CSE 502: Computer Architecture

Out-of-Order Execution and Register Rename

In Search of Parallelism

- "Trivial" Parallelism is limited
  - What is trivial parallelism?
    - In-order: sequential instructions do not have dependencies
    - In all previous cases, all insns. executed with or after earlier insns.
  - Superscalar execution quickly hits a ceiling due to deps.

- So what is "non-trivial" parallelism? …

Instruction-Level Parallelism (ILP)

ILP is a measure of inter-dependencies between insns.

Average ILP = num. instruction / num. cyc required

code1: ILP = 1

i.e. must execute serially

code2: ILP = 3

i.e. can execute at the same time

```
code1:    r1 ← r2 + 1
          r3 ← r1 / 17
          r4 ← r0 - r3
```

```
code2:    r1 ← r2 + 1
          r3 ← r9 / 17
          r4 ← r0 - r10
```

The Problem with In-Order Pipelines

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
`addf f0,f1,f2`	F	D	E+	E+	E+	W										
`mulf f2,f3,f2`		F	D	d*	d*	E*	E*	E*	E*	E*	W					
`subf f0,f1,f4`			F	p*	p*	D	E+	E+	E+	W						

- What's happening in cycle 4?
  - **`mulf`** stalls due to **RAW hazard**
    - OK, this is a fundamental problem
  - **`subf`** stalls due to **pipeline hazard**
    - Why? **`subf`** can't proceed into D because **`mulf`** is there
    - That is the only reason, and it isn't a fundamental one
- Why can't **`subf`** go to D in cycle 4 and E+ in cycle 5?

ILP != IPC

- ILP usually assumes
  - Infinite resources
  - Perfect fetch
  - Unit-latency for all instructions
- ILP is a property of the program dataflow
- IPC is the “real” observed metric
  - How many insns. are executed per cycle
- ILP is an upper-bound on the attainable IPC
  - Specific to a particular program

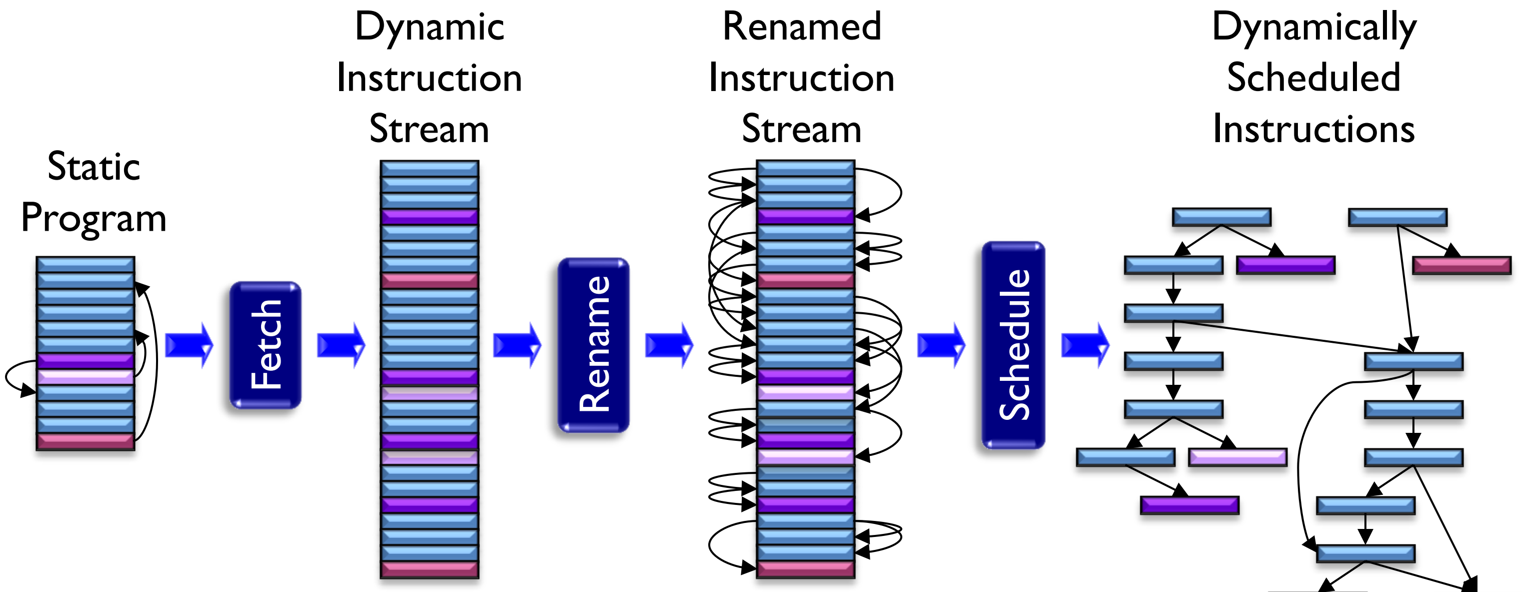
OoO Execution (1/3)

- Dynamic scheduling
  - Totally in the hardware
  - Also called *Out-of-Order execution (OoO)*
- 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+ insns. scheduling window

Out-of-Order Execution (2/3)

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

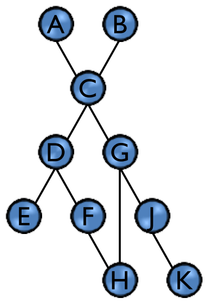
Out-of-Order Execution (3/3)

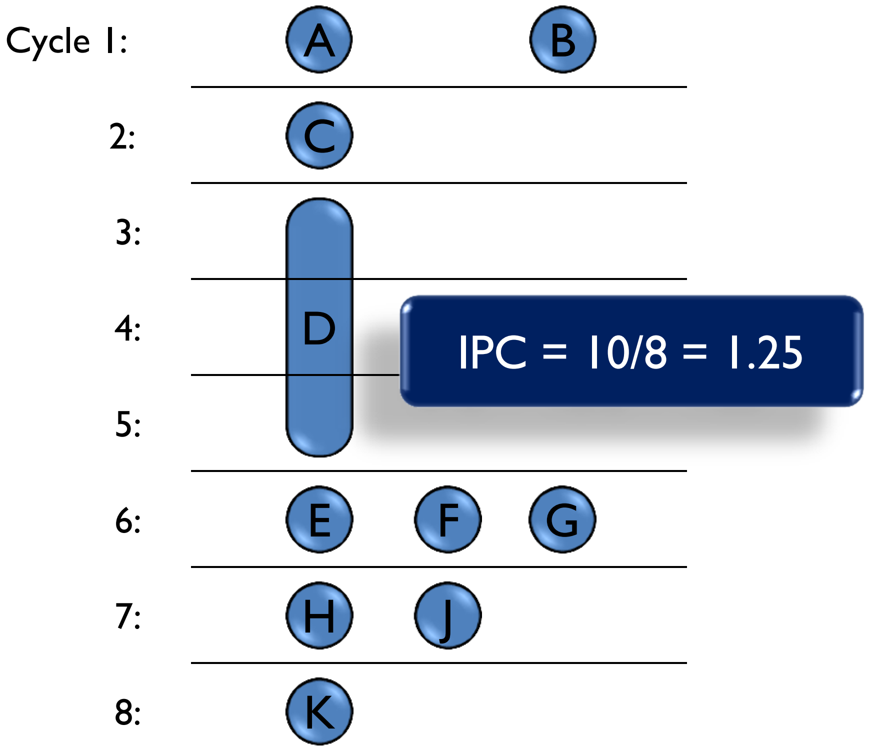

Static Program
Fetch
Dynamic Instruction Stream
Rename
Renamed Instruction Stream
Schedule
Dynamically Scheduled Instructions

Out-of-order = out of the original sequential order

OoO Example (1/2)

A: R1 = R2 + R3
B: R4 = R5 + R6
C: R1 = R1 * R4
D: R7 = LD 0[R1]
E: BEQZ R7, +32
F: R4 = R7 - 3
G: R1 = R1 + 1
H: R4 → ST 0[R1]
J: R1 = R1 – 1
K: R3 → ST 0[R1]





IPC = 10/8 = 1.25

OoO Example (2/2)

A: R1 = R2 + R3

B: R4 = R5 + R6

C: R1 = R1 * R4

D: R9 = LD 0[R1]

E: BEQZ R7, +32

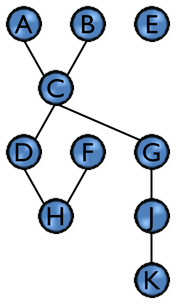
F: R4 = R7 - 3

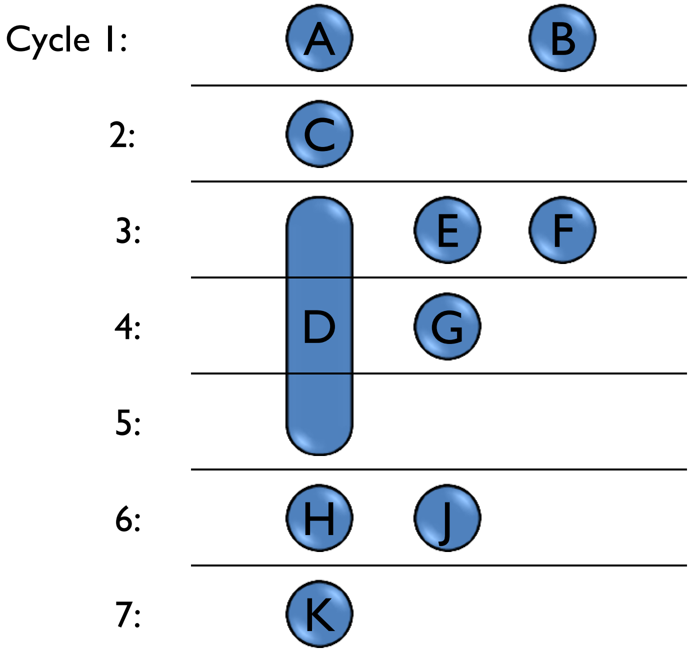
G: R1 = R1 + 1

H: R4 → ST 0[R9]

J: R1 = R9 – 1

K: R3 → ST 0[R1]

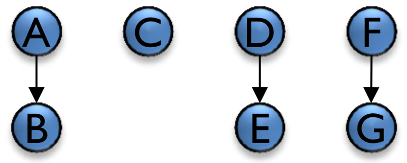


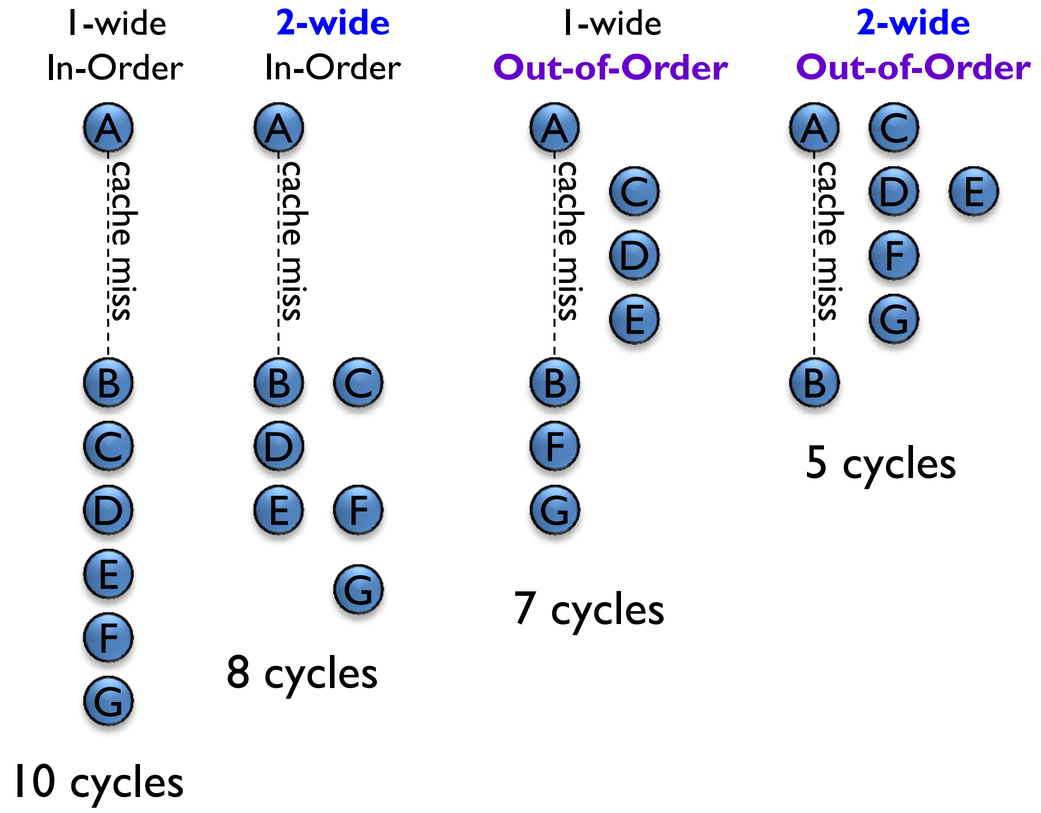


IPC = 10/7 = 1.43

Superscalar != Out-of-Order

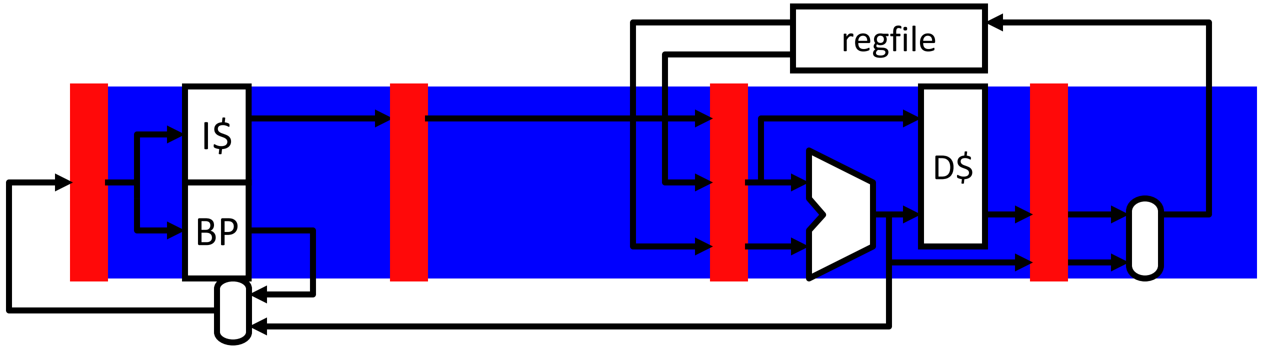
A: R1 = Load 16[R2]
B: R3 = R1 + R4
C: R6 = Load 8[R9]
D: R5 = R2 – 4
E: R7 = Load 20[R5]
F: R4 = R4 – 1
G: BEQ R4, #0


$$\begin{tikzcd} A \arrow[d] & C & D \arrow[d] & F \arrow[d] \\ B & & E & G \end{tikzcd}$$



Example Pipeline Terminology

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

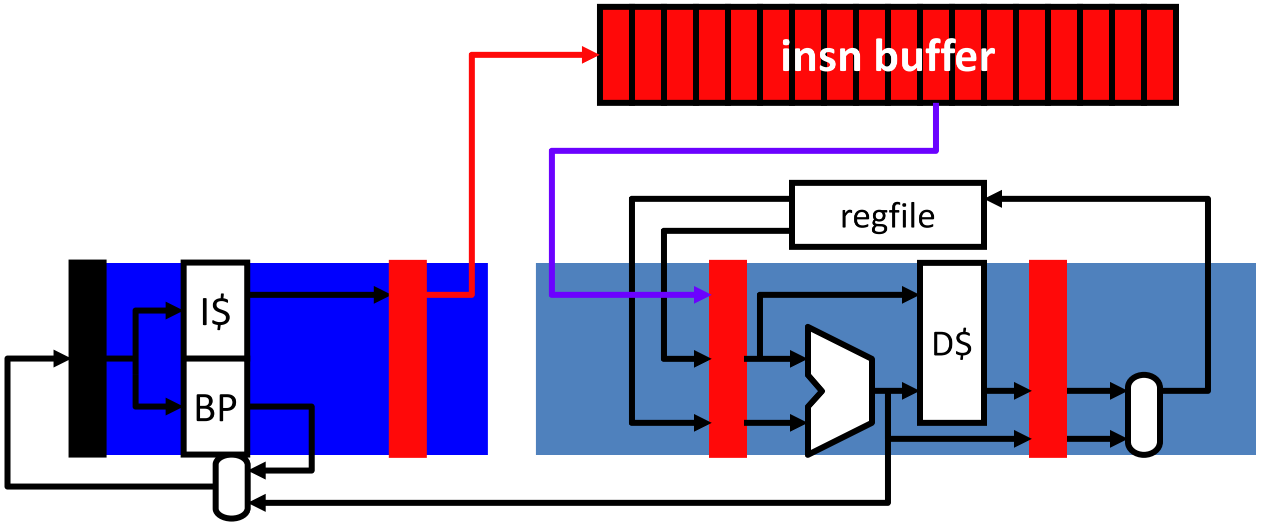


Example Pipeline Diagram

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

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

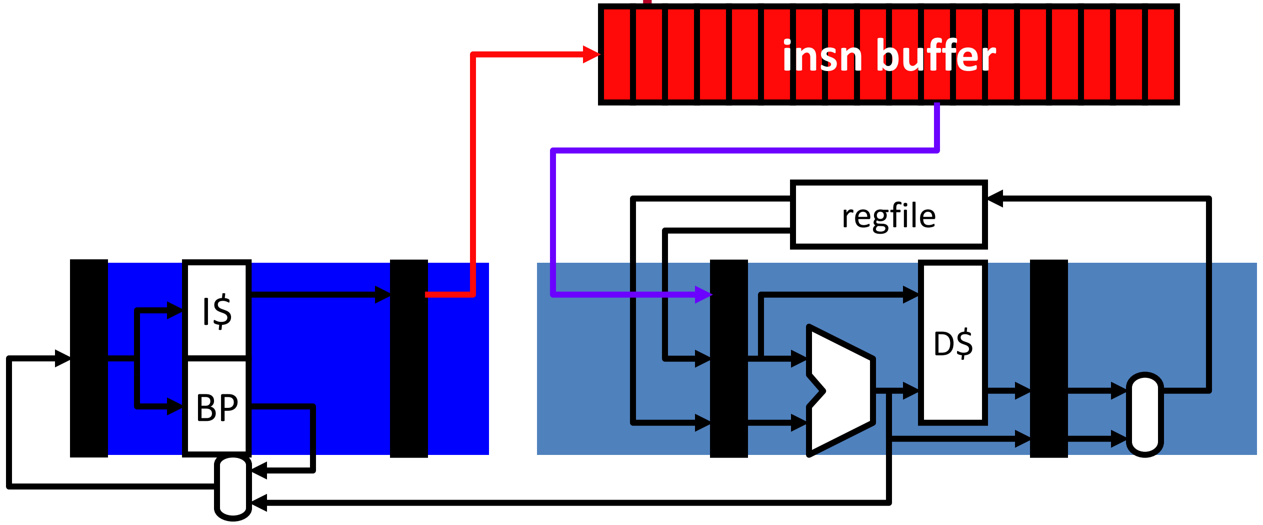
Instruction Buffer



- Trick: ***instruction buffer*** (a.k.a. *instruction window*)
  - A bunch of registers for holding insns.

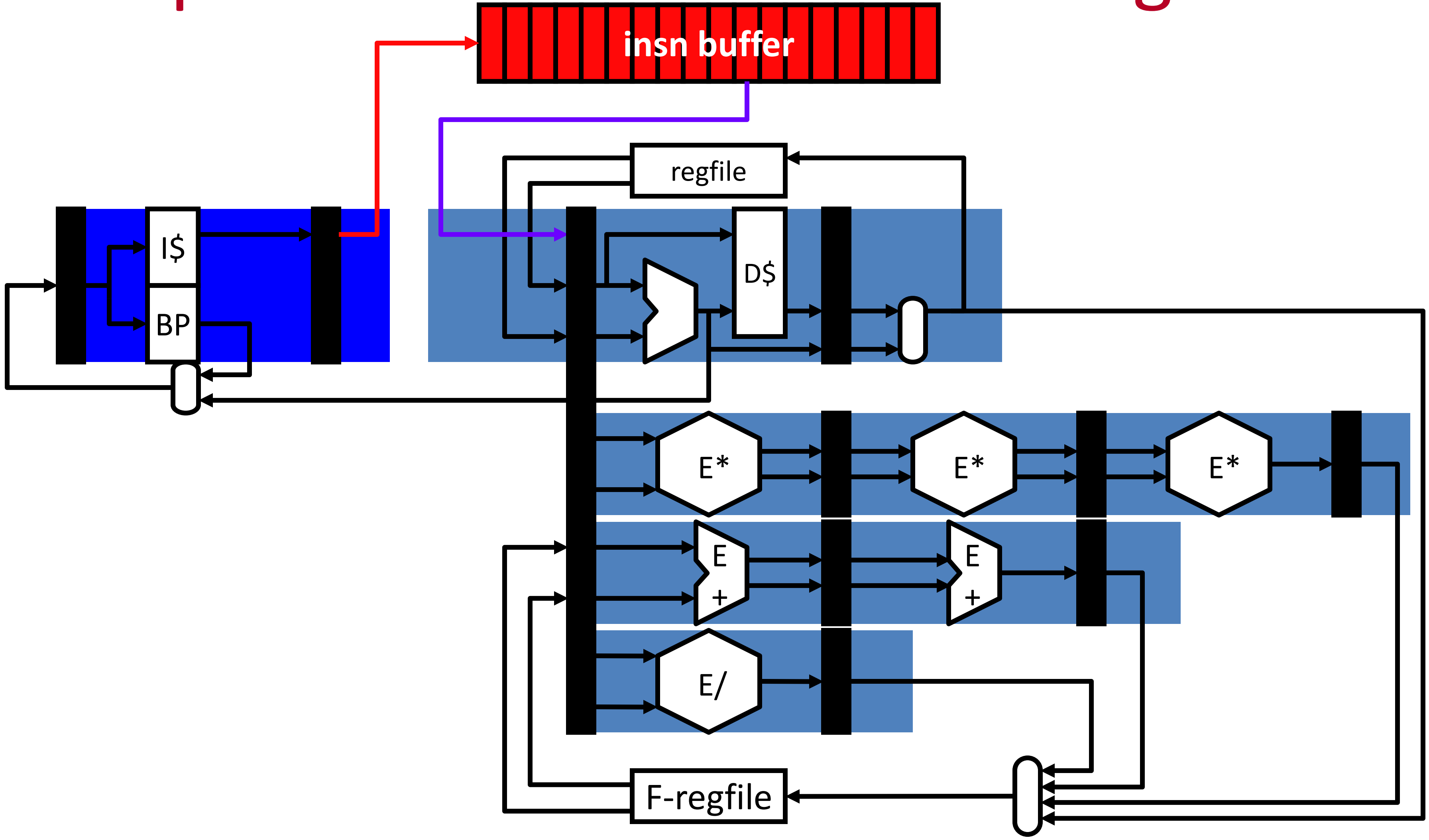
- Split D into two parts
  - Accumulate decoded insns. in buffer **in-order**
  - Buffer sends insns. down rest of pipeline **out-of-order**

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 with Floating-Point

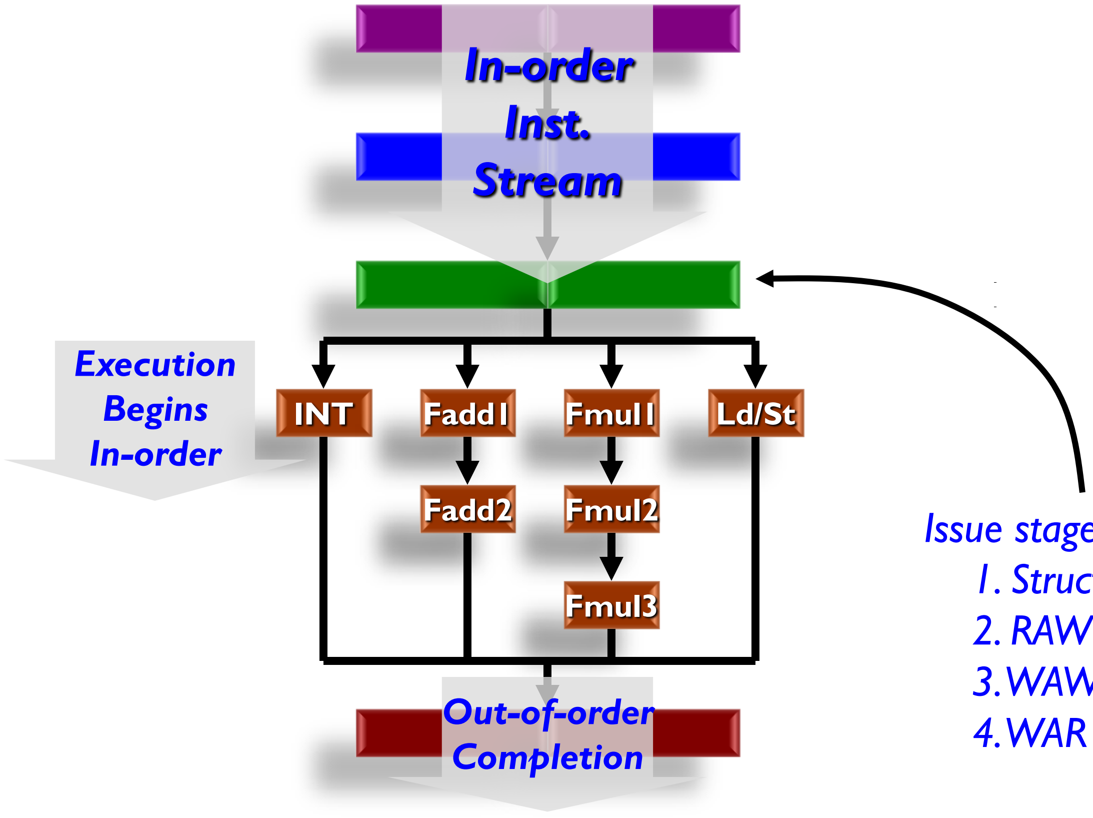


Number of pipeline stages per FU can vary

Our-of-Order Topics

- “*Scoreboarding*”
  - First OoO, no register renaming
- “*Tomasulo’s algorithm*”
  - OoO with register renaming
- Handling precise state and speculation
  - P6-style execution (Intel Pentium Pro)
  - R10k-style execution (MIPS R10k)
- Handling memory dependencies

In-Order Issue, OoO Completion



Issue stage needs to check:
1. *Structural Dependence*
2. *RAW Hazard*
3. *WAW Hazard*
4. *WAR Hazard*

Issue = send an instruction to execution

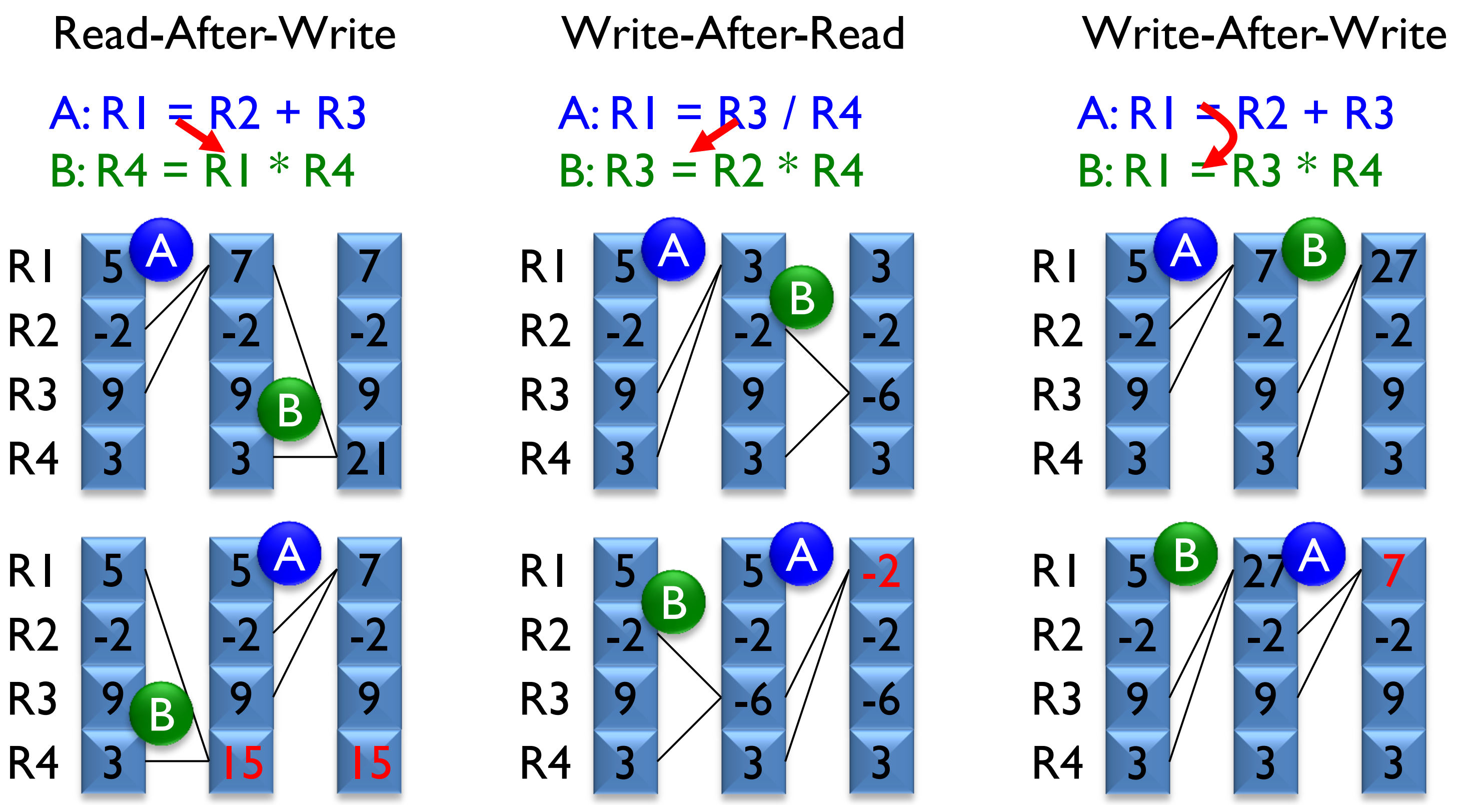
Track with Simple Scoreboarding

- Scoreboard: a bit-array, 1-bit for each GPR
  - If the bit is *not* set: the register has valid data
  - If the bit is set: the register has stale data

    i.e., some outstanding instruction is going to change it
- Issue in Order: RD ← Fn (RS, RT)
  - If SB[RS] or SB[RT] is set → RAW, stall
  - If SB[RD] is set → WAW, stall
  - Else, dispatch to FU (Fn) and set SB[RD]
- Complete out-of-order
  - Update GPR[RD], clear SB[RD]

Finite number of regs. will force WAR and WAW

Review of Register Dependencies

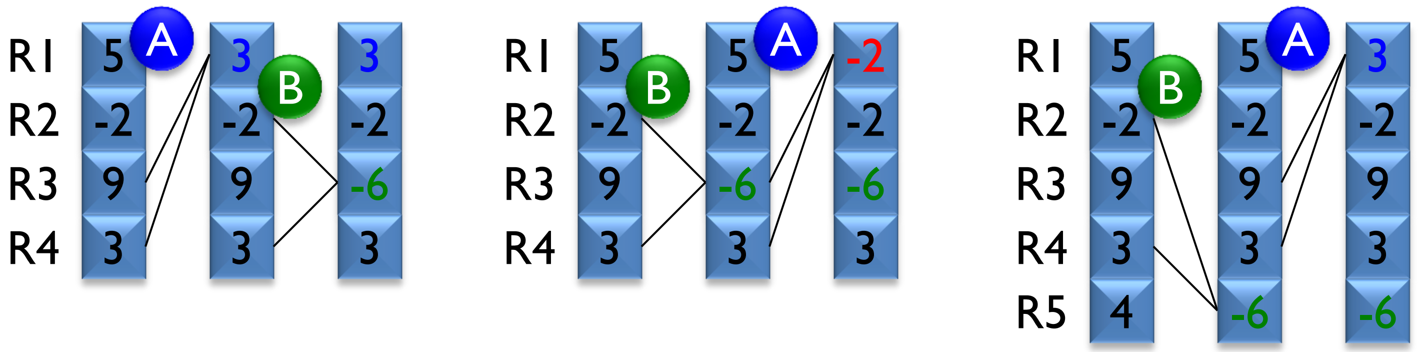


Eliminating WAR Dependencies

- WAR dependencies are from reusing registers

A: R1 = R3 / R4
B: R3 = R2 * R4

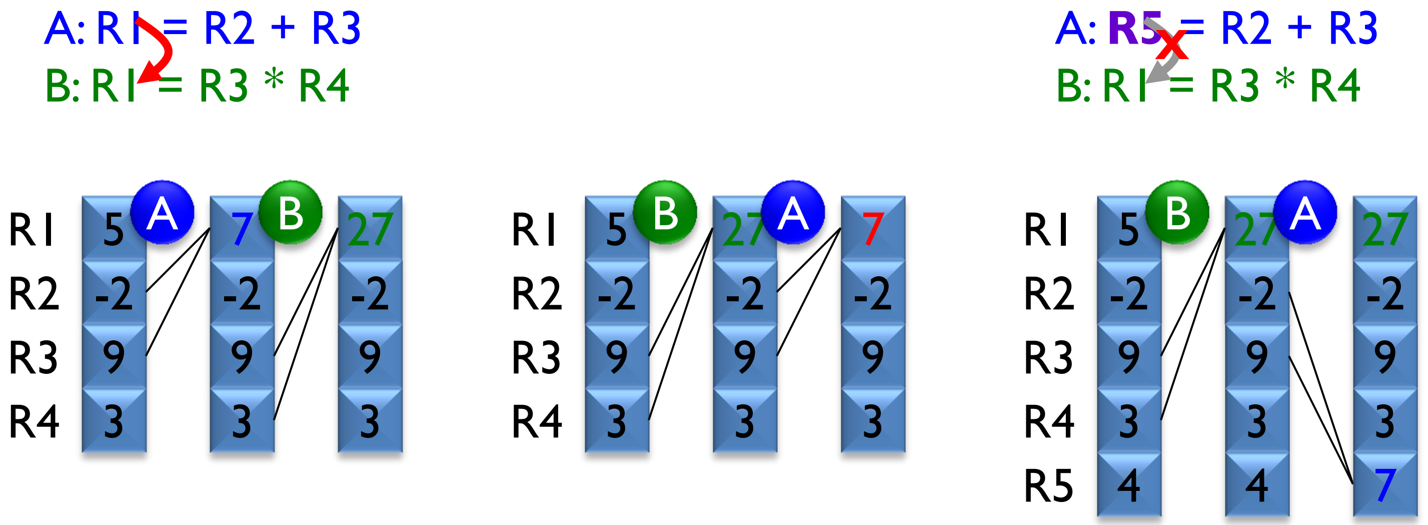
A: R1 = R3 / R4
B: **R5** = R2 * R4



Can get correct result just by using different reg.

Eliminating WAW Dependencies

- WAW dependencies are also from reusing registers



Can get correct result just by using different reg.

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, 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 Locations: p1,p2,p3,p4,p5,p6,p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"

MapTable

r1	r2	r3
p1	p2	p3

FreeList

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

Original insns.

```
add r2,r3,r1
```

Renamed insns.

```
add p2,p3,p4
```

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   Locations: p1,p2,p3,p4,p5,p6,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

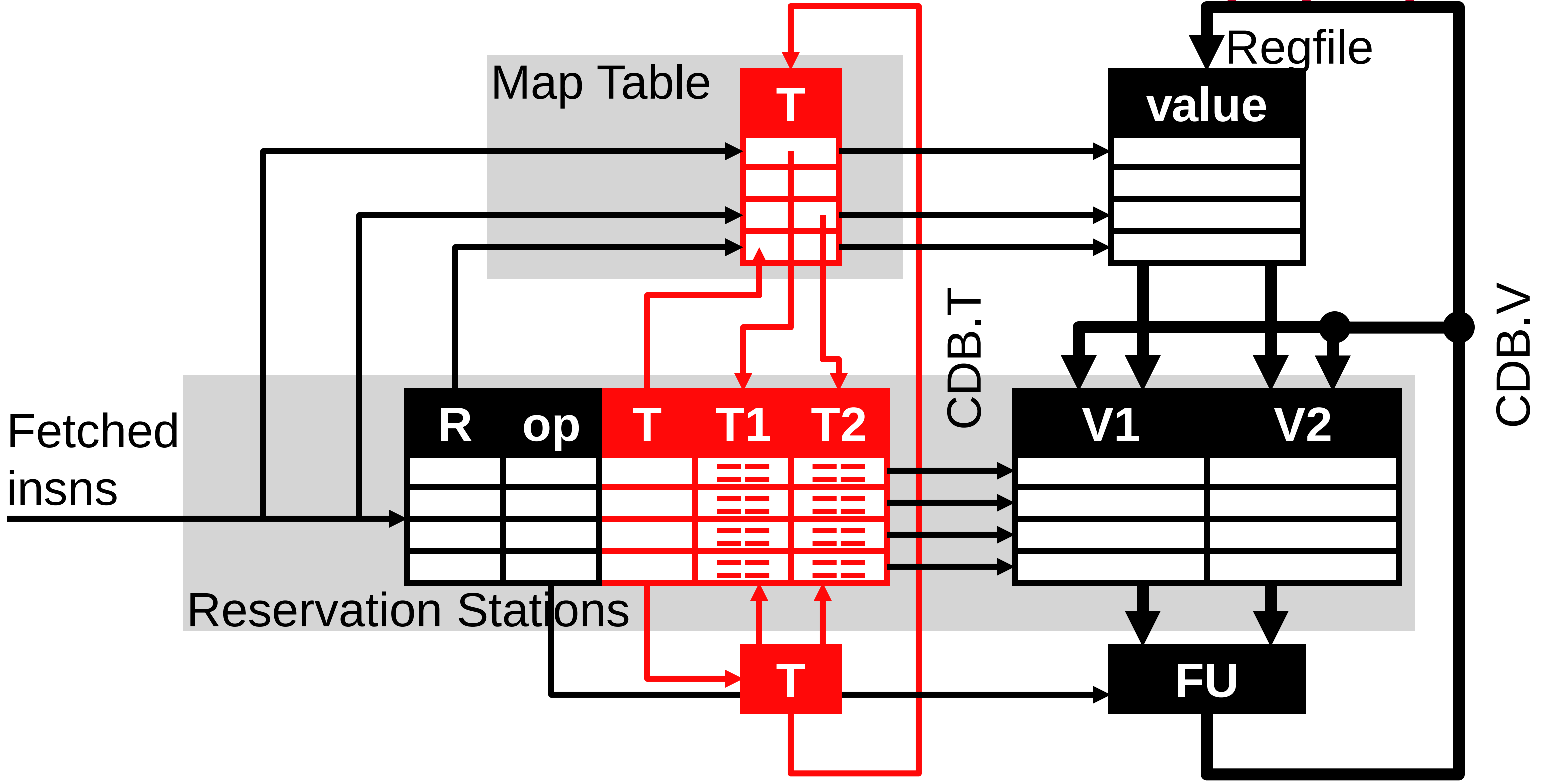
Tomasulo’s Algorithm

- *Reservation Stations* (RS): instruction buffer
- Common data bus (CDB): broadcasts results to RS
- Register renaming: removes WAR/WAW hazards
- Bypassing (not shown here to make example simpler)

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., RAT)
  - **T**: tag (RS#) that will write this register
- Common Data Bus (CDB)
  - Broadcasts <RS#, value> of completed insns.
- Valid tags indicate the RS# that will produce result

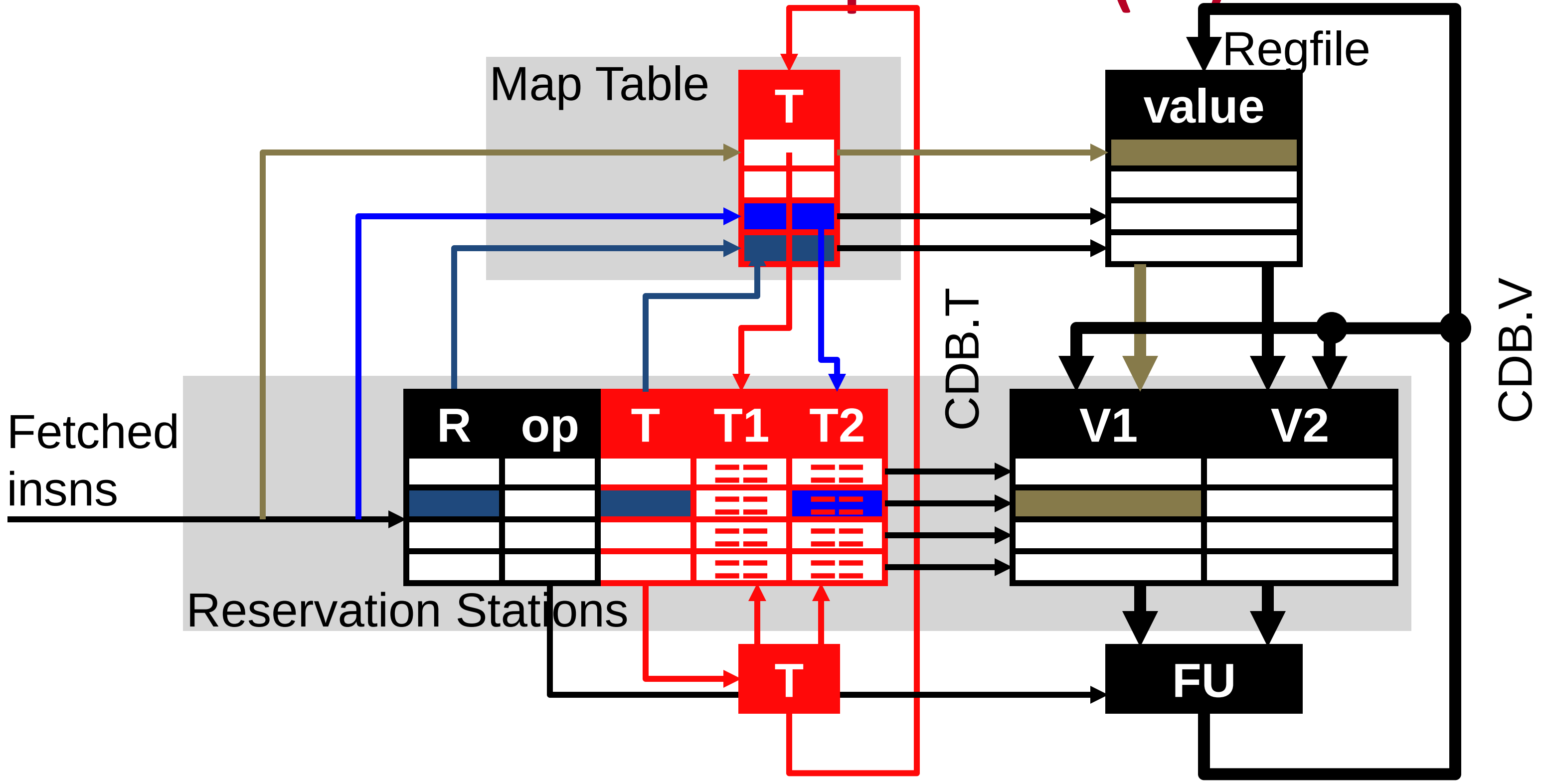
Tomasulo Data Structures (2/2)



Tomasulo Pipeline

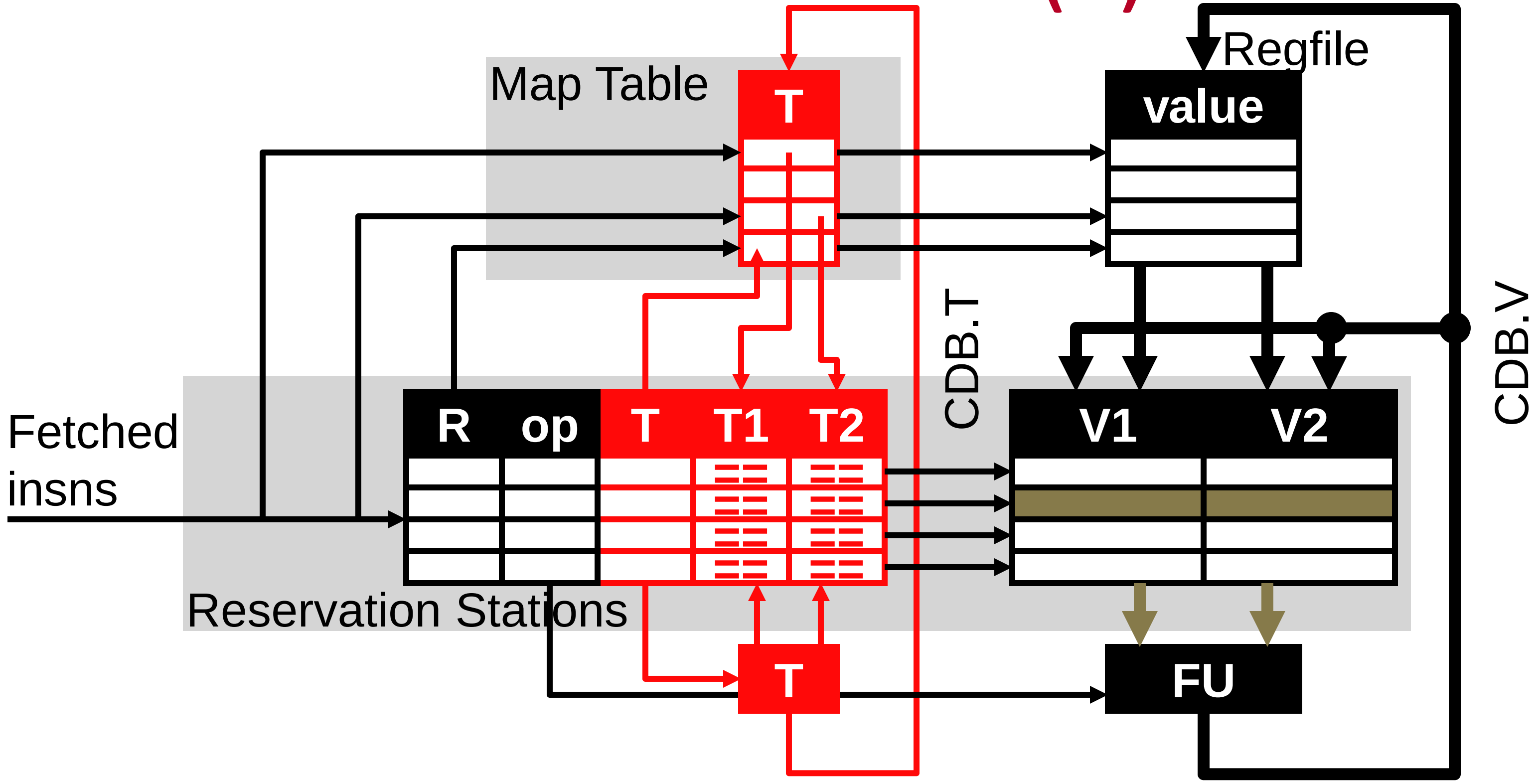
- New pipeline structure: F, **D**, **S**, X, **W**
  - **D (dispatch)**
    - **Structural** hazard ? **stall** : allocate RS entry
  - **S (issue)**
    - **RAW** hazard ? **wait** (monitor CDB) : go to execute
  - **W (writeback)**
    - **W**rite register, free RS entry
    - W and RAW-dependent S in same cycle
    - W and structural-dependent D in same cycle

Tomasulo Dispatch (D)



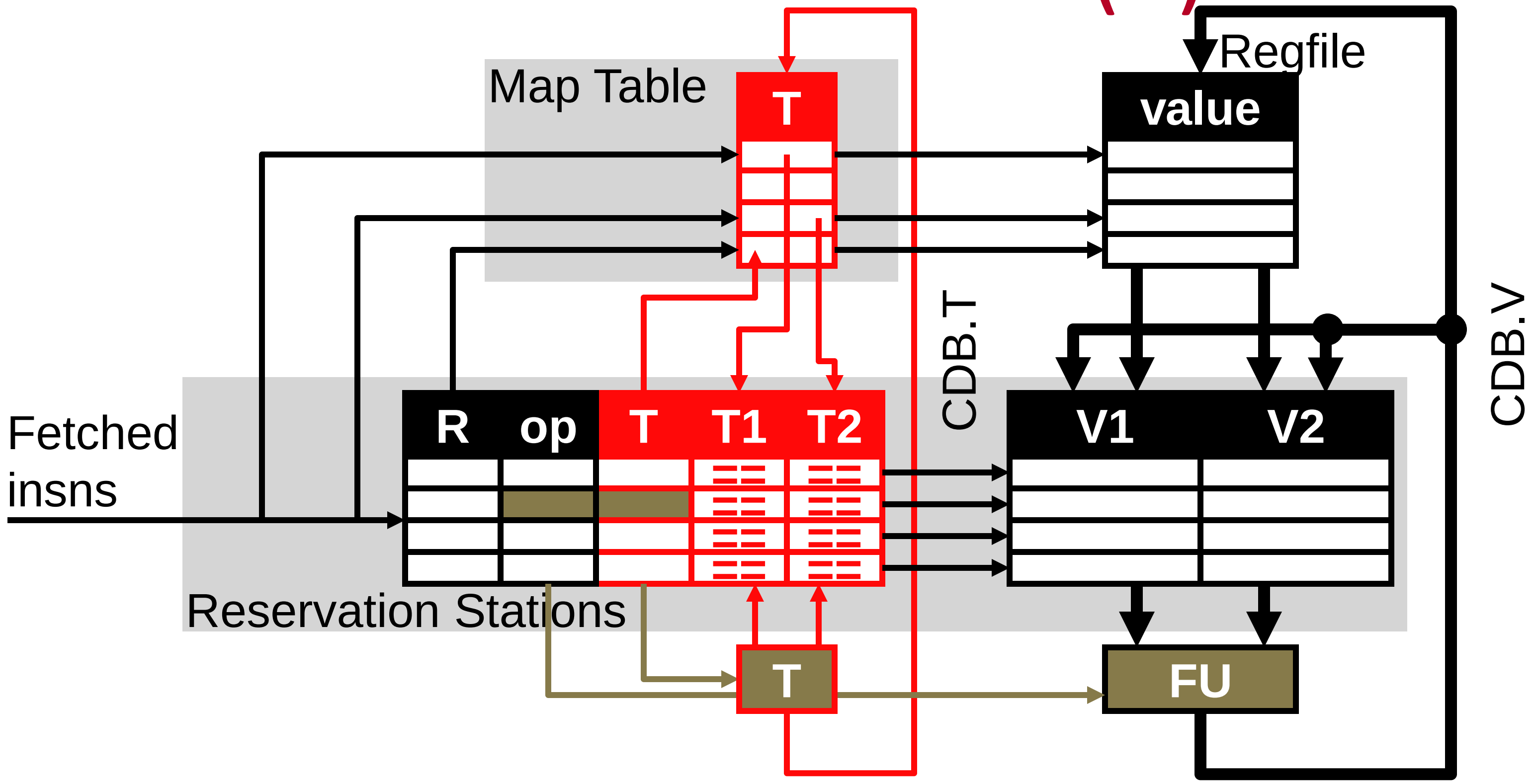
- Allocate RS entry (structural stall if busy)
  - Input register ready ? read value into RS : read tag into RS
  - Set register status (i.e., rename) for output register

Tomasulo Issue (S)

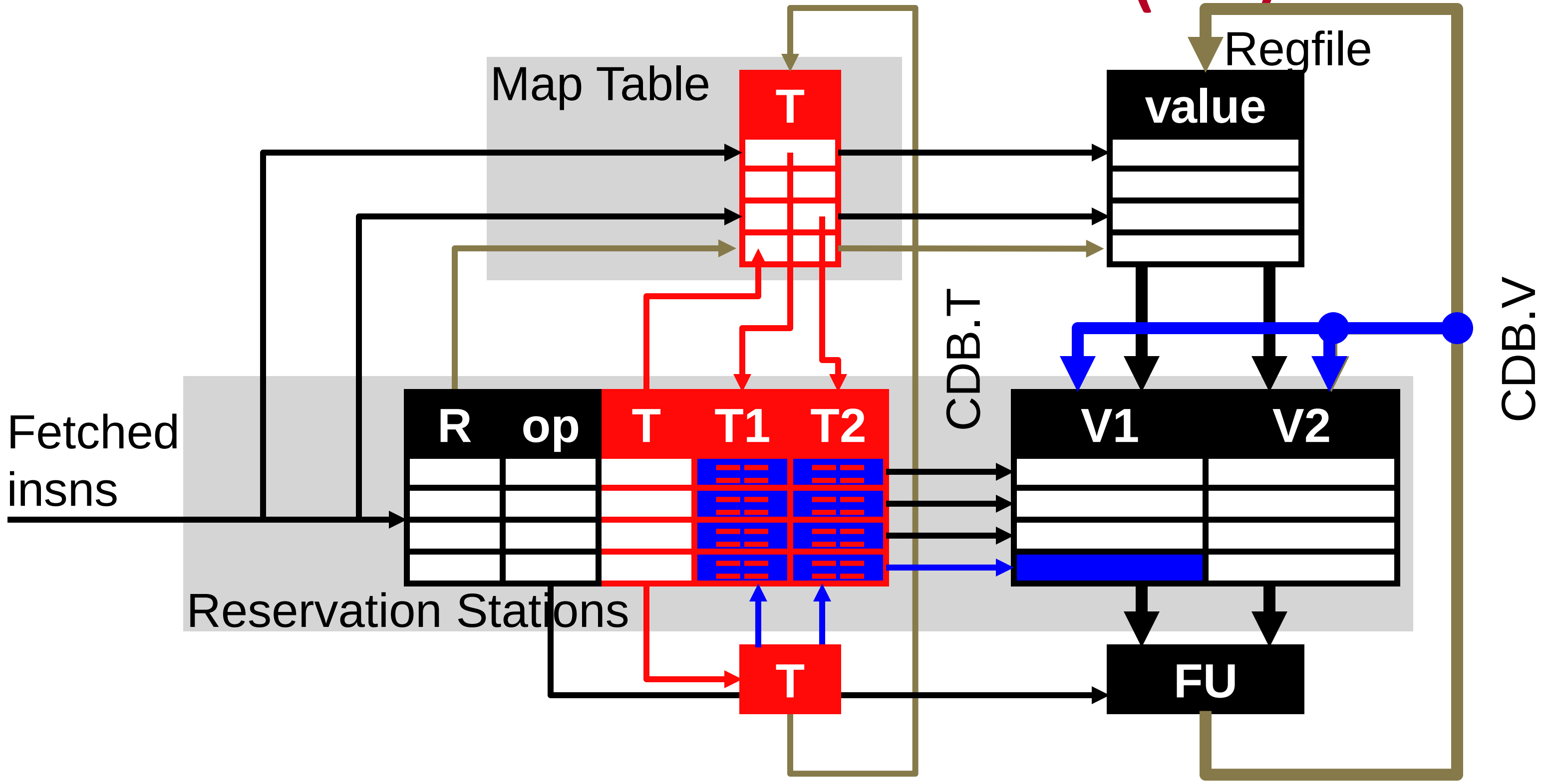


- Wait for RAW hazards
  - Read register values from RS

Tomasulo Execute (X)

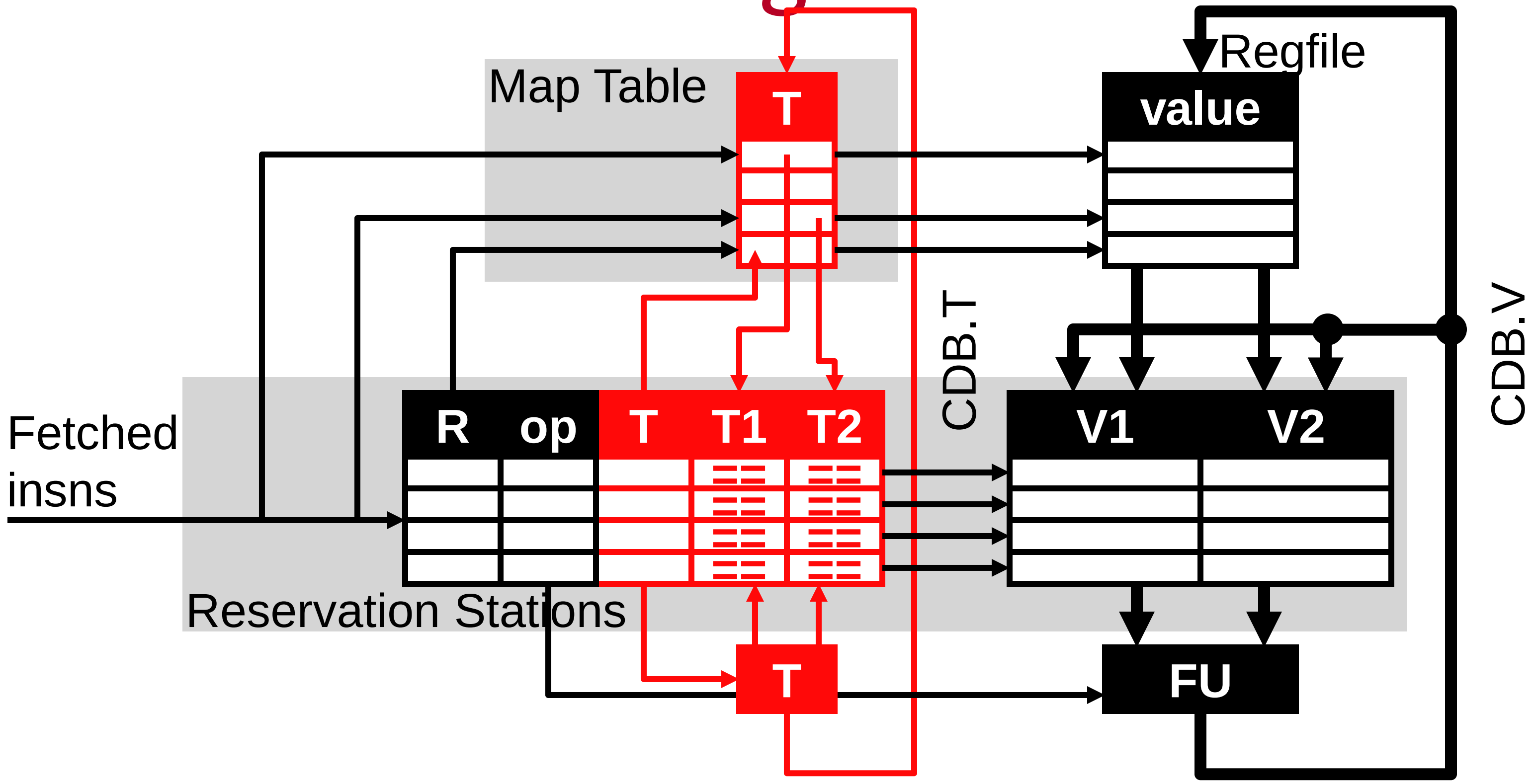


Tomasulo Writeback (W)



- Wait for structural (CDB) hazards
  - Output Reg tag still matches? 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

Tomasulo Data Structures

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

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

CDB	
T	V

no stall on WAW: scoreboard overwrites `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
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7		
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

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

CDB	
T	V
RS#1	[r1]

no W wait on WAR: scoreboard ensures 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
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	c8
stf f2,Z(r1)	c3	c8		
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7	c8	
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

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

CDB	
T	V
RS#4	[f2]

mulf finished (W)
don't clear f2 RegStatus
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
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	c8
stf f2,Z(r1)	c3	c8	c9	
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7	c8	c9
mulf f0,f1,f2	c6	c9		
stf f2,Z(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
`ldf X(r1),f1`	c1	c2	c3	c4
`mulf f0,f1,f2`	c2	c4	c5+	c8
`stf f2,Z(r1)`	c3	c8	c9	c10
`addi r1,4,r1`	c4	c5	c6	c7
`ldf X(r1),f1`	c5	c7	c8	c9
`mulf f0,f1,f2`	c6	c9	c10	
`stf f2,Z(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

Scoreboard vs. Tomasulo

	Scoreboard				Tomasulo			
Insn	D	S	X	W	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	c4	c1	c2	c3	c4
`mulf f0,f1,f2`	c2	c4	c5+	c8	c2	c4	c5+	c8
`stf f2,Z(r1)`	c3	c8	c9	c10	c3	c8	c9	c10
`addi r1,4,r1`	c4	c5	c6	c9	c4	c5	c6	c7
`ldf X(r1),f1`	c5	c9	c10	c11	c5	c7	c8	c9
`mulf f0,f1,f2`	c8	c11	c12+	c15	c6	c9	c10+	c13
`stf f2,Z(r1)`	c10	c15	c16	c17	c10	c13	c14	c15

Hazard	Scoreboard	Tomasulo
Insn buffer	stall in D	stall in D
FU	wait in S	wait in S
RAW	wait in S	wait in S
WAR	wait in W	none
WAW	stall in D	none

Can We Add Superscalar?

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