Register Renaming
&
Out-of-Order Execution

Nima Honarmand

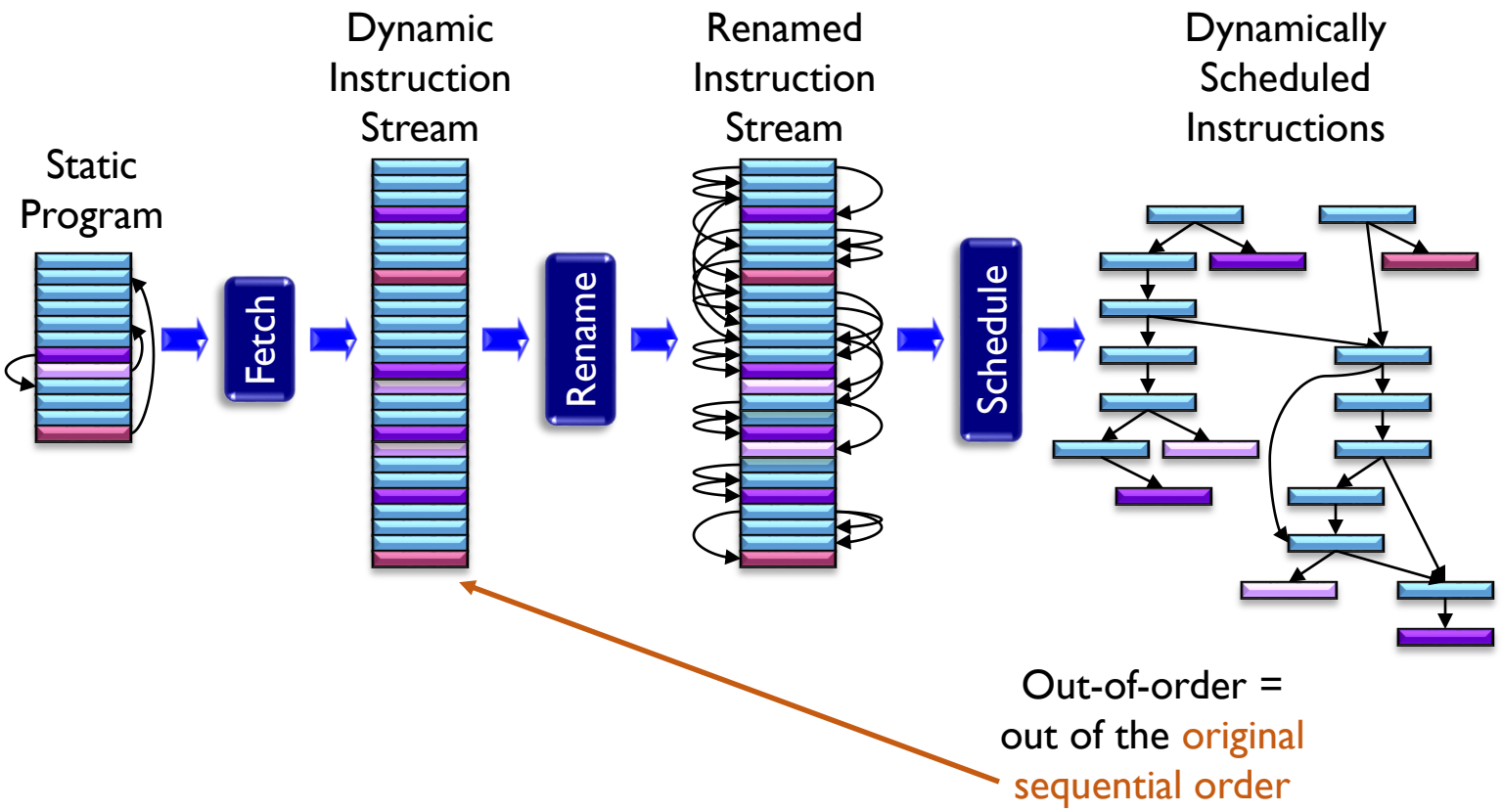
OoO Execution (1/3)

- Dynamic scheduling
  - Totally in the hardware
  - Also called *Out-of-Order execution (OoO)*
  - As opposed to static scheduling (in-order execution)
- Fetch many instructions into *instruction window*
  - Use branch prediction to speculate past branches
- Rename regs. to avoid false deps. (WAW and WAR)
- Execute insns. as soon as possible
  - As soon as deps. (regs and memory) are known
- Today's machines: 100+ insnstruction window

Out-of-Order Execution (2/3)

- Execute insns. in *dataflow* order
  - Often similar to, but not the same as, *program* order
- Register renaming removes false deps.
  - WAR and WAW
- Scheduler identifies when to run insns.
  - Wait for all deps. to be satisfied

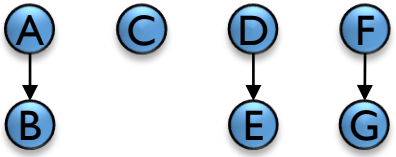
Out-of-Order Execution (3/3)

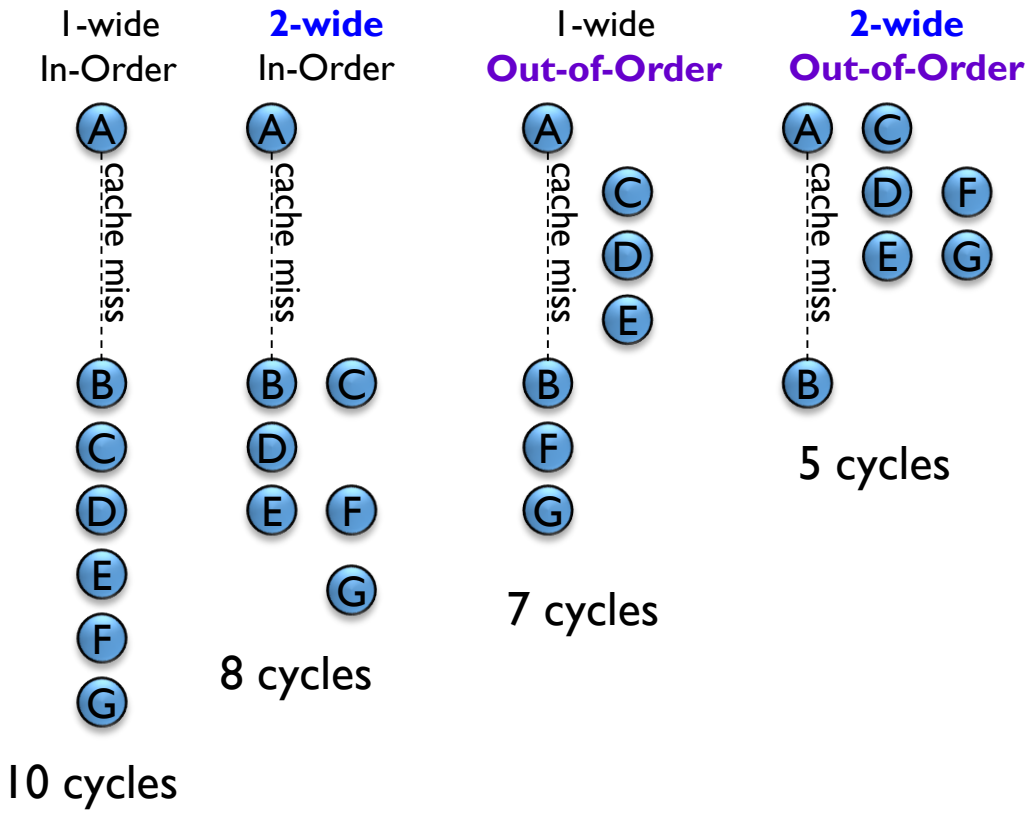


Recall: Superscalar != Out-of-Order

- These are **orthogonal** concepts
  - All combinations are possible (but not equally common)

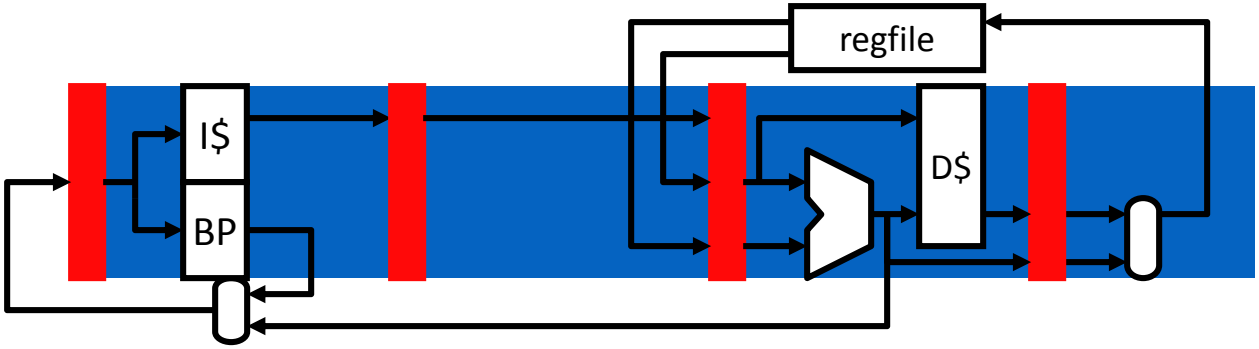
A: R1 = Load 16[R2]
B: R3 = R1 + R4
C: R6 = Load 8[R9]
D: R5 = R2 – 4
E: R7 = Load 20[R5]
F: R4 = R4 – 1
G: BEQ R4, #0





Example Pipeline Terminology

- In-order pipeline
  - F: Fetch
  - D: Decode
  - X: Execute
  - W: Writeback

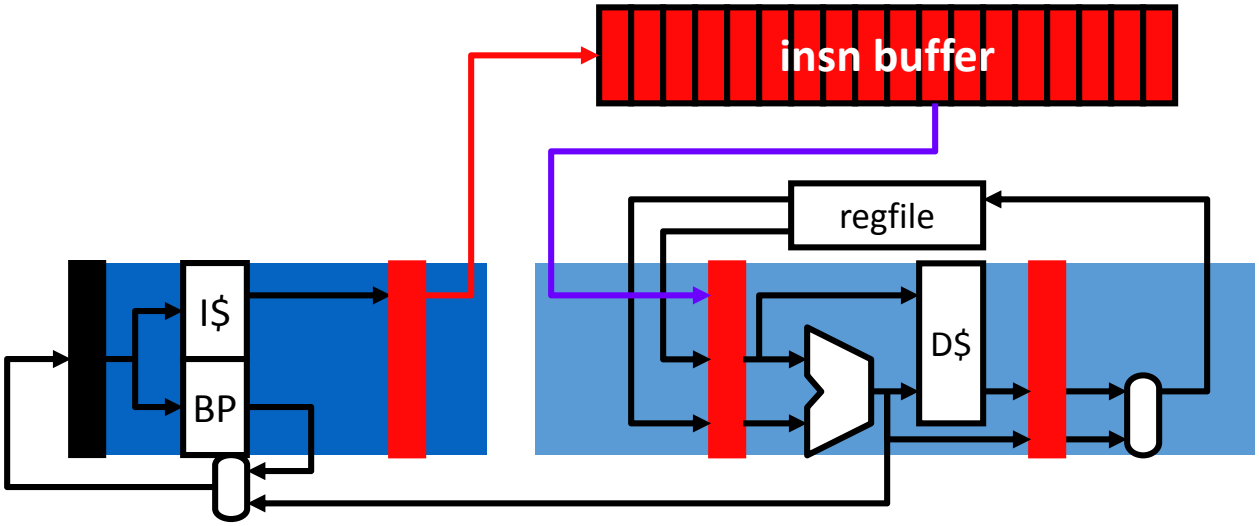


Example Pipeline Diagram

- Alternative pipeline diagram
  - Down: insns
  - Across: pipeline stages
  - In boxes: cycles
  - Basically: stages ↔ cycles
  - Convenient for out-of-order

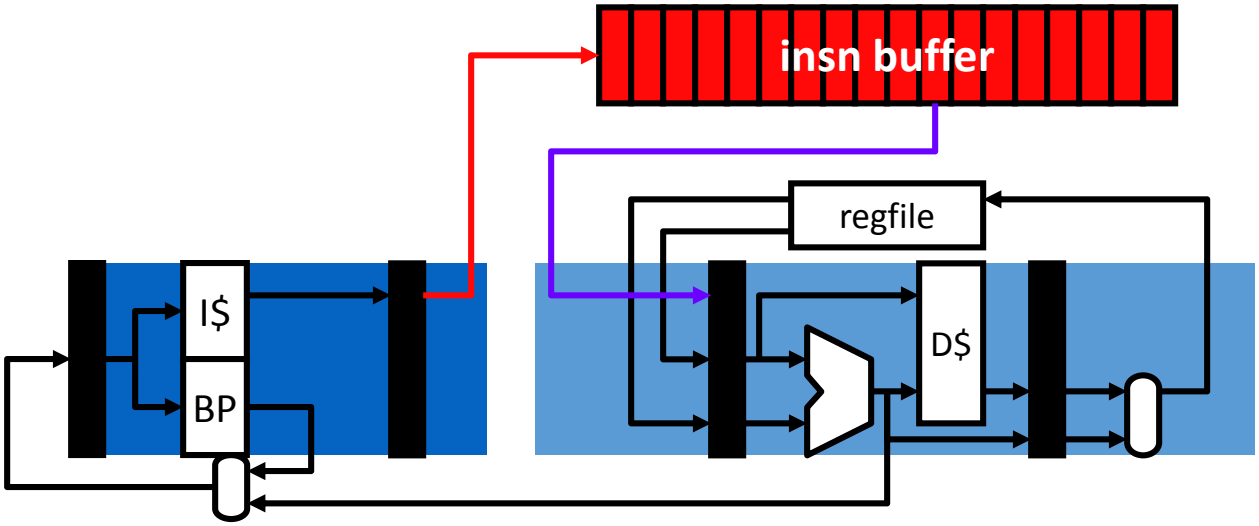
Insn	D	X	W
`f1 = ldf (r1)`	c1	c2	c3
`f2 = mulf f0,f1`	c3	c4+	c7
`stf f2,(r1)`	c7	c8	c9
`r1 = addi r1,4`	c8	c9	c10
`f1 = ldf (r1)`	c10	c11	c12
`f2 = mulf f0,f1`	c12	c13+	c16
`stf f2,(r1)`	c16	c17	c18

Instruction Buffer



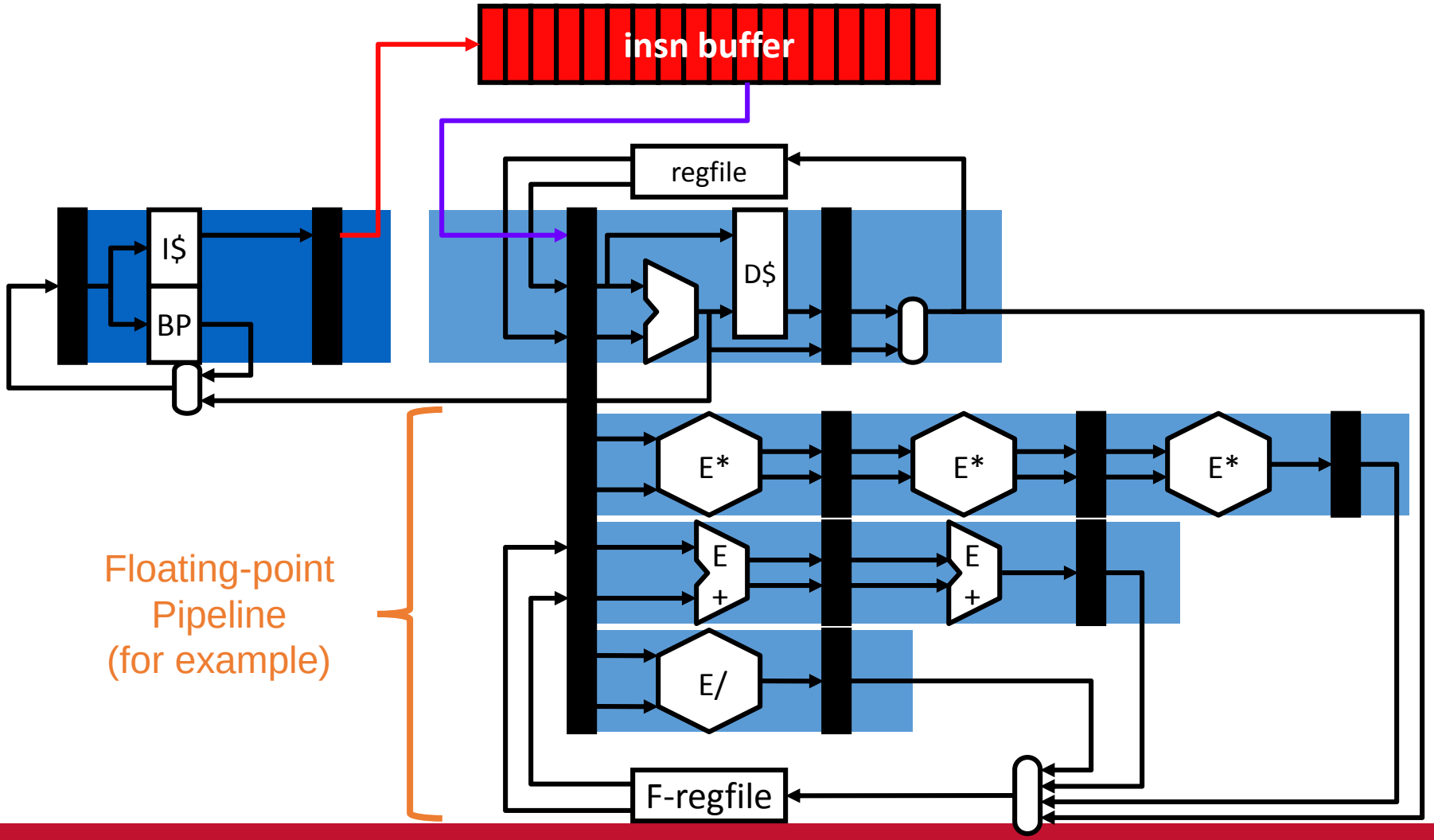
- Trick: ***instruction buffer*** (a.k.a. *instruction window*)
  - A bunch of registers for holding insns.
- Split D into two parts
  - Accumulate decoded insns. in buffer **in-order**
  - Buffer sends insns. down rest of pipeline **out-of-order**

Dispatch and Issue



- **Dispatch (D)**: first part of decode
  - Allocate slot in insn. buffer (if buffer is not full)
  - In order: blocks younger insns.
- **Issue (S)**: second part of decode
  - Send insns. from insn. buffer to execution units
  - Out-of-order: doesn't block younger insns.

Dispatch and Issue in Diversified Pipelines



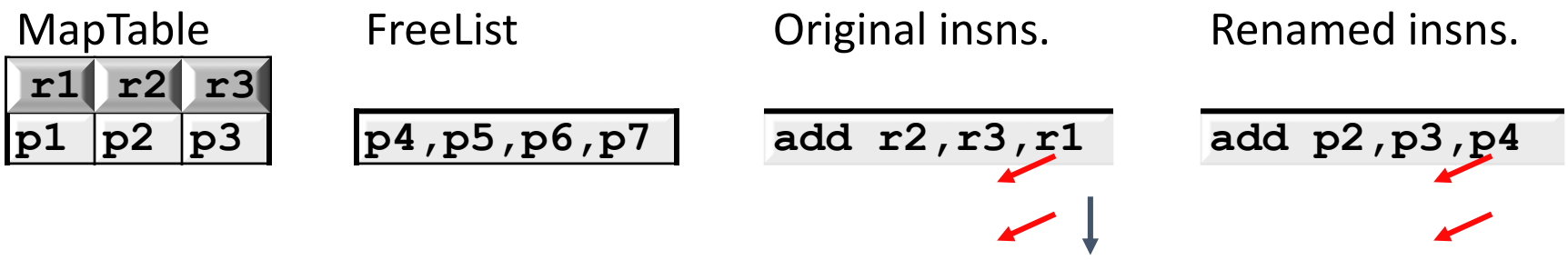
Number of pipeline stages per FU can vary

Register Renaming

- *Register renaming* (in hardware)
  - "Change" register names to eliminate WAR/WAW hazards
  - Arch. registers (r1,f0…) are *names*, not storage locations
  - Can have more locations than names
  - Can have multiple active versions of same name
- How does it work?
  - **Map-table**: maps names to most recent locations
  - On a write: allocate new location (from a **free list**), note in map-table
  - On a read: find location of most recent write via map-table

Register Renaming

- Anti (**WAR**) and output (**WAW**) deps. are false
  - Dep. is on name/location, not on data
  - Given infinite registers, WAR/WAW don't arise
  - Renaming removes WAR/WAW, but leaves **RAW** intact
- Example
  - Names: r1,r2,r3 Physical Locations: p1–p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"



Register Renaming

- Anti (**WAR**) and output (**WAW**) deps. are false
  - Dep. is on name/location, not on data
  - Given infinite registers, WAR/WAW don't arise
  - Renaming removes WAR/WAW, but leaves **RAW** intact
- Example
  - Names: r1,r2,r3 Physical Locations: p1–p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

Original insns.

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

Renamed insns.

```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```

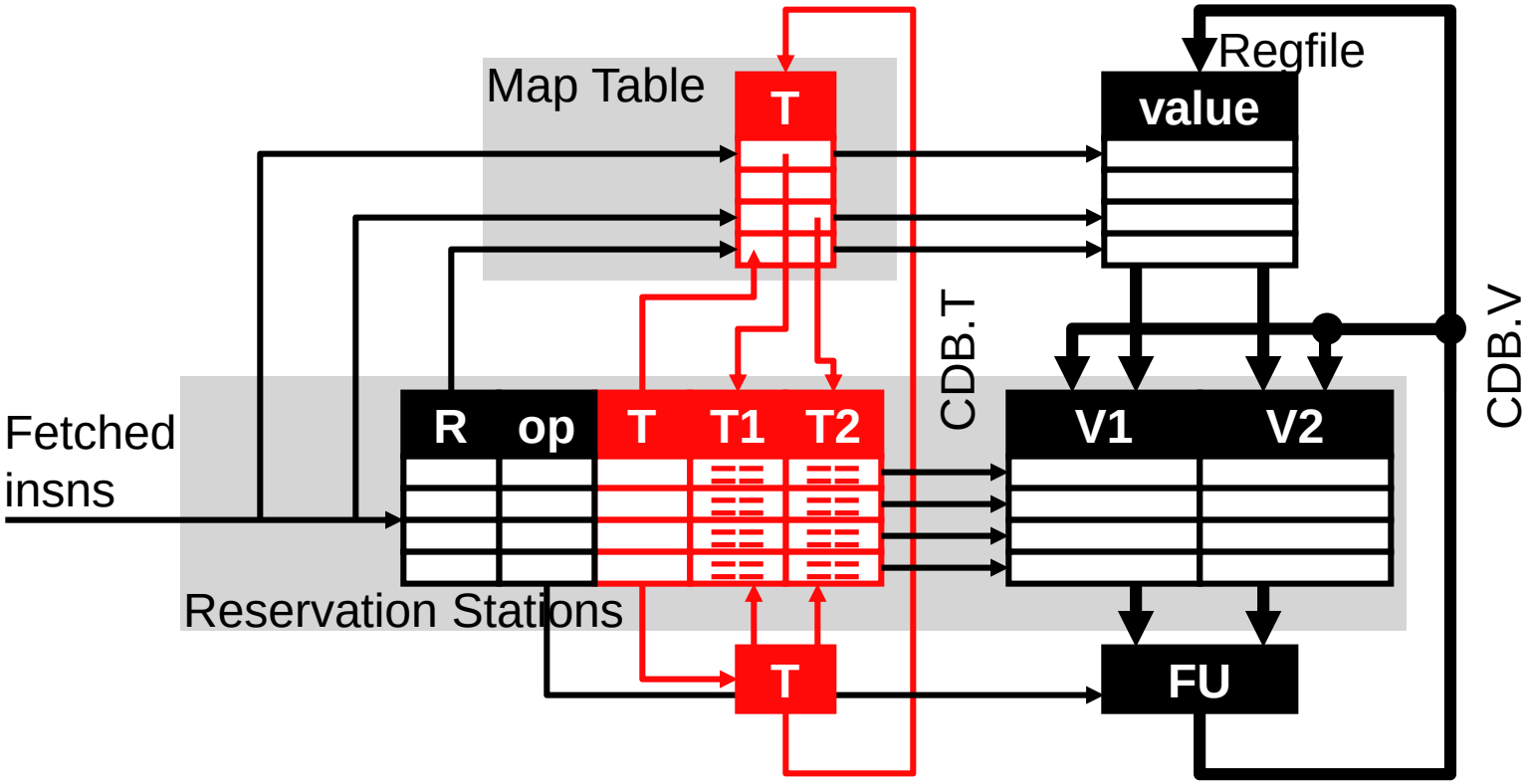
Tomasulo's Algorithm

- *Reservation Stations* (RS): buffers to hold insns
- Common data bus (CDB): broadcasts results to RS
- Register renaming: removes WAR/WAW hazards
- Forwarding (not shown for now to make example simpler)
  - Will discuss later

Tomasulo Data Structures (1/2)

- Reservation Stations (RS)
  - **FU**, **busy**, **op**, **R** (destination register name)
  - **T**: destination register tag (RS# of this RS)
  - **T1**, **T2**: source register tag (RS# of RS that will output value)
  - **V1**, **V2**: source register values
- Map Table – a.k.a. Register Alias Table (RAT)
  - **T**: tag (RS#) that will write this register
  - Valid tags indicate the RS# that will produce result
- Common Data Bus (CDB)
  - Broadcasts <RS#, value> of completed insns.

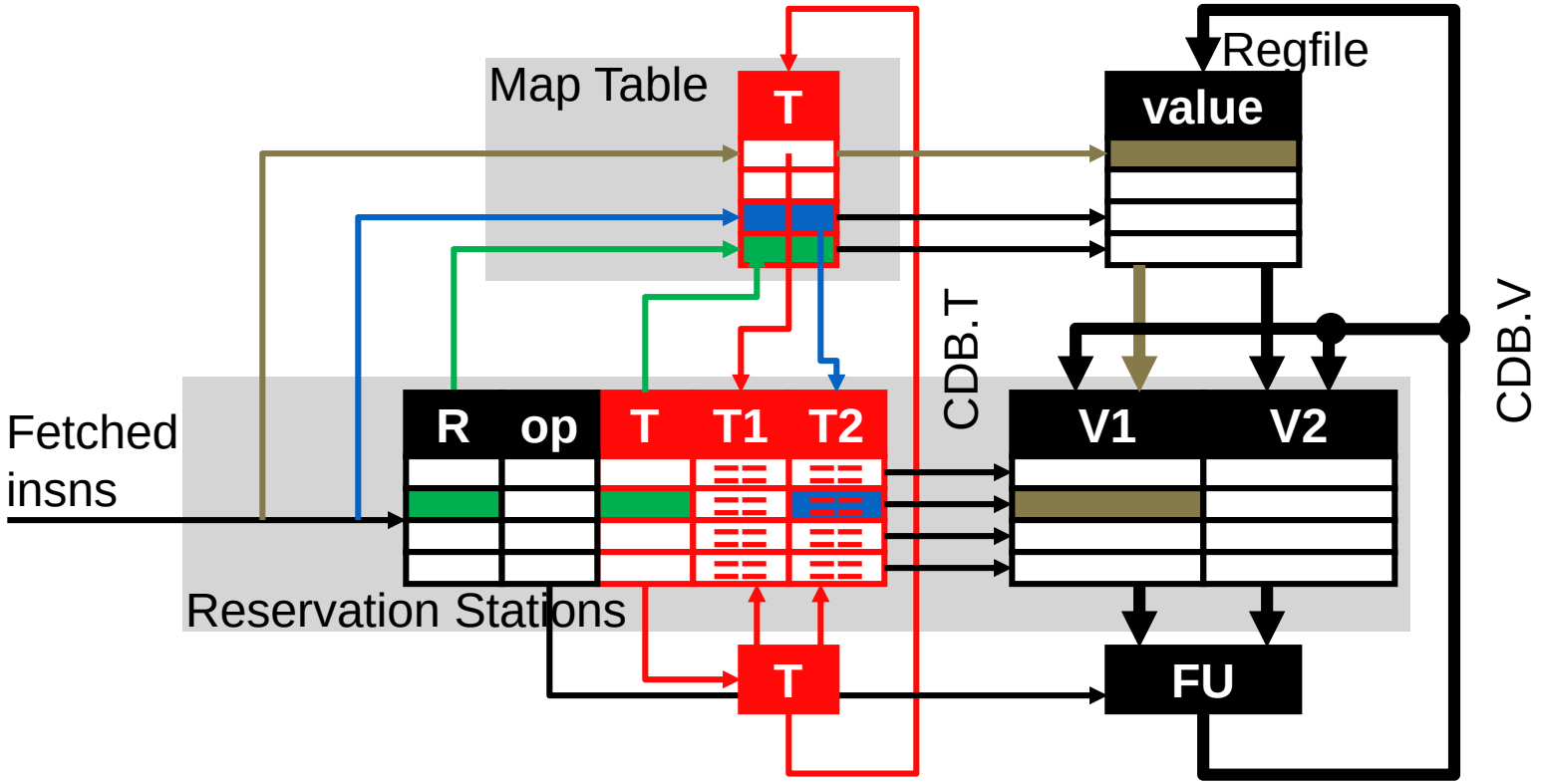
Tomasulo Data Structures (2/2)



Tomasulo Pipeline

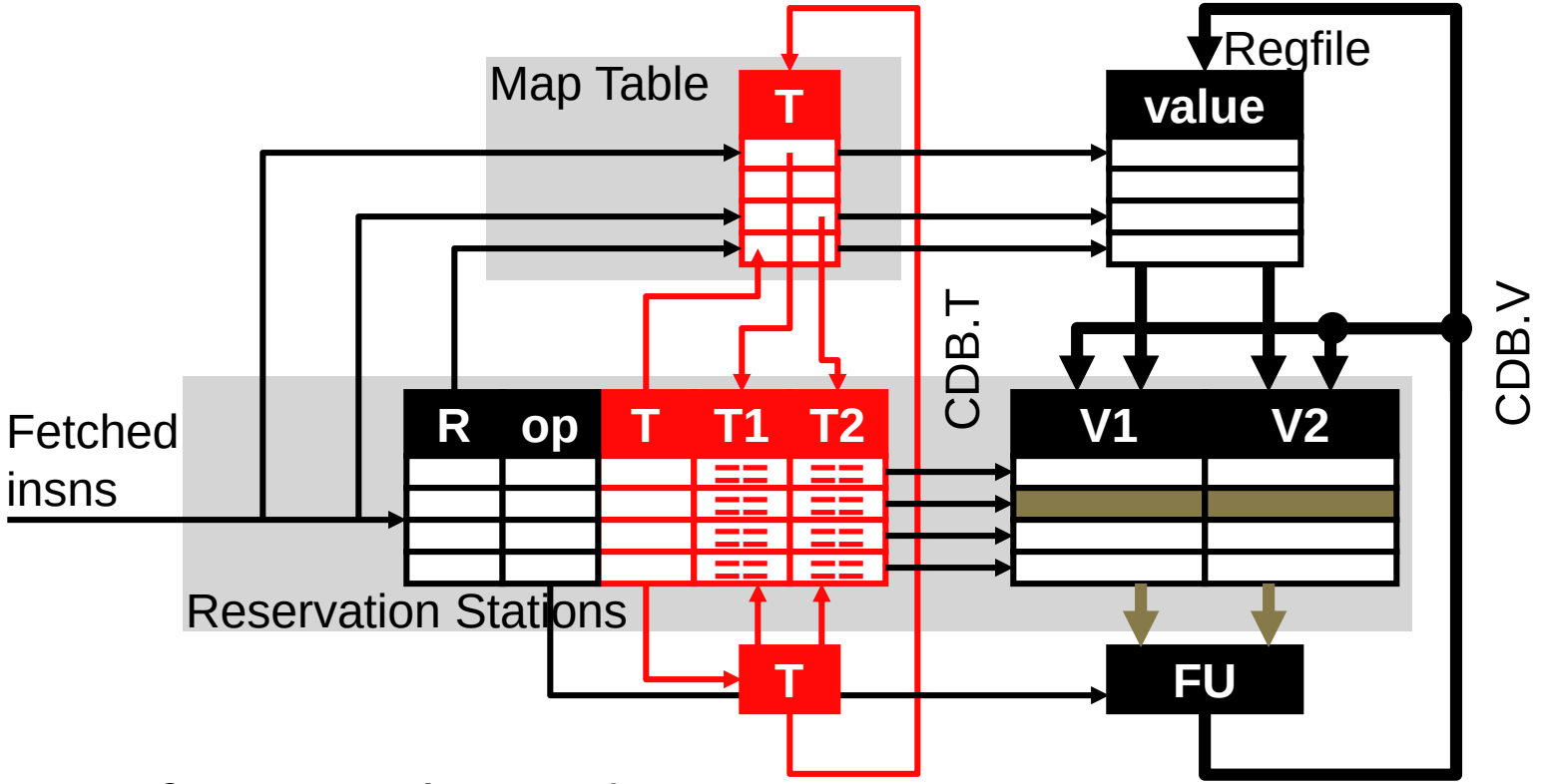
- New pipeline structure: F, **D**, **S**, X, **W**
  - **D (dispatch)**
    - **Structural** hazard ? **stall** : allocate RS entry
      - In this case, structural hazard means there is not a free RS entry for the required FU
  - **S (issue)**
    - **RAW** hazard ? **wait** (monitor CDB) : go to execute
  - **W (writeback)**
    - **W**rite register, free RS entry
    - W and RAW-dependent S in same cycle
      - Instruction(s) waiting for this result to be produced can now issue
    - W and structurally-stalled D in same cycle
      - Instruction waiting for a free RS entry can now be dispatched

Tomasulo Dispatch (D)



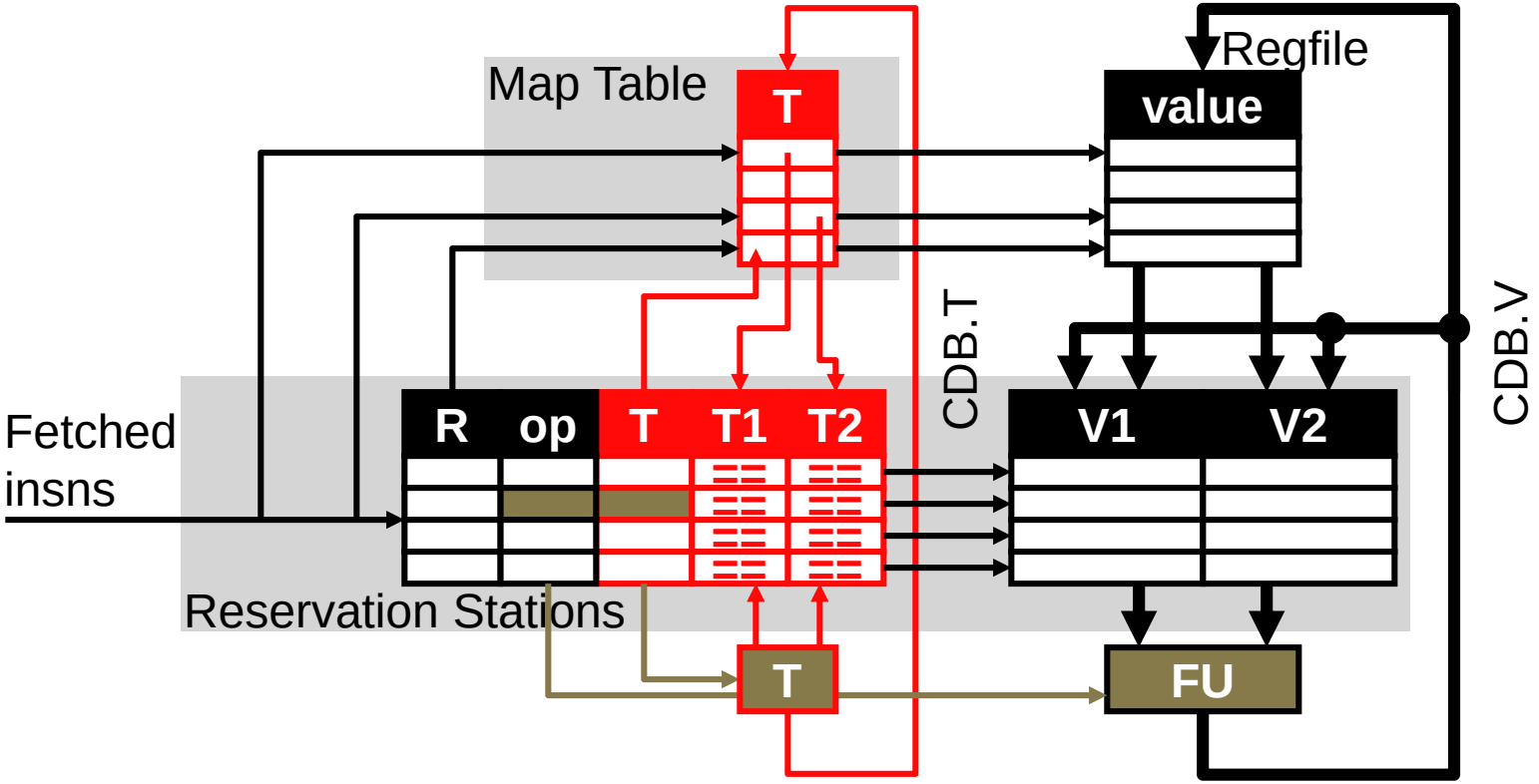
- Allocate RS entry (structural stall if no free entry)
  - Input register ready ? read value into RS : read tag into RS
  - Set register status (i.e., rename) for output register

Tomasulo Issue (S)

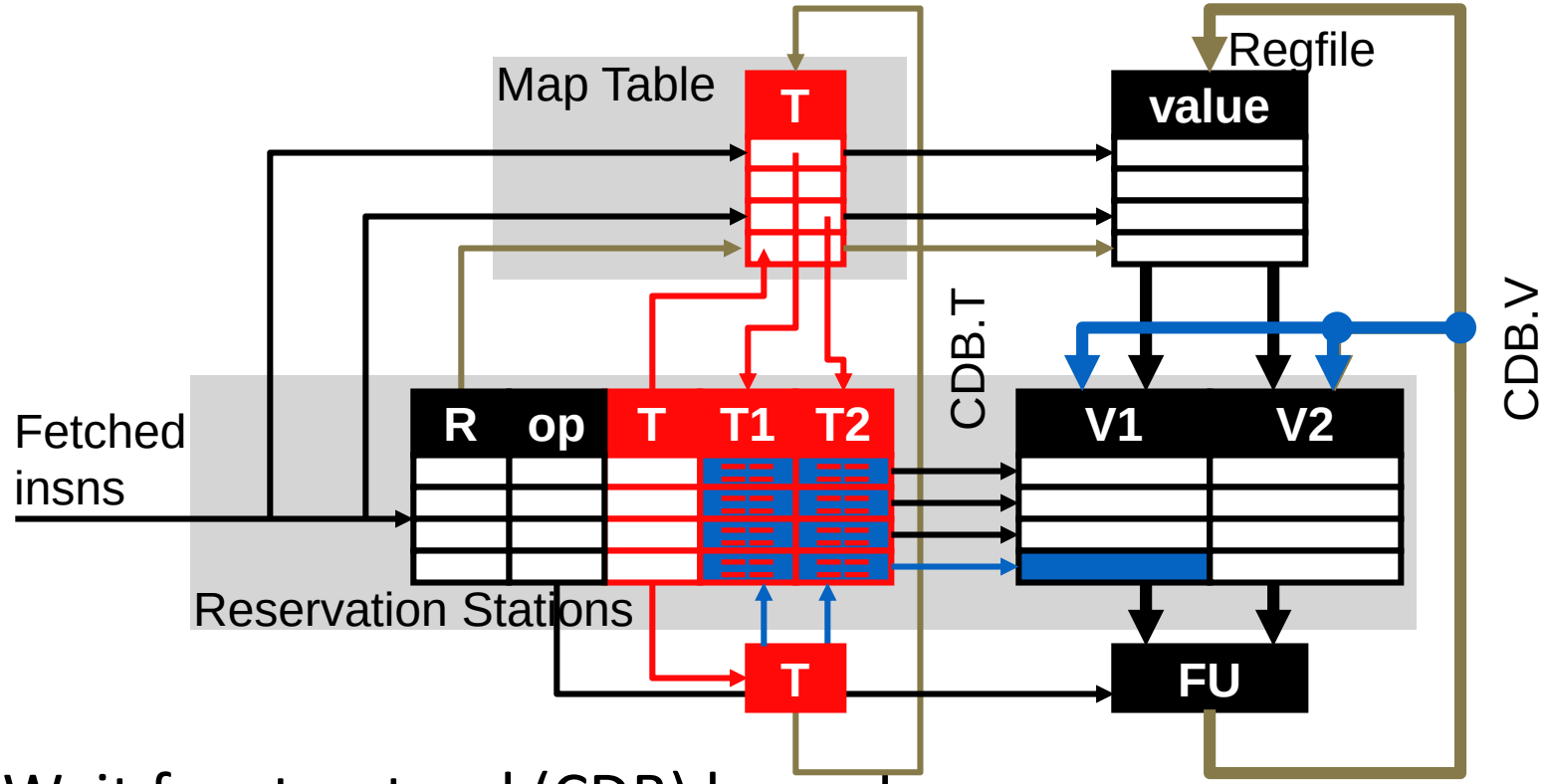


- Wait for RAW hazards
  - Read register values from RS

Tomasulo Execute (X)

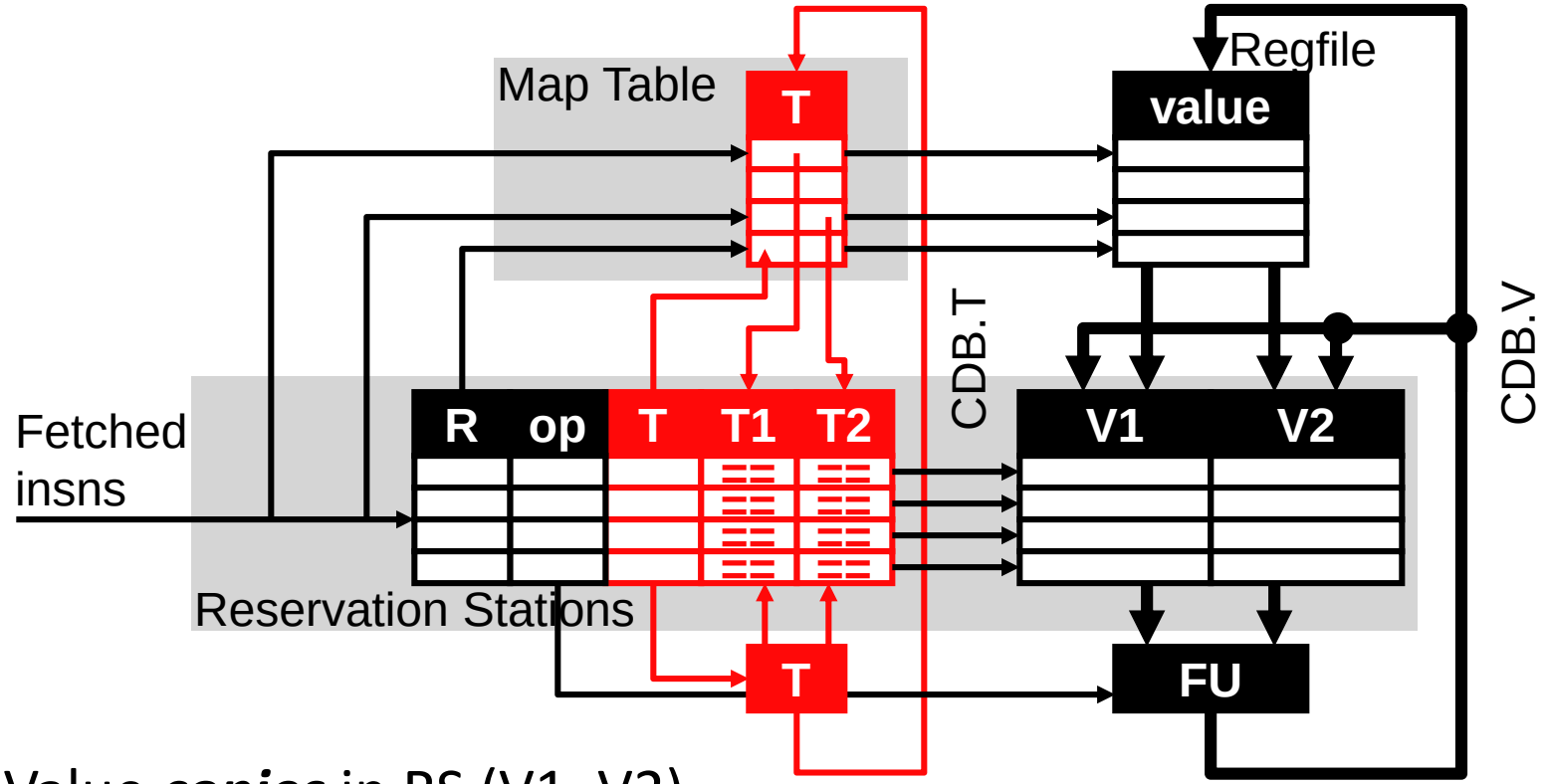


Tomasulo Writeback (W)



- Wait for structural (CDB) hazards
  - R still matches Map Table entry? clear, write result to register
  - CDB broadcast to RS: tag match ? clear tag, copy value

Where is the "register rename"?



- Value ***copies*** in RS (V1, V2)
- Insn. stores correct input values in its own RS entry
- "Free list" is implicit (allocate/deallocate as part of RS)

Tomasulo Data Structures

Insn Status				
Insn	D	S	X	W
`f1 = ldf (r1)`				
`f2 = mulf f0,f1`				
`stf f2,(r1)`				
`r1 = addi r1,4`				
`f1 = ldf (r1)`				
`f2 = mulf f0,f1`				
`stf f2,(r1)`				

Map Table	
Reg	T
`f0`	
`f1`	
`f2`	
`r1`	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	`ALU`	`no`						
2	`LD`	`no`						
3	`ST`	`no`						
4	`FP1`	`no`						
5	`FP2`	`no`						

Tomasulo: Cycle 1

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1			
f2 = mulf f0,f1				
stf f2,(r1)				
r1 = addi r1,4				
f1 = ldf (r1)				
f2 = mulf f0,f1				
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	no						
4	FP1	no						
5	FP2	no						

allocate

Tomasulo: Cycle 2

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2		
f2 = mulf f0,f1	c2			
stf f2,(r1)				
r1 = addi r1,4				
f1 = ldf (r1)				
f2 = mulf f0,f1				
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	no						
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-
5	FP2	no						

allocate

Tomasulo: Cycle 3

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	
f2 = mulf f0,f1	c2			
stf f2,(r1)	c3			
r1 = addi r1,4				
f1 = ldf (r1)				
f2 = mulf f0,f1				
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-
5	FP2	no						

allocate

Tomasulo: Cycle 4

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	c4
f2 = mulf f0,f1	c2	c4		
stf f2,(r1)	c3			
r1 = addi r1,4	c4			
f1 = ldf (r1)				
f2 = mulf f0,f1				
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	RS#1

CDB	
T	V
RS#2	[f1]

`ldf` finished (W)
clear `f1` RegStatus
CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	no						
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	RS#2	[f0]	CDB.V
5	FP2	no						

allocate

free

RS#2 ready → grab CDB value

Tomasulo: Cycle 5

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	c4
f2 = mulf f0,f1	c2	c4	c5	
stf f2,(r1)	c3			
r1 = addi r1,4	c4	c5		
f1 = ldf (r1)	c5			
f2 = mulf f0,f1				
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	RS#1

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	yes	ldf	f1	-	RS#1	-	-
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	no						

allocate

Tomasulo: Cycle 6

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	c4
f2 = mulf f0,f1	c2	c4	c5+	
stf f2,(r1)	c3			
r1 = addi r1,4	c4	c5	c6	
f1 = ldf (r1)	c5			
f2 = mulf f0,f1	c6			
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	~~RS#4~~RS#5
r1	RS#1

CDB	
T	V

no stall on WAW:
overwrite f2 RegStatus
anyone who needs old f2 tag has it

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	yes	ldf	f1	-	RS#1	-	-
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

allocate

Tomasulo: Cycle 7

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	c4
f2 = mulf f0,f1	c2	c4	c5+	
stf f2,(r1)	c3			
r1 = addi r1,4	c4	c5	c6	c7
f1 = ldf (r1)	c5	c7		
f2 = mulf f0,f1	c6			
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	*RS#1*

CDB	
T	V
RS#1	[r1]

no stall on WAR:
anyone who needs old r1 has RS copy

D stall on store RS: structural (no space)

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	RS#1	-	CDB.V
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

addi finished (W)
clear r1 RegStatus
CDB broadcast

RS#1 ready → grab CDB value

Tomasulo: Cycle 8

Insn Status				
Insn	D	S	X	W
`f1 = ldf (r1)`	c1	c2	c3	c4
`f2 = mulf f0,f1`	c2	c4	c5+	c8
`stf f2,(r1)`	c3	c8		
`r1 = addi r1,4`	c4	c5	c6	c7
`f1 = ldf (r1)`	c5	c7	c8	
`f2 = mulf f0,f1`	c6			
`stf f2,(r1)`				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	

CDB	
T	V
RS#4	[f2]

`mulf` finished (W), `f2` already overwritten by 2nd `mulf` (`RS#5`)
CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	yes	stf	-	RS#4	-	CDB.V	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

`RS#4` ready → grab CDB value

Tomasulo: Cycle 9

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	c4
f2 = mulf f0,f1	c2	c4	c5+	c8
stf f2,(r1)	c3	c8	c9	
r1 = addi r1,4	c4	c5	c6	c7
f1 = ldf (r1)	c5	c7	c8	c9
f2 = mulf f0,f1	c6	c9		
stf f2,(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	

CDB	
T	V
RS#2	[f1]

2nd ldf finished (W)
clear f1 RegStatus
CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	-	-	-	[f2]	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	RS#2	[f0]	CDB.V

RS#2 ready → grab CDB value

Tomasulo: Cycle 10

Insn Status				
Insn	D	S	X	W
f1 = ldf (r1)	c1	c2	c3	c4
f2 = mulf f0,f1	c2	c4	c5+	c8
stf f2,(r1)	c3	c8	c9	c10
r1 = addi r1,4	c4	c5	c6	c7
f1 = ldf (r1)	c5	c7	c8	c9
f2 = mulf f0,f1	c6	c9	c10	
stf f2,(r1)	c10			

Map Table	
Reg	T
f0	
f1	
f2	RS#5
r1	

CDB	
T	V

`stf` finished (W)
no output register → no CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	-	RS#5	-	-	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	-	[f0]	[f1]

free → allocate

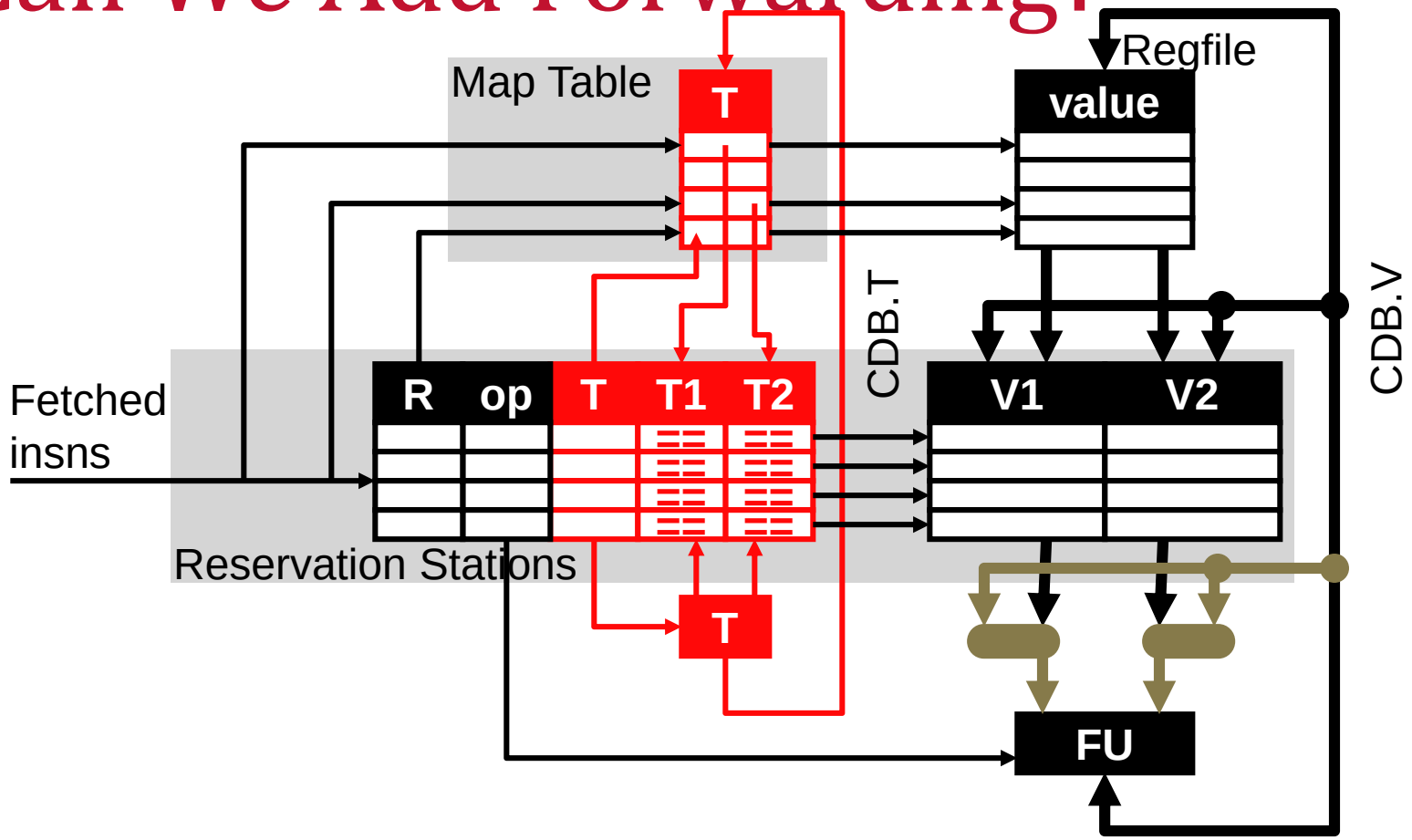
Superscalar Tomasulo Pipeline

- Recall: Dynamic scheduling and multi-issue are orthogonal
  - **N**: superscalar width (number of parallel operations)
  - **WS**: window size (number of reservation stations)
- What is needed for an **N**-by-**WS** Tomasulo?
  - RS: **N** tag/value write (D), **N** value read (S), **2WS** tag cmp (W)
  - Select logic: **WS**→**N** priority encoder (S)
  - Map Table: **2N** read (D), **N** write (D)
  - Register File: **2N** read (D), **N** write (W)
  - CDB: **N** (W)

Superscalar Select Logic

- Superscalar select logic: **WS→N** priority encoder
  - Somewhat complicated (**N**^{2} $\log_2$ **WS**)
  - Can simplify using different RS designs
- **Split design**
  - Divide RS into **N** banks: 1 per FU?
  - Implement **N** separate **WS/N**→1 encoders
  - \+ Simpler: **N** * $\log_2$ **WS/N**
  - Less scheduling flexibility
- **FIFO design**
  - Can issue only head of each RS bank
  - \+ Simpler: no select logic at all
  - Less scheduling flexibility (but surprisingly not that bad)

Can We Add Forwarding?



- Yes, but it's more complicated than you might think
  - In fact: requires a completely new pipeline

Why Out-of-Order Forwarding Is Hard

	No Forwarding				Forwarding			
Insn	D	S	X	W	D	S	X	W
`f1 = ldf (r1)`	c1	c2	c3	c4	c1	c2	c3	c4
`f2 = mulf f0,f1`	c2	c4	c5+	c8	c2	c3	c4+	c7

- Forwarding: **`ldf`** X in c3 → **`mulf`** X in c4 → **`mulf`** S in c3
  - But how can **`mulf`** S in c3 if **`ldf`** W in c4? Must change pipeline
- Modern OoO schedulers
  - Split CDB tag and value, move tag broadcast to S
    - **`ldf`** tag broadcast now in cycle 2 → mulf S in cycle 3
  - How do multi-cycle operations work?
    - Delay tag broadcast according
  - How do variable-latency operations (e.g., cache misses) work?
    - **Speculatively** broadcast tag assuming best-case delay
    - If wrong, **kill and replay** the dependent insns (and their dependent insns, etc.)

  → Very complex scheduler used in high-performance processors