Dynamic Instruction Scheduling (II)

Hung-Wei Tseng

Recap: Three pipeline hazards

- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that's not yet generated or propagated when the instruction needs that

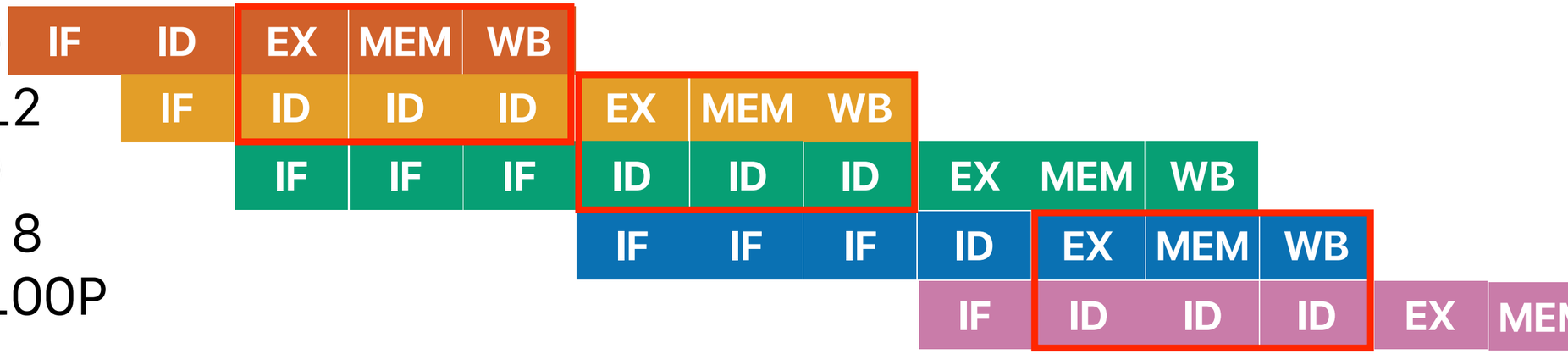
Recap: addressing hazards

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

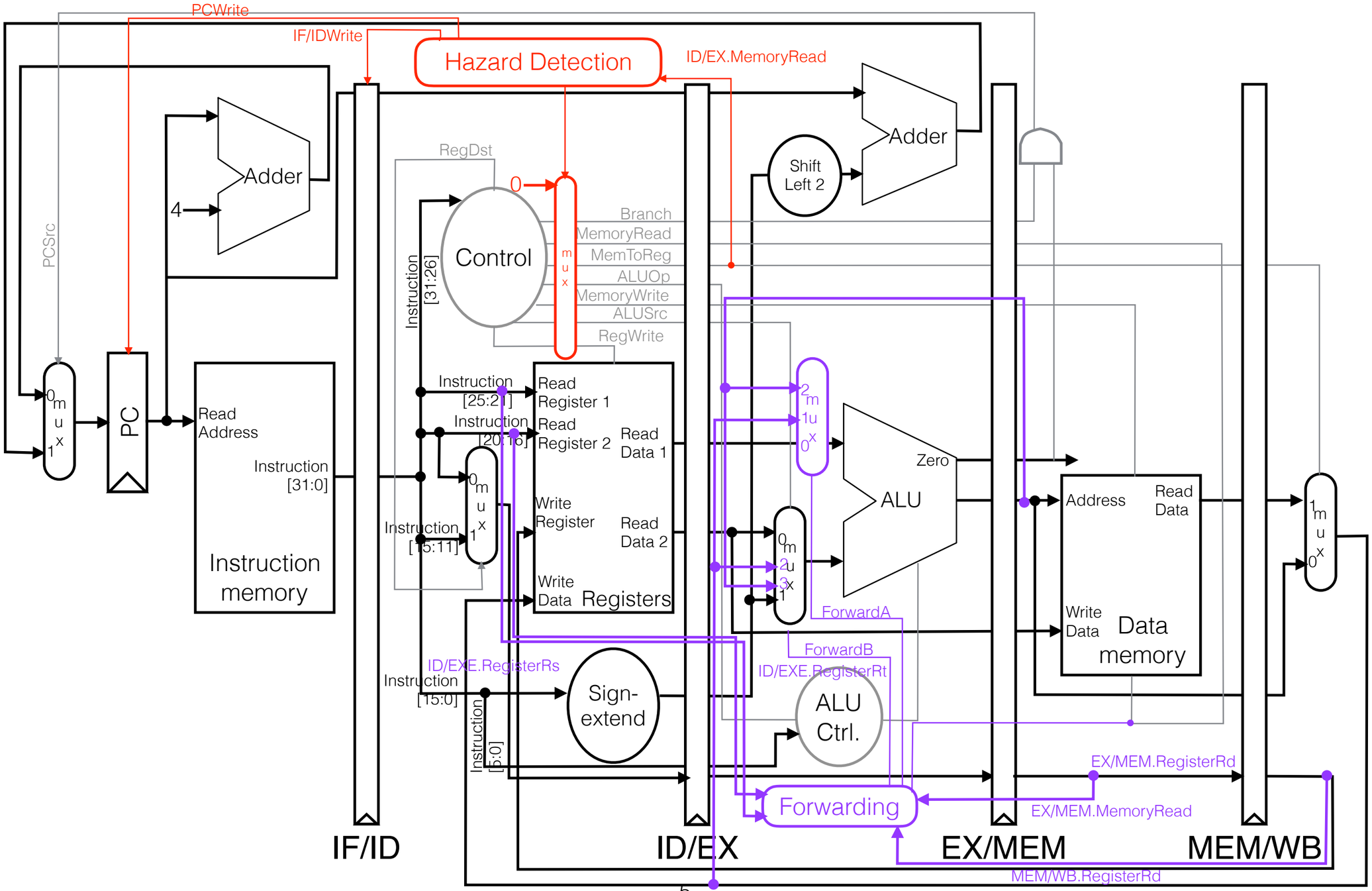
How many of data hazards?

- How many pairs of instructions in the following RISC-V instructions will results in data hazards/stalls in a basic 5-stage RISC-V pipeline?

```
ld   X6,0(X10)
add  X7,X6, X12
sd   X7,0(X10)
addi X10,X10, 8
bne  X10,X5, LOOP
```



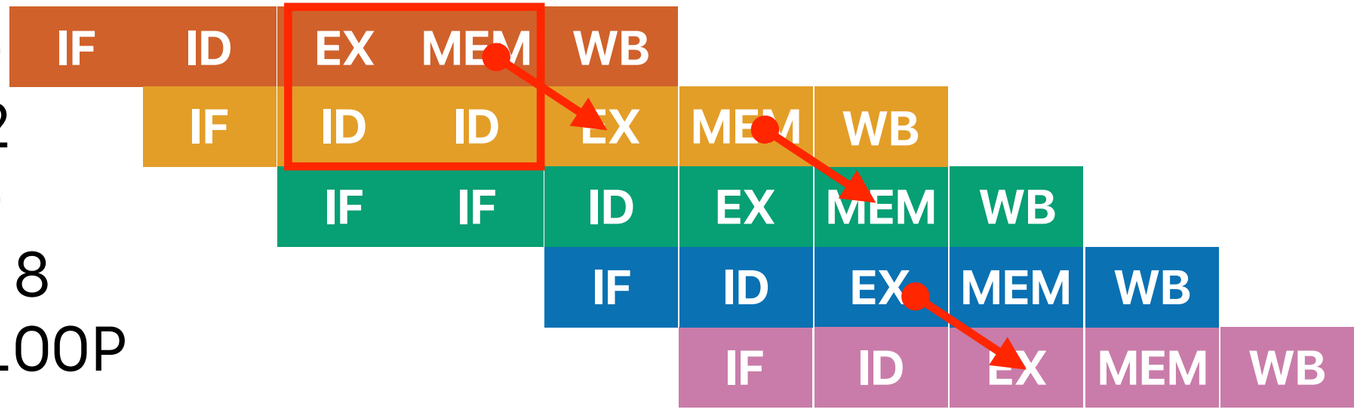
A. 1
B. 2
C. 3
D. 4
E. 5


PCWrite
IF/IDWrite
Hazard Detection
ID/EX.MemoryRead
Adder
4
Adder
Shift Left 2
PCSrc
RegDst
0
Branch
MemoryRead
MemToReg
ALUOp
MemoryWrite
ALUSrc
RegWrite
Control
mux
Instruction [31:26]
PC
Read Address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Read Register 1
Read Register 2
Write Register
Write Data
Read Data 1
Read Data 2
Registers
2
1
0
ALU
Zero
ForwardA
ForwardB
ID/EXE.RegisterRs
ID/EXE.RegisterRt
Instruction [15:0]
Instruction [5:0]
Sign-extend
ALU Ctrl.
Address
Read Data
Write Data
Data memory
EX/MEM.RegisterRd
EX/MEM.MemoryRead
Forwarding
MEM/WB.RegisterRd
IF/ID
ID/EX
EX/MEM
MEM/WB

Do we still have to stall?

- How many pairs of instructions in the following RISC-V instructions will results in data hazards/stalls in a basic 5-stage RISC-V pipeline with "full" data forwarding?

```
ld   X6,0(X10)
add  X7,X6,X12
sd   X7,0(X10)
addi X10,X10, 8
bne  X10,X5, LOOP
```



A. 0

B. 1

C. 2

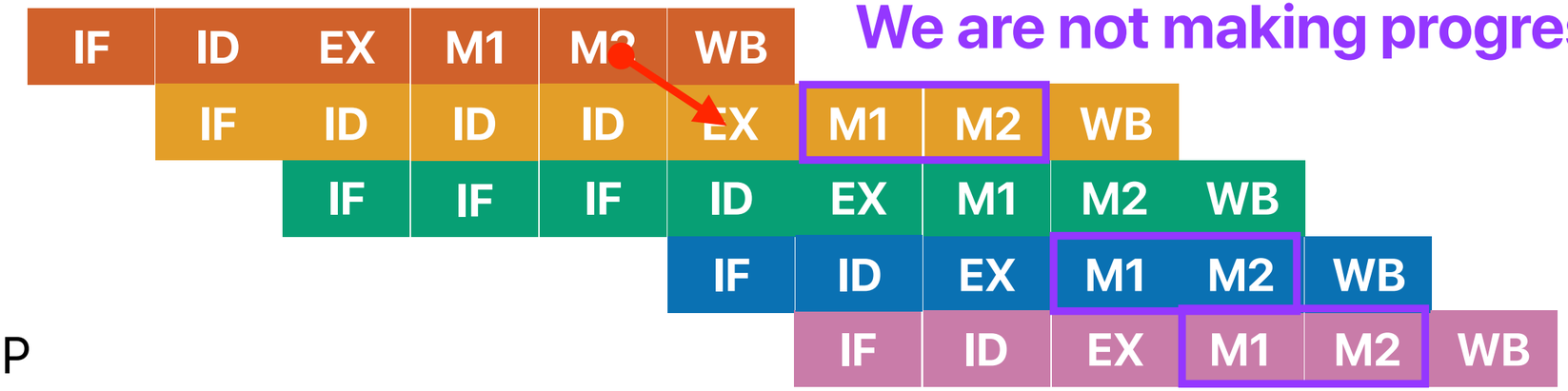
D. 3

E. 4

Problems with data forwarding

- What if our pipeline gets deeper? — Considering a newly designed pipeline where memory stage is split into 2 stages and the memory access finishes at the 2nd memory stage. By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

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`



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
```

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

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

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

Outline

- Concepts of dynamic scheduling
- Tomasulo Algorithm
- Register renaming
- SuperScalar
- Reorder buffer & Speculative execution

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

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.

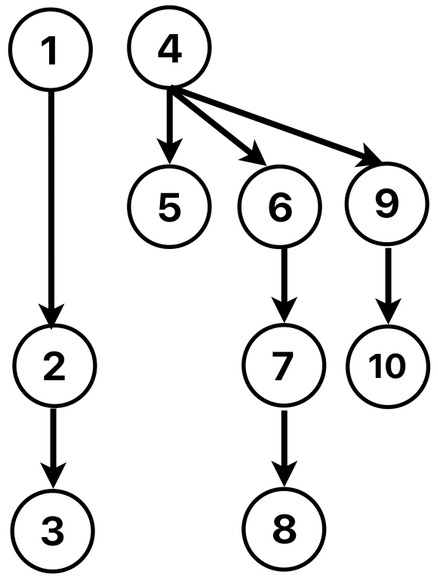
What do you need to execution an instruction?

- 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

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
```

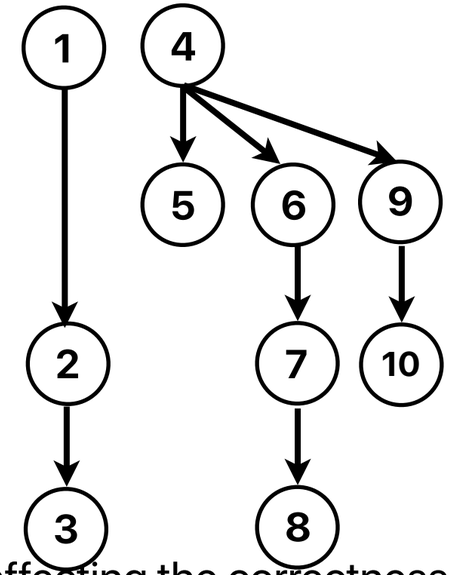


- **In theory**, instructions without dependencies can be executed in parallel or out-of-order
- Instructions with dependencies can never be reordered

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**?

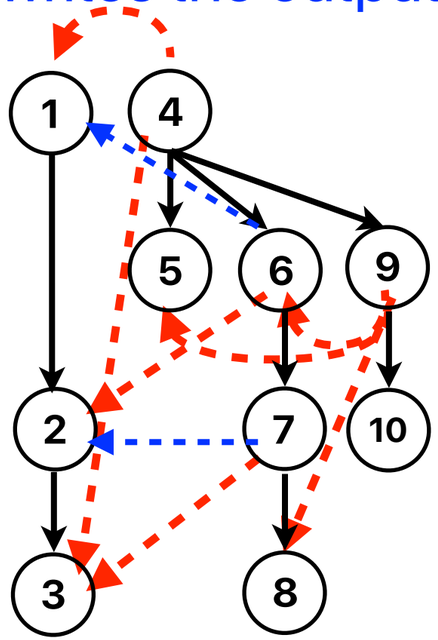
A. (2) and (4)
B. (3) and (5)
C. (5) and (6)
D. (6) and (9)
E. (9) and (10)

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

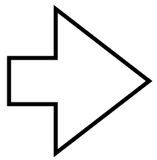
```
①   ld   X6,0(X10)
②   add  X7,X6,X12
③   sd   X7,0(X10)
④   addi X10,X10,8
⑤   bne  X10,X5,LOOP
⑥   ld   X6,0(X10)
⑦   add  X7,X6,X12
⑧   sd   X7,0(X10)
⑨   addi X10,X10,8
⑩   bne  X10,X5,LOOP
```

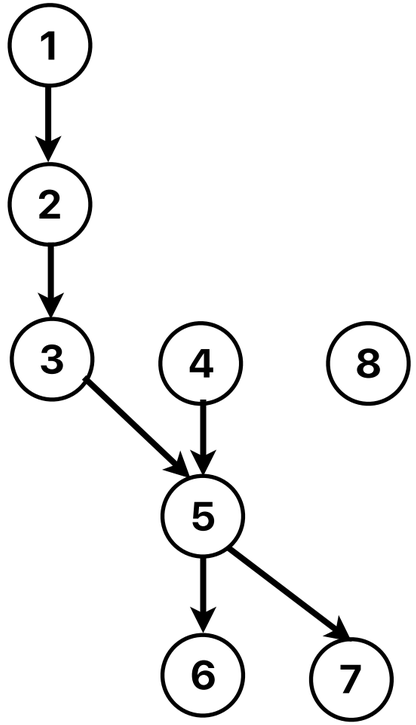


False dependencies

- 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
```



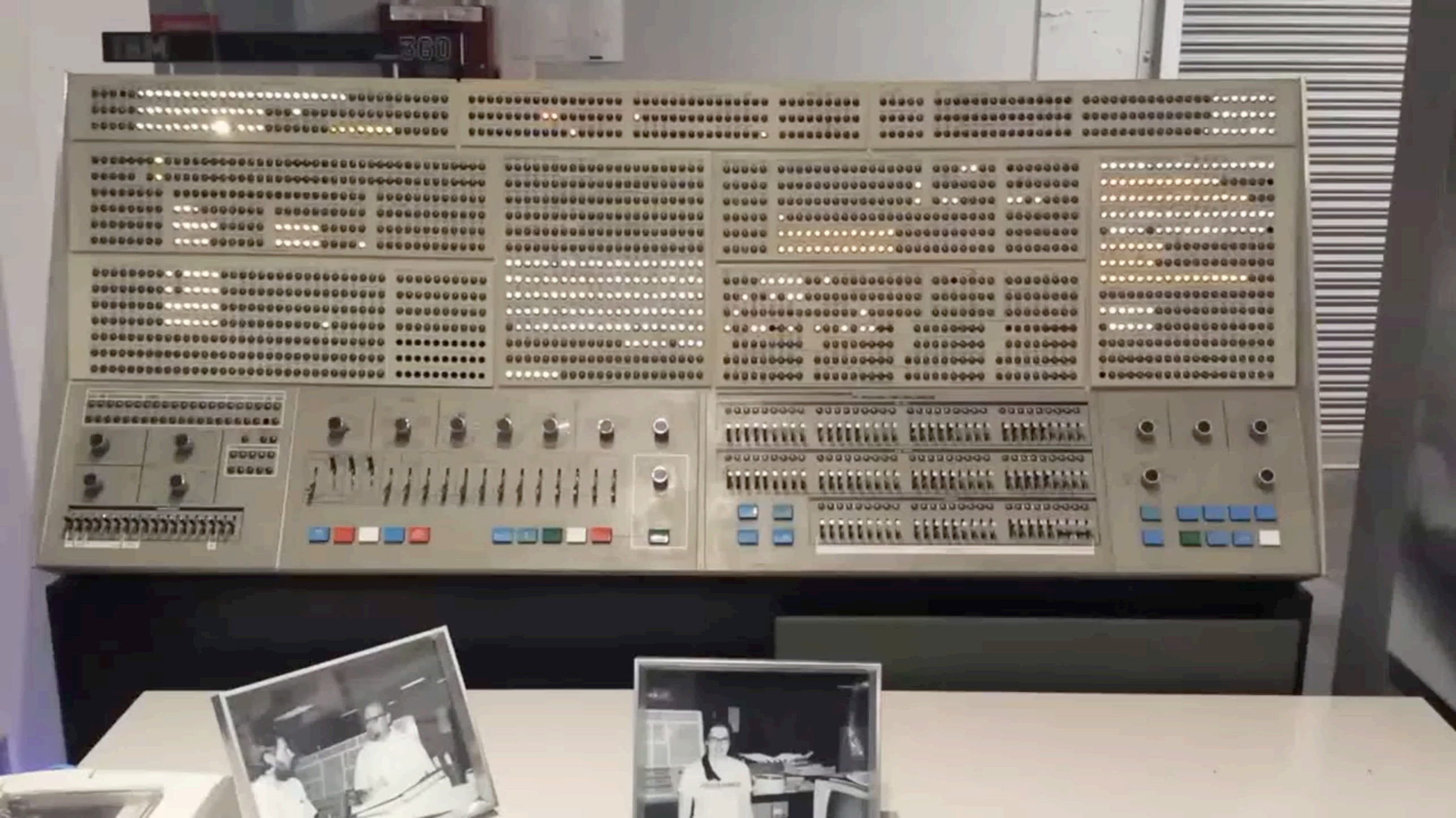


which of the following pair is not a "false dependency"

A. (1) and (4) — 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?


IBM
360

Tomasulo’s Algorithm

IBM 360 Model 91

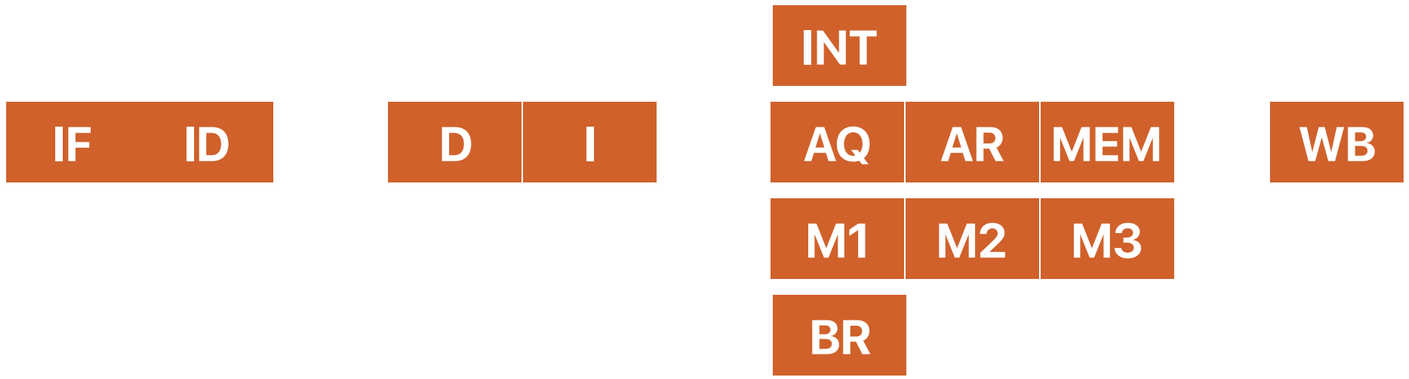
- The most powerful commercialized machine in 1968
-

CDC 6600

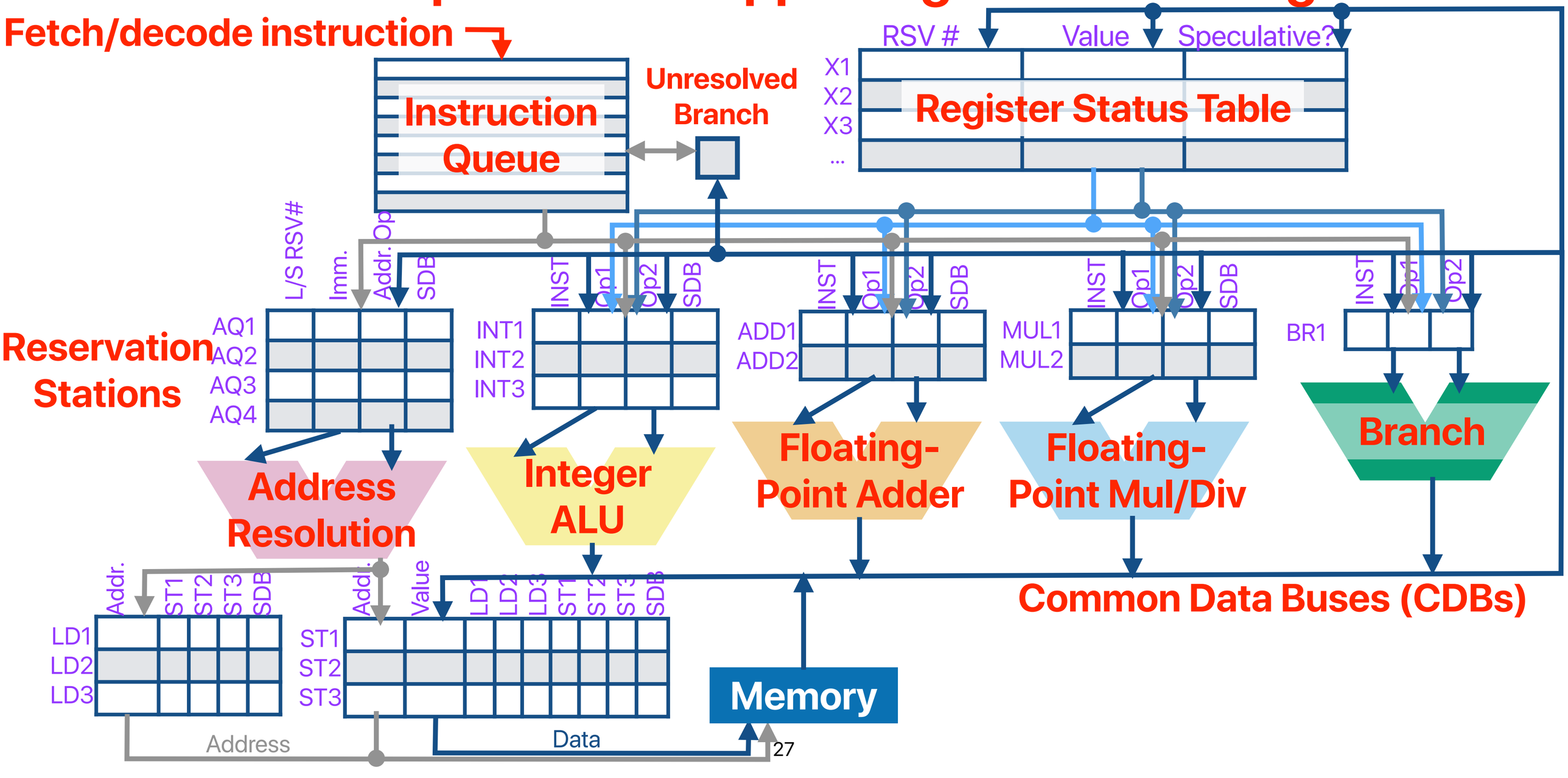
- The first machine with out-of-order execution.
- With performance of up to 3 “M”FLOPS
- It uses score-boarding to achieve so.
  - Instruction storage added to each functional execution unit
  - Instructions issue to FU when no structural hazards, begin execution when dependences satisfied. Thus, instructions 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
```

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1									
LD2									
LD3									
ST1									
ST2									
ST3									
ADD1									
ADD2									
MUL1									
MUL2									
BR									

	RSV #	Value	Spec?
X5			
X6			
X7			
X10			
X12			

Tomasulo in motion

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
```

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	ld	0	[X10]						1
LD2									
LD3									
ST1									
ST2									
ST3									
ADD1									
ADD2									
MUL1									
MUL2									
BR									

	RSV #	Value	Spec?
X5			
X6	LD1		
X7			
X10			
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
```

	D	AQ
	D	

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	ld	0	[X10]						1
LD2									
LD3									
ST1									
ST2									
ST3									
ADD1	add		[X12]		LD1				2
ADD2									
MUL1									
MUL2									
BR									

	RSV #	Value	Spec?
X5			
X6	LD1		
X7	ADD1		
X10			
X12			

Tomasulo in motion

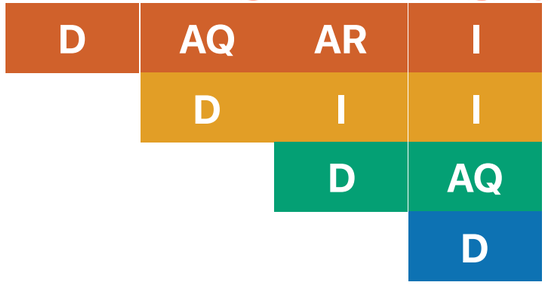
① ld X6,0(X10)
② add X7,X6,X12
③ sd X7,0(X10)
④ addi X10,X10,8
⑤ bne X10,X5,LOOP
⑥ ld X6,0(X10)
⑦ add X7,X6,X12
⑧ sd X7,0(X10)
⑨ addi X10,X10,8
⑩ bne X10,X5,LOOP

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

	RSV #	Value	Spec?
X5			
X6	LD1		
X7	ADD1		
X10			
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
```

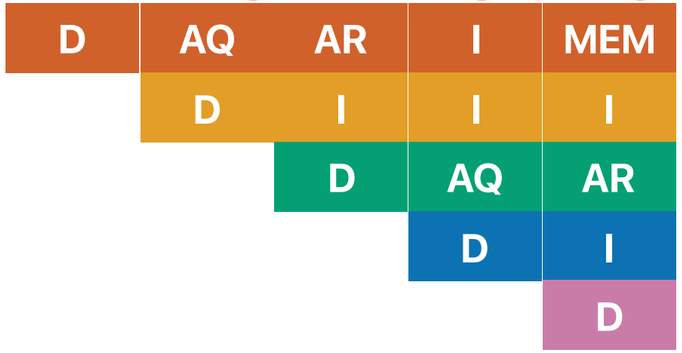


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	ld	0	[X10]						1
LD2									
LD3									
ST1	sd	0	[X10]				ADD1		3
ST2									
ST3									
ADD1	add		[X12]		LD1				2
ADD2	addi	8	[X10]						4
MUL1									
MUL2									
BR									

	RSV #	Value	Spec?
X5			
X6	LD1		
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
```

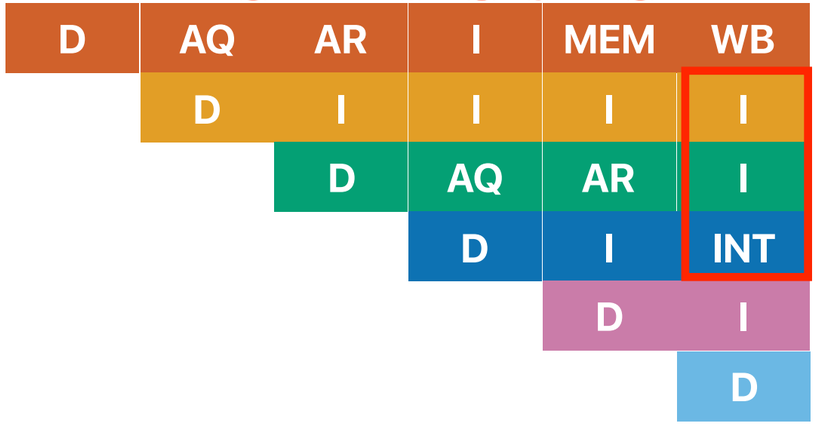


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	ld	0	[X10]						1
LD2									
LD3									
ST1	sd	0	[X10]				ADD1		3
ST2									
ST3									
ADD1	add		[X12]		LD1				2
ADD2	addi	8	[X10]						4
MUL1									
MUL2									
BR	bne		[X5]		ADD2				5

	RSV #	Value	Spec?
X5			
X6	LD1		
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
```



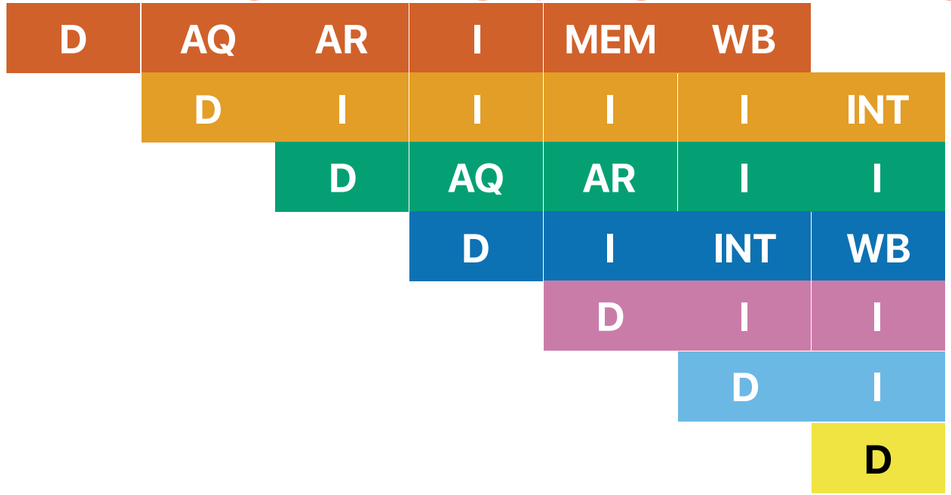
addi is now ahead of **add**!

	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
```



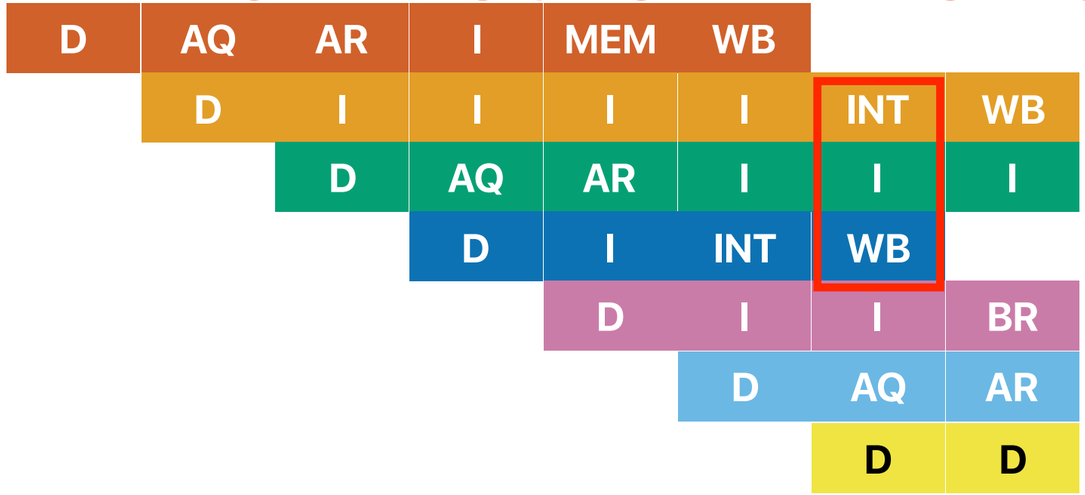
no reservation station for add!

	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
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	ADD2	[X5]						5

	RSV #	Value	Spec?
X5			
X6	LD2		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
```



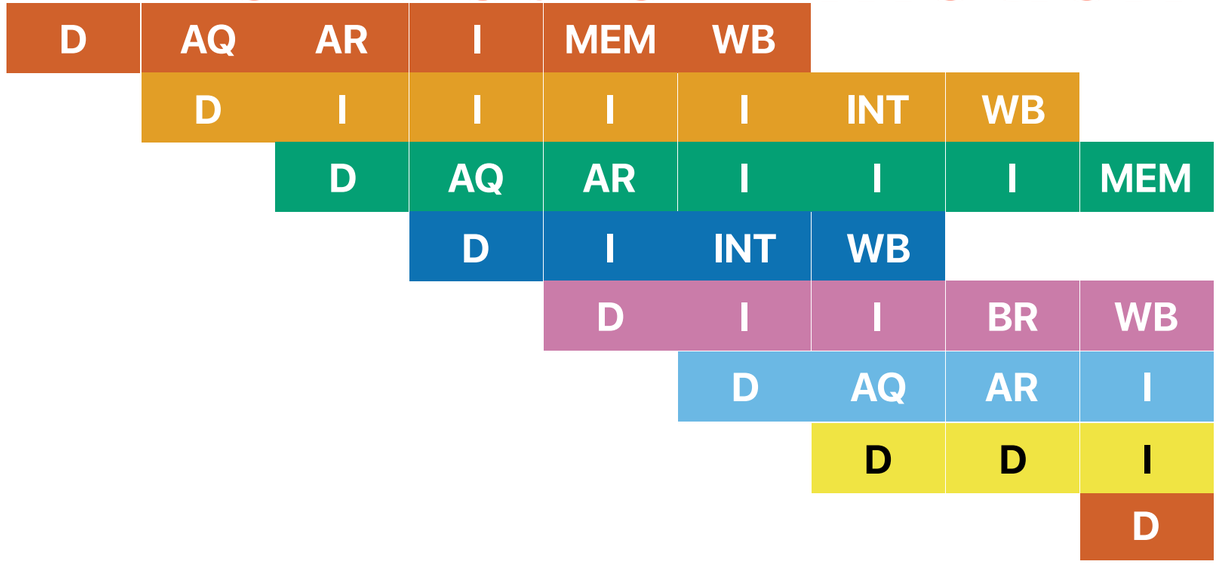
addi is finished ahead of **add** and **sd**!

	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]						5

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

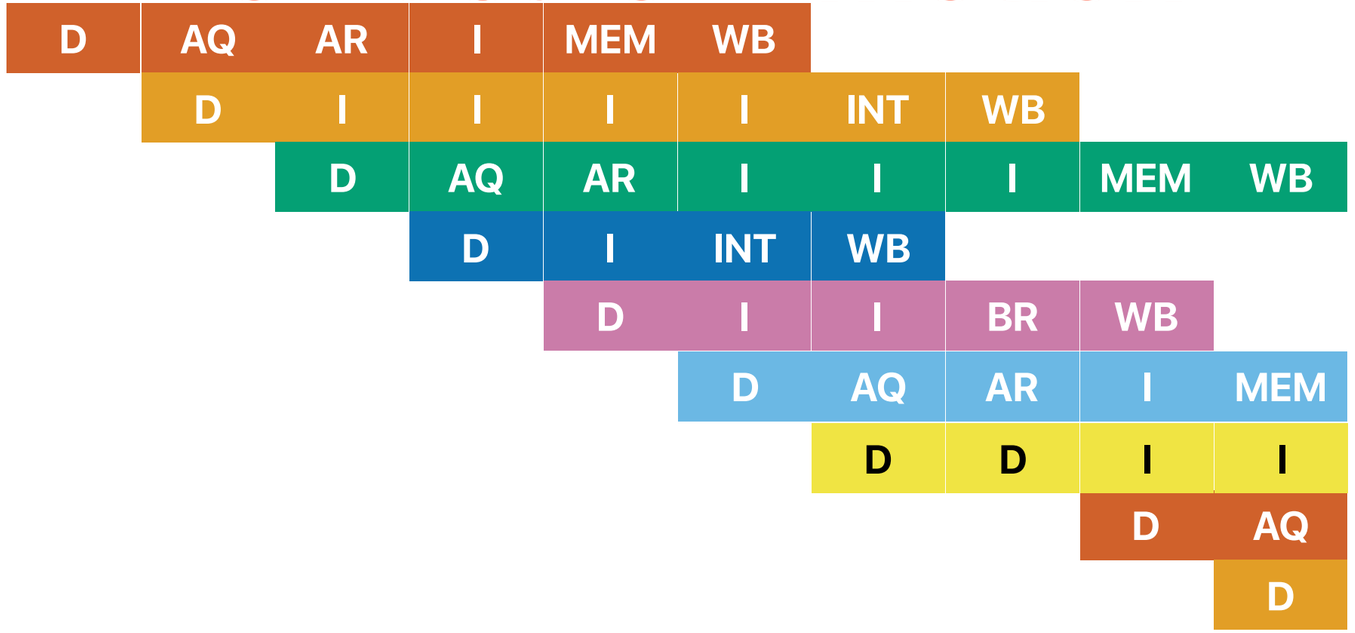


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	ld	0	ADD2						6
LD3									
ST1	sd	0	[X10]	ADD1					3
ST2	sd	0	[X10]				ADD2		8
ST3									
ADD1	~~add~~	~~LD1~~	~~[X12]~~						~~2~~
ADD2	add		[X12]		[LD2]				7
MUL1									
MUL2									
BR	~~bne~~	~~ADD2~~	~~[X5]~~						~~5~~

	RSV #	Value	Spec?
X5			
X6	LD2		
X7	ADD2		
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
```

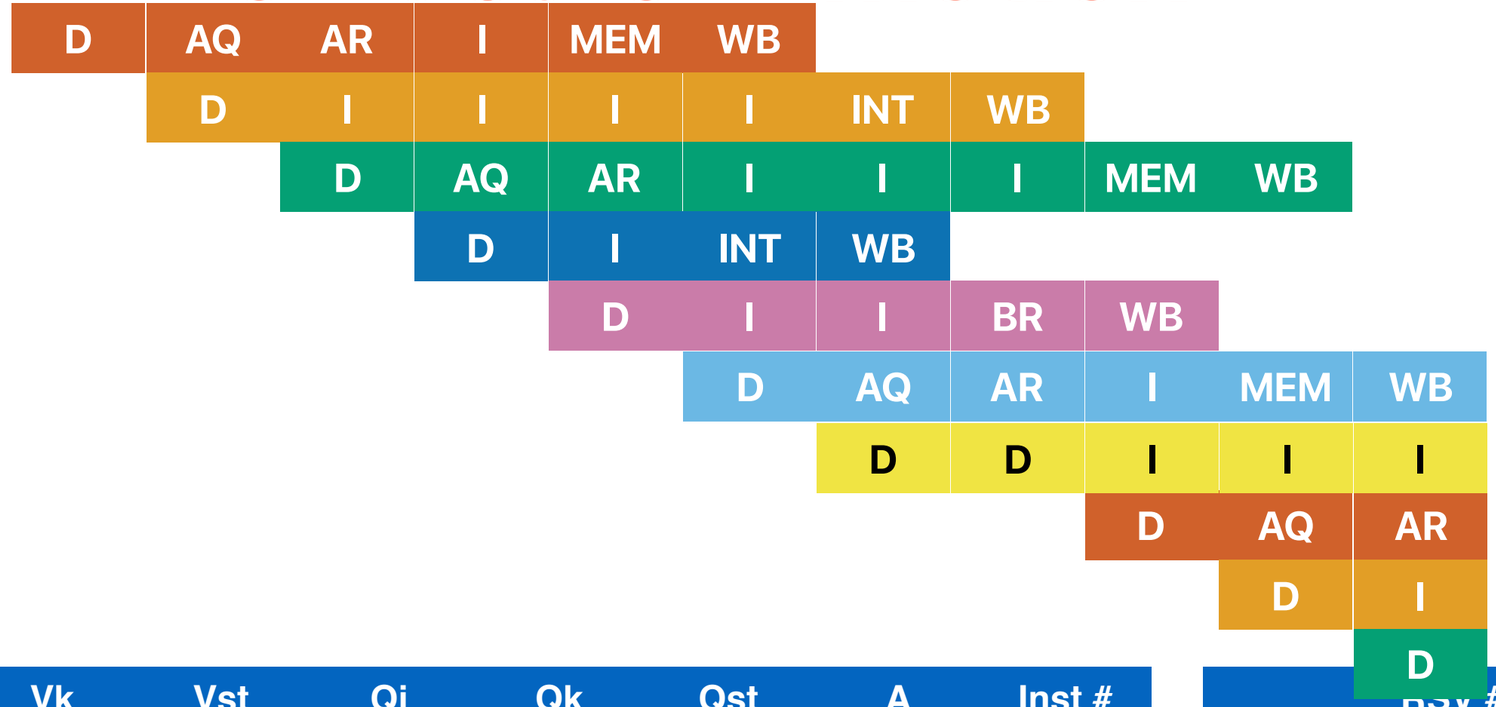


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	ld	0	ADD2						6
LD3									
ST1	sd	0	[X10]	ADD1					3
ST2	sd	0	[X10]				ADD2		8
ST3									
ADD1	add	8	ADD2						9
ADD2	add		[X12]		[LD2]				7
MUL1									
MUL2									
BR	~~bne~~	~~ADD2~~	~~[X5]~~						~~5~~

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

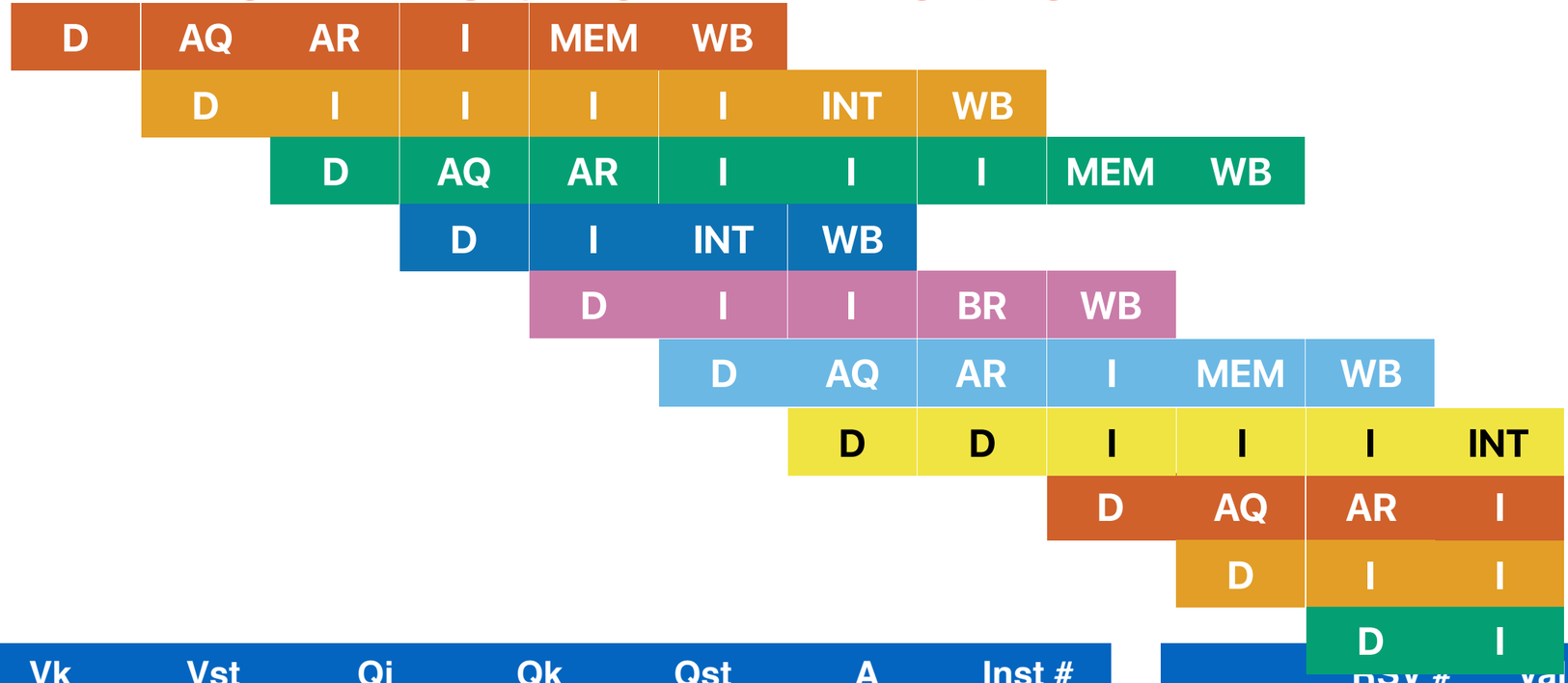


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	ld	0	ADD2						6
LD3									
ST1	~~sd~~	~~0~~	~~[X10]~~	~~ADD1~~					~~3~~
ST2	sd	0	[X10]				ADD2		8
ST3									
ADD1	add	8	ADD2						9
ADD2	add		[X12]		[LD2]				7
MUL1									
MUL2									
BR	br		[X5]		ADD1				10

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

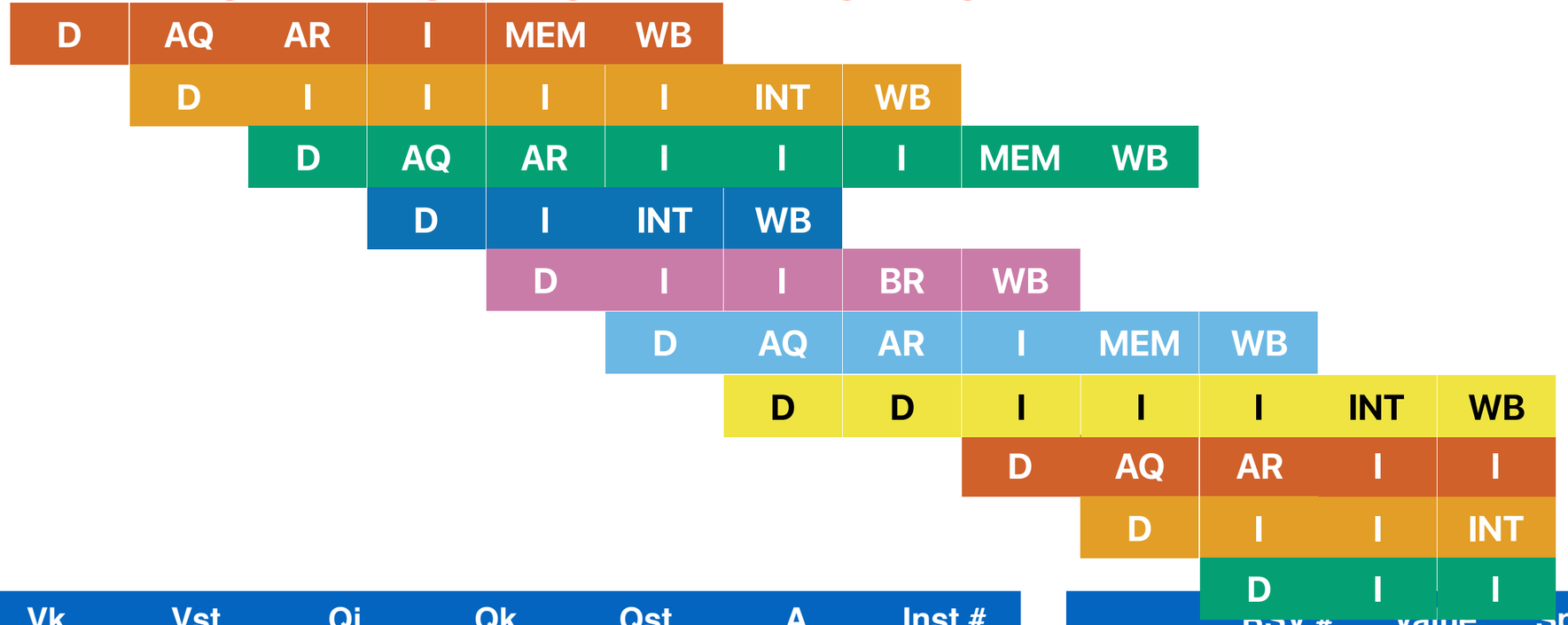


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	~~ld~~	~~0~~	~~ADD2~~						~~6~~
LD3									
ST1	~~sd~~	~~0~~	~~[X10]~~	~~ADD1~~					~~3~~
ST2	sd	0	[X10]				ADD2		8
ST3									
ADD1	add	8	ADD2						9
ADD2	add		[X12]		[LD2]				7
MUL1									
MUL2									
BR	br		[X5]		ADD1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

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

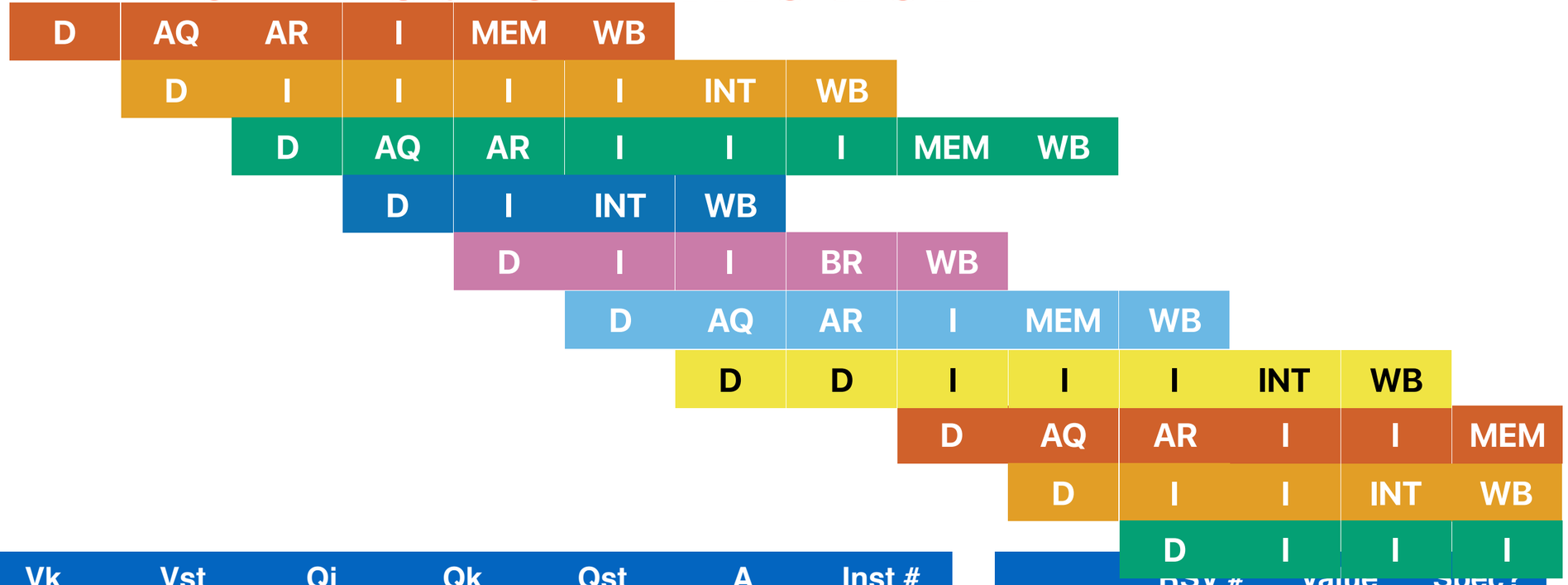


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	~~ld~~	~~0~~	~~ADD2~~						~~6~~
LD3									
ST1	~~sd~~	~~0~~	~~[X10]~~	~~ADD1~~					~~3~~
ST2	sd	0	[X10]	ADD2					8
ST3									
ADD1	add	8	ADD2						9
ADD2	add		[X12]		[LD2]				7
MUL1									
MUL2									
BR	br		[X5]		ADD1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

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

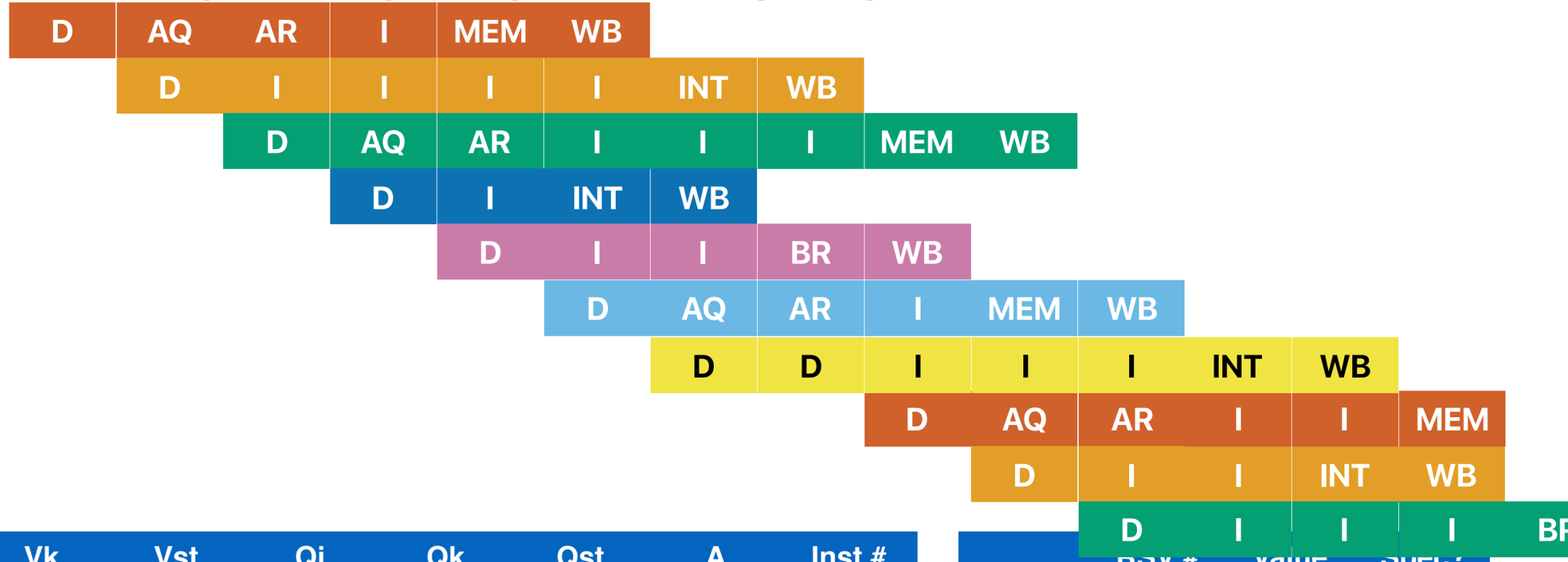


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	~~ld~~	~~0~~	~~ADD2~~						~~6~~
LD3									
ST1	~~sd~~	~~0~~	~~[X10]~~	~~ADD1~~					~~3~~
ST2	sd	0	[X10]	ADD2					8
ST3									
ADD1	add	8	ADD2						9
ADD2	~~add~~		~~[X12]~~		~~[LD2]~~				~~7~~
MUL1									
MUL2								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

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

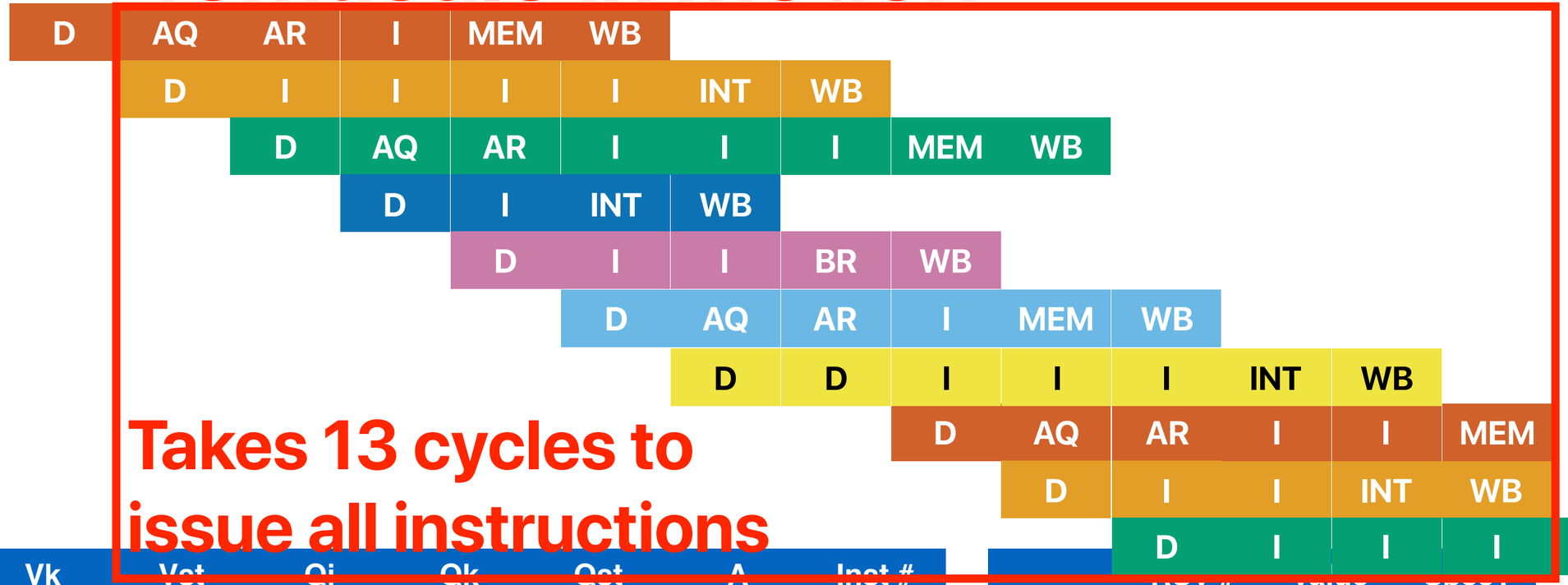


	INST	Vj	Vk	Vst	Qj	Qk	Qst	A	Inst #
LD1	~~ld~~	~~0~~	~~[X10]~~						~~1~~
LD2	~~ld~~	~~0~~	~~ADD2~~						~~6~~
LD3									
ST1	~~sd~~	~~0~~	~~[X10]~~	~~ADD1~~					~~3~~
ST2	~~sd~~	~~0~~	~~[X10]~~	~~ADD2~~					~~8~~
ST3									
ADD1	add	8	ADD2						9
ADD2	~~add~~		~~[X12]~~		~~[LD2]~~				~~7~~
MUL1		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

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



Takes 13 cycles to issue all instructions

	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	~~sd~~	~~0~~	~~[X10]~~	~~ADD2~~					~~8~~
ST3									
ADD1	add	8	ADD2						9
ADD2	~~add~~		~~[X12]~~		~~[LD2]~~				~~7~~
MUL1									
MUL2									
BR	br		[X5]	ADD1					10

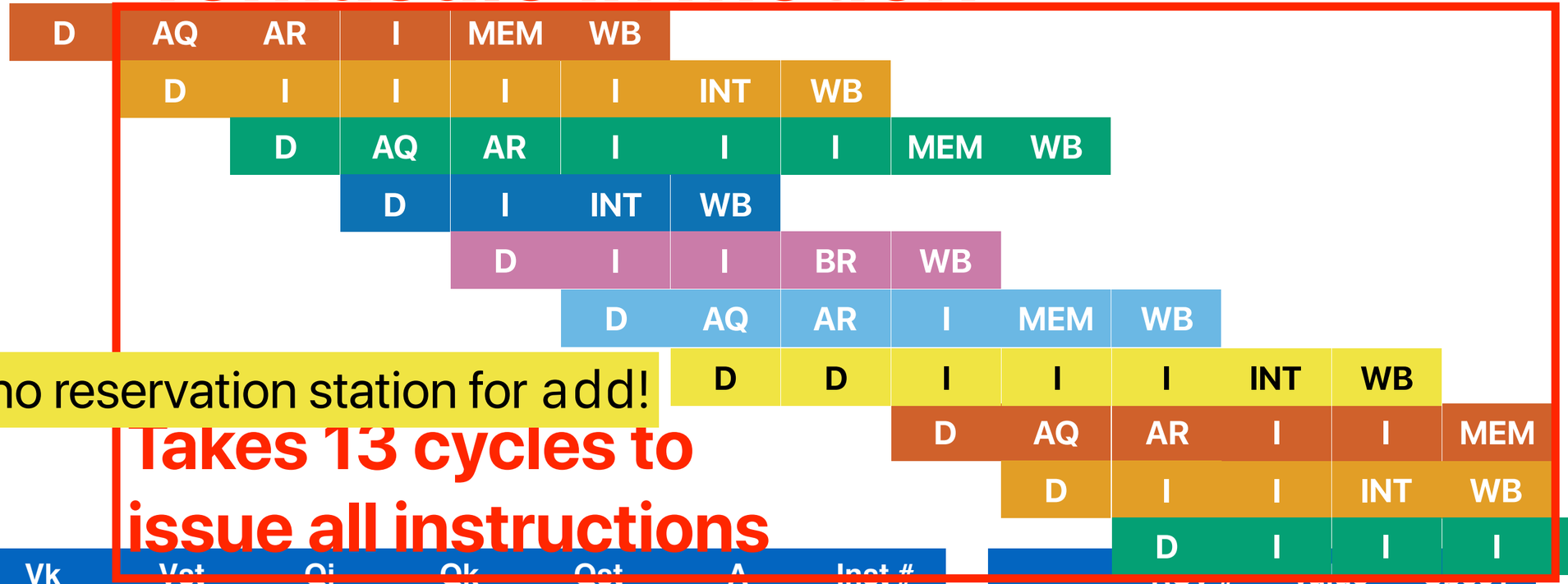
X5	
X6	LD2
X7	ADD2
X10	ADD1
X12	

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)
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!

Takes 13 cycles to issue all instructions

	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	~~sd~~	~~0~~	~~[X10]~~	~~ADD2~~					~~8~~
ST3									
ADD1	add	8	ADD2						9
ADD2	~~add~~		~~[X12]~~		~~[LD2]~~				~~7~~
MUL1									
MUL2									
BR	br		[X5]	ADD1					10

	Inst #	Value	Qest
X5			
X6		LD2	
X7		ADD2	
X10		ADD1	
X12			

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;
}
```

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

4*n instructions

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
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

Only one branch for four iterations in A

Recap: Why is B better than A?

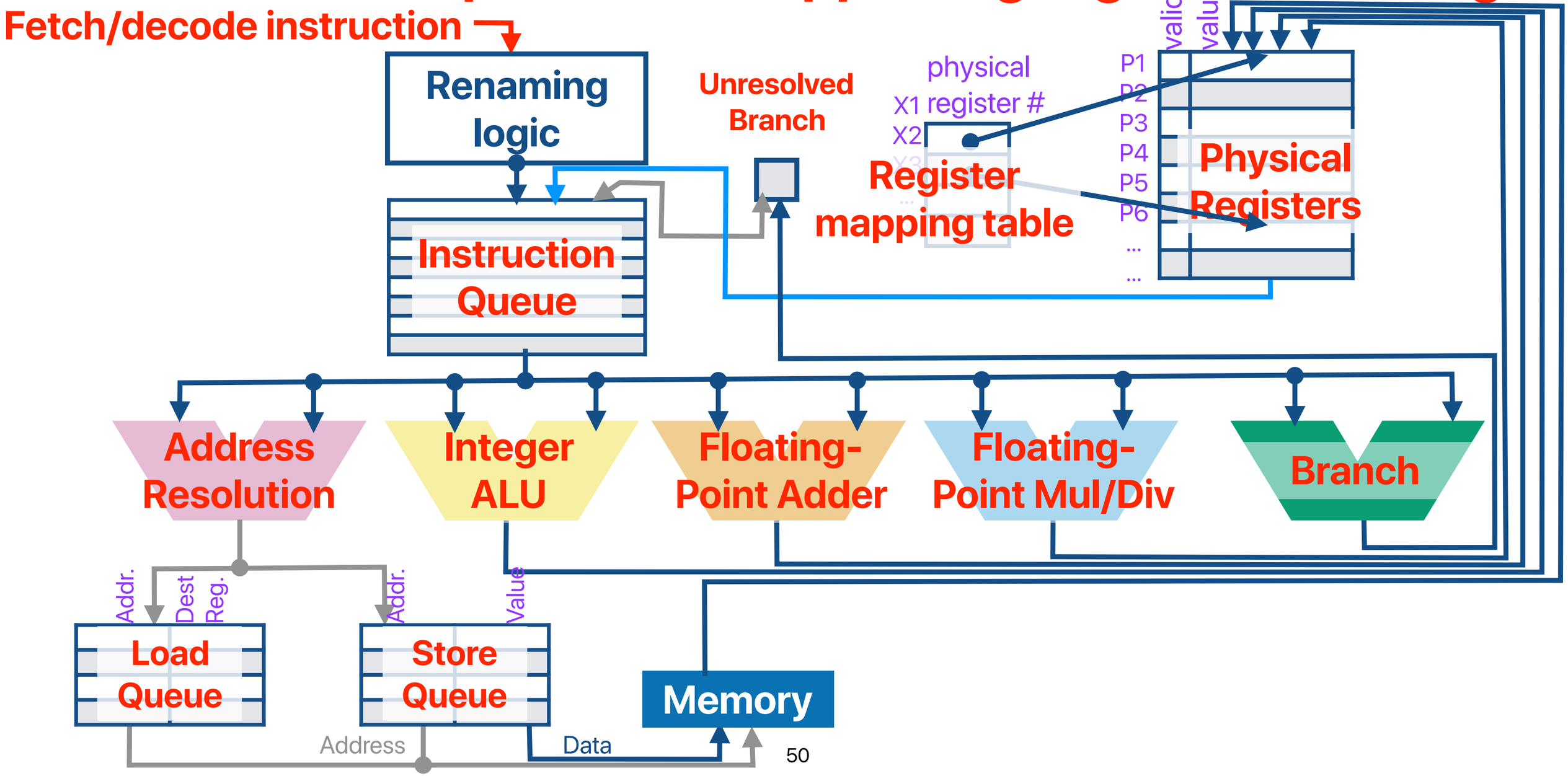
```
and  x2, x1, 1
add  x3, x3, x2
shr  x1, x1, 1
and  x2, x1, 1
add  x3, x3, x2
shr  x1, x1, 1
and  x2, x1, 1
add  x3, x3, x2
shr  x1, x1, 1
and  x2, x1, 1
add  x3, x3, x2
shr  x1, x1, 1
bne  x1, x0, LOOP
```

```
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

R

```
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	
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

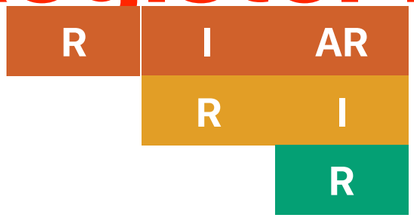

```
①   ld   X6,0(X10)
②   add  X7,X6,X12
③   sd   X7,0(X10)
④   addi X10,X10,8
⑤   bne  X10,X5,LOOP
⑥   ld   X6,0(X10)
⑦   add  X7,X6,X12
⑧   sd   X7,0(X10)
⑨   addi X10,X10,8
⑩   bne  X10,X5,LOOP
```

	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
```

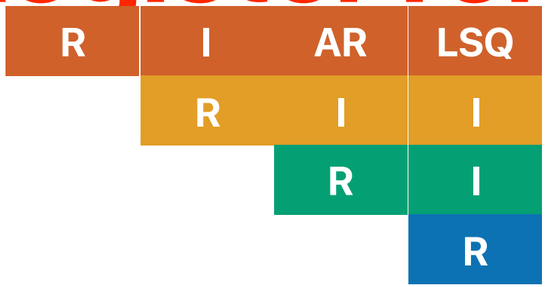
	Renamed instruction
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	
5	
6	
7	
8	
9	
10	

	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

```
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
```



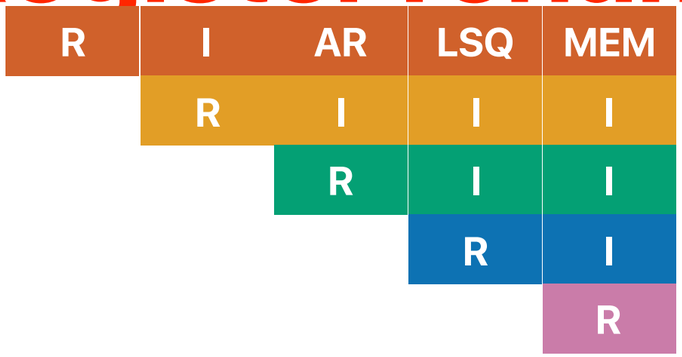
	Renamed instruction
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	
6	
7	
8	
9	
10	

	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			

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
```



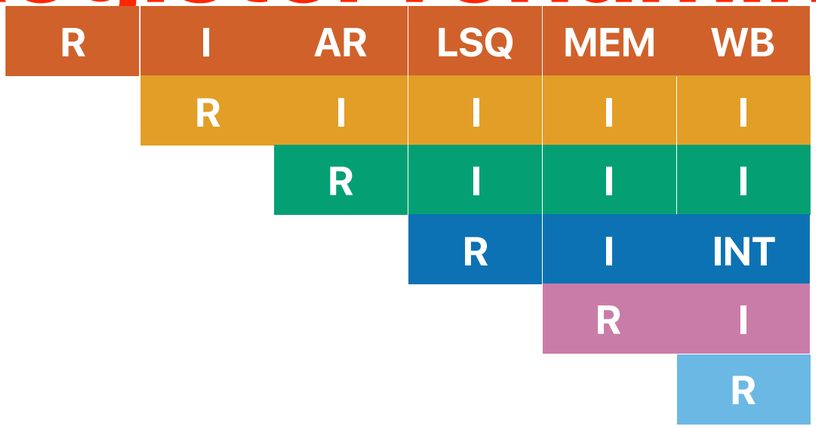
	Renamed instruction
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	
7	
8	
9	
10	

	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

```
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
```



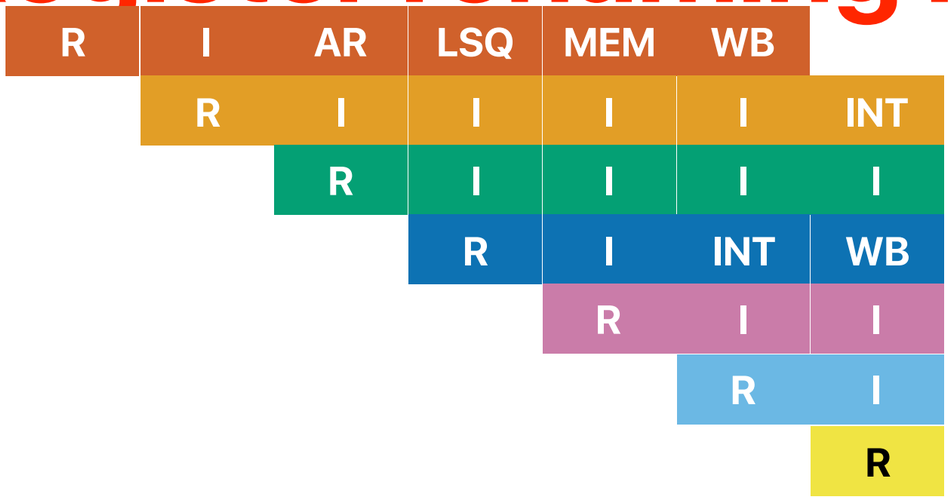
	Renamed instruction
1	ld P1, 0(X10)
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	
8	
9	
10	

	Physical Register
X5	
X6	P1
X7	P2
X10	P3
X12	

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6			
P2	0		1	P7			
P3	0		1	P8			
P4	0		1	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
```



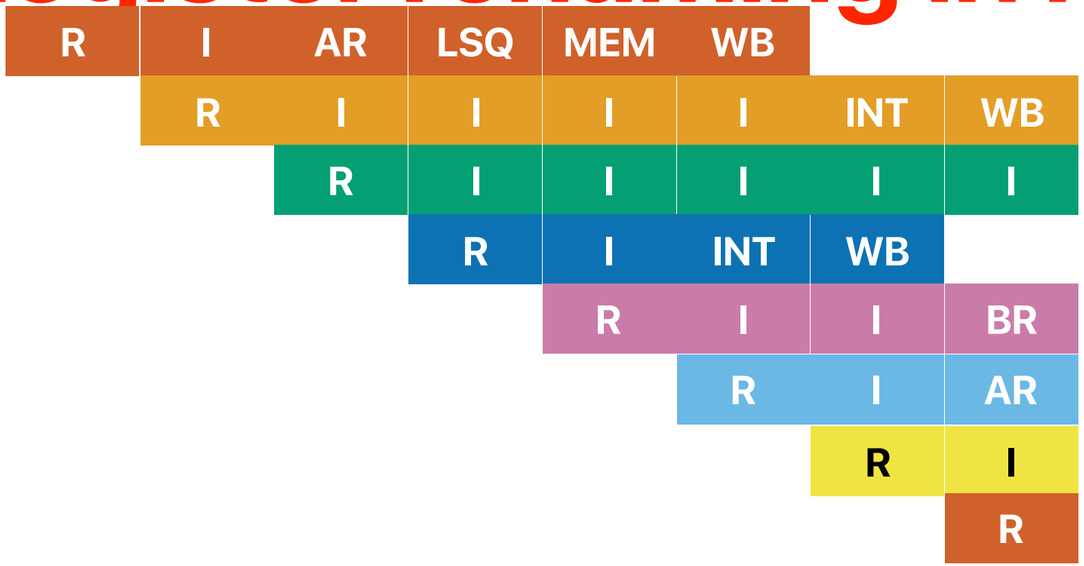
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	add P2, P1, X12
3	sd P2, 0(X10)
4	addi P3, X10, 8
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	
9	
10	

	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

```
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
```



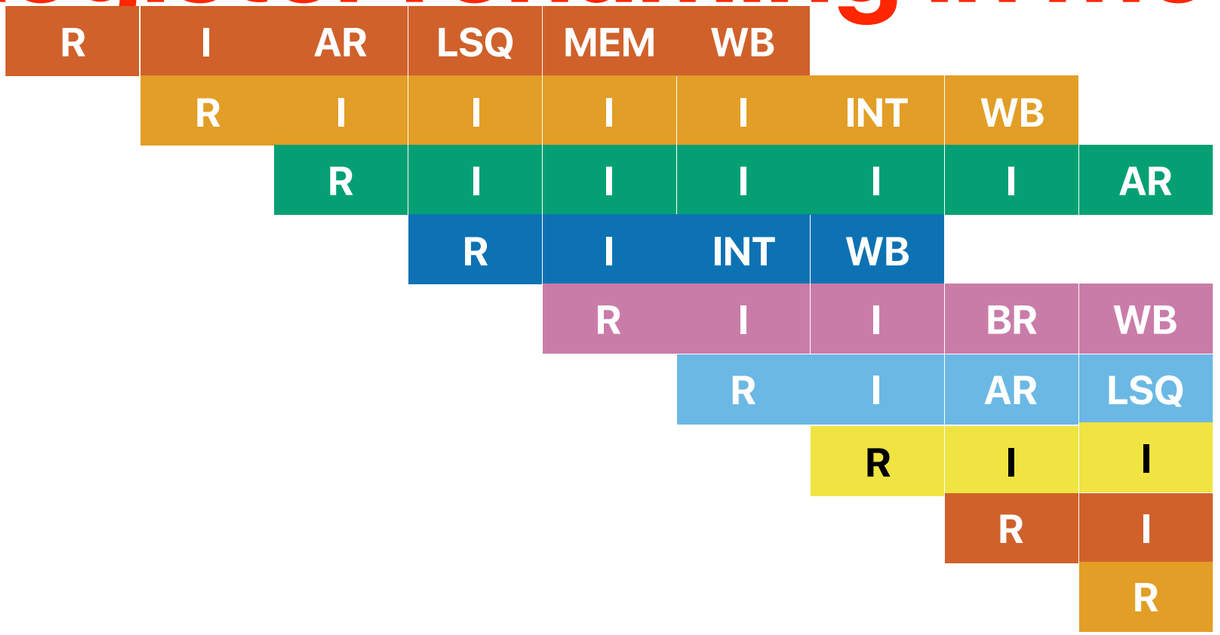
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	add P2, P1, X12
3	sd P2, 0(X10)
4	~~addi P3, X10, 8~~
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	
10	

	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

```
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
```



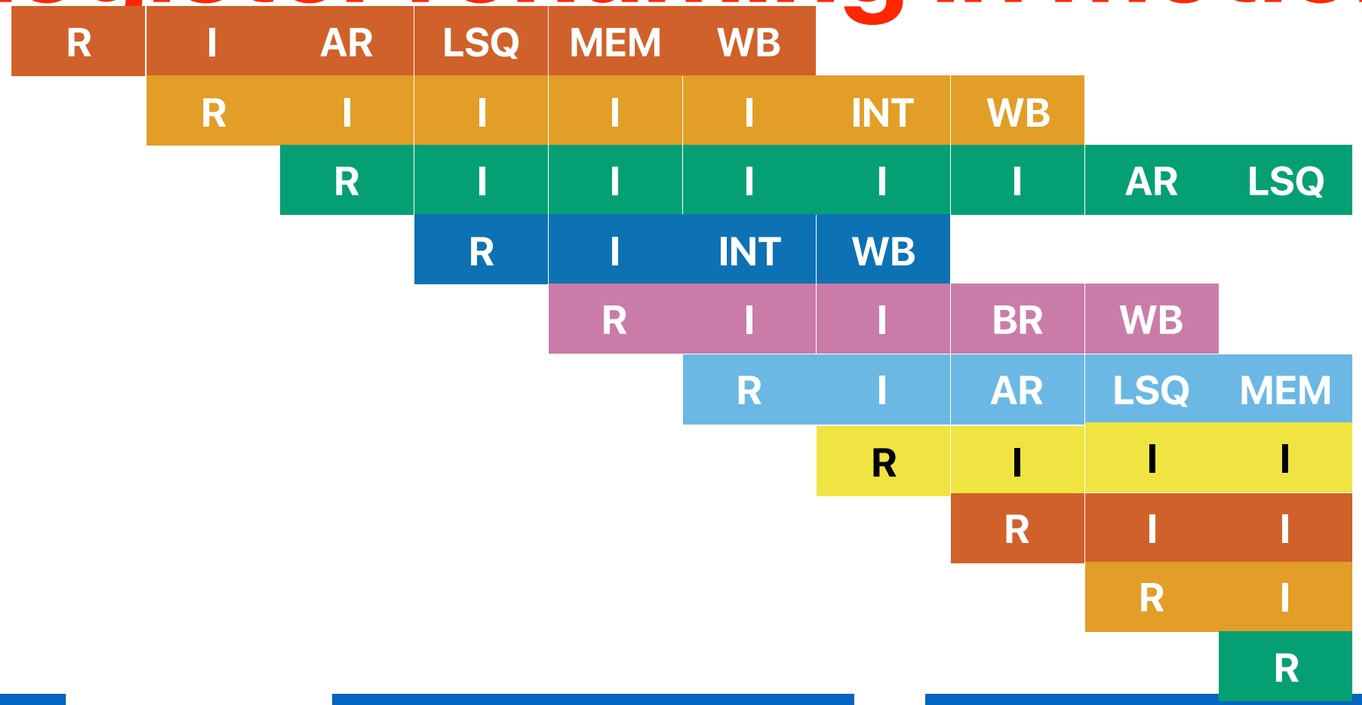
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	~~add P2, P1, X12~~
3	sd P2, 0(X10)
4	~~addi P3, X10, 8~~
5	bne P3, X5, LOOP
6	ld P4, 0(P3)
7	add P5, P1, X12
8	sd P5, 0(P3)
9	addi P6, P3, 8
10	

	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

```
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
```



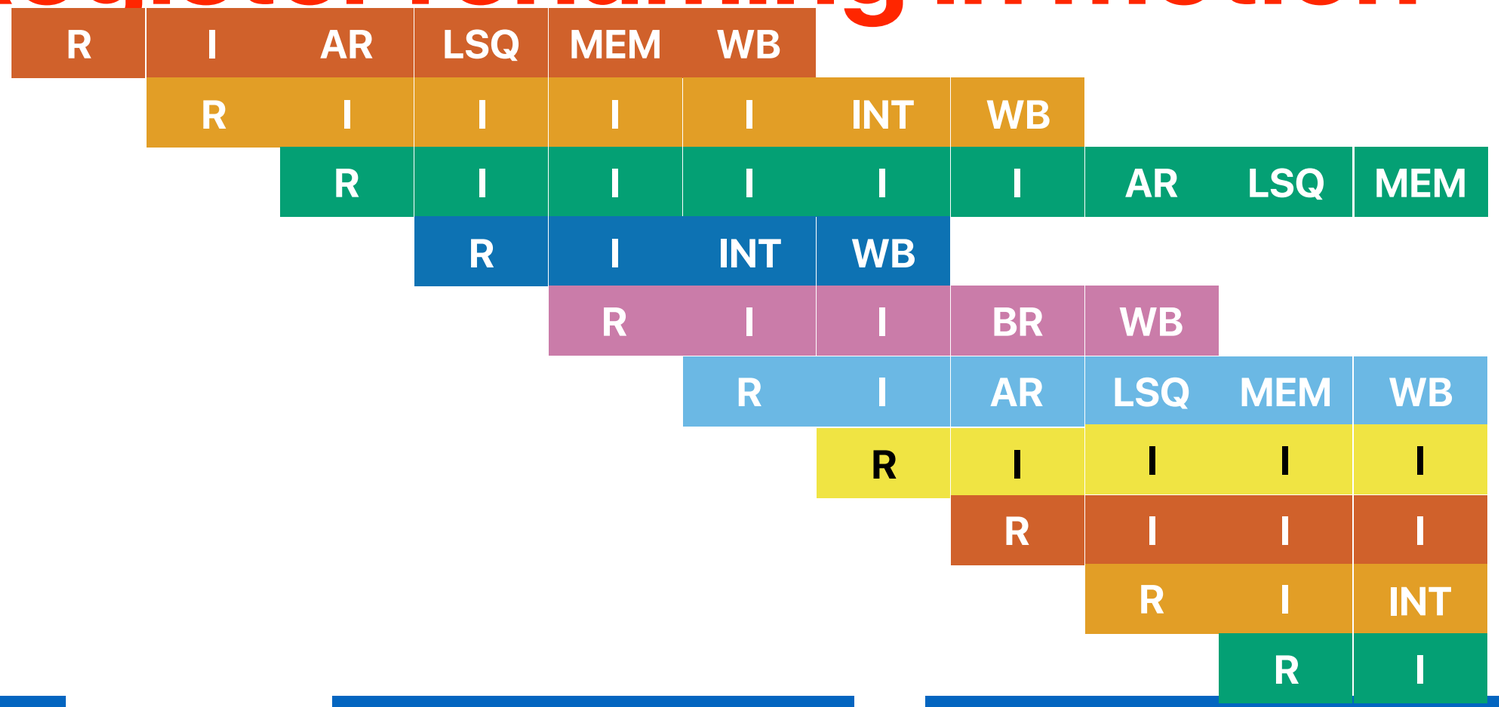
	Renamed instruction		
1	~~ld~~	~~P1, 0(X10)~~	
2	~~add~~	~~P2, P1, X12~~	
3	sd	P2, 0(X10)	
4	~~addi~~	~~P3, X10, 8~~	
5	~~bne~~	~~P3, X5, LOOP~~	
6	ld	P4, 0(P3)	
7	add	P5, P1, X12	
8	sd	P5, 0(P3)	
9	addi	P6, P3, 8	
10	bne	P6, 0(X10)	

	Physical Register
X5	
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

```
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
```



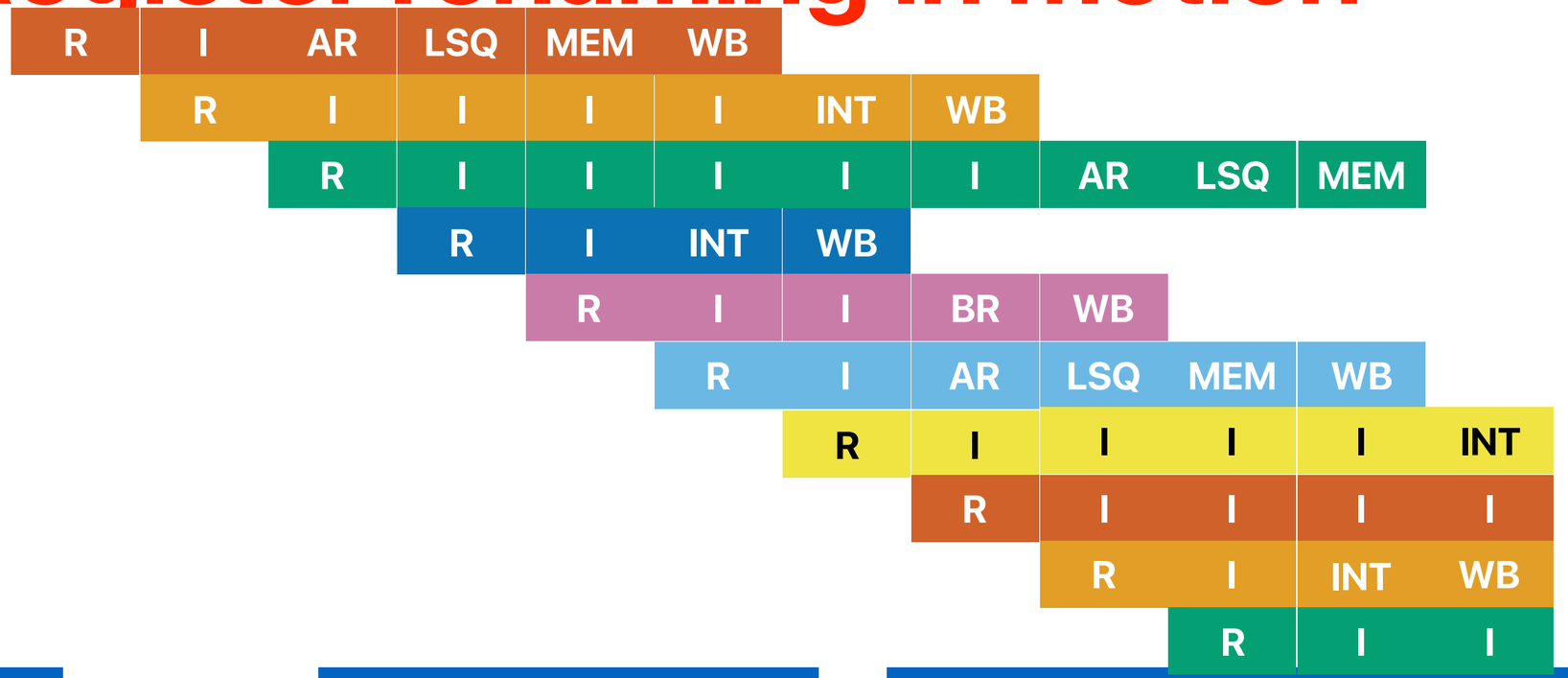
	Renamed instruction		
1	~~ld~~	~~P1, 0(X10)~~	
2	~~add~~	~~P2, P1, X12~~	
3	sd	P2, 0(X10)	
4	~~addi~~	~~P3, X10, 8~~	
5	~~bne~~	~~P3, X5, LOOP~~	
6	ld	P4, 0(P3)	
7	add	P5, P1, X12	
8	sd	P5, 0(P3)	
9	addi	P6, P3, 8	
10	bne	P6, 0(X10)	

	Physical Register
X5	
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

```
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
```



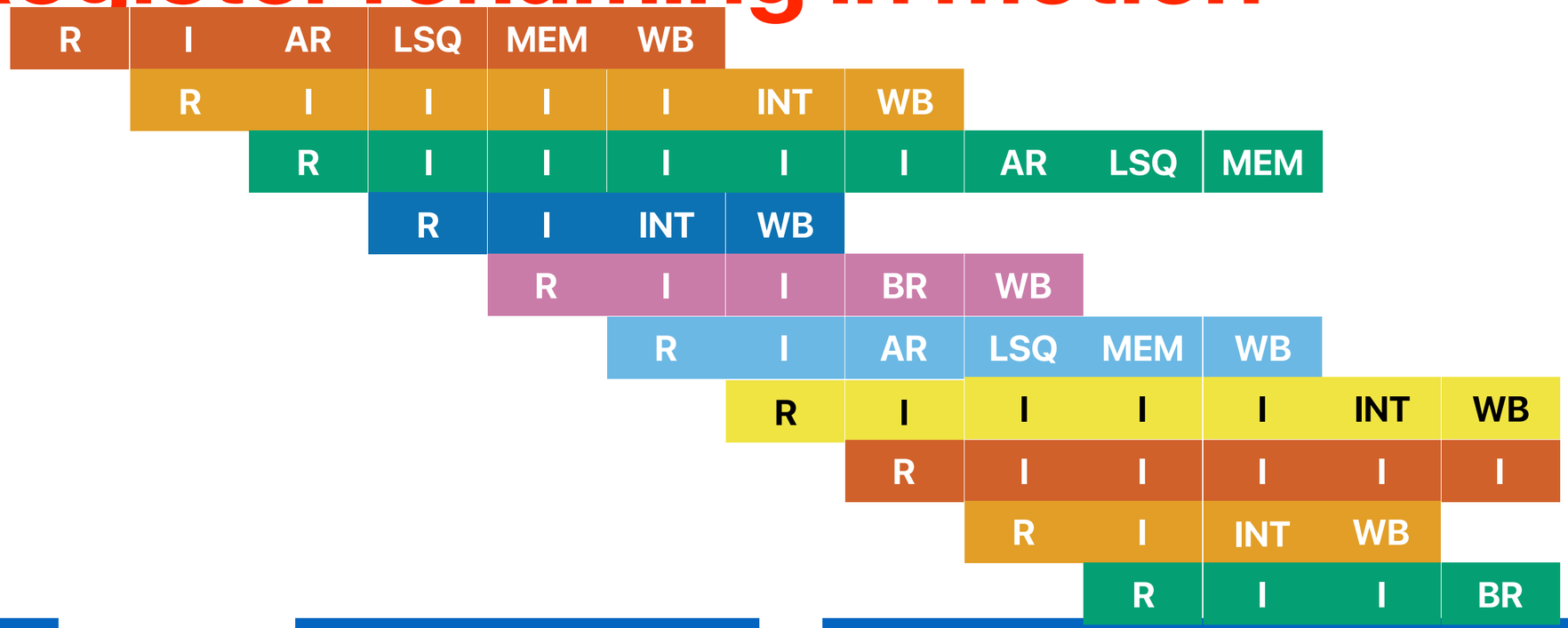
	Renamed instruction		
1	~~ld~~	~~P1, 0(X10)~~	
2	~~add~~	~~P2, P1, X12~~	
3	~~sd~~	~~P2, 0(X10)~~	
4	~~addi~~	~~P3, X10, 8~~	
5	~~bne~~	~~P3, X5, LOOP~~	
6	~~ld~~	~~P4, 0(P3)~~	
7	add	P5, P1, X12	
8	sd	P5, 0(P3)	
9	addi	P6, P3, 8	
10	bne	P6, 0(X10)	

	Physical Register
X5	
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

```
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
```



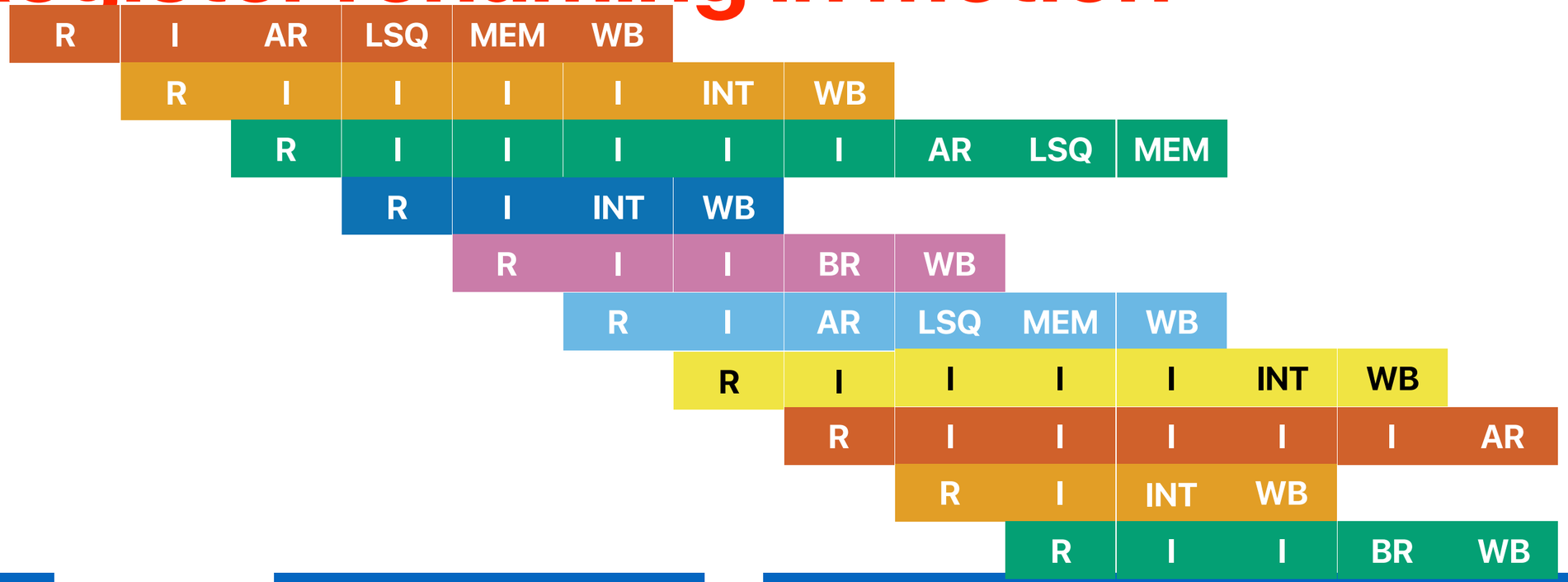
	Renamed instruction		
1	~~ld~~	~~P1, 0(X10)~~	
2	~~add~~	~~P2, P1, X12~~	
3	~~sd~~	~~P2, 0(X10)~~	
4	~~addi~~	~~P3, X10, 8~~	
5	~~bne~~	~~P3, X5, LOOP~~	
6	~~ld~~	~~P4, 0(P3)~~	
7	add	P5, P1, X12	
8	sd	P5, 0(P3)	
9	~~addi~~	~~P6, P3, 8~~	
10	bne	P6, 0(X10)	

	Physical Register
X5	
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
```



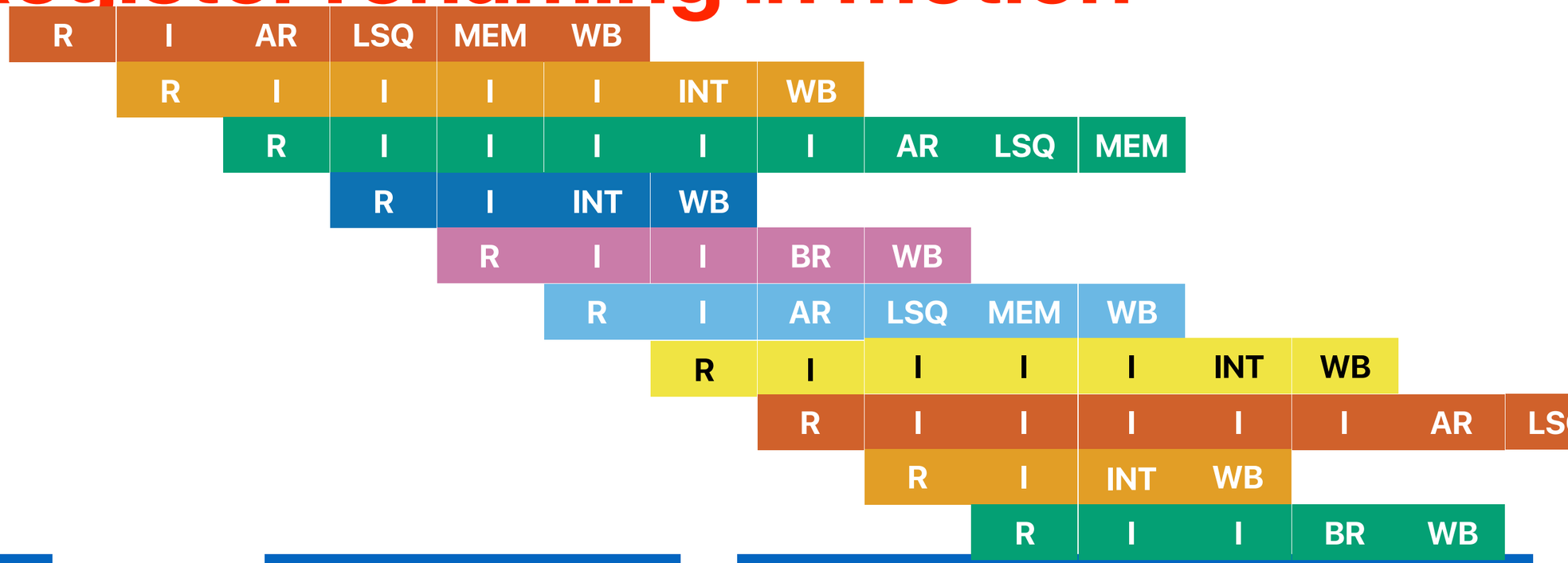
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	~~add P2, P1, X12~~
3	~~sd P2, 0(X10)~~
4	~~addi P3, X10, 8~~
5	~~bne P3, X5, LOOP~~
6	~~ld P4, 0(P3)~~
7	~~add P5, P1, X12~~
8	sd P5, 0(P3)
9	~~addi P6, P3, 8~~
10	bne P6, 0(X10)

	Physical Register
X5	
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
```



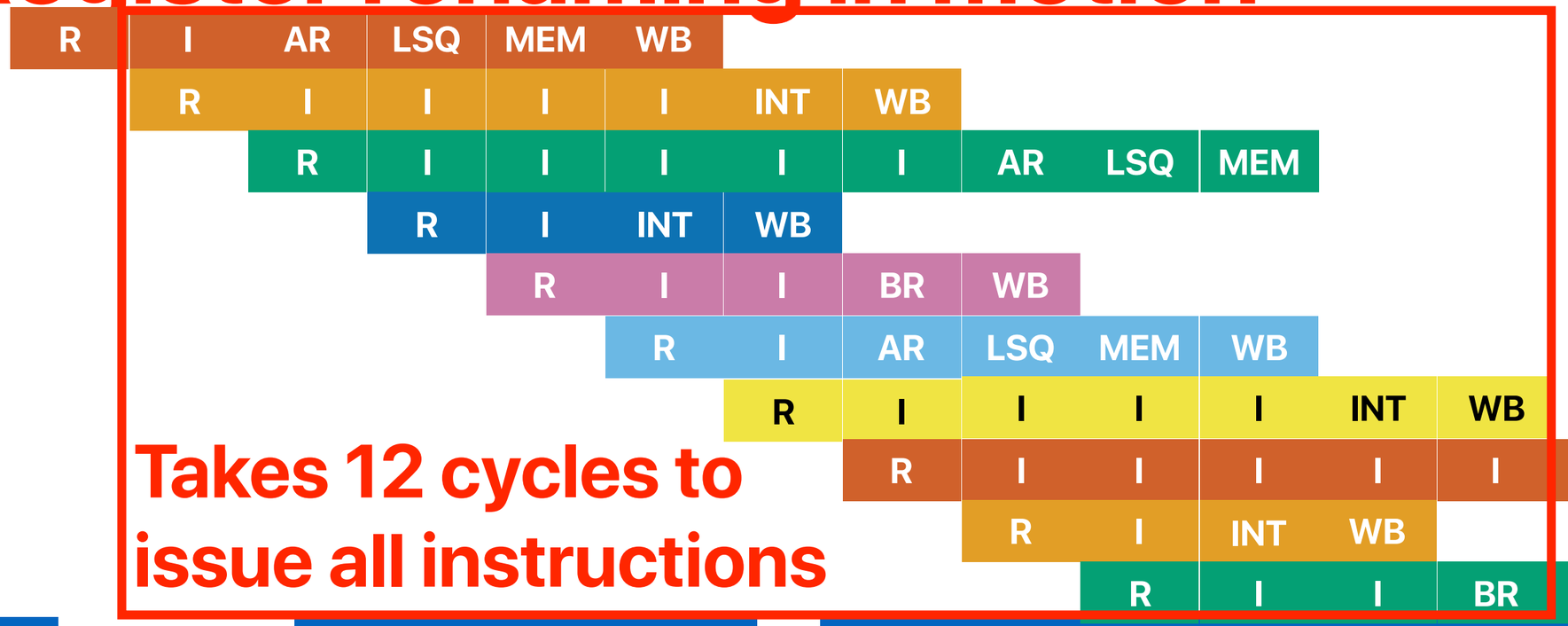
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	~~add P2, P1, X12~~
3	~~sd P2, 0(X10)~~
4	~~addi P3, X10, 8~~
5	~~bne P3, X5, LOOP~~
6	~~ld P4, 0(P3)~~
7	~~add P5, P1, X12~~
8	sd P5, 0(P3)
9	~~addi P6, P3, 8~~
10	~~bne P6, 0(X10)~~

	Physical Register
X5	
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
```



Takes 12 cycles to issue all instructions

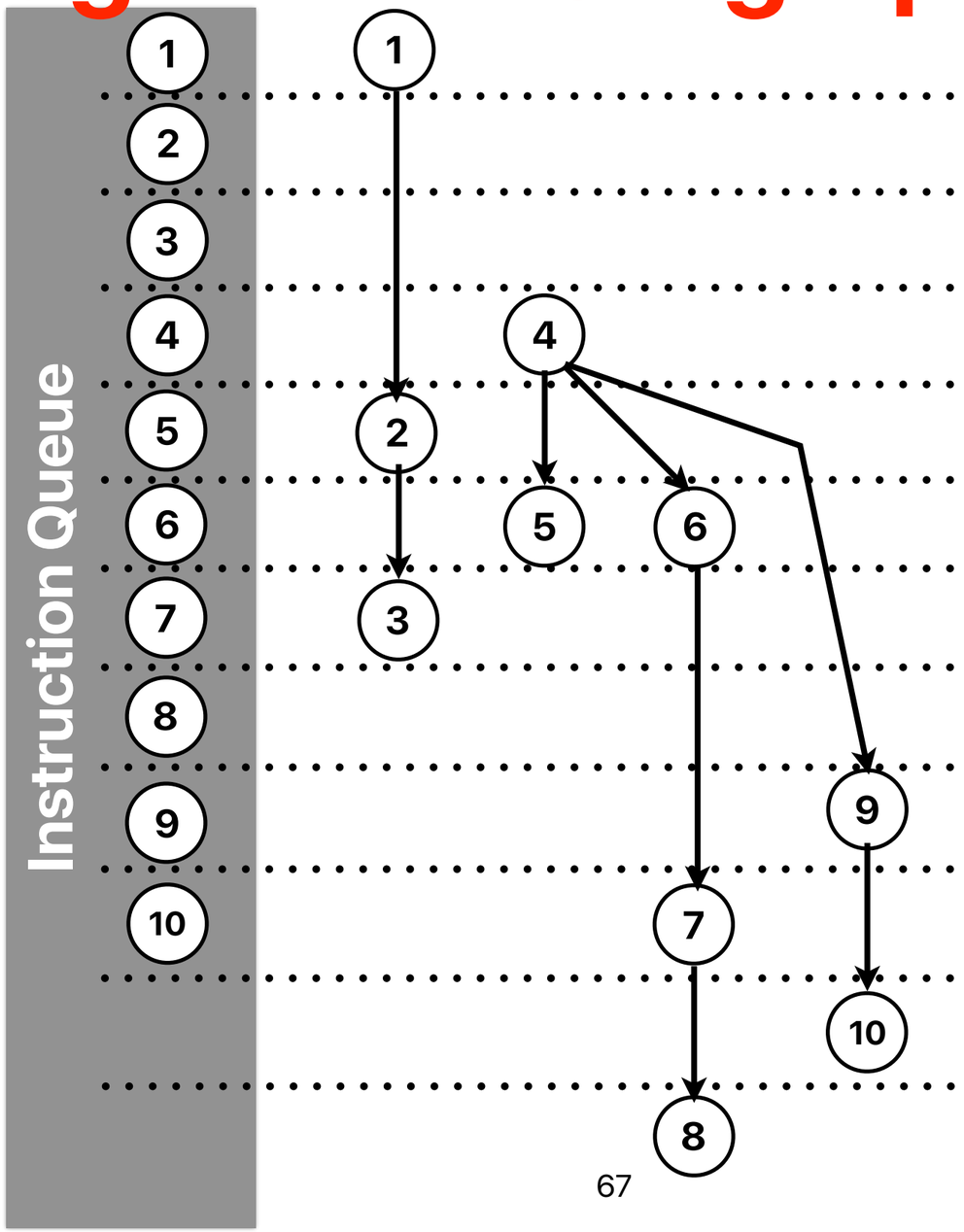
	Renamed instruction
1	~~ld P1, 0(X10)~~
2	~~add P2, P1, X12~~
3	~~sd P2, 0(X10)~~
4	~~addi P3, X10, 8~~
5	~~bne P3, X5, LOOP~~
6	~~ld P4, 0(P3)~~
7	~~add P5, P1, X12~~
8	sd P5, 0(P3)
9	~~addi P6, P3, 8~~
10	~~bne P6, 0(X10)~~

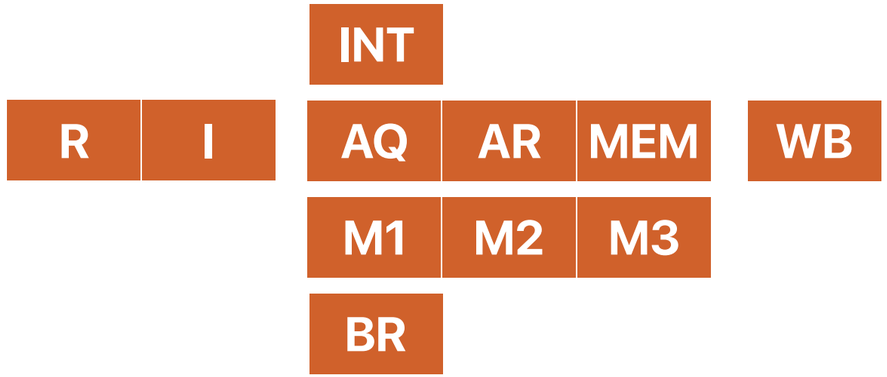
	Physical Register
X5	
X6	P1
X7	P5
X10	P3
X12	

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

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

Announcement

- Homework #3 due tonight
- Reading quiz due next Monday
- iEval submission — attach your "confirmation" screen, you get an extra/bonus homework
- Project due on 12/2 — roughly three weeks from now
  - You can only turn-in "helper.c"
    - `mcfutil.c:refresh_potential()` creates helper threads
    - `mcfutil.c:refresh_potential()` calls `helper_thread_sync()` function periodically
    - It's your task to think what to do in `helper_thread_sync()` and `helper_thread()` functions
    - Please DO READ papers before you ask what to do
  - Formula for grading — **min(100, speedup*100)**
  - No extension
- Office hour for Hung-Wei this week — MWF 1p-2p — no office hour next week