

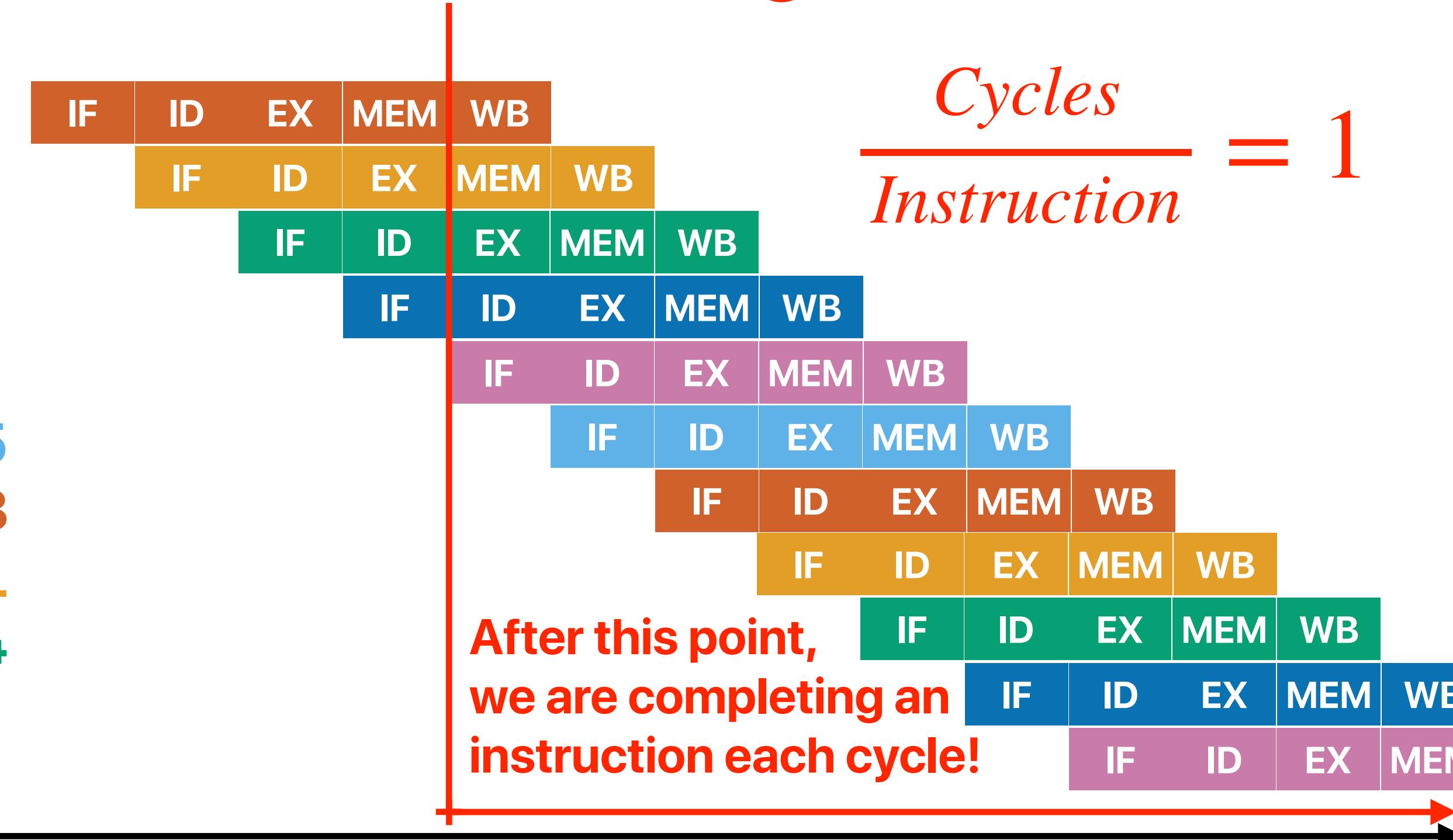
Dynamic Instruction Scheduling

(II)

Hung-Wei Tseng

Recap: Pipelining

add x1, x2, x3
 ld x4, 0(x5)
 sub x6, x7, x8
 sub x9, x10, x11
 sd x1, 0(x12)
 xor x13, x14, x15
 and x16, x17, x18
 add x19, x20, x21
 sub x22, x23, x24
 ld x25, 4(x26)
 sd x27, 0(x28)



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

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 Scheduling**

Outline

- The concept of dynamic instruction schedule
- Tomasulo Algorithm
- Register renaming

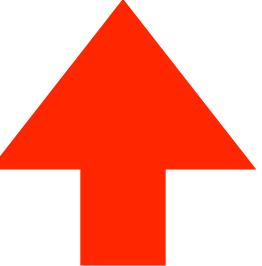
Dynamic instruction scheduling/ Out-of-order (OoO) execution

Recap: Tips of drawing a pipeline diagram

- Each instruction has to go through all 5 pipeline stages: IF, ID, EXE, MEM, WB in order — only valid if it's single-issue, RISC-V 5-stage pipeline
- An instruction can enter the next pipeline stage in the next cycle if
 - No other instruction is occupying the next stage
 - This instruction has completed its own work in the current stage
 - The next stage has all its inputs ready
- Fetch a new instruction only if
 - We know the next PC to fetch
 - We can predict the next PC
 - Flush an instruction if the branch resolution says it's mis-predicted.

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

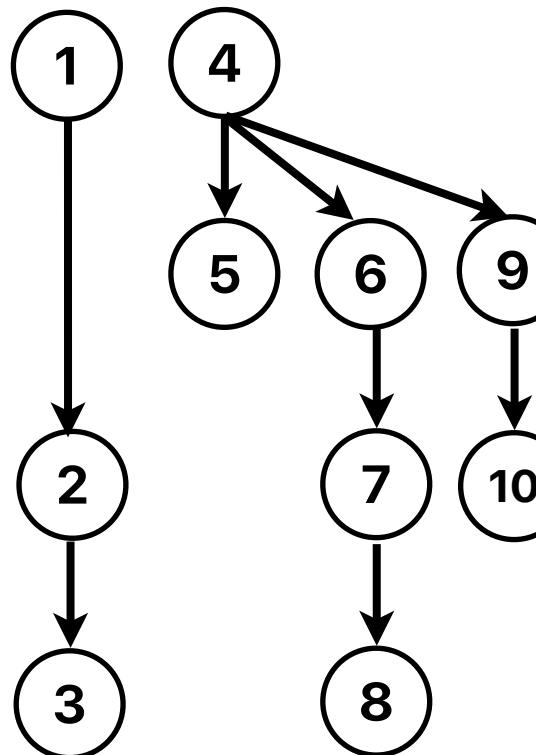


- This instruction has completed its own work in the current stage
- No other instruction is occupying the next stage
- The next stage has all its inputs ready

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

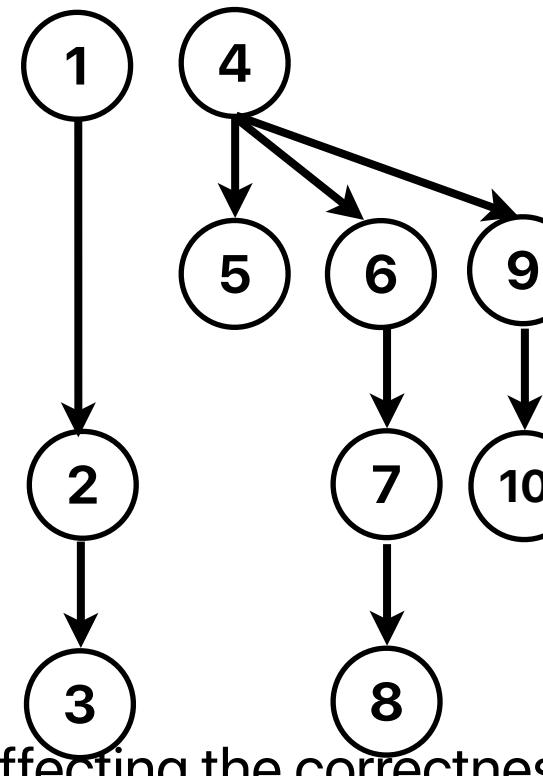
If we can predict the future ...

- Consider the following dynamic instructions:

- 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

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

- A. (2) and (4)
- B. (3) and (5)
- C. (5) and (6)
- D. (6) and (9)
- E. (9) and (10)

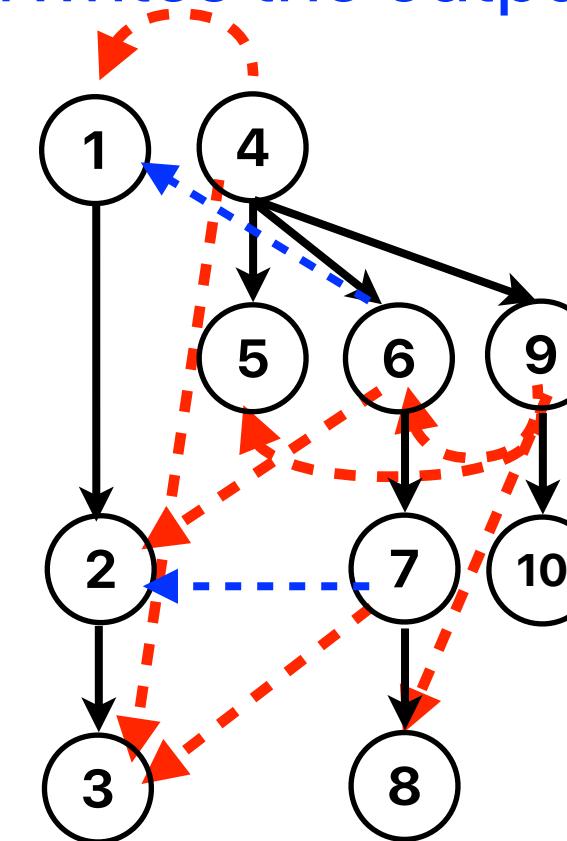


We still can only reorder (5) and (6)
even though (2) & (4) are not
depending on each other!

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



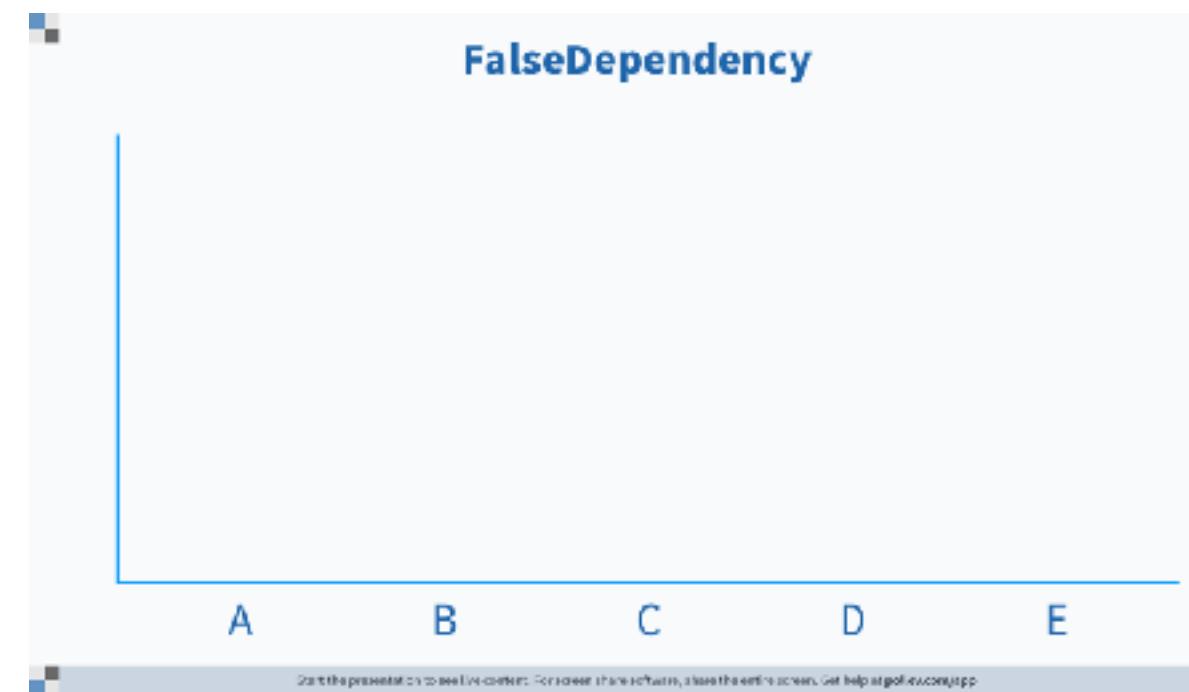
False dependencies

- Consider the following dynamic instructions

- ① ld X12, 0(X20)
- ② add X12, X10, X12
- ③ sub X18, X12, X10
- ④ ld X12, 8(X20)
- ⑤ add X14, X18, X12
- ⑥ add X18, X14, X14
- ⑦ sd X14, 16(X20)
- ⑧ addi X20, X20, 8

which of the following pair is not a “false dependency”

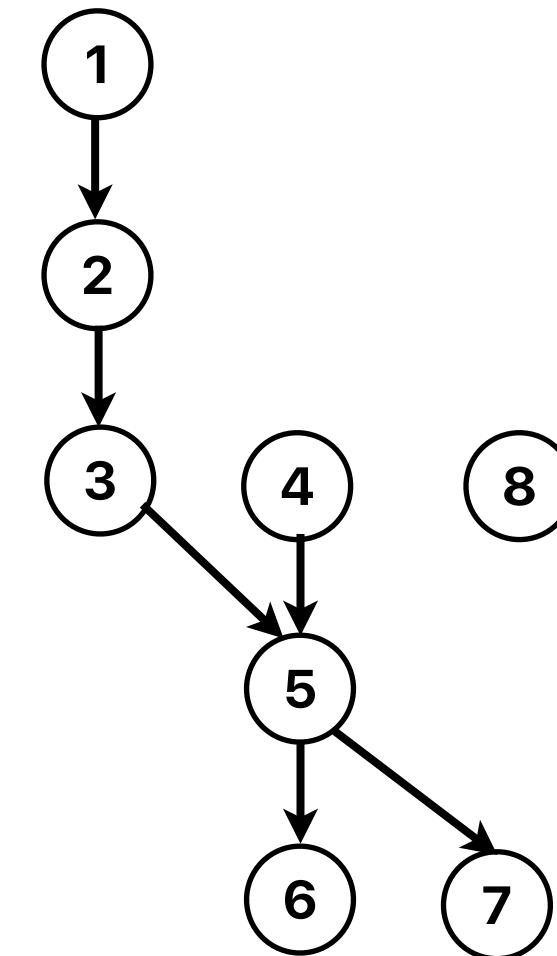
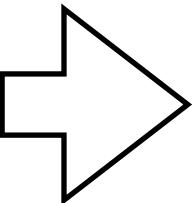
- A. (1) and (4)
- B. (1) and (8)
- C. (5) and (7)
- D. (4) and (8)
- E. (7) and (8)



False dependencies

- Consider the following dynamic instructions

- ① ld X12, 0(X20)
- ② add X12, X10, X12
- ③ sub X18, X12, X10
- ④ ld X12, 8(X20)
- ⑤ add X14, X18, X12
- ⑥ add X18, X14, X14
- ⑦ sd X14, 16(X20)
- ⑧ addi X20, X20, 8



which of the following pair is not a “false dependency”

- | | |
|----------------|------------------------------|
| A. (1) and (4) | WAW |
| B. (1) and (8) | WAR |
| C. (5) and (7) | True dependency (RAW) |
| D. (4) and (8) | WAR |
| E. (7) and (8) | WAR |

Out-of-order execution

- Any sequence of instructions has set of RAW, WAW, and WAR hazards that constrain its execution.
- Can we design a processor that extracts as much parallelism as possible, while still respecting these dependences?



Tomasulo's Algorithm

IBM 360 Model 91

- The most powerful commercialized machine in 1968

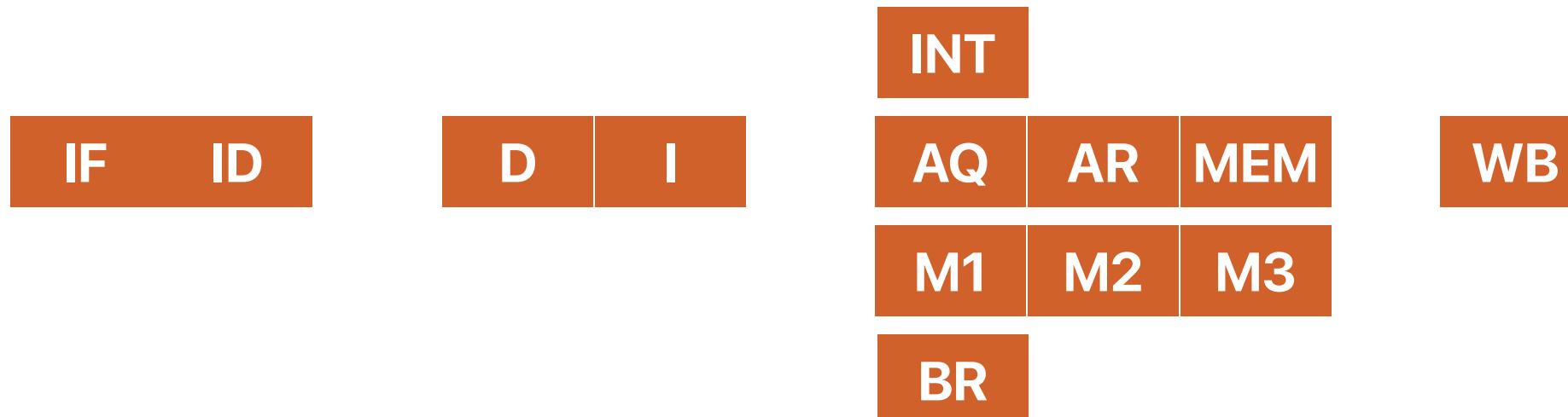


CDC 6600

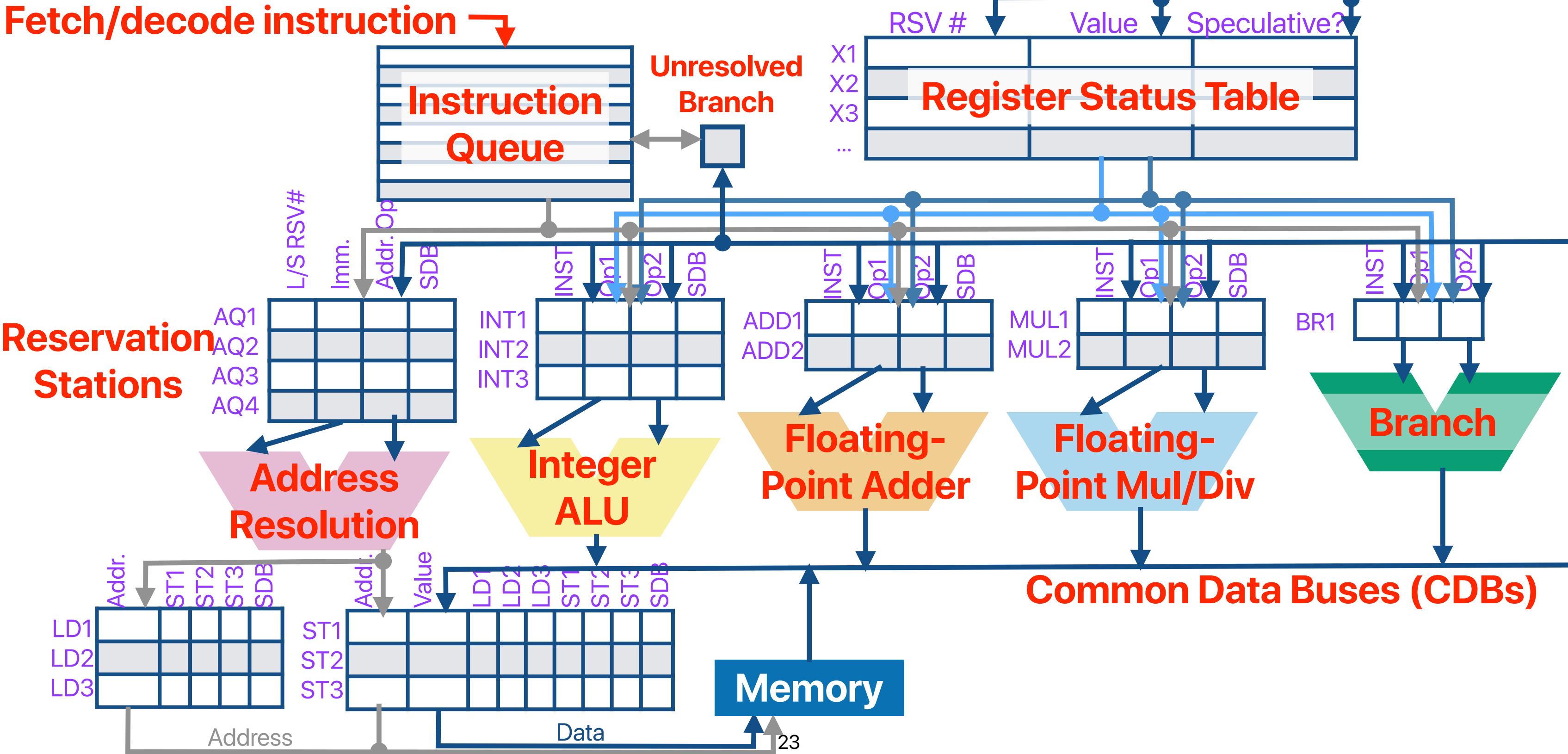
- The first machine with out-of-order execution.
- With performance of up to 3 "M" FLOPS
- It uses score-boarding to achieve so.
 - Instruction storage added to each functional execution unit
 - Instructions issue to FU when no structural hazards, begin execution when dependences satisfied. Thus, instructions issued to different FUs can execute out of order.
 - "scoreboard" tracks RAW, WAR, WAW hazards, tells each instruction when to proceed.
 - No forwarding
 - No register renaming

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

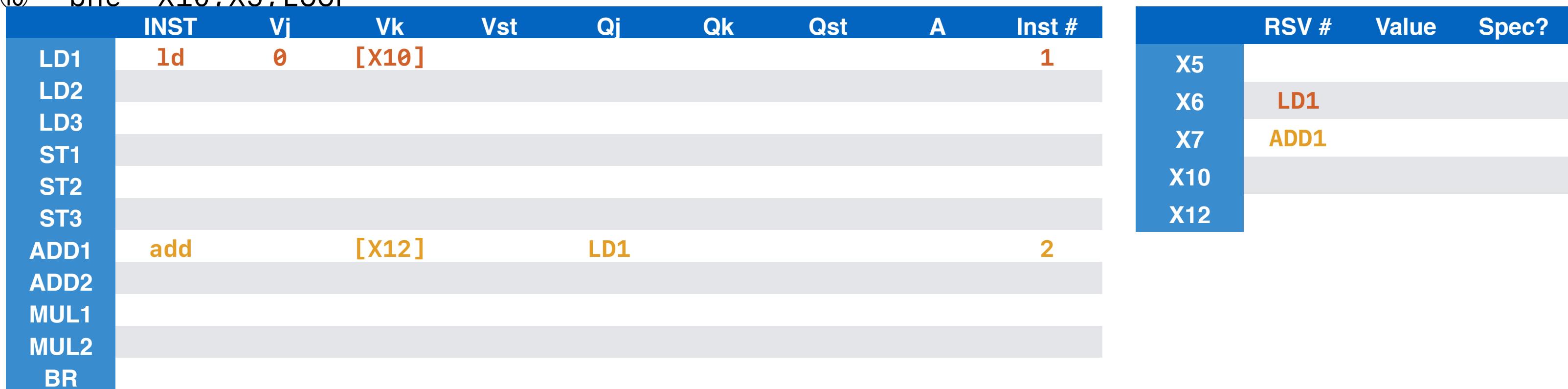
- ① ld X6, 0(X10)
 - ② add X7, X6, X12
 - ③ sd X7, 0(X10)
 - ④ addi X10, X10, 8
 - ⑤ bne X10, X5, LOOP
 - ⑥ ld X6, 0(X10)
 - ⑦ add X7, X6, X12
 - ⑧ sd X7, 0(X10)
 - ⑨ addi X10, X10, 8
 - ⑩ bne X10, X5, LOOP

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)
 - 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

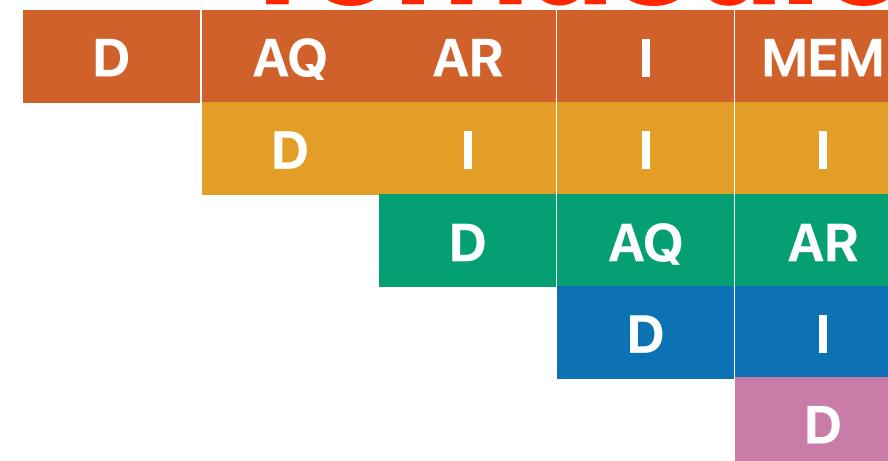
Tomasulo in motion

			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

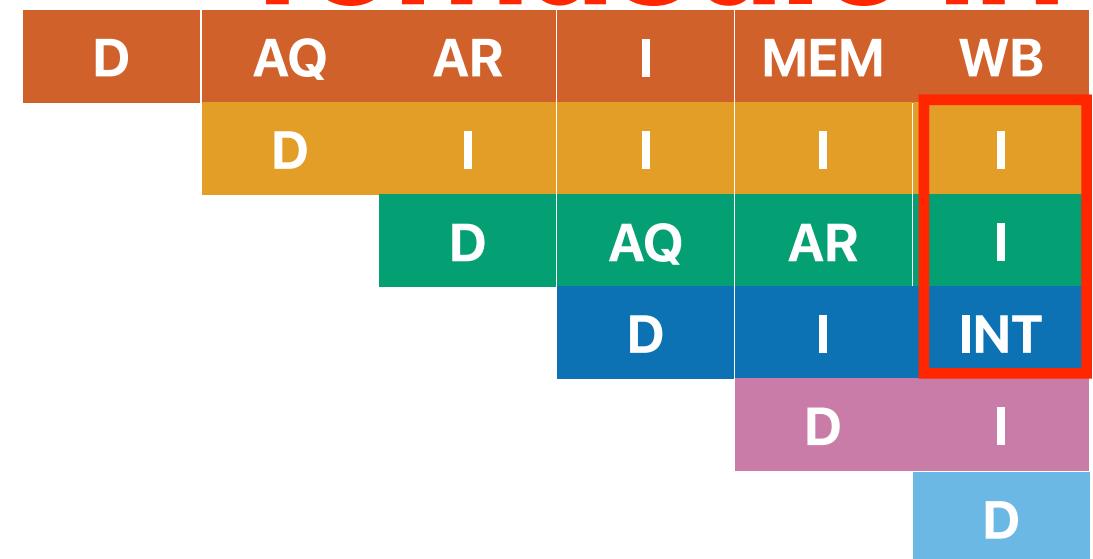
- ① ld X6, 0(X10)
- ② add X7, X6, X12
- ③ sd X7, 0(X10)
- ④ addi X10, X10, 8
- ⑤ bne X10, X5, LOOP
- ⑥ ld X6, 0(X10)
- ⑦ add X7, X6, X12
- ⑧ sd X7, 0(X10)
- ⑨ addi X10, X10, 8
- ⑩ bne X10, X5, LOOP



	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2										X6	LD1	
LD3										X7	ADD1	
ST1	sd	0	[X10]				ADD1		3	X10	ADD2	
ST2										X12		
ST3												
ADD1	add		[X12]		LD1				2			
ADD2	addi	8	[X10]						4			
MUL1												
MUL2												
BR	bne		[X5]		ADD2				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



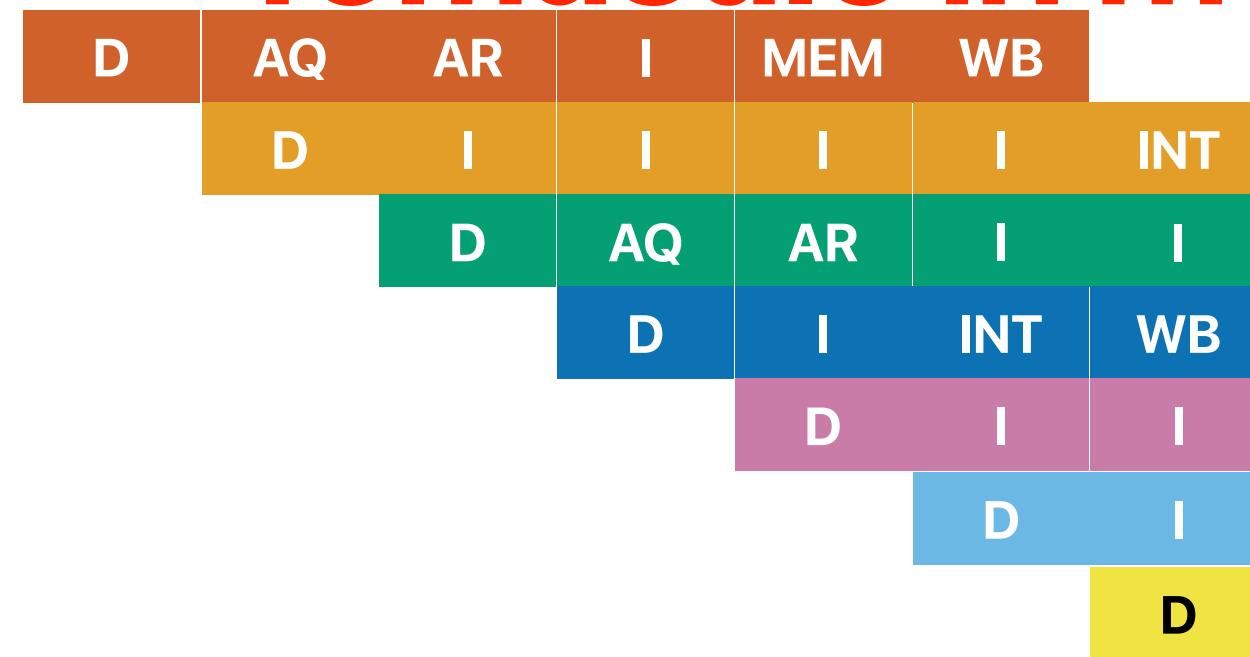
addi is now ahead of add!

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	ld	0	[X10]						1
LD2	ld	0			ADD2				6
LD3									
ST1	sd	0	[X10]				ADD1		3
ST2									
ST3									
ADD1	add	LD1	[X12]						2
ADD2	addi	8	[X10]						4
MUL1									
MUL2									
BR	bne		[X5]		ADD2				5

RSV #	Value	Spec?
X5		
X6	LD2	LD1 1
X7	ADD1	
X10	ADD2	
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

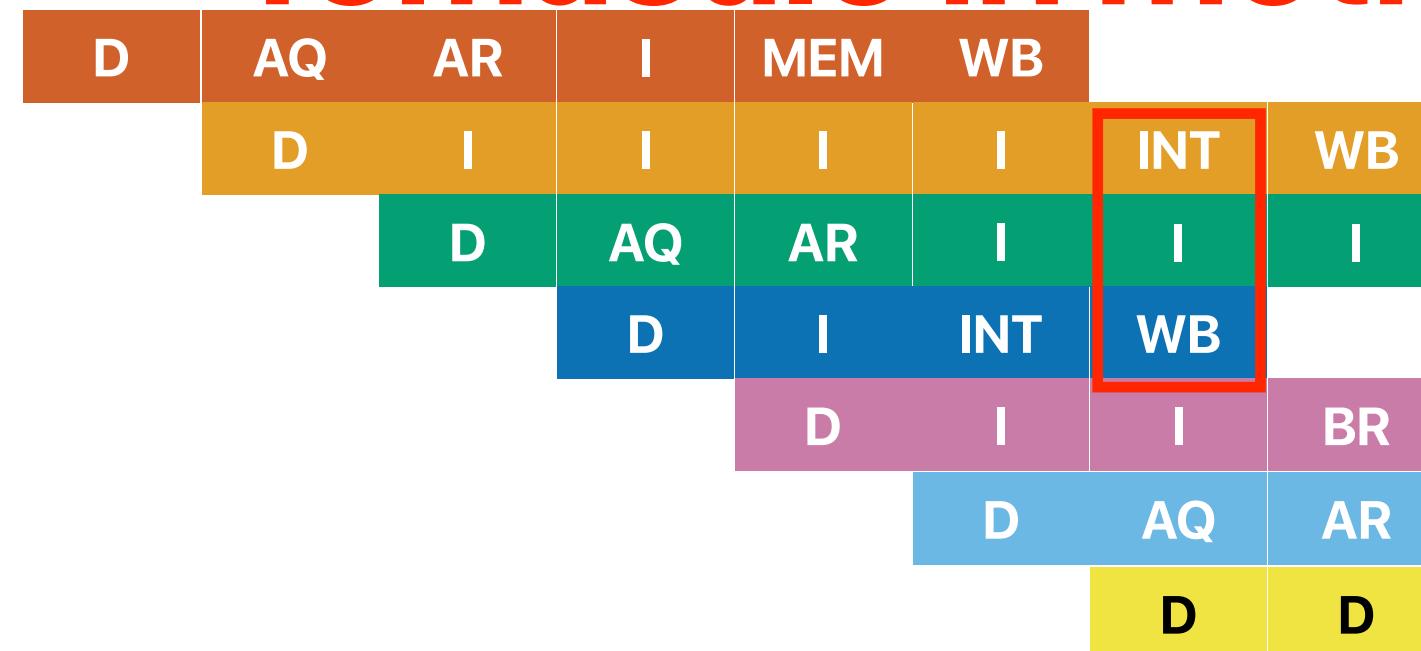


no reservation station for add!

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2	ld	0	ADD2						6	X6	LD2	1
LD3										X7	ADD1	
ST1	sd	0	[X10]				ADD1		3	X10		ADD2
ST2										X12		
ST3												
ADD1	add	LD1	[X12]						2			
ADD2	addi	8	[X10]						4			
MUL1												
MUL2												
BR	bne	ADD2	[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	ADD2					6	
LD3									
ST1	sd	0	[X10]	ADD1				3	
ST2									
ST3									
ADD1	add	LD1	[X12]					2	
ADD2	add		[X12]		[LD2]			7	
MUL1									
MUL2									
BR	bne	ADD2	[X5]						

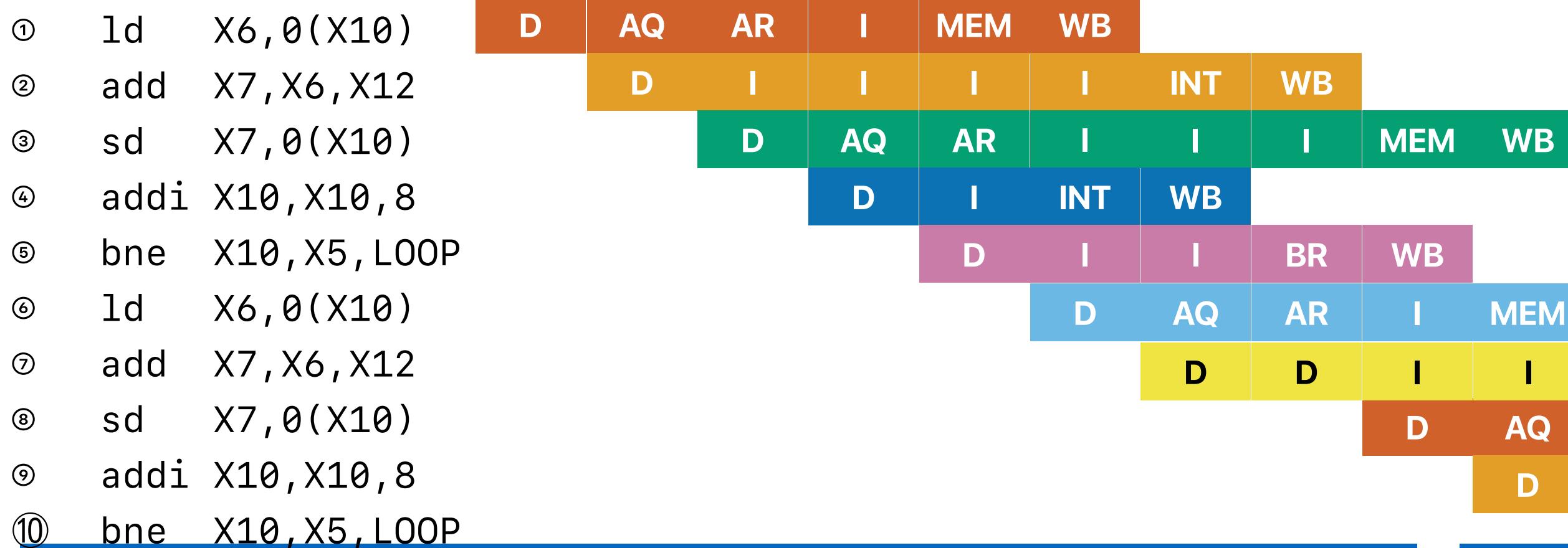
	RSV #	Value	Spec?
X5			
X6	LD2		1
X7	ADD2	ADD1	1
X10			
X12			

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	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	ADD2						6	X6	LD2	
LD3										X7	ADD2	
ST1	sd	0	[X10]	ADD1					3	X10		ADD2
ST2	sd	0	[X10]				ADD2		8	X12		
ST3												
ADD1	add	LD1	[X12]						2			
ADD2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	bne	ADD2	[X5]						5			

Tomasulo in motion



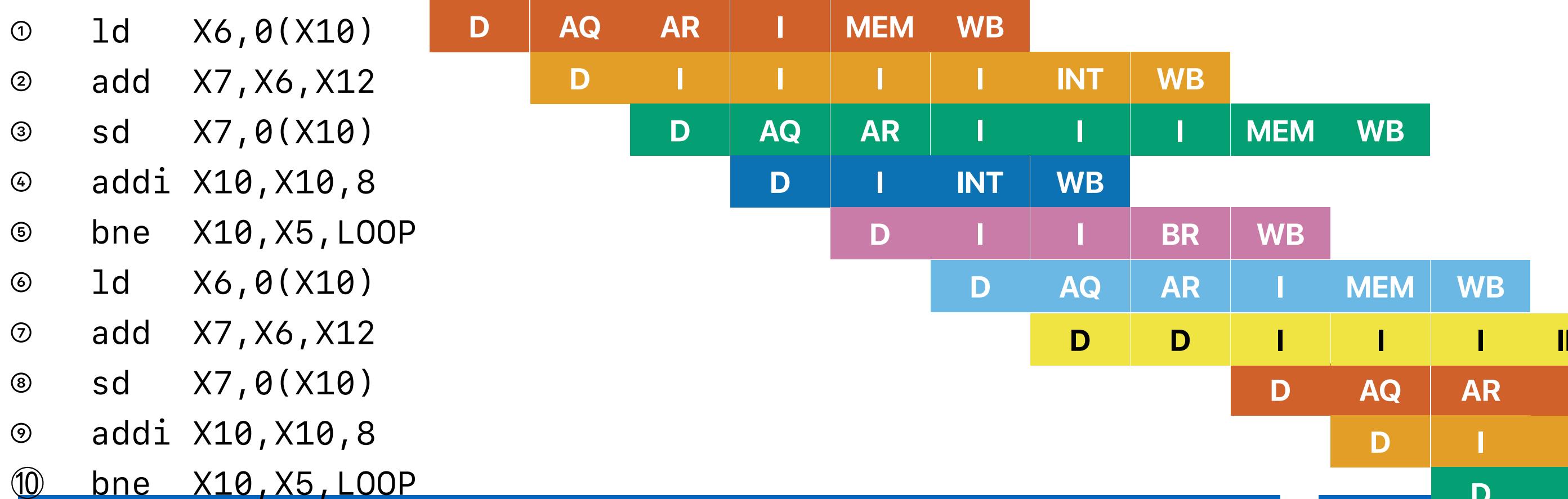
	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	RSV #	Value	Spec?
LD1	ld	0	[X10]						1	X5		
LD2	ld	0	ADD2						6	X6	LD2	
LD3										X7	ADD2	
ST1	sd	0	[X10]	ADD1					3	X10	ADD1	
ST2	sd	0	[X10]				ADD2		8	X12		
ST3												
ADD1	add	8	ADD2						9			
ADD2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	bne	ADD2	[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	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	I				
⑧	sd	X7, 0(X10)	D	AQ	AR							
⑨	addi	X10, X10, 8	D	I								
⑩	bne	X10, X5, LOOP	D									

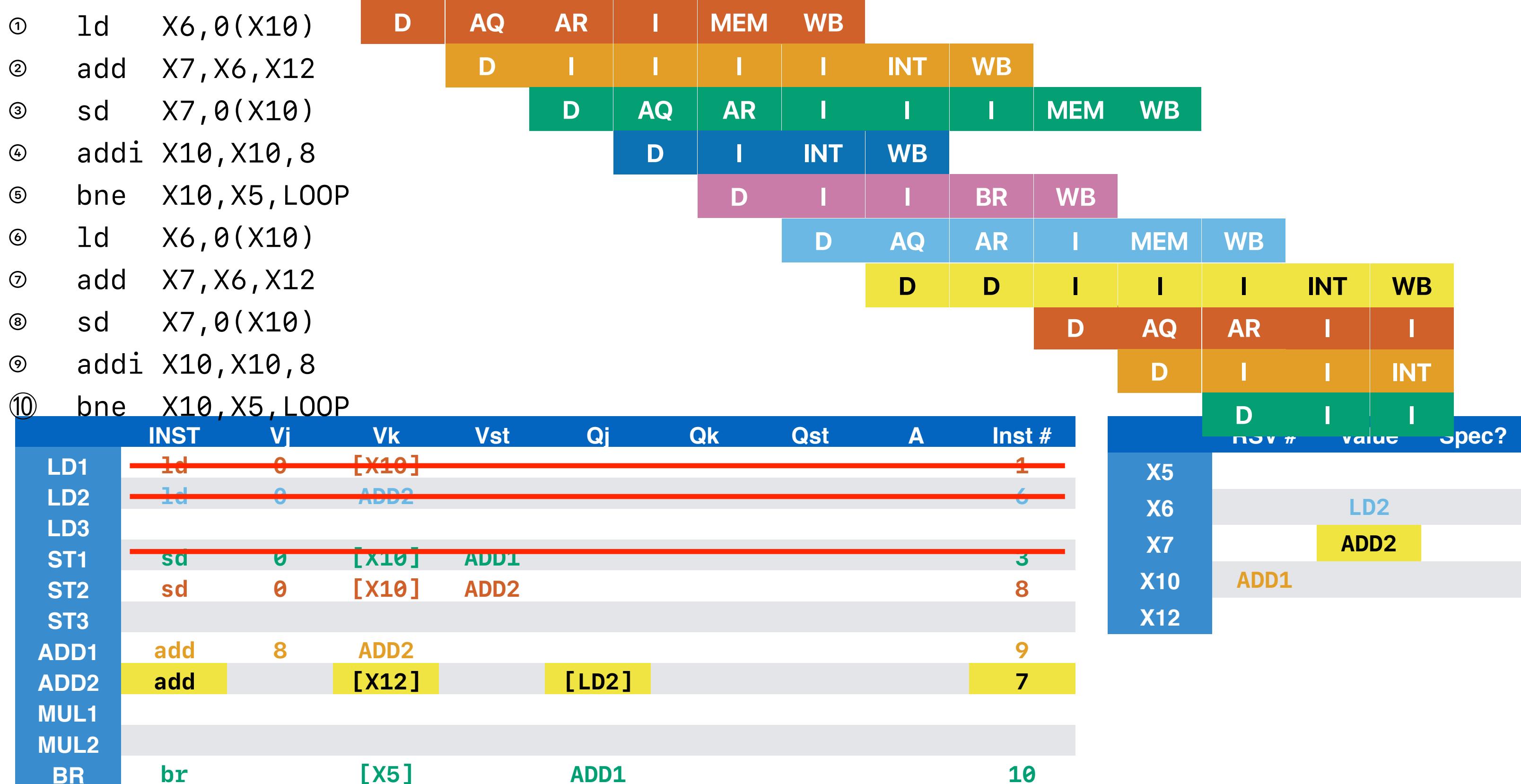
	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	Resv #	Value	Spec?
LD1	ld	0	[X10]						1		X5	
LD2	ld	0	ADD2						6		X6	LD2
LD3											X7	ADD2
ST1	sd	0	[X10]	ADD1					3		X10	ADD1
ST2	sd	0	[X10]				ADD2		8		X12	
ST3												
ADD1	add	8	ADD2						9			
ADD2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	br		[X5]		ADD1				10			

Tomasulo in motion

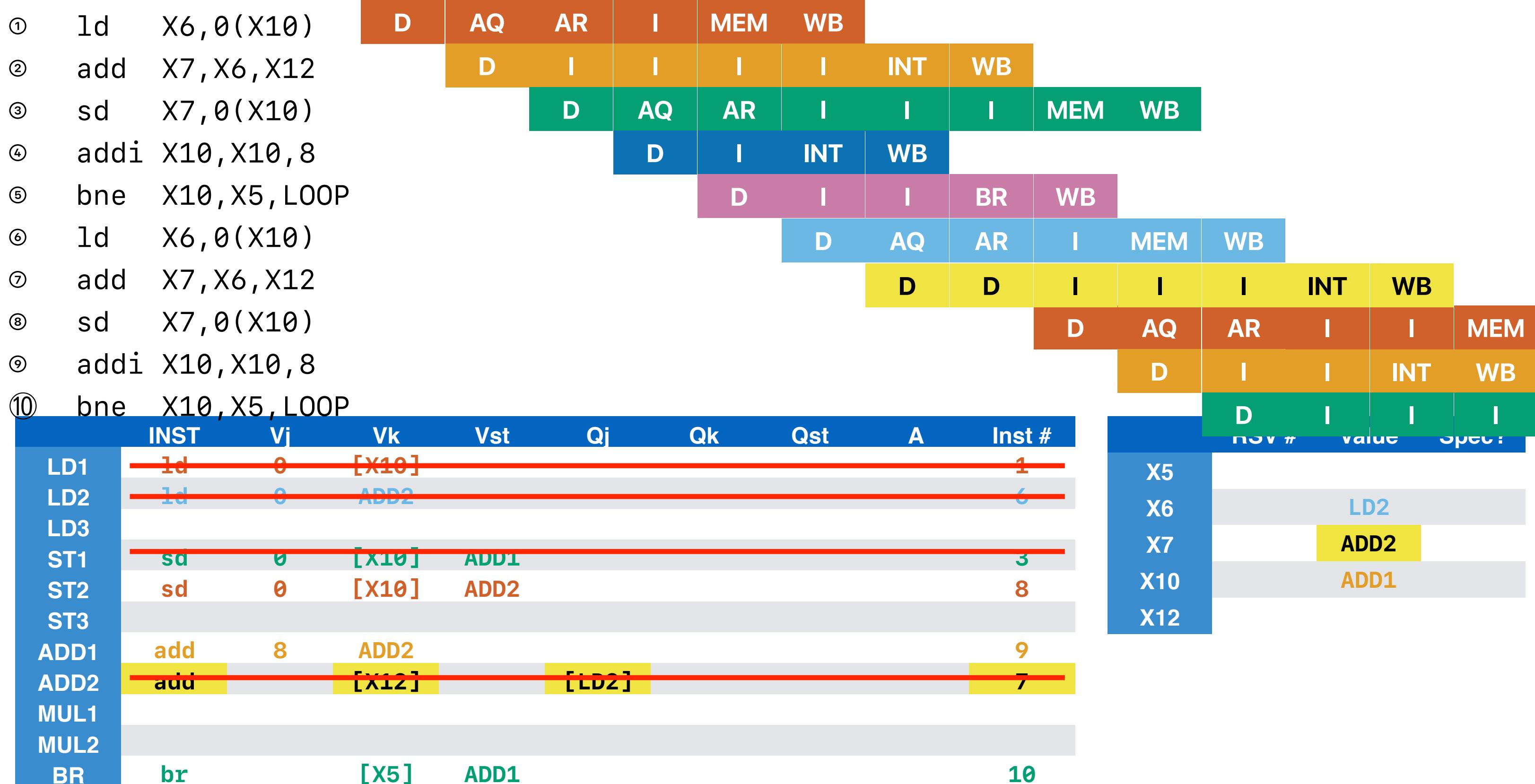


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #	Resv	Value	Spec?
LD1	ld	0	[X10]						1		X5	
LD2	ld	0	ADD2						6		X6	LD2
LD3											X7	ADD2
ST1	sd	0	[X10]	ADD1					3		X10	ADD1
ST2	sd	0	[X10]				ADD2		8		X12	
ST3												
ADD1	add	8	ADD2						9			
ADD2	add		[X12]		[LD2]				7			
MUL1												
MUL2												
BR	br		[X5]	ADD1					10			

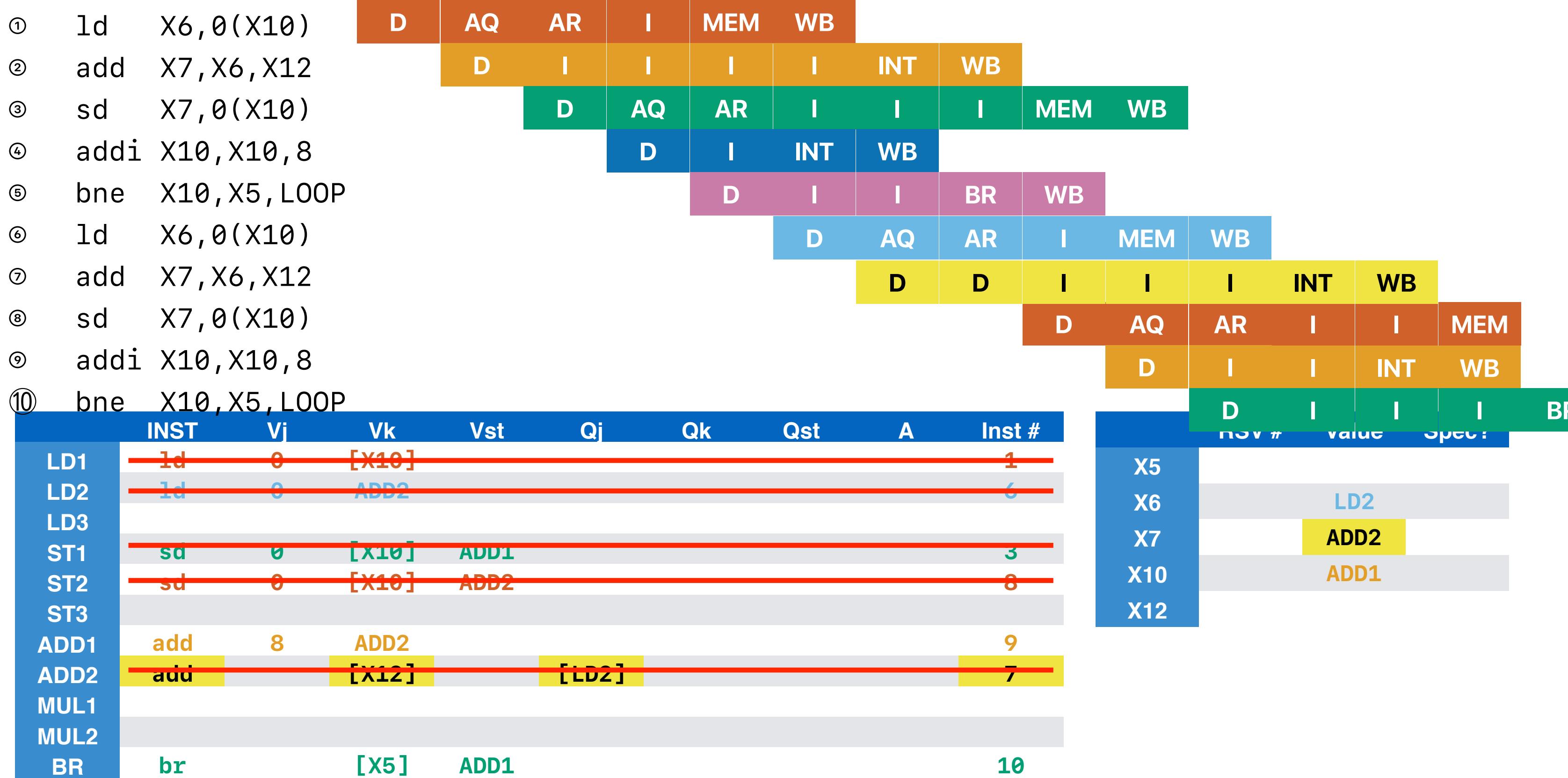
Tomasulo in motion



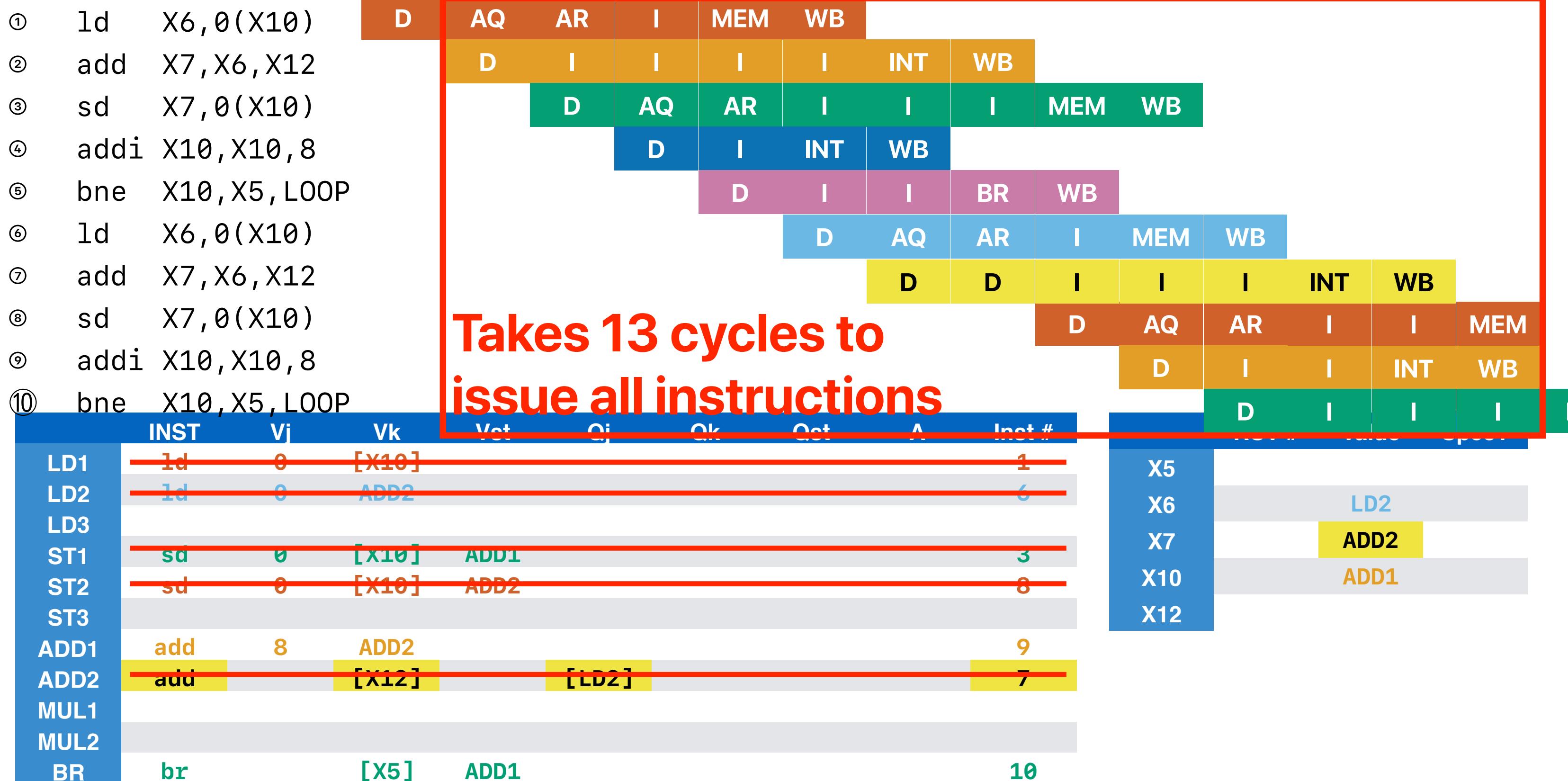
Tomasulo in motion



Tomasulo in motion



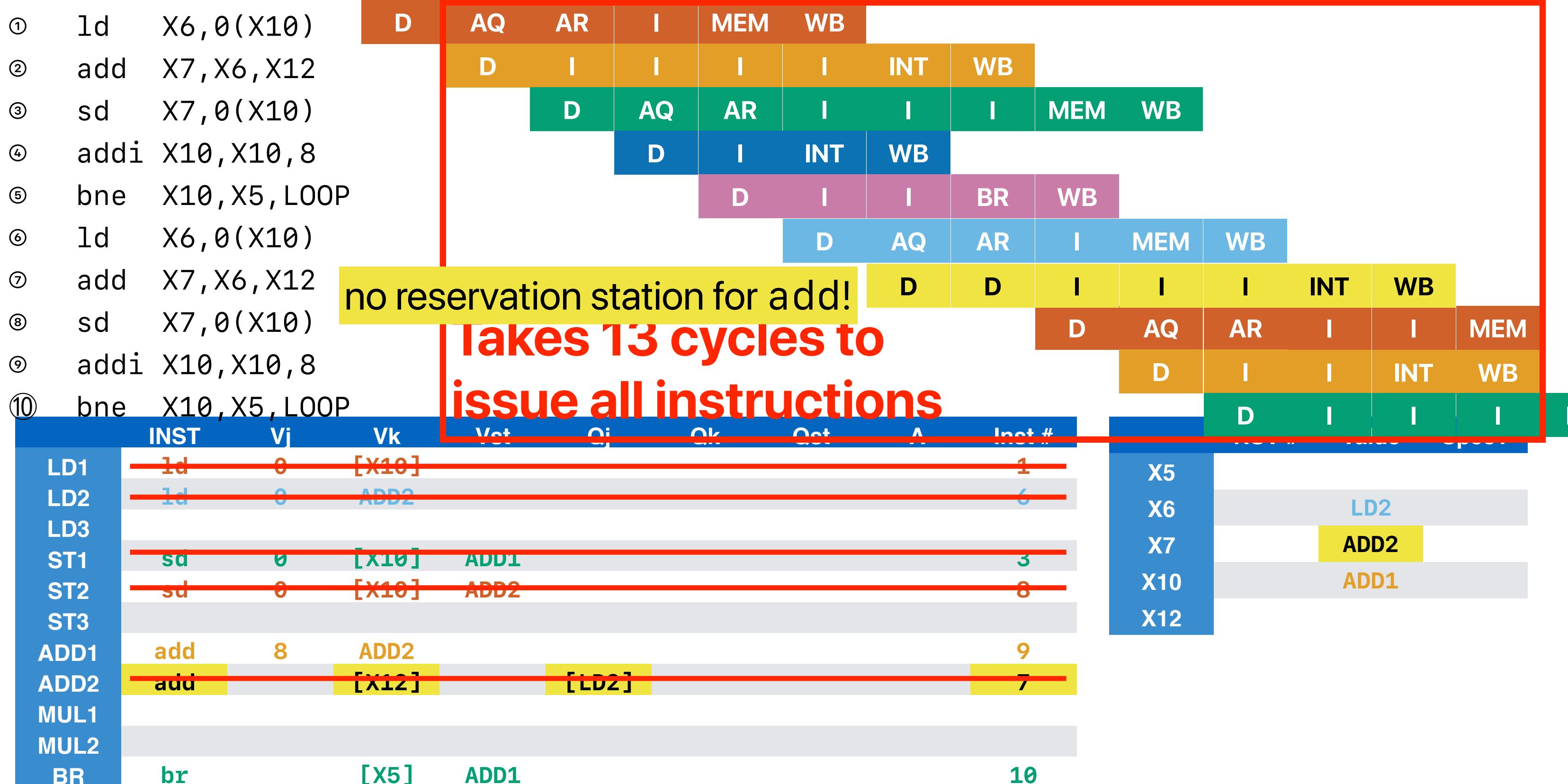
Tomasulo in motion



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



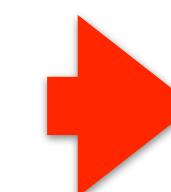
Recap: Why is B better than A?

A

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

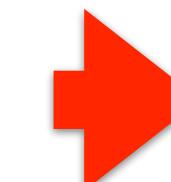
B

```
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;  
}
```

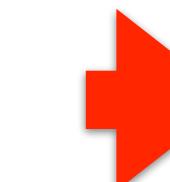


```
and x2, x1, 1  
add x3, x3, x2  
shr x1, x1, 1  
bne x1, x0, LOOP
```

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
```



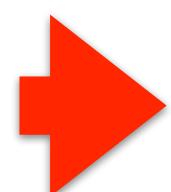
```
and x2, x1, 1  
shr x4, x1, 1  
shr x5, x1, 2  
shr x6, x1, 3  
shr x1, x1, 4  
and x7, x4, 1  
and x8, x5, 1  
and x9, x6, 1  
add x3, x3, x2  
add x3, x3, x7  
add x3, x3, x8  
add x3, x3, x9  
bne x1, x0, LOOP
```

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

43 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

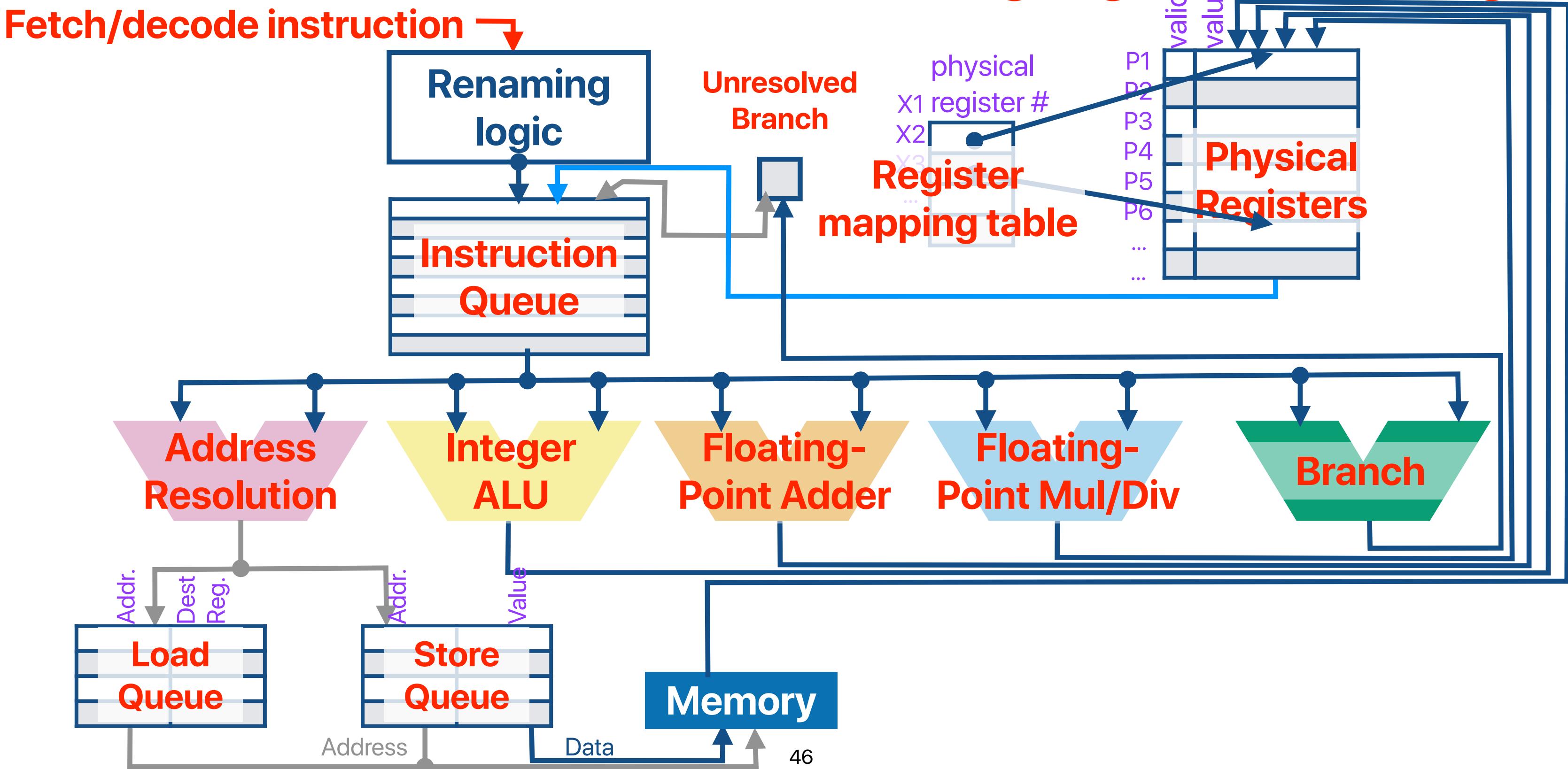


and x2, x1, 1
shr x4, x1, 1
shr x5, x1, 2
shr x6, x1, 3
shr x1, x1, 4
and x7, x4, 1
and x8, x5, 1
and x9, x6, 1
add x3, x3, x2
add x3, x3, x7
add x3, x3, x8
add x3, x3, x9
bne x1, x0, LOOP

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



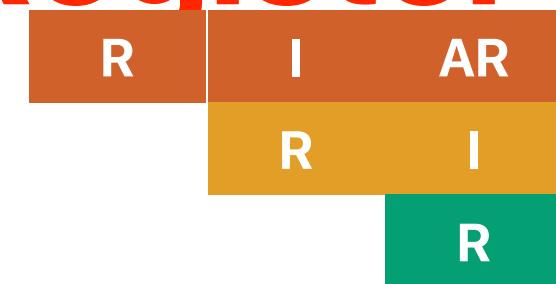
Renamed instruction		
1	ld P1, 0(X10)	
2	add P2, P1, X12	
3		
4		
5		
6		
7		
8		
9		
10		

Physical Register	
X5	
X6	P1
X7	P2
X10	
X12	

	Valid	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 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		
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

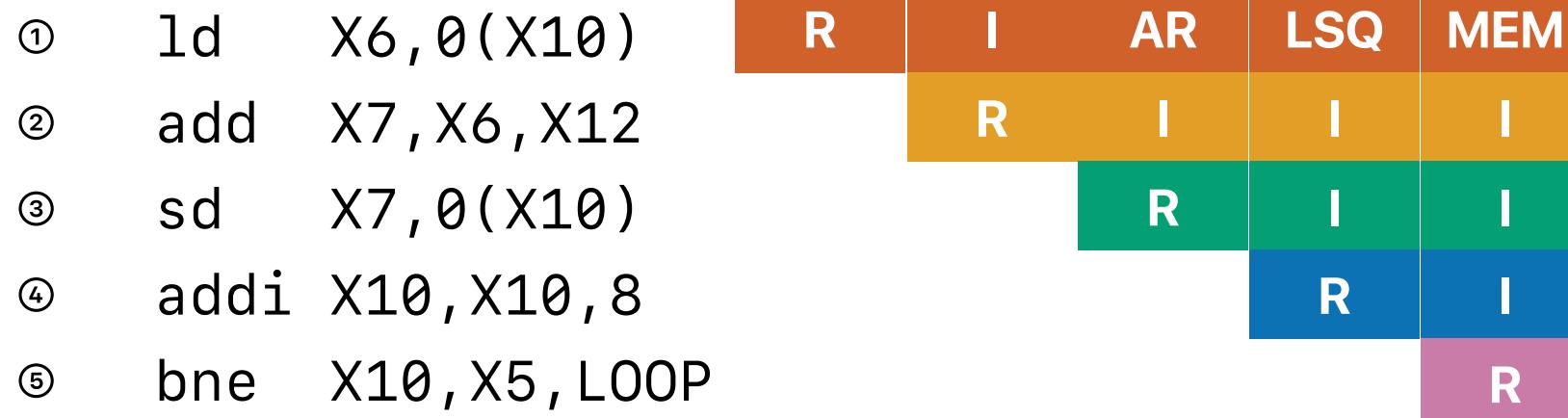


Renamed instruction		
1	ld	P1, 0(X10)
2	add	P2, P1, X12
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
5		
6		
7		
8		
9		
10		

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



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

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	P3

	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

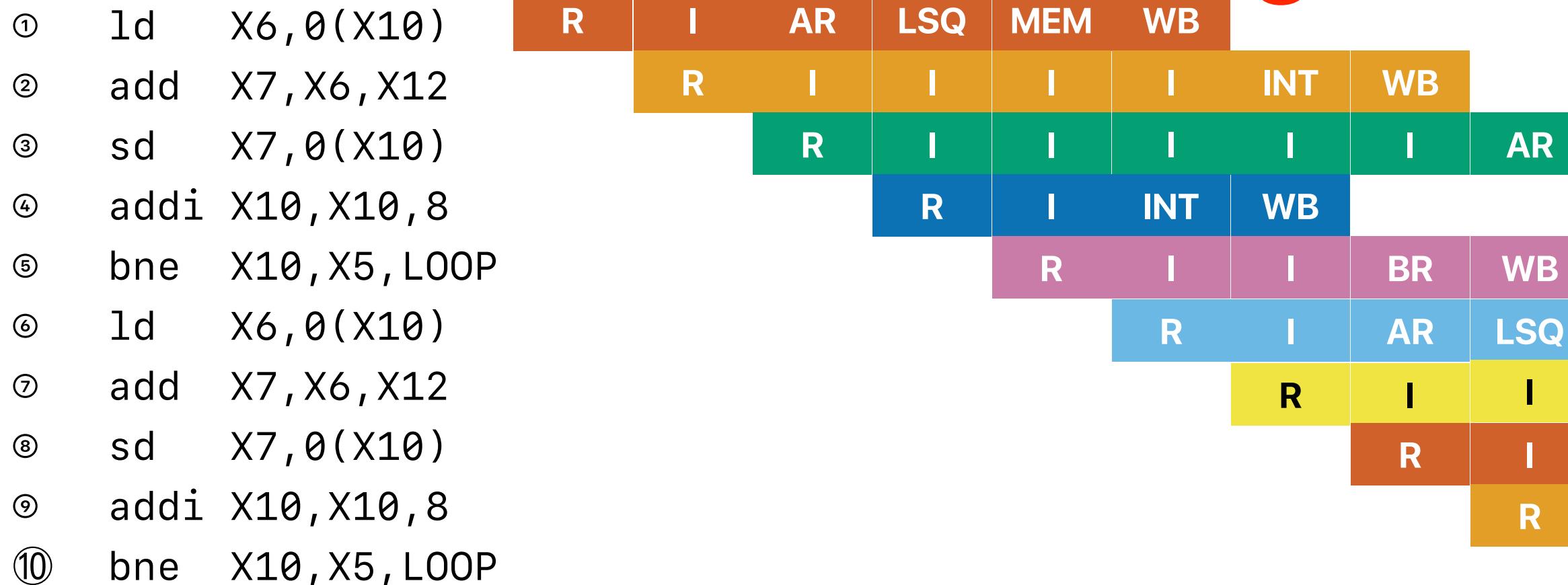
			R	I	AR	LSQ	MEM	WB	
①	ld	X6, 0(X10)	R	I	AR	LSQ	MEM	WB	
②	add	X7, X6, X12	R	I	I	I	I	INT	WB
③	sd	X7, 0(X10)	R	I	I	I	I	I	I
④	addi	X10, X10, 8		R	I	INT		WB	
⑤	bne	X10, X5, LOOP			R	I	I	BR	
⑥	ld	X6, 0(X10)			R	I		AR	
⑦	add	X7, X6, X12			R	I			
⑧	sd	X7, 0(X10)				R			
⑨	addi	X10, X10, 8							
⑩	bne	X10, X5,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

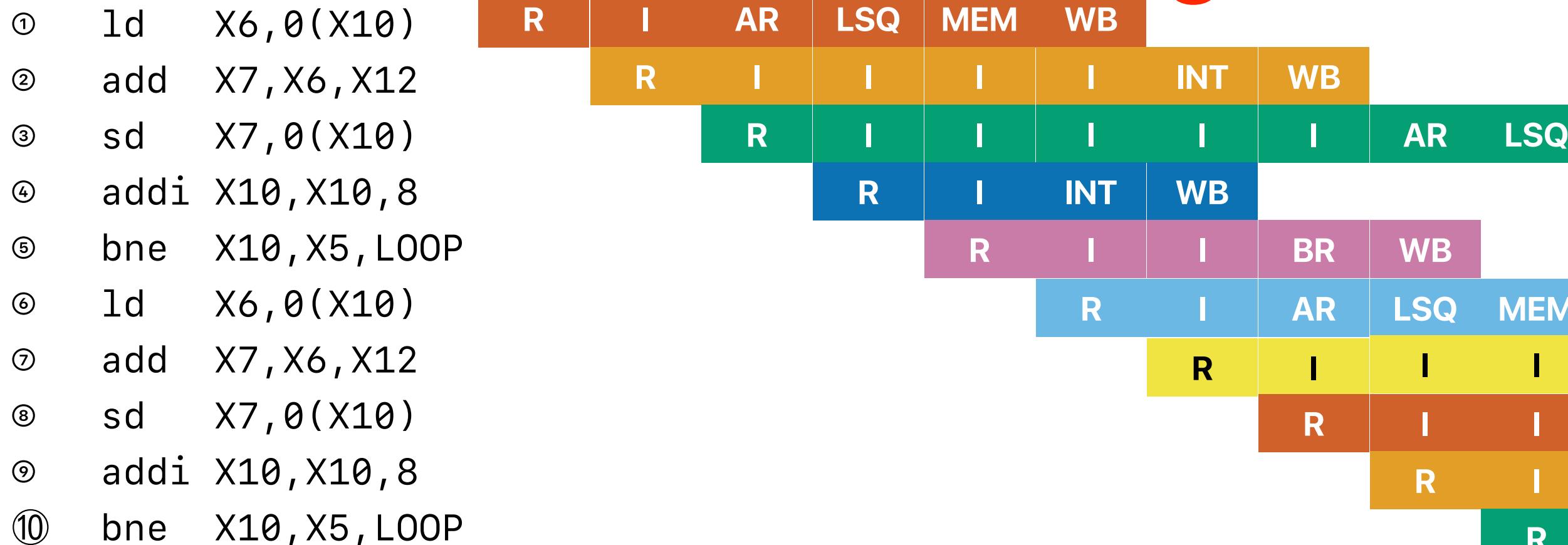


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	

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

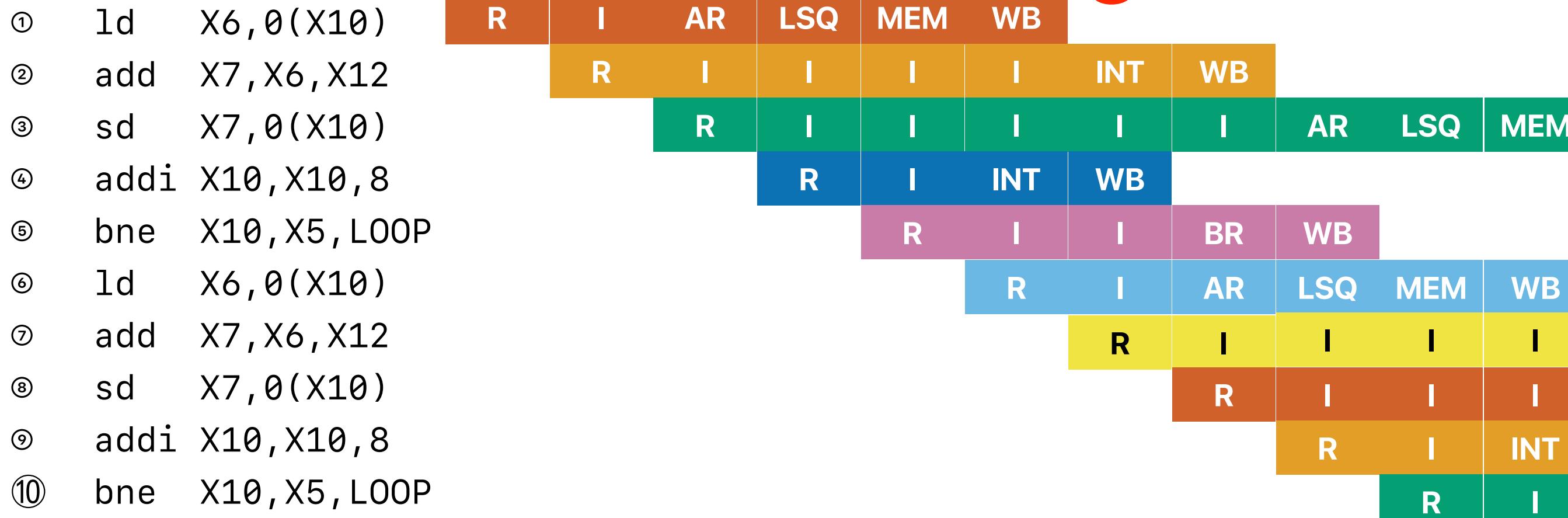


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	
X6	P1
X7	P5
X10	P3
X12	

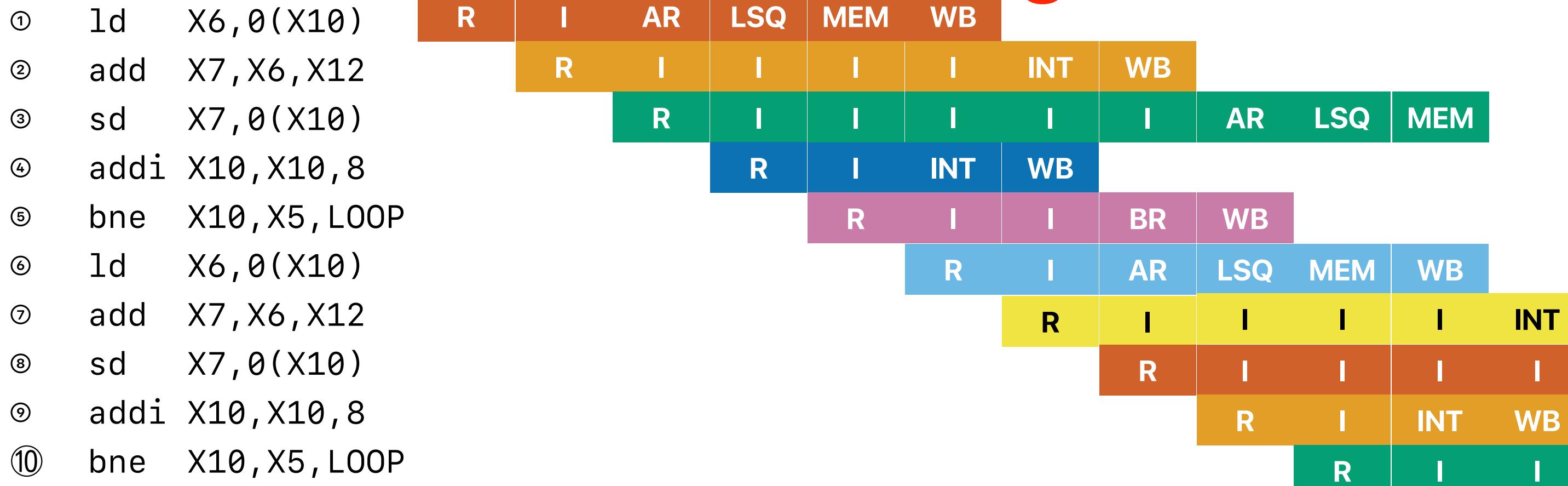
Valid Value In use			Valid Value In use		
P1	1	1	P6	0	1
P2	1	1	P7		
P3	1	1	P8		
P4	0	1	P9		
P5	0	1	P10		

Register renaming in motion



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

Register renaming in motion

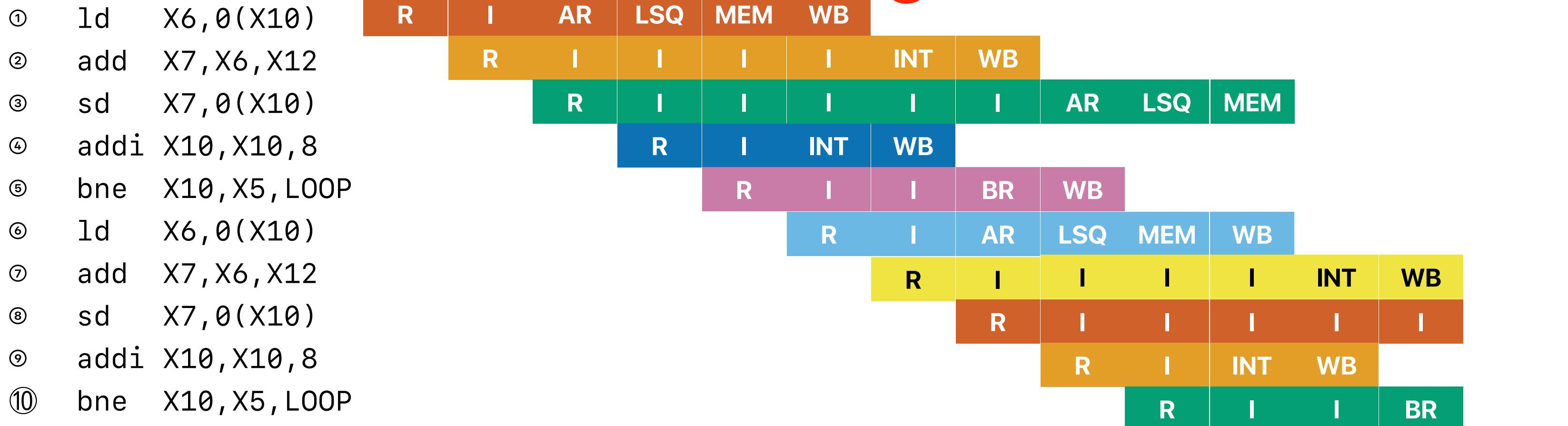


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

Physical Register	
X5	
X6	P1
X7	P5
X10	P3
X12	

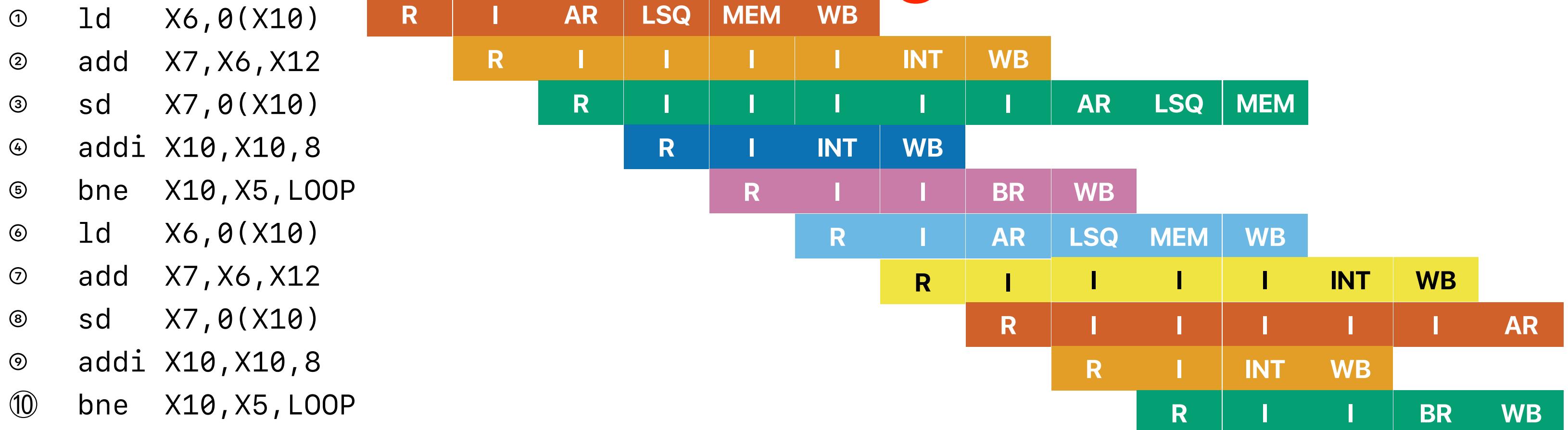
Valid Value In use			Valid Value In use		
P1	1	1	P6	1	1
P2	1	1	P7		
P3	1	1	P8		
P4	1	1	P9		
P5	0	1	P10		

Register renaming in motion



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

Register renaming in motion

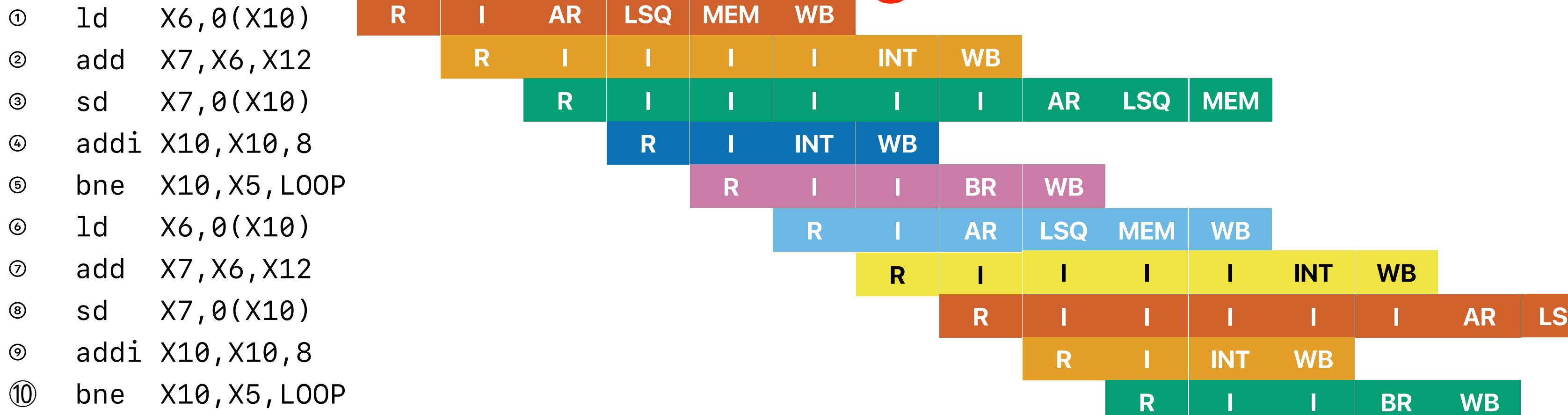


	Renamed instruction	
1	ld	P1, 0(X10)
2	add	P2, P1, X12
3	sd	P2, 0(X10)
4	addi	P3, X10, 8
5	bne	P3, X5, LOOP
6	ld	P4, 0(P3)
7	add	P5, P1, X12
8	sd	P5, 0(P3)
9	addi	P6, P3, 8
10	bne	P6, 0(X10)

	Physical Register	
X5		
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

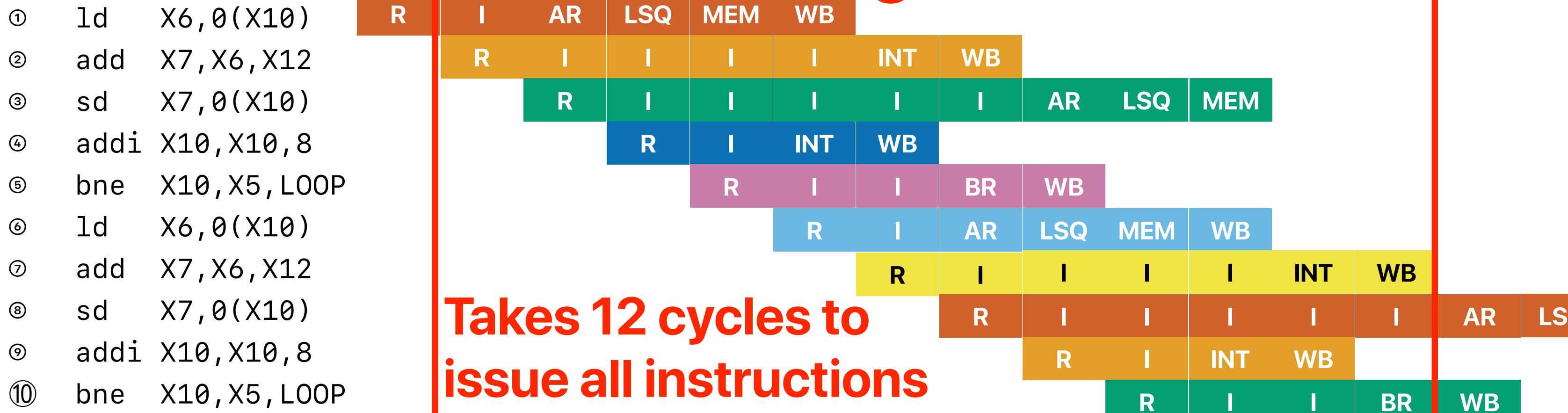


Renamed instruction	
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	bne P6, 0(X10)

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		

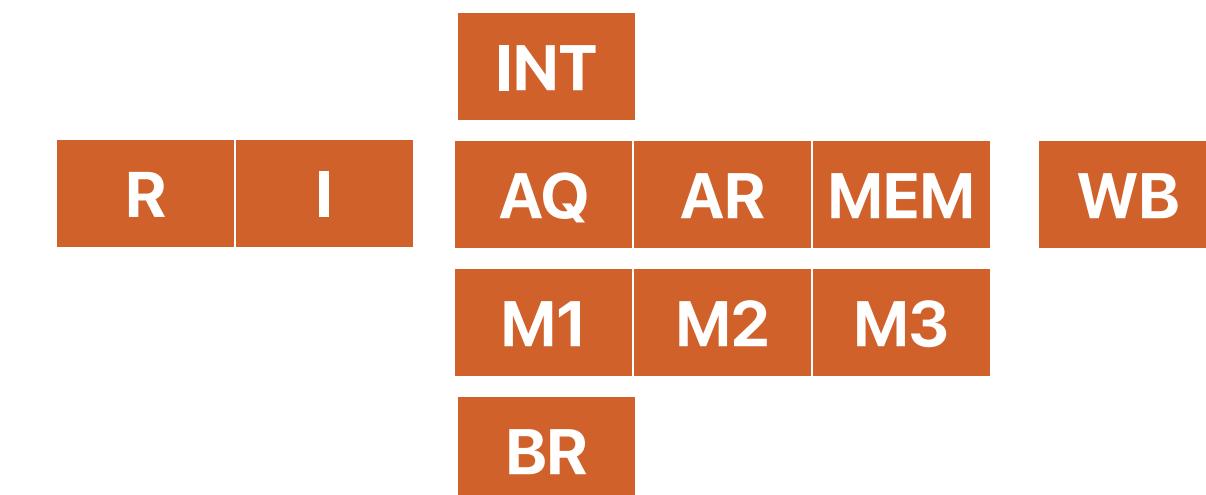
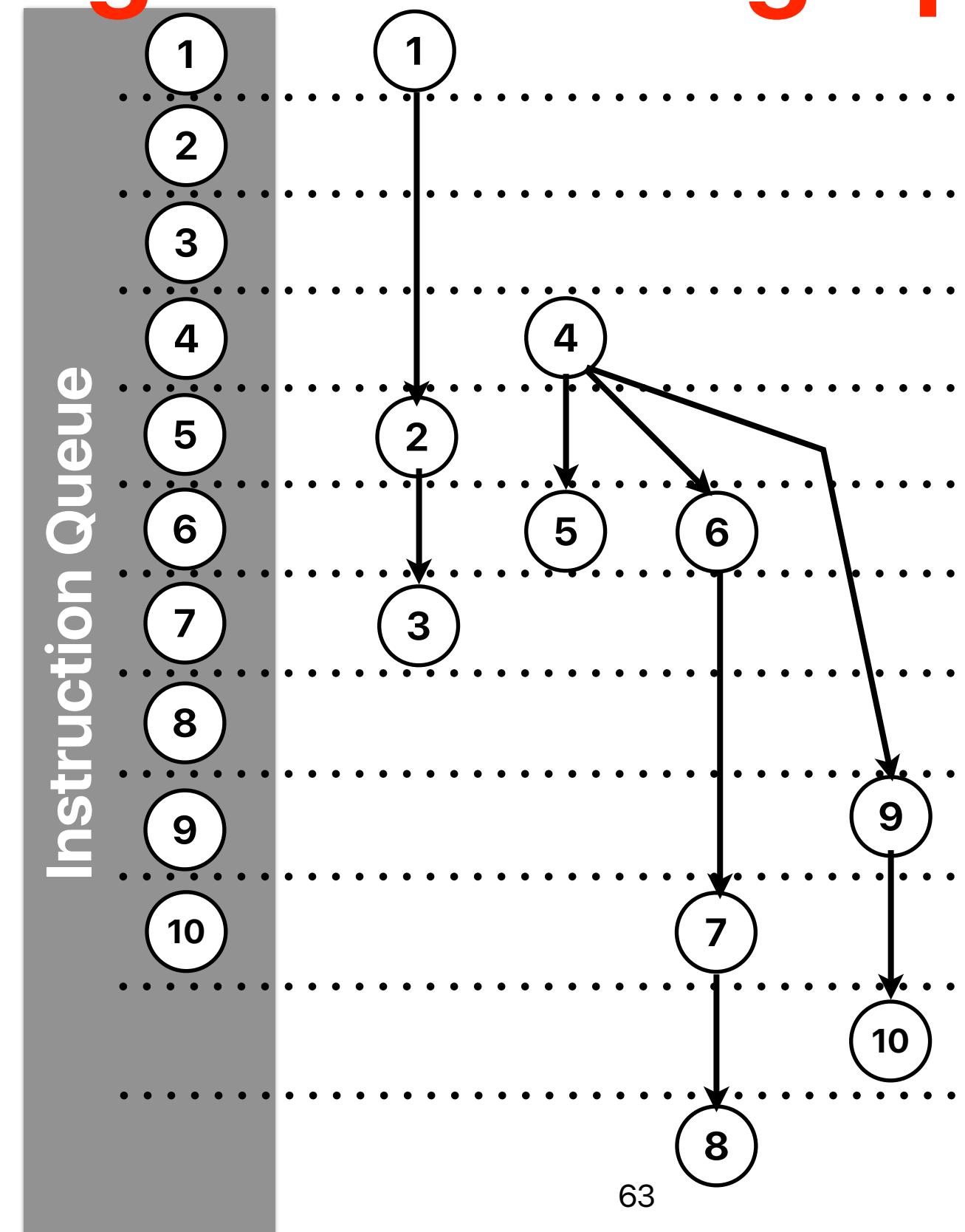
Register renaming in motion



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

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



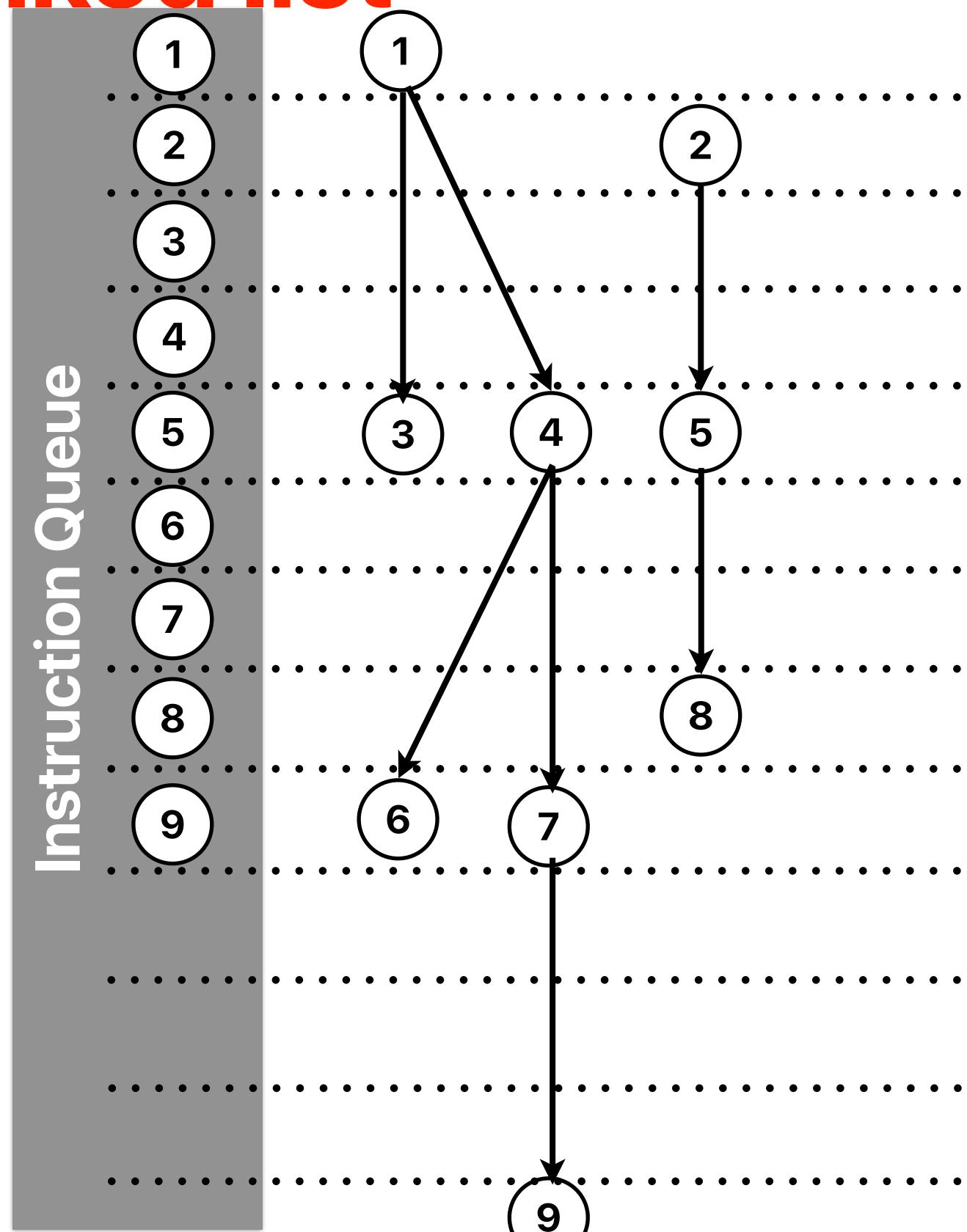
What about “linked list”

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)	R	I	AR	LSQ
②	addi	X7, X7, 1	R	I	INT	
③	bne	X10, X0, LOOP		R	I	
④	ld	X10, 8(X10)			R	
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		Physical Register		Valid Value In use			Valid Value In use		
1	ld P1, 0(X10)	X5		P1	0	1	P6		
2	add P2, X7, 1	X6		P2	0	1	P7		
3	bne P1, X0, LOOP	X7	P2	P3	0	1	P8		
4	ld P3, 0(P1)	X10	P3	P4			P9		
5		X12		P5			P10		
6									
7									
8									
9									
10									

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

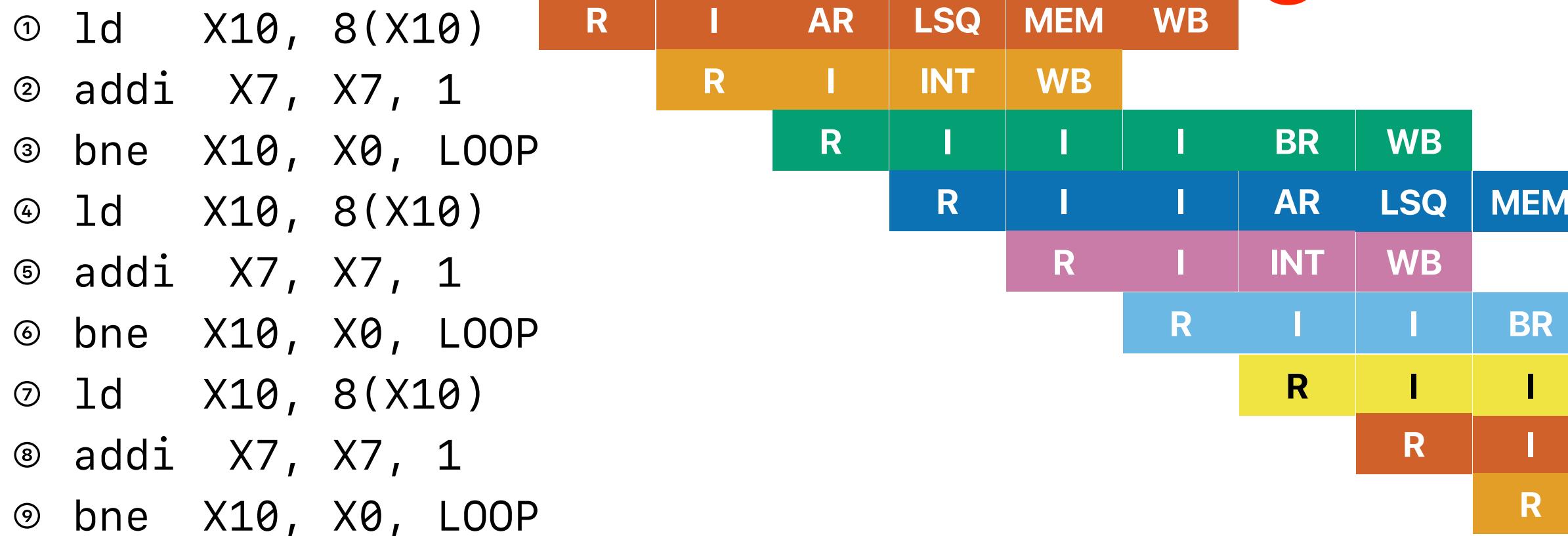
		R	I	AR	LSQ	MEM	WB
①	ld X10, 8(X10)	R					
②	addi X7, X7, 1		R	I	INT	WB	
③	bne X10, X0, LOOP			R	I	I	BR WB
④	ld X10, 8(X10)			R	I	I	AR LSQ
⑤	addi X7, X7, 1			R	I	INT	WB
⑥	bne X10, X0, LOOP			R	I	I	
⑦	ld X10, 8(X10)			R	I	I	
⑧	addi X7, X7, 1						R
⑨	bne X10, X0, LOOP						

	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	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

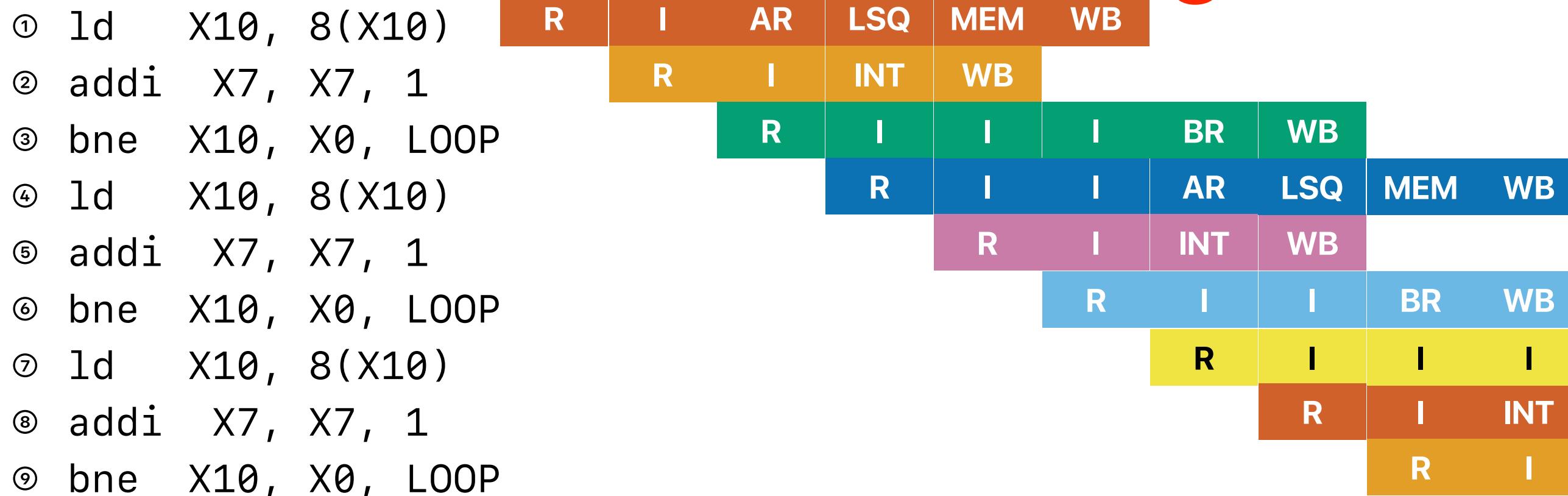


	Renamed instruction
1	ld P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	ld P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	ld P5, 0(P3)
8	add P6, P4, 1
9	bne P5, X0, LOOP
10	

	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

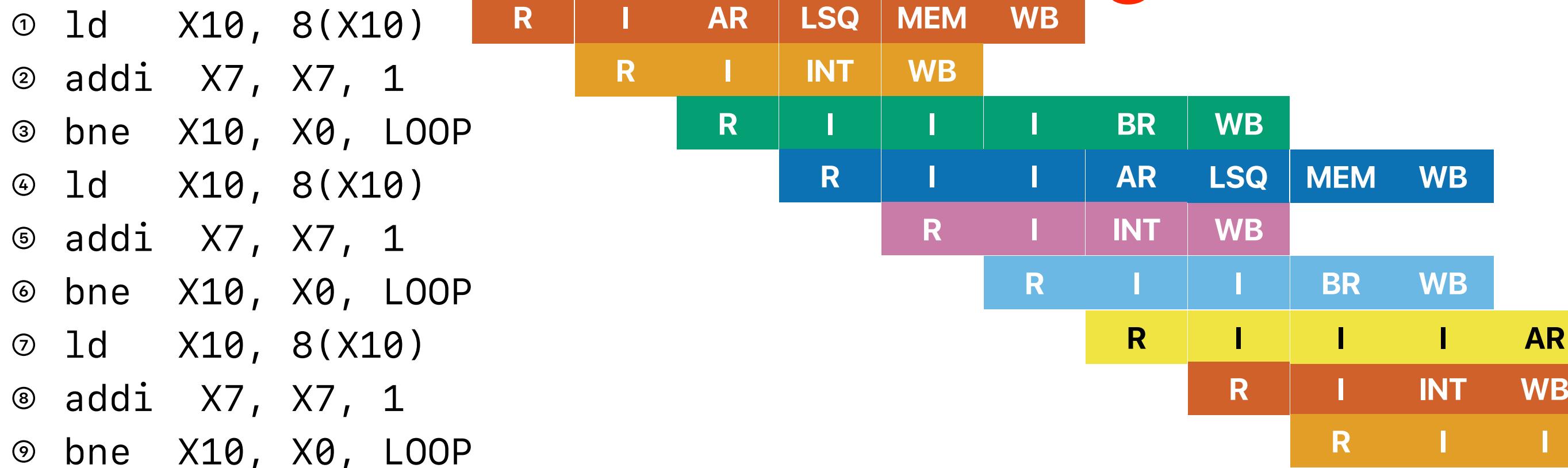


	Renamed instruction
1	ld P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	ld P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	ld P5, 0(P3)
8	add P6, P4, 1
9	bne P5, X0, LOOP
10	

	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

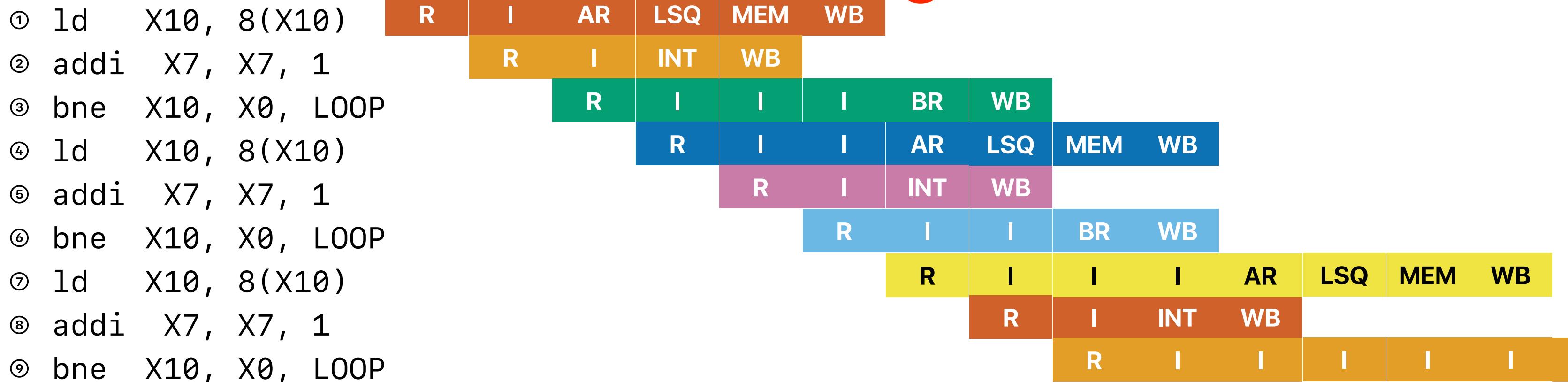


	Renamed instruction
1	1d P1, 0(X10)
2	add P2, X7, 1
3	bne P1, X0, LOOP
4	1d P3, 0(P1)
5	add P4, P2, 1
6	bne P3, X0, LOOP
7	1d P5, 0(P3)
8	add P6, P4, 1
9	bne P5, X0, LOOP
10	

	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	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	

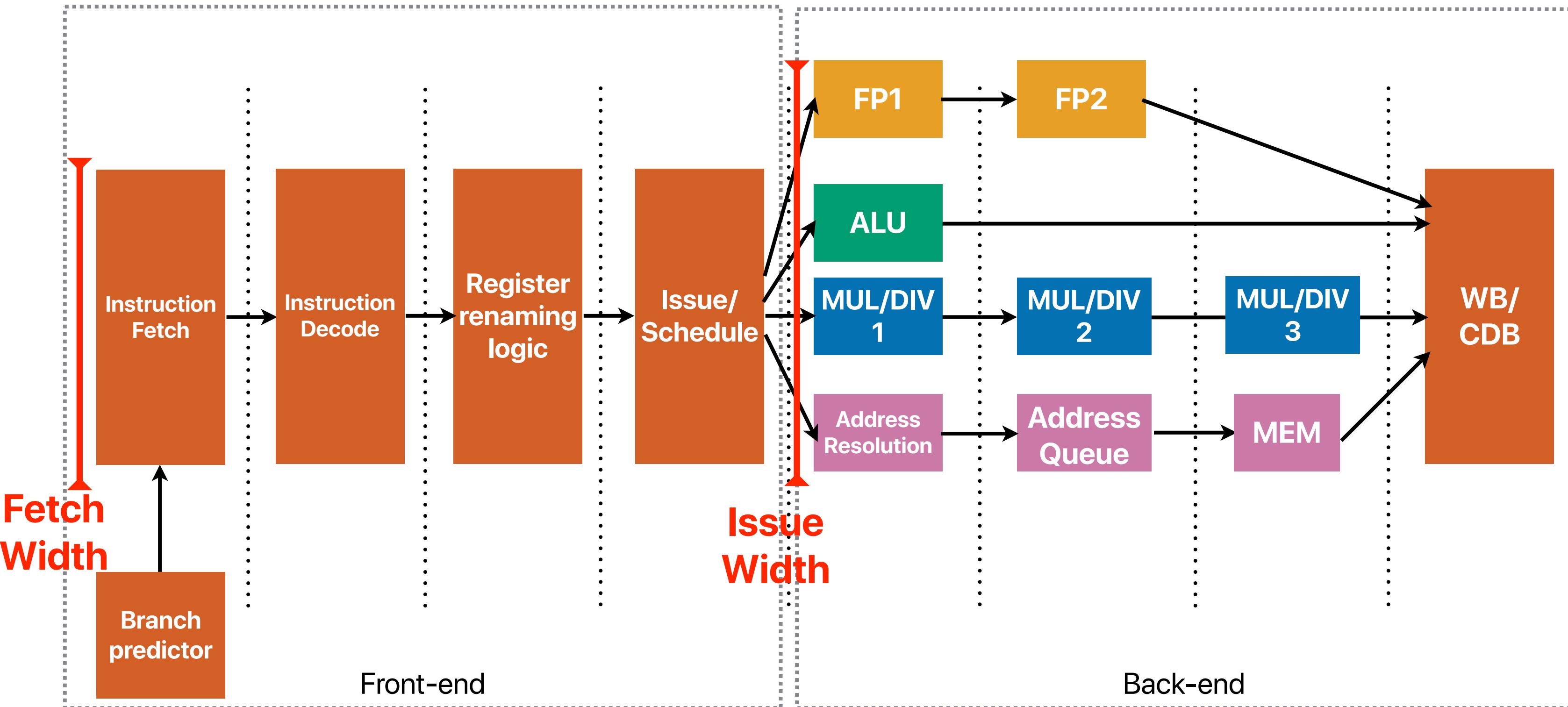
	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6	0		1
P2	1		1	P7			
P3	0		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Super Scalar

Superscalar

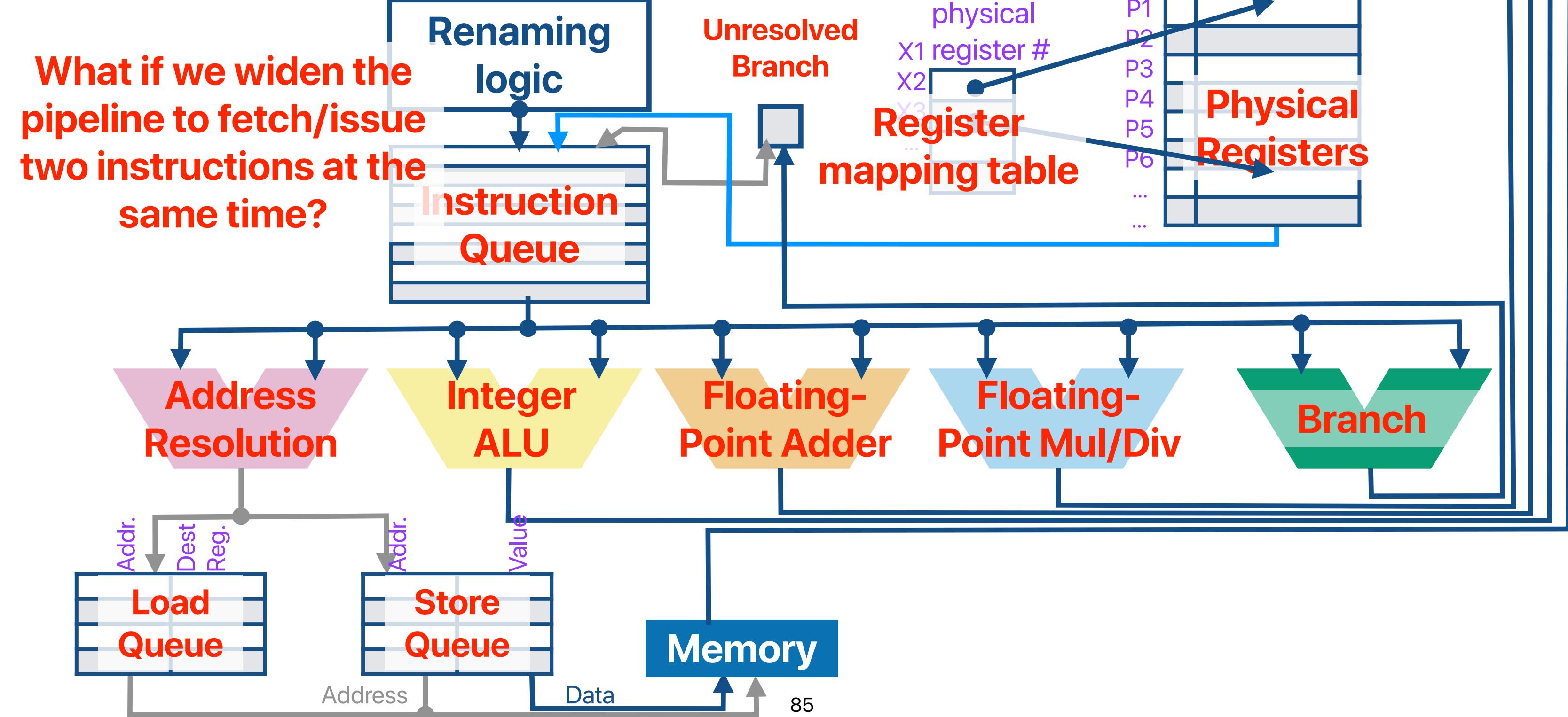
- Since we have more functional units now, we should fetch/decode more instructions each cycle so that we can have more instructions to issue!
- Super-scalar: fetch/decode/issue more than one instruction each cycle
 - Fetch width: how many instructions can the processor fetch/decode each cycle
 - Issue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



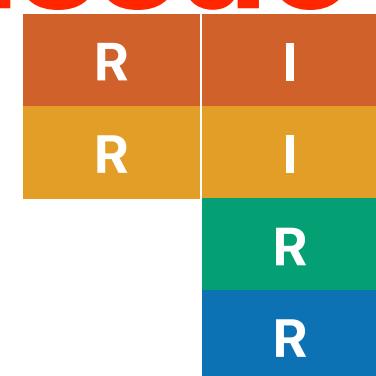
Renamed instruction		
1	ld	P1, 0(X10)
2	add	P2, P1, X12
3		
4		
5		
6		
7		
8		
9		
10		

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

- 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



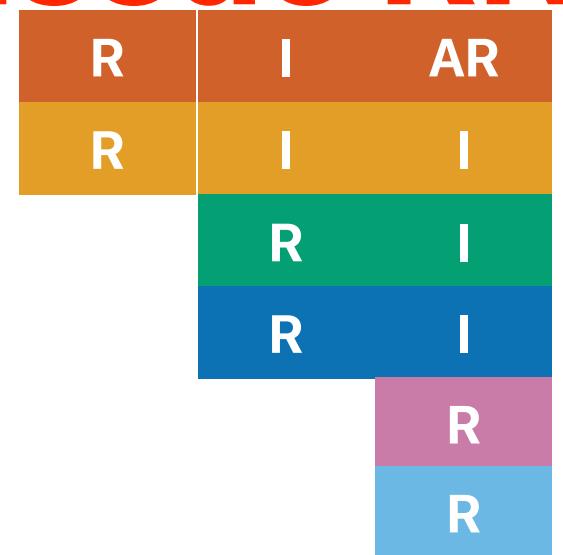
Renamed instruction		
1	ld P1, 0(X10)	
2	add P2, P1, X12	
3	sd P2, 0(X10)	
4	addi P3, X10, 8	
5		
6		
7		
8		
9		
10		

Physical Register	
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



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	0	1	P6		
P2	0	1	P7		
P3	0	1	P8		
P4	0	1	P9		
P5			P10		

2-issue RR processor in motion

			R	I	AR	AQ
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	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

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		
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)								

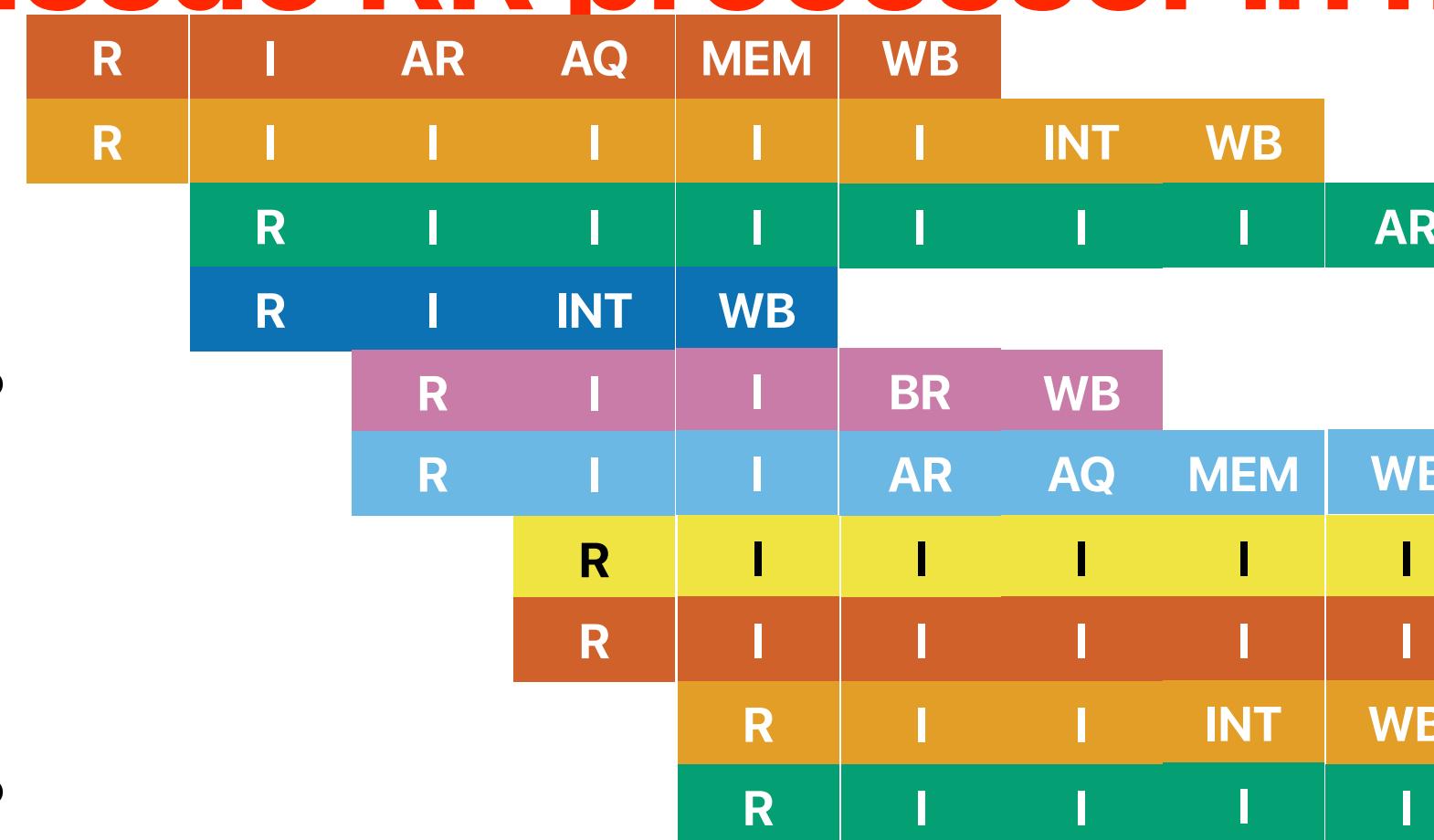
2-issue RR processor in motion

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

Renamed instruction	Physical Register	Valid			Valid		
		Value	In use		Value	In use	
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	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

- 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)

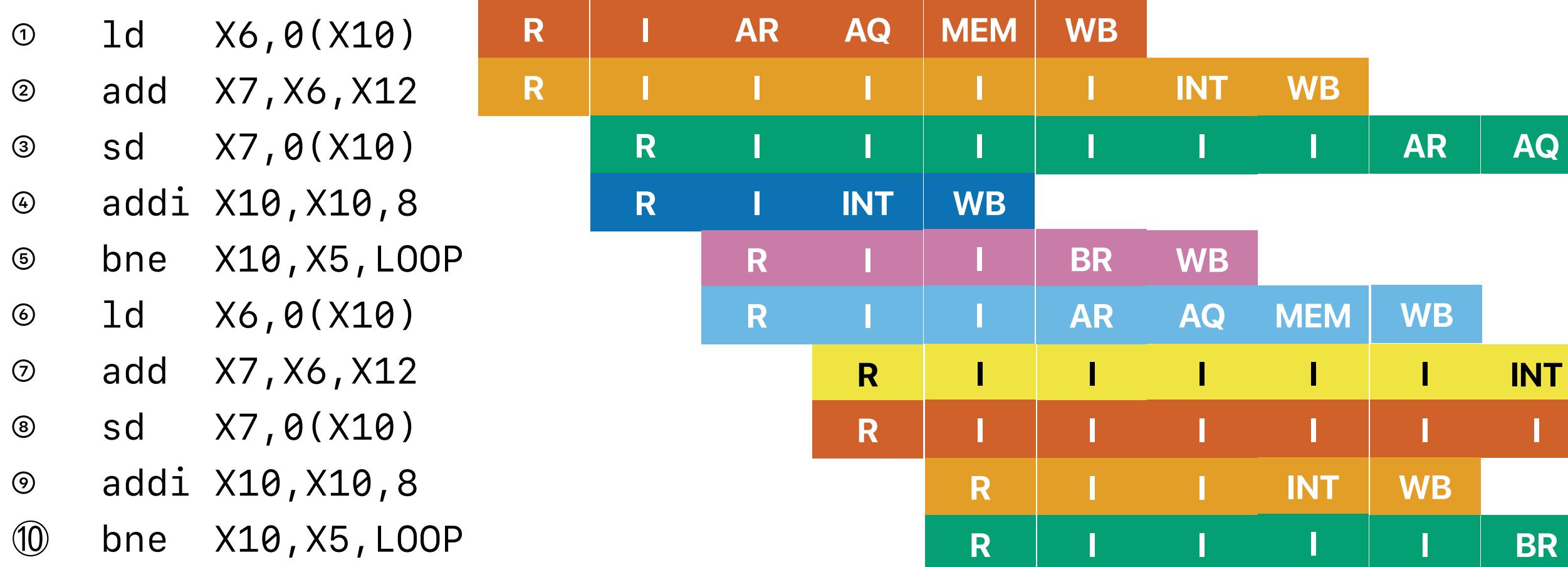
	Physical Register	Valid			Valid		
		Value	In use		Value	In use	
X5		P1	1	1	P6		
X6	P1	P2	1	1	P7		
X7	P5	P3	1	1	P8		
X10	P3	P4	1	1	P9		
X12		P5	0	1	P10		

2-issue RR processor in motion

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
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	
⑧	sd	X7, 0(X10)		R	I	I	I	I	I	I	
⑨	addi	X10, X10, 8		R	I	I	INT	WB			
⑩	bne	X10, X5, LOOP		R	I	I	I	I	I	BR	

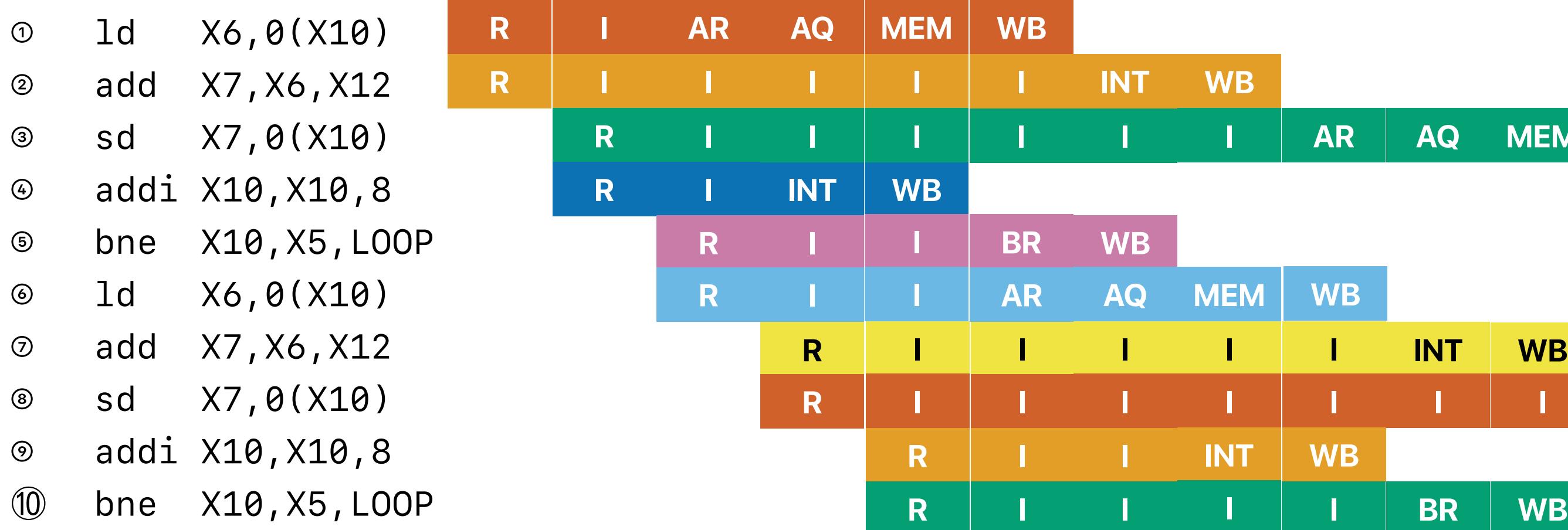
Renamed instruction	Physical Register	Valid			Valid		
		Value	In use		Value	In use	
1 ld P1, 0(X10)	X5			P1	1	1	P6
2 add P2, P1, X12	X6	P1		P2	1	1	P7
3 sd P2, 0(X10)	X7	P5		P3	1	1	P8
4 addi P3, X10, 8	X10	P3		P4	1	1	P9
5 bne P3, X5, LOOP	X12			P5	0	1	P10
6 ld P4, 0(P3)							
7 add P5, P1, X12						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

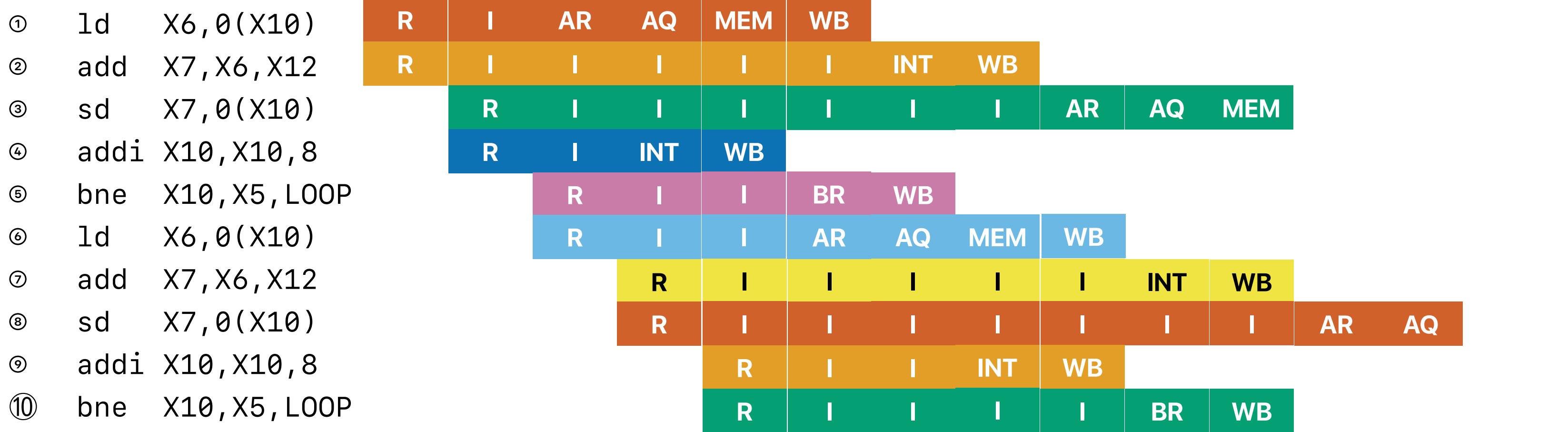
2-issue RR processor in motion



Renamed instruction	Physical Register	Valid			Valid		
		Value	In use		Value	In use	
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				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)							

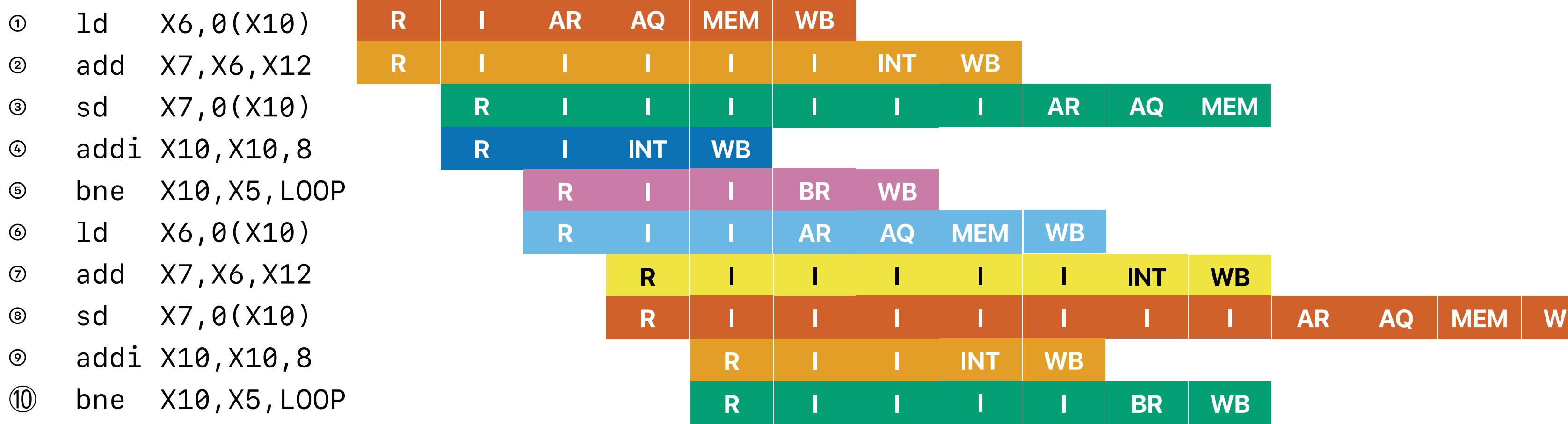
2-issue RR processor in motion

2-issue RR processor in motion



Renamed instruction	Physical Register	Valid			Valid		
		Value	In use		Value	In use	
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	1	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



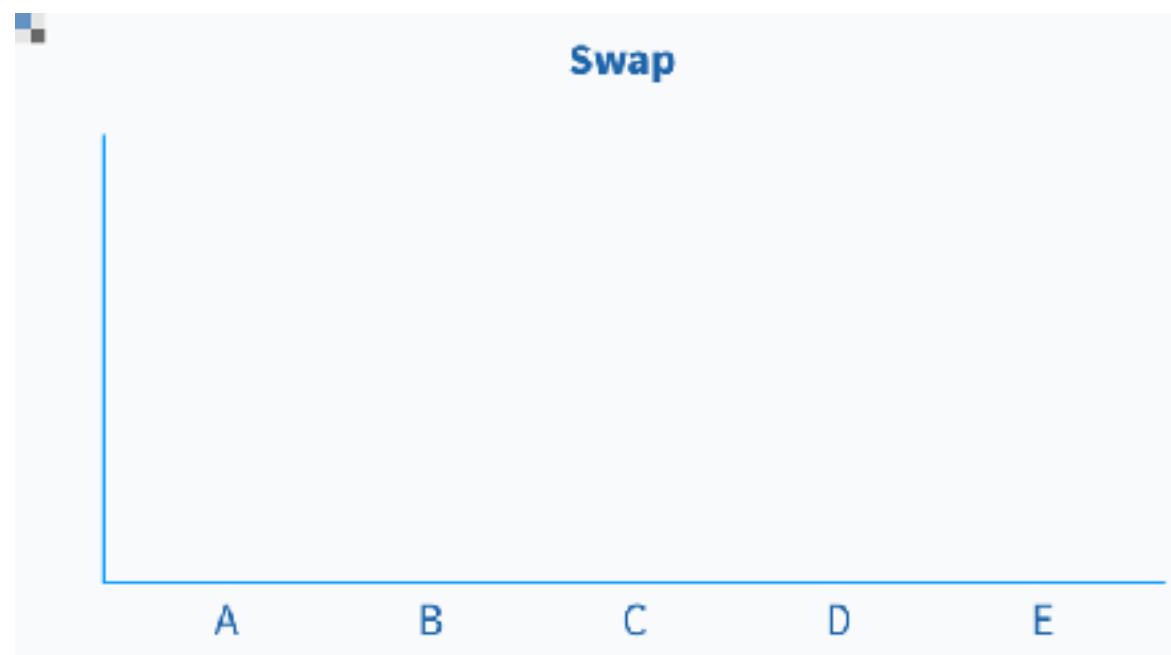
Renamed instruction		Physical Register		Valid Value In use			Valid Value In use		
1	1d P1, 0(X10)	X5		P1	1	1	P6		
2	add P2, P1, X12	X6	P1	P2	1	1	P7		
3	sd P2, 0(X10)	X7	P5	P3	1	1	P8		
4	addi P3, X10, 8	X10	P3	P4	1	1	P9		
5	bne P3, X5, LOOP	X12		P5	1	1	P10		
6	1d P4, 0(P3)								
7	add P5, P1, X12		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

Revisit the swap

- For the following RISC-V implementation of the swap function using XOR, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction. This processor fetches 2 instruction per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

① 1d X6, 0(X10)
② 1d X7, 0(X11)
③ add X8, X6, X0
④ add X6, X7, X0
⑤ add X7, X8, X0
⑥ sd X6, 0(X10)
⑦ sd X7, 0(X11)

- A. 5
- B. 7
- C. 9
- D. 11
- E. 13



Announcements

- Assignment #3 due this **Wednesday**
- Reading Quiz due next Monday
- Project is released
 - Please check website to the link of GitHub repo
 - You may discuss, but each needs an individual/distinguishable version of code
 - You need to write a brief report
 - Grading rubrics
 - 20% — report
 - 20% — if your code can compile and run
 - 60% — performance based. The sample prefetcher is the baseline. We calculate your score at this part using $\min(\text{Speedup}-1, 1)$. If you can speedup by 2, you score full credits in this part
 - Due 11/29 — **no extension**

Computer Science & Engineering

203

つづく

