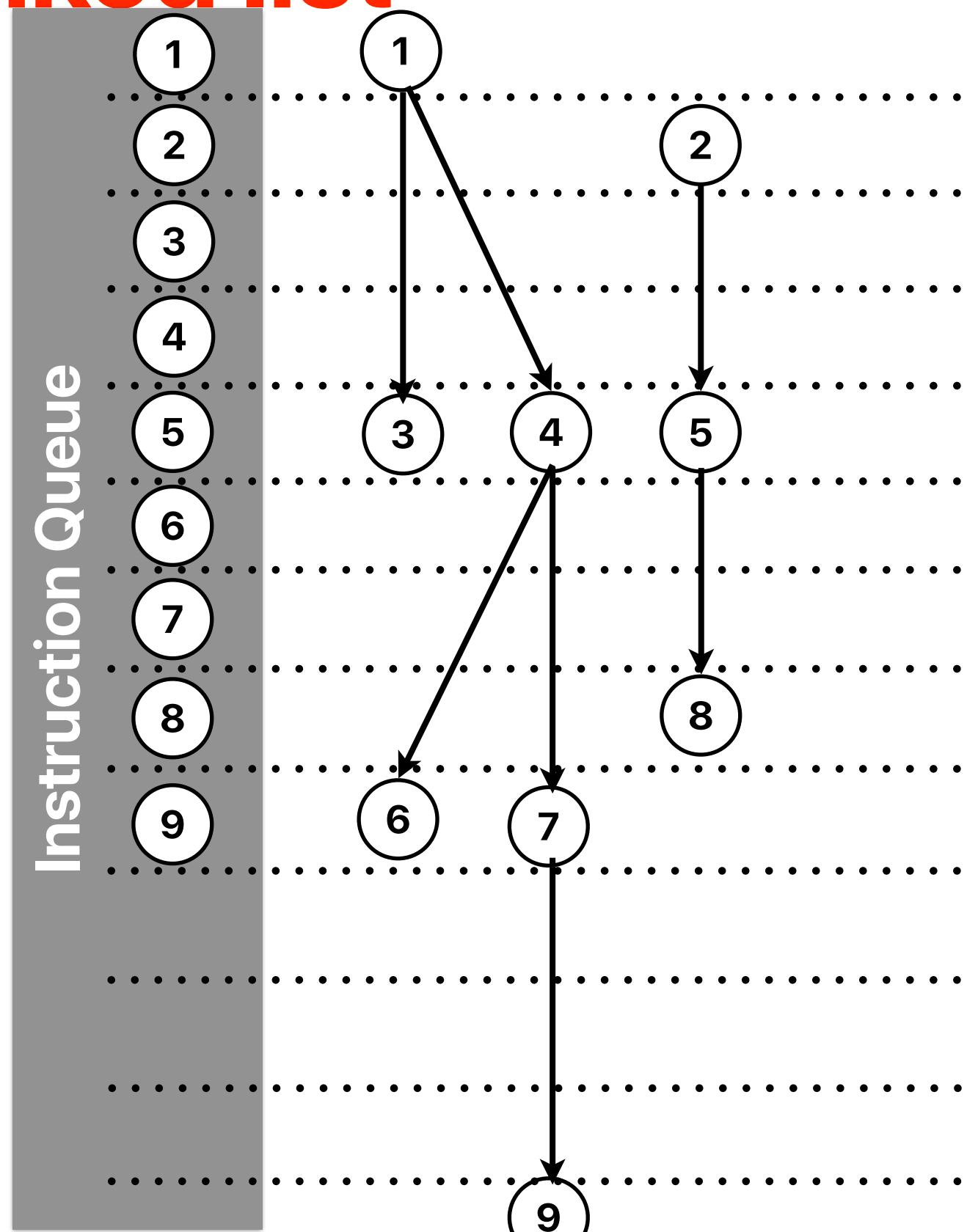
What about “linked list”

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



What about “linked list”

- For the following C code and its translation in RISC-V, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor only fetches 1 instruction 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 )
```

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

- A. 9
- B. 10
- C. 11
- D. 12
- E. 13

Register renaming in motion

- 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

R

Renamed instruction	
1	ld P1, 0(X10)
2	
3	
4	
5	
6	
7	
8	
9	
10	

Physical Register	
X5	
X6	P1
X7	
X10	
X12	

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

Register renaming in motion

- 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



Renamed instruction		
1	ld P1, 0(X10)	
2	add P2, X7, 1	
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		

Register renaming in motion

- 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



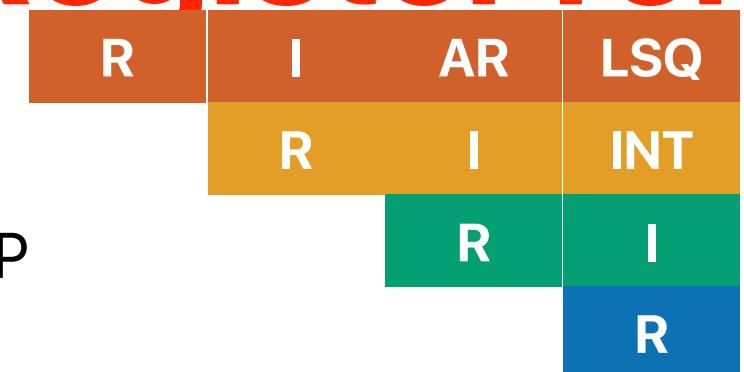
Renamed instruction		
1	ld P1, 0(X10)	
2	add P2, X7, 1	
3	bne P1, X0, LOOP	
4		
5		
6		
7		
8		
9		
10		

Physical Register	
X5	
X6	
X7	P2
X10	P1
X12	

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

Register renaming in motion

- 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



Renamed instruction		
1	ld P1, 0(X10)	
2	add P2, X7, 1	
3	bne P1, X0, LOOP	
4	ld P3, 0(P1)	
5		
6		
7		
8		
9		
10		

Physical Register	
X5	
X6	
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		

Register renaming in motion

①	ld	X10, 8(X10)	R	I	AR	LSQ	MEM
②	addi	X7, X7, 1	R	I	INT	WB	
③	bne	X10, X0, LOOP	R	I	I		
④	ld	X10, 8(X10)	R	R	I		
⑤	addi	X7, X7, 1			R		
⑥	bne	X10, X0, LOOP					
⑦	ld	X10, 8(X10)					
⑧	addi	X7, X7, 1					
⑨	bne	X10, X0, LOOP					

Renamed instruction		
1	ld	P1, 0(X10)
2	add	P2, X7, 1
3	bne	P1, X0, LOOP
4	ld	P3, 0(P1)
5	add	P4, P2, 1
6		
7		
8		
9		
10		

Physical Register	
X5	
X6	
X7	P4
X10	P3
X12	

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 in motion

			R	I	AR	LSQ	MEM	WB
①	ld	X10, 8(X10)	R					
②	addi	X7, X7, 1		R	I	INT	WB	
③	bne	X10, X0, LOOP			R	I	I	I
④	ld	X10, 8(X10)			R	I	I	
⑤	addi	X7, X7, 1				R	I	
⑥	bne	X10, X0, LOOP					R	
⑦	ld	X10, 8(X10)						
⑧	addi	X7, X7, 1						
⑨	bne	X10, X0, LOOP						

	Renamed instruction
1	ld P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	ld P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	
8	
9	
10	

	Physical Register
X5	
X6	
X7	P4
X10	P3
X12	

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

Register renaming in motion

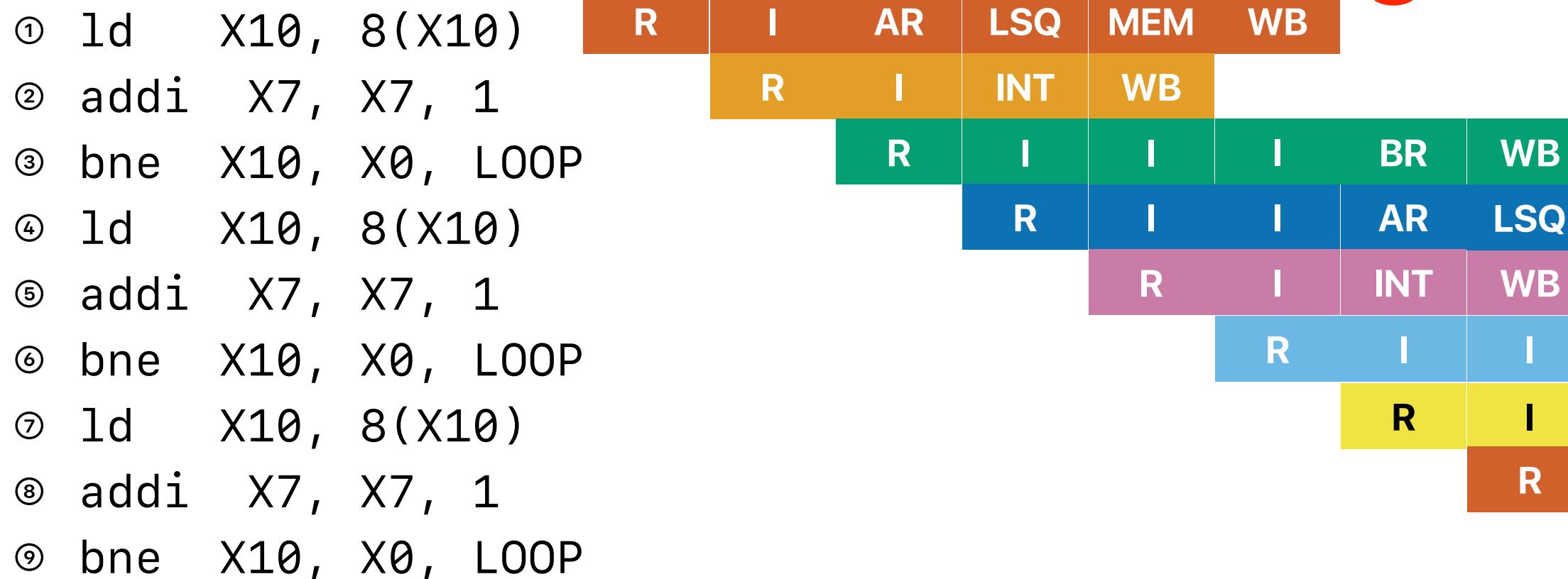
		R	I	AR	LSQ	MEM	WB
①	ld X10, 8(X10)	R					
②	addi X7, X7, 1		R	I	INT	WB	
③	bne X10, X0, LOOP			R	I	I	BR
④	ld X10, 8(X10)			R	I	I	AR
⑤	addi X7, X7, 1				R	I	INT
⑥	bne X10, X0, LOOP					R	I
⑦	ld X10, 8(X10)						R
⑧	addi X7, X7, 1						
⑨	bne X10, X0, LOOP						

	Renamed instruction
1	ld P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	ld P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	ld P5, 0(P3)
8	
9	
10	

	Physical Register
X5	
X6	
X7	P4
X10	P5
X12	

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

Register renaming in motion

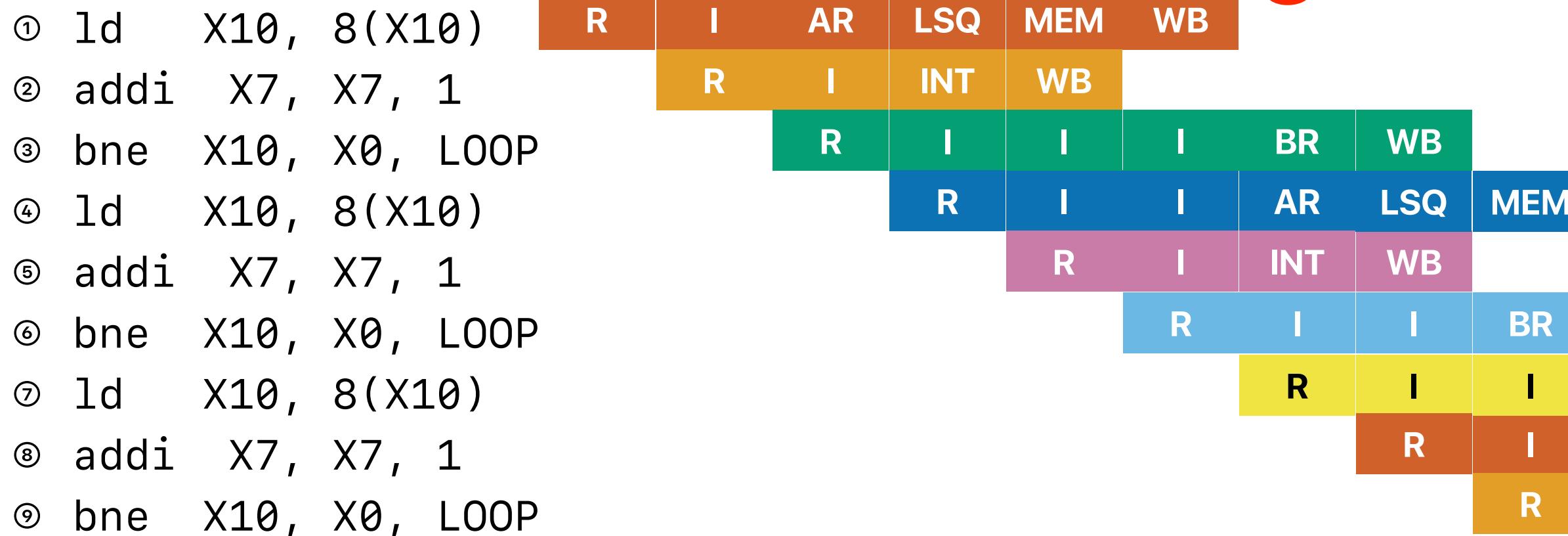


	Renamed instruction
1	ld P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	ld P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	ld P5, 0(P3)
8	add P6, P4, 1
9	
10	

	Physical Register
X5	
X6	
X7	P6
X10	P5
X12	

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

Register renaming in motion

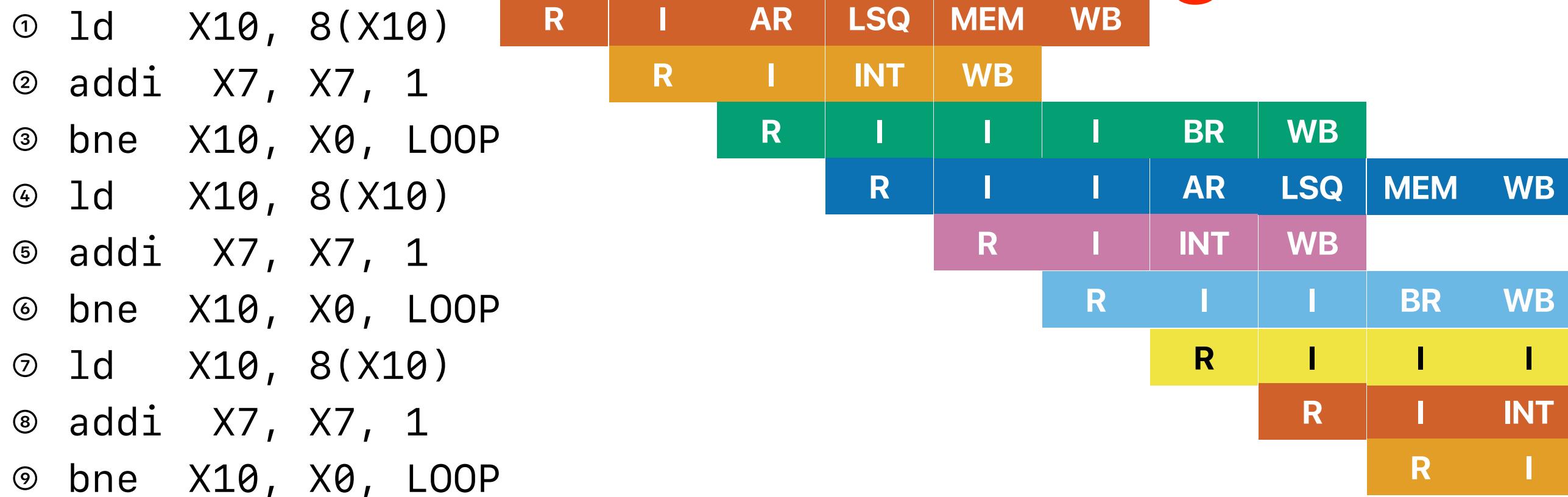


	Renamed instruction
1	1d P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	1d P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	1d P5, 0(P3)
8	add P6, P4, 1
9	bne P5, X0, LOOP
10	

	Physical Register
X5	
X6	
X7	P6
X10	P5
X12	

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

Register renaming in motion

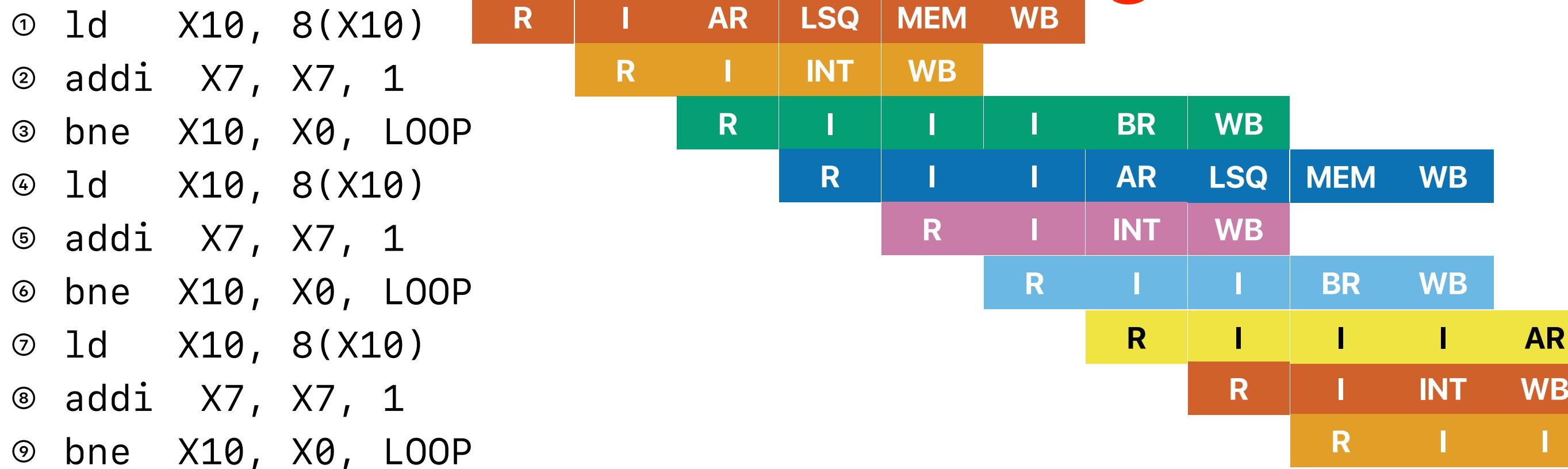


Renamed instruction		
1	ld P1, 0(X10)	
2	add P2, X7, 1	
3	bne P1, X0, LOOP	
4	ld P3, 0(P1)	
5	add P4, P2, 1	
6	bne P3, X0, LOOP	
7	ld P5, 0(P3)	
8	add P6, P4, 1	
9	bne P5, X0, LOOP	
10		

Physical Register	
X5	
X6	
X7	P6
X10	P5
X12	

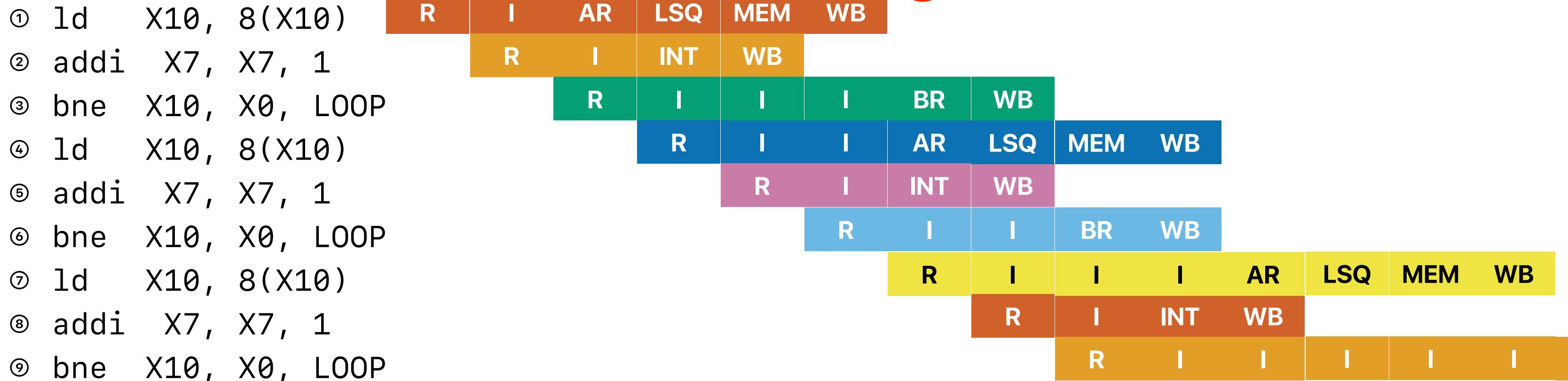
Valid Value In use			Valid Value In use		
P1	1	1	P6	0	1
P2	1	1	P7		
P3	0	1	P8		
P4	0	1	P9		
P5	0	1	P10		

Register renaming in motion



	Renamed instruction	Physical Register	Valid	Value	In use	Valid	Value	In use
1	1d P1, 0(X10)	X5	P1	1	1	P6	0	1
2	add P2, X7, 1	X6	P2	1	1	P7		
3	bne P1, X0, LOOP	X7	P6			P8		
4	1d P3, 0(P1)	X10	P5			P9		
5	add P4, P2, 1	X12				P10		
6	bne P3, X0, LOOP							
7	1d P5, 0(P3)							
8	add P6, P4, 1							
9	bne P5, X0, LOOP							
10								

Register renaming in motion



	Renamed instruction
1	ld P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	ld P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	ld P5, 0(P3)
8	add P6, P4, 1
9	bne P5, X0, LOOP
10	

	Physical Register
X5	
X6	
X7	P6
X10	P5
X12	

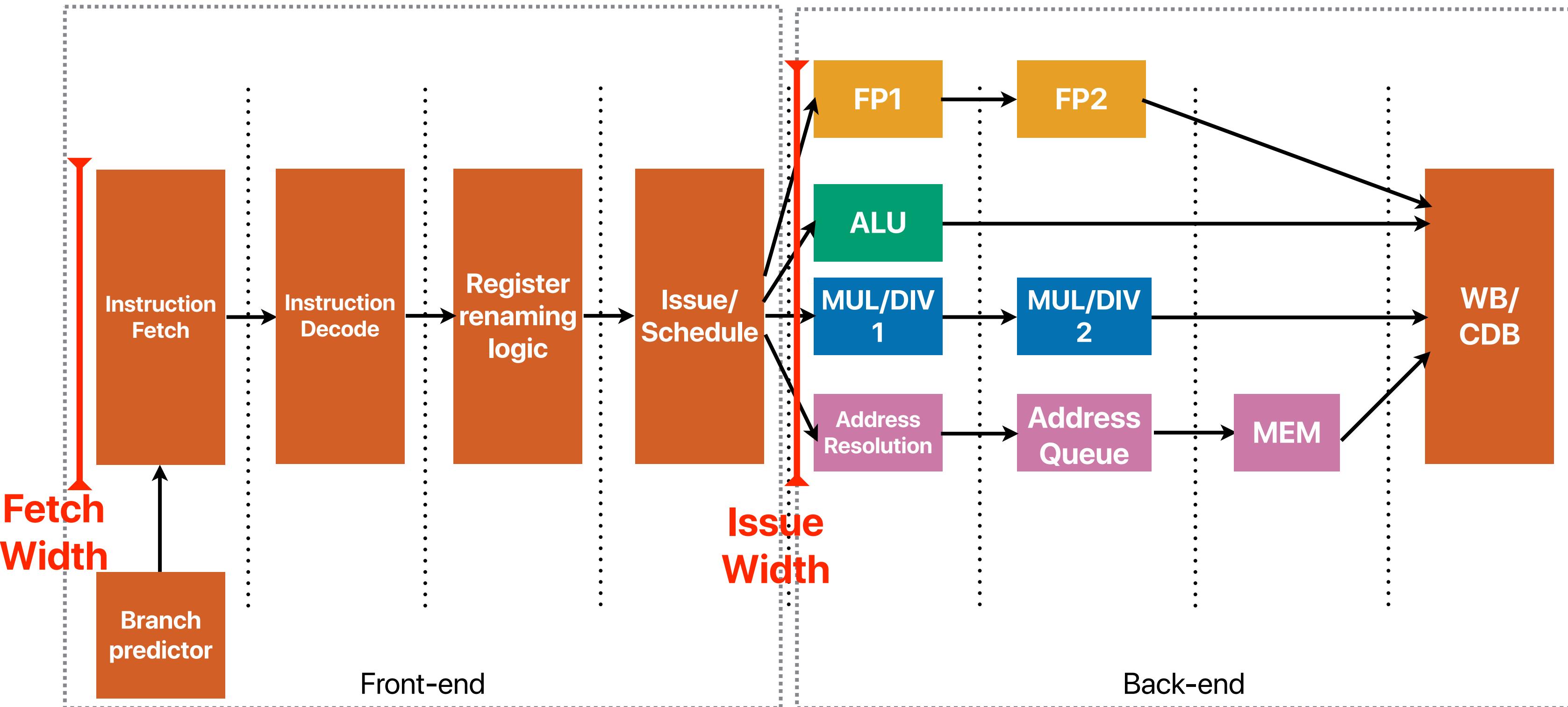
	Valid	Value	In use	Valid	Value	In use
P1	1		1	P6	0	1
P2	1		1	P7		
P3	0		1	P8		
P4	0		1	P9		
P5	0		1	P10		

Super Scalar

Superscalar

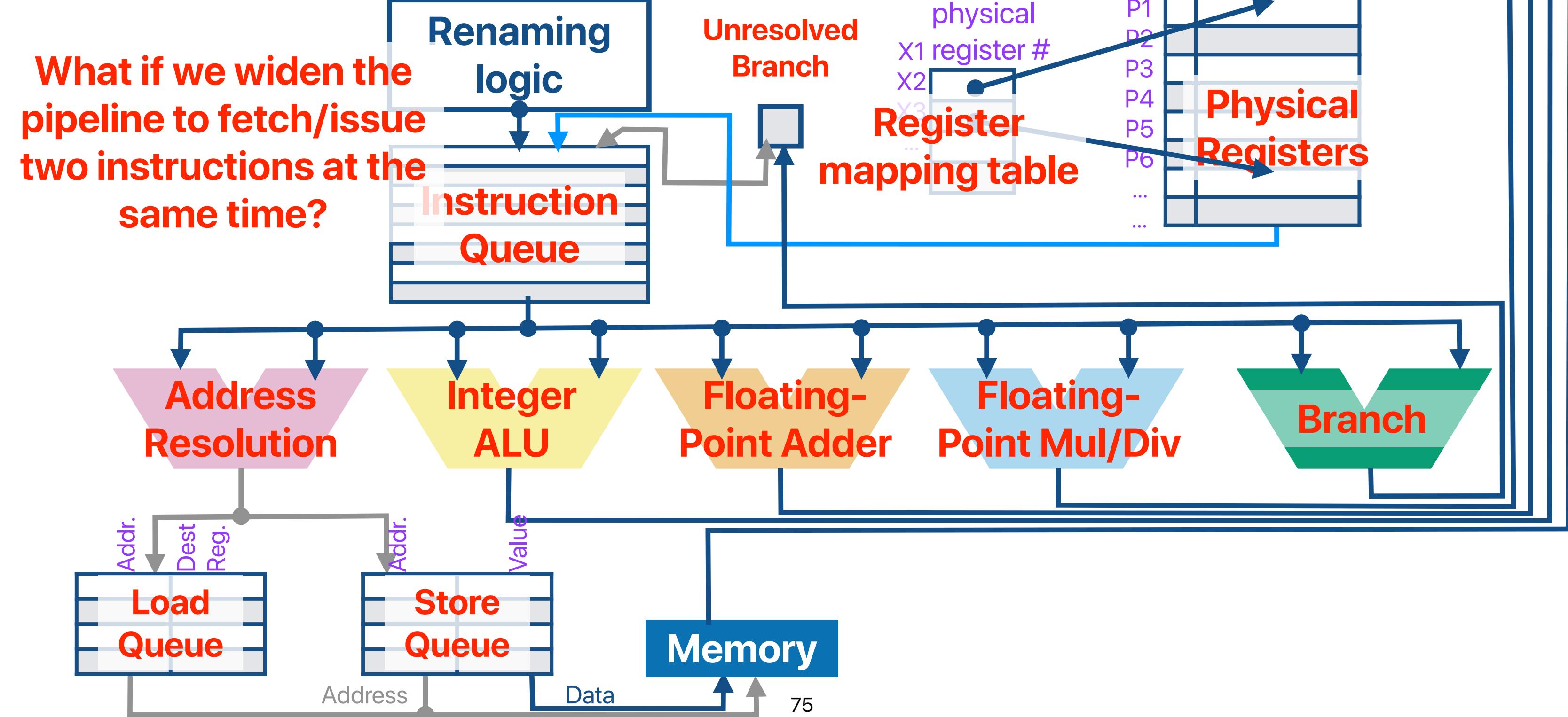
- Since we have more functional units now, we should fetch/decode more instructions each cycle so that we can have more instructions to issue!
- Super-scalar: fetch/decode/issue more than one instruction each cycle
 - Fetch width: how many instructions can the processor fetch/decode each cycle
 - Issue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

Super Scalar Pipeline



Overview of a processor supporting register renaming

Fetch/decode instruction →



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 | |

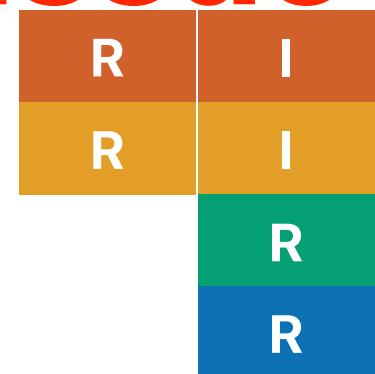
Renamed instruction		
1	ld	P1, 0(X10)
2	add	P2, P1, X12
3		
4		
5		
6		
7		
8		
9		
10		

Physical Register	
X5	
X6	P1
X7	P2
X10	
X12	

Valid			Valid		
Value			Value		
In use			In use		
P1	0	1	P6		
P2	0	1	P7		
P3			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



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

	Physical Register
1	X5
2	X6
3	X7
4	X10
5	X12

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

2-issue RR processor in motion

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

Renamed instruction		Physical Register		Valid			Valid		
		X5	X6	Value	In use	P1	Value	In use	
1	ld P1, 0(X10)			0	1	P6			
2	add P2, P1, X12	X5		0	1	P7			
3	sd P2, 0(X10)	X6	P1	0	1	P8			
4	addi P3, X10, 8	X7	P2	0	1	P9			
5	bne P3, X5, LOOP	X10	P3	0	1	P10			
6	ld P4, 0(P3)	X12							
7									
8									
9									
10									

2-issue RR processor in motion

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	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)
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		
10		

Physical Register	
X5	
X6	P1
X7	P5
X10	P3
X12	

Valid			Valid		
Value			Value		
In use			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

			R	I	AR	AQ	MEM
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	I
④	addi	X10, X10, 8	R	I	INT	WB	
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)
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	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

			R	I	AR	AQ	MEM	WB
①	ld	X6, 0(X10)	R	I				
②	add	X7, X6, X12	R	I	I	I	I	I
③	sd	X7, 0(X10)	R	I	I	I	I	I
④	addi	X10, X10, 8	R	I	INT	WB		
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	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)

	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

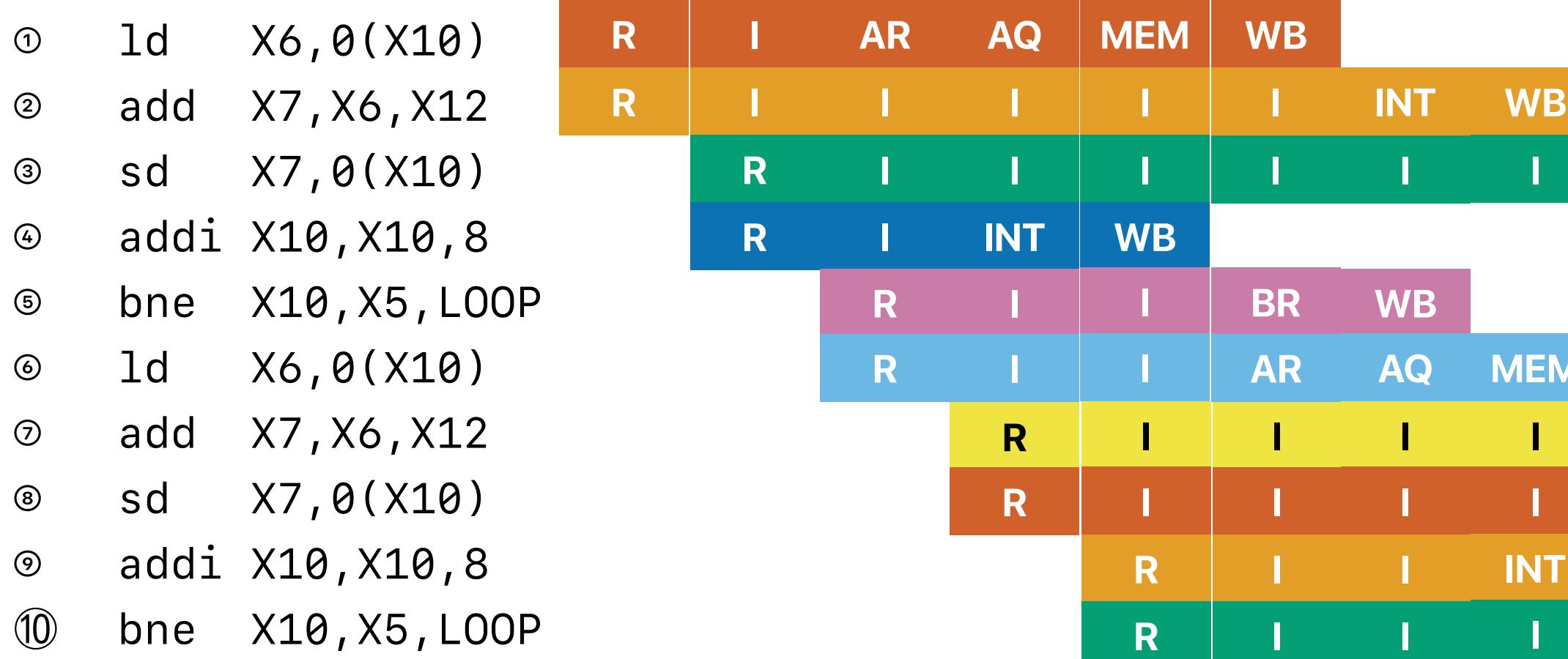
	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	0		1	P7			
P3	1		1	P8			
P4	0		1	P9			
P5	0		1	P10			

2-issue RR processor in motion

			R	I	AR	AQ	MEM	WB	
①	ld	X6, 0(X10)	R	I					
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		WB		
⑤	bne	X10, X5, LOOP		R	I	I	BR	WB	
⑥	ld	X6, 0(X10)		R	I	I	AR	AQ	
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			
⑩	bne	X10, X5, LOOP		R	I	I			

Renamed instruction		Physical Register		Valid	Value	In use	Valid	Value	In use
1	ld P1, 0(X10)	X5		P1	1	1	P6		
2	add P2, P1, X12	X6	P1	P2	0	1	P7		
3	sd P2, 0(X10)	X7	P5	P3	1	1	P8		
4	addi P3, X10, 8	X10	P3	P4	0	1	P9		
5	bne P3, X5, LOOP	X12		P5	0	1	P10		
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)								

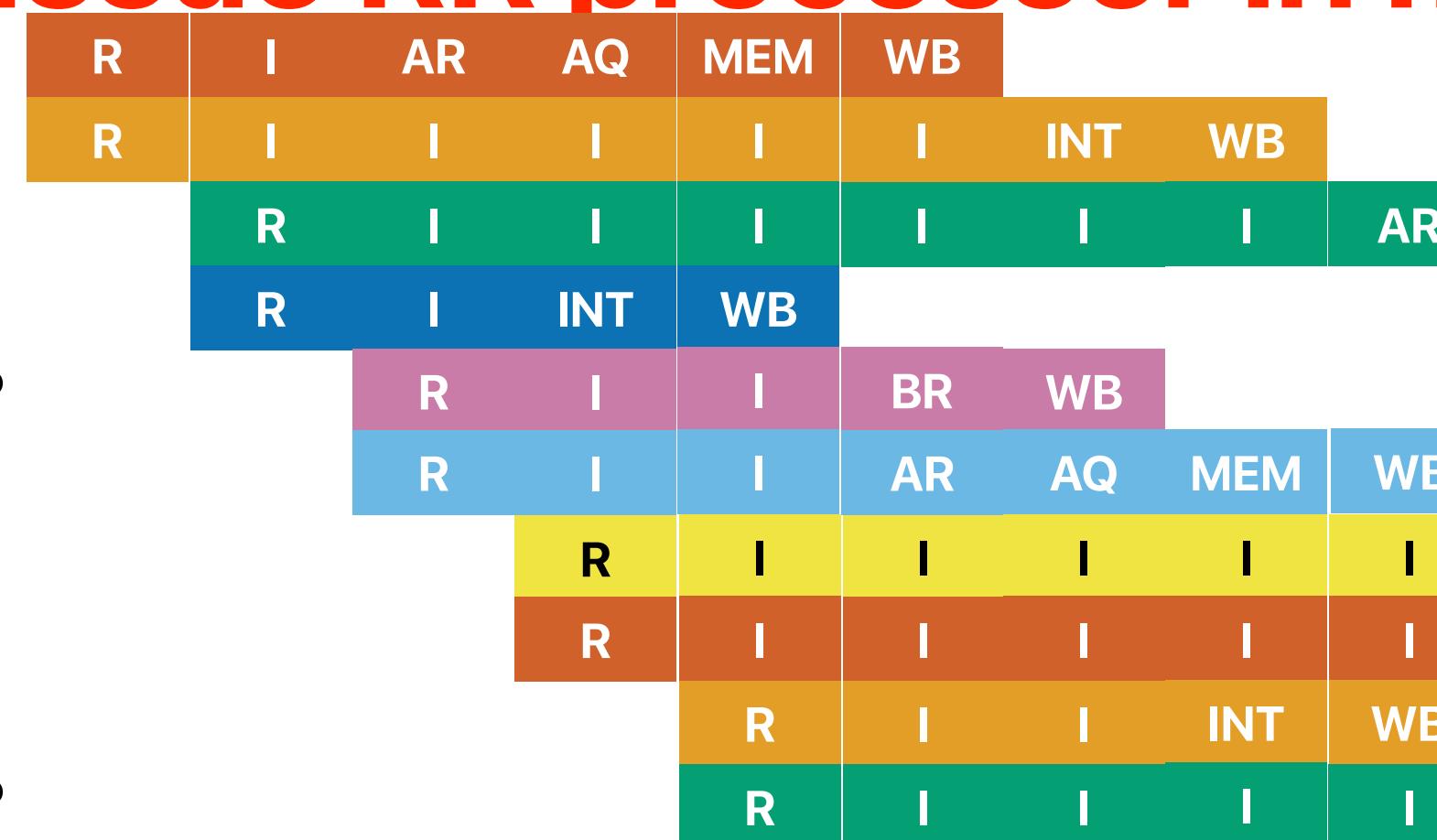
2-issue RR processor in motion



Renamed instruction		Physical Register		Valid Value In use			Valid Value In use		
		X5	X6	P1	1	1	P6		
1	1d P1, 0(X10)			P2	1	1	P7		
2	add P2, P1, X12			P3	1	1	P8		
3	sd P2, 0(X10)			P4	0	1	P9		
4	addi P3, X10, 8			P5	0	1	P10		
5	bne P3, X5, LOOP								
6	1d P4, 0(P3)								
7	add P5, P1, X12								
8	sd P5, 0(P3)								
9	addi P6, P3, 8								
10	bne P6, 0(X10)								

2-issue RR processor in motion

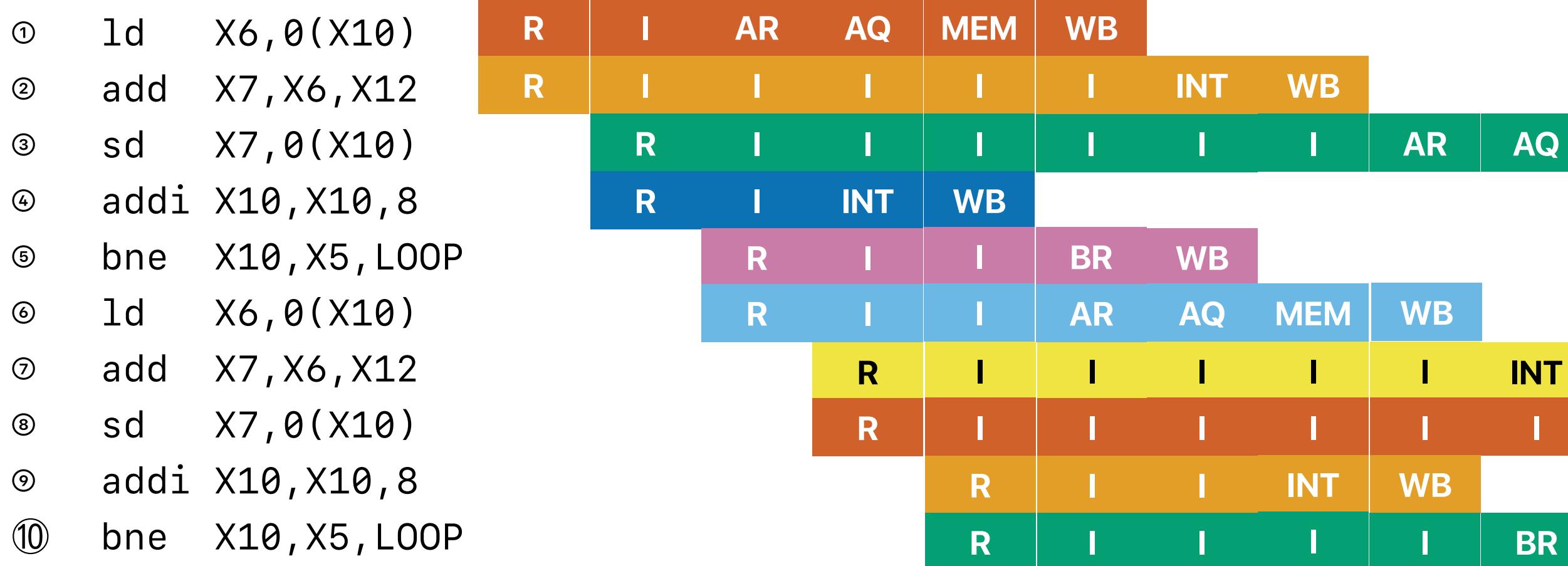
- ① ld X6, 0(X10)
- ② add X7, X6, X12
- ③ sd X7, 0(X10)
- ④ addi X10, X10, 8
- ⑤ bne X10, X5, LOOP
- ⑥ ld X6, 0(X10)
- ⑦ add X7, X6, X12
- ⑧ sd X7, 0(X10)
- ⑨ addi X10, X10, 8
- ⑩ bne X10, X5, LOOP



	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)

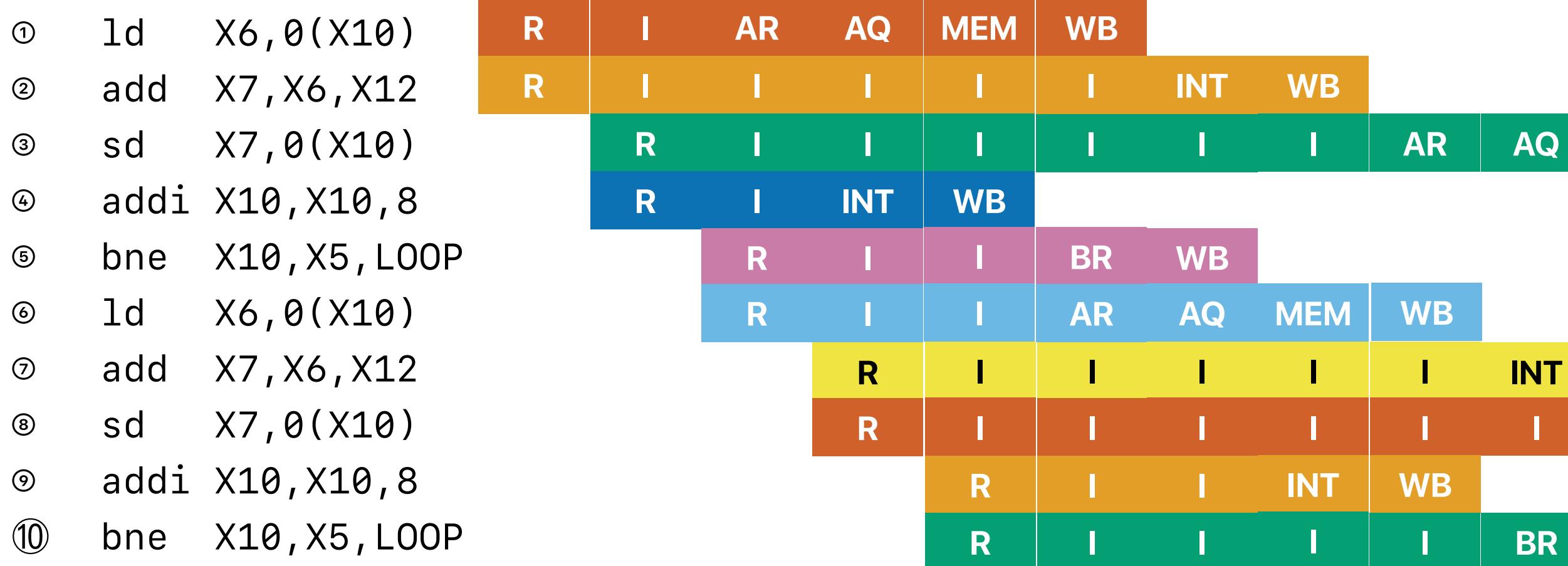
	Physical Register	Valid			Valid		
		Value	In use		Value	In use	
	X5			P1	1	1	P6
	X6	P1		P2	1	1	P7
	X7	P5		P3	1	1	P8
	X10	P3		P4	1	1	P9
	X12			P5	0	1	P10

2-issue RR processor in motion



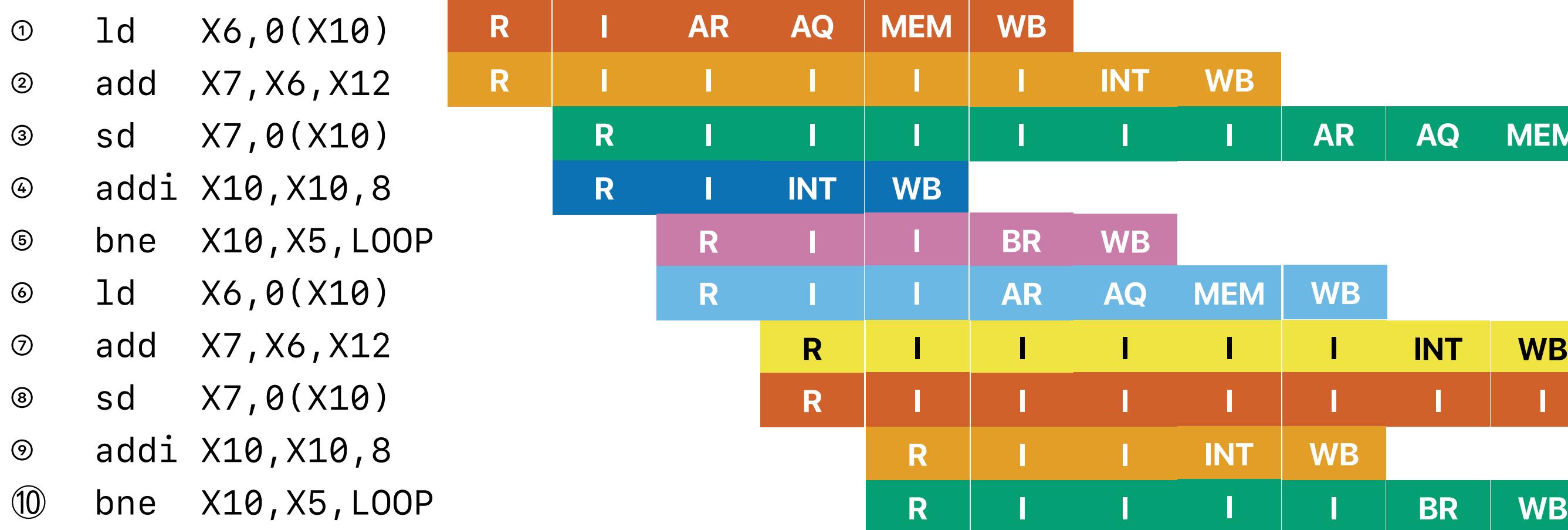
Renamed instruction		Physical Register		Valid Value In use			Valid Value In use		
		X5		P1	1	1	P6		
1	ld P1, 0(X10)	X5		P1	1	1	P6		
2	add P2, P1, X12	X6	P1	P2	1	1	P7		
3	sd P2, 0(X10)	X7	P5	P3	1	1	P8		
4	addi P3, X10, 8	X10	P3	P4	1	1	P9		
5	bne P3, X5, LOOP	X12		P5	0	1	P10		
6	ld P4, 0(P3)								
7	add P5, P1, X12								
8	sd P5, 0(P3)								
9	addi P6, P3, 8								
10	bne P6, 0(X10)								

2-issue RR processor in motion



Renamed instruction		Physical Register		Valid Value In use			Valid Value In use		
		X5		P1	1	1	P6		
1	ld P1, 0(X10)	X5		P1	1	1	P6		
2	add P2, P1, X12	X6	P1	P2	1	1	P7		
3	sd P2, 0(X10)	X7	P5	P3	1	1	P8		
4	addi P3, X10, 8	X10	P3	P4	1	1	P9		
5	bne P3, X5, LOOP	X12		P5	0	1	P10		
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)								

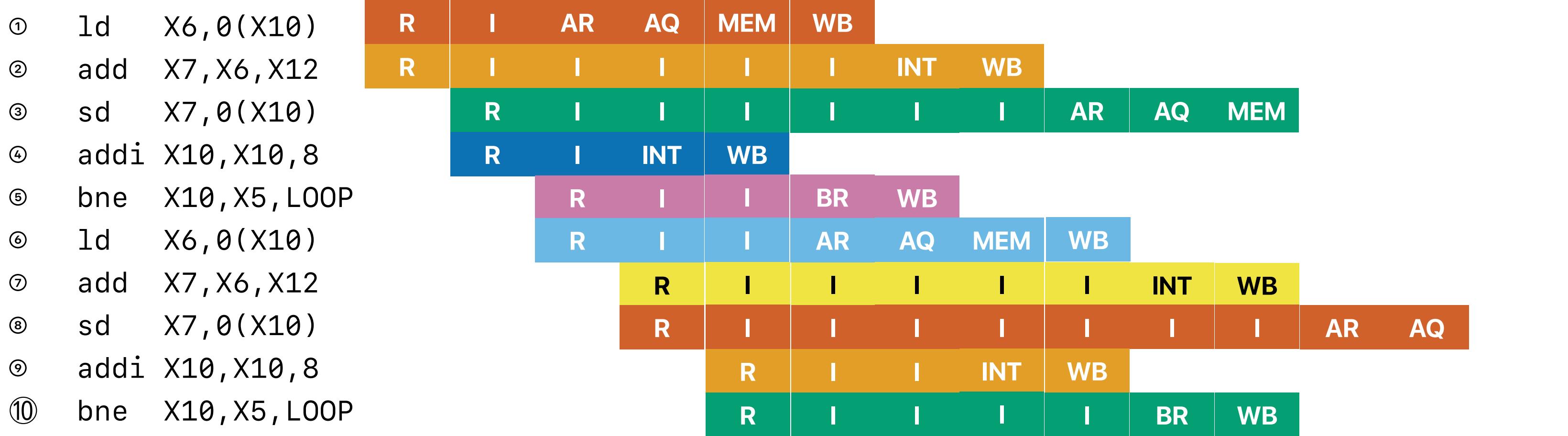
2-issue RR processor in motion



Renamed instruction		Physical Register		Valid Value In use			Valid Value In use		
		X5		P1	1	1	P6		
1	1d P1, 0(X10)	X5		P1	1	1	P6		
2	add P2, P1, X12	X6	P1	P2	1	1	P7		
3	sd P2, 0(X10)	X7	P5	P3	1	1	P8		
4	addi P3, X10, 8	X10	P3	P4	1	1	P9		
5	bne P3, X5, LOOP	X12		P5	1	1	P10		
6	1d P4, 0(P3)								
7	add P5, P1, X12								
8	sd P5, 0(P3)								
9	addi P6, P3, 8								
10	bne P6, 0(X10)								

2-issue RR processor in motion

2-issue RR processor in motion



Renamed instruction		Physical Register		Valid Value In use			Valid Value In use		
		X5		P1	1	1	P6		
1	1d P1, 0(X10)	X5		P1	1	1	P6		
2	add P2, P1, X12	X6	P1	P2	1	1	P7		
3	sd P2, 0(X10)	X7	P5	P3	1	1	P8		
4	addi P3, X10, 8	X10	P3	P4	1	1	P9		
5	bne P3, X5, LOOP	X12		P5	1	1	P10		
6	1d P4, 0(P3)								
7	add P5, P1, X12								
8	sd P5, 0(P3)								
9	addi P6, P3, 8								
10	bne P6, 0(X10)								

2-issue RR processor in motion

		R	I	AR	AQ	MEM	WB				
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	I	I	I	I	I	AR	AQ MEM
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	I	I	I	I	AR AQ MEM WB
⑨	addi	X10, X10, 8		R	I	I	INT	WB			
⑩	bne	X10, X5, LOOP		R	I	I	I	I	I	BR	WB

Renamed instruction	Physical Register	Valid			Valid	Value	In use
		1	2	3			
1 ld P1, 0(X10)	X5				P1	1	1
2 add P2, P1, X12	X6	P1			P2	1	1
3 sd P2, 0(X10)	X7	P5			P3	1	1
4 addi P3, X10, 8	X10	P3			P4	1	1
5 bne P3, X5, LOOP	X12				P5	1	1
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)							

What about “linked list”

- For the following C code and its translation in RISC-V, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor can fetch 2 instruction 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 )
```

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

- A. 9
- B. 10
- C. 11
- D. 12
- E. 13

What about “linked list”

- For the following C code and its translation in RISC-V, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor can fetch 2 instruction 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 )
```

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

- A. 9
- B. 10
- C. 11
- D. 12
- E. 13

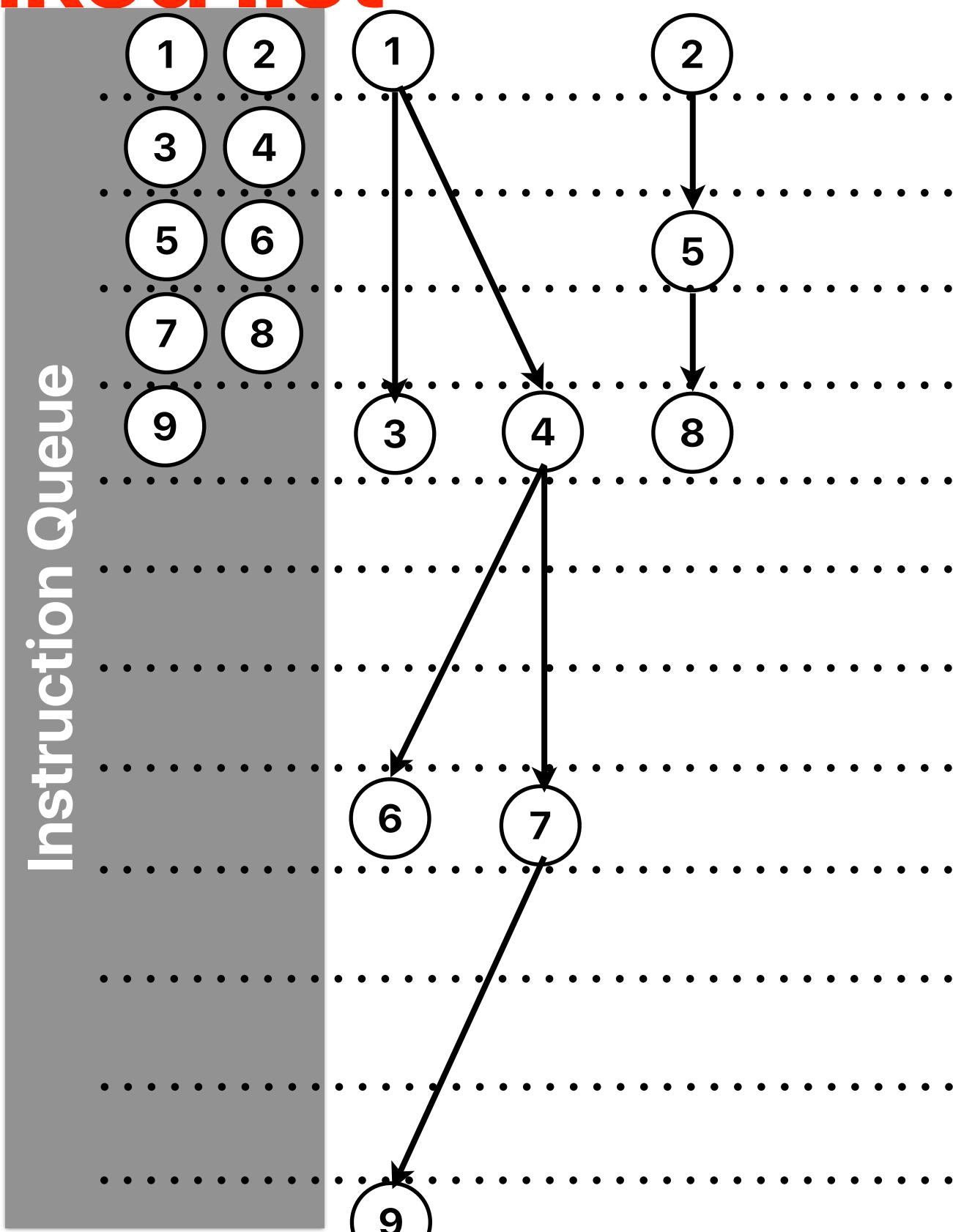
What about “linked list”

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



What about “linked list”

- For the following C code and its translation in RISC-V, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor can fetch 2 instruction 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 )
```

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

- A. 9
- B. 10
- C. 11
- D. 12
- E. 13

SuperScalar does not help!!!

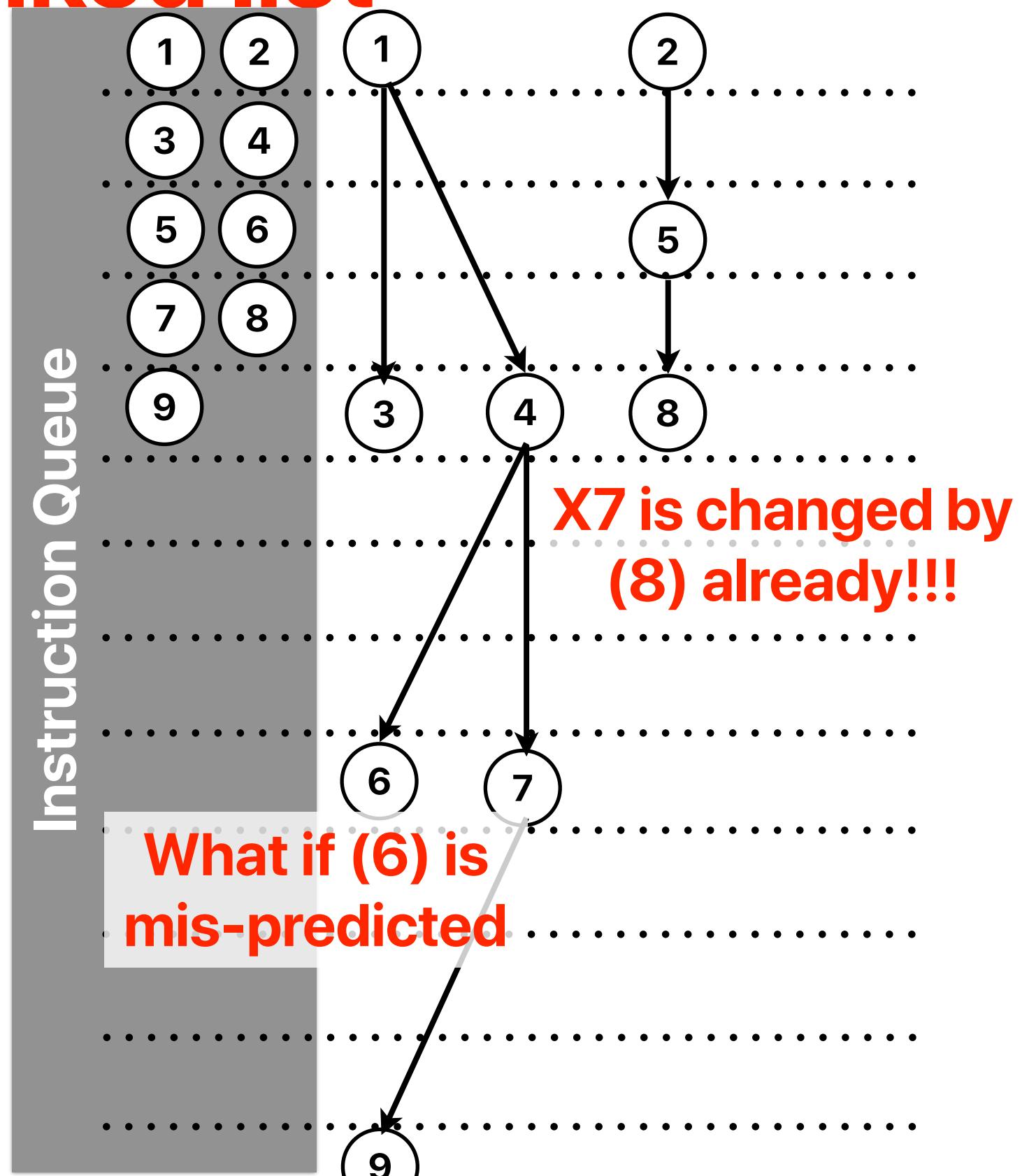
What about “linked list”

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



Announcement

- Project is up — check the website
- Assignment #4 is up — start EARLY!!!
- 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
- Regarding projected grades
 - Based on your “weighted total” column in iLearn — we only have 50% offered so far, that’s why the max is only 48 now
 - Our final grading is based on “relative ranking” and scale may change

Computer Science & Engineering

203

つづく

