Precise State Recovery in Out-of-Order Pipelines

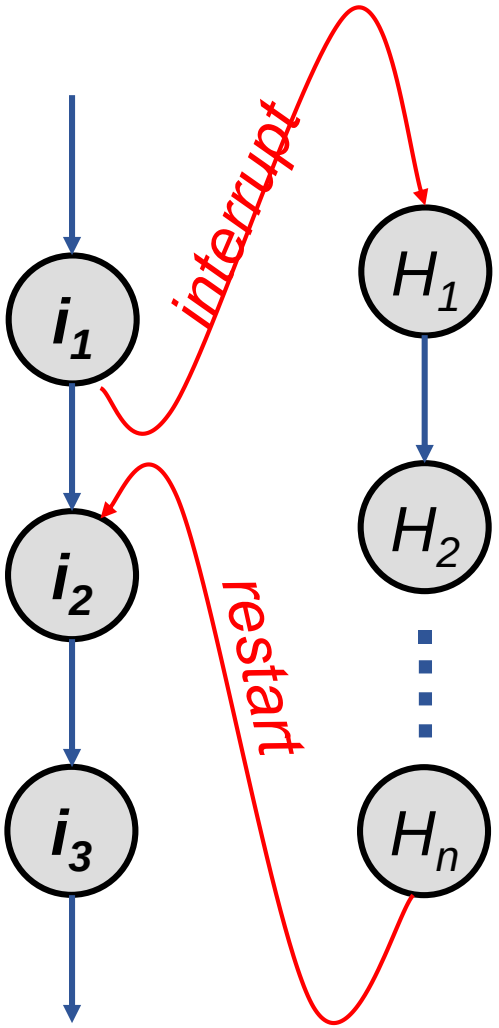
Nima Honarmand

Interrupts

- An unexpected transfer of control flow
  - Pick up where you left off once handled (***restartable***)
  - Transparent to interrupted program

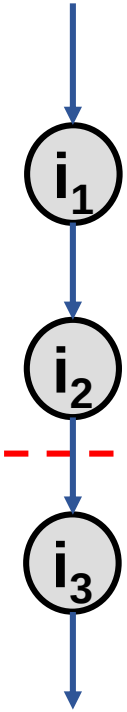
Kinds:

- Asynchronous
  - I/O device wants attention
  - Can "defer" interrupt until convenient
- Synchronous (aka exceptions, traps)
  - Unusual condition for some instruction
  - OS system calls

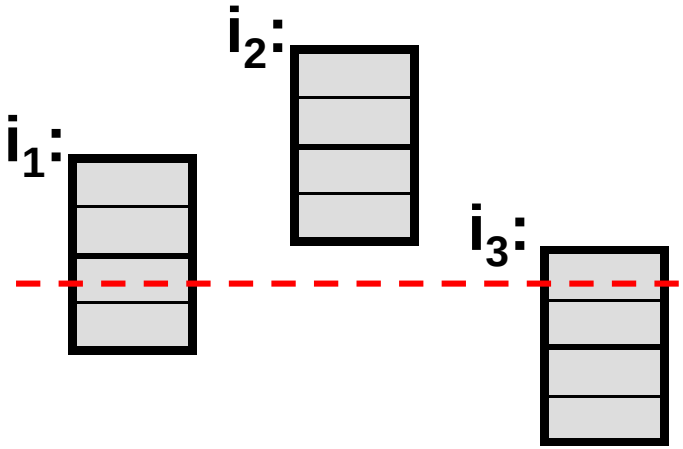


Precise Interrupts

Sequential Code Semantics



Overlapped/OoO Execution



***Precise** interrupt should appear to happen between two instructions*

Speculation and Precise Interrupts

- Why discussing these together:
  - Branch mis-speculation: must reset state (e.g., regs) to time of br.
    - All instructions before branch should be complete
    - All instructions after branch should look as if never started (abort)
  - We want sequential semantics for interrupts
    - All instructions before interrupt should be complete
    - All instructions after interrupt should look as if never started (abort)

  → Same problem, same (or similar) solution

- What makes this difficult?
  - OoO completion → must undo post-interrupt/branch writebacks

- Problems with Tomasulo:
  1. Don't know the relative order of instructions in RS
  2. How to undo post-interrupt/branch writebacks?

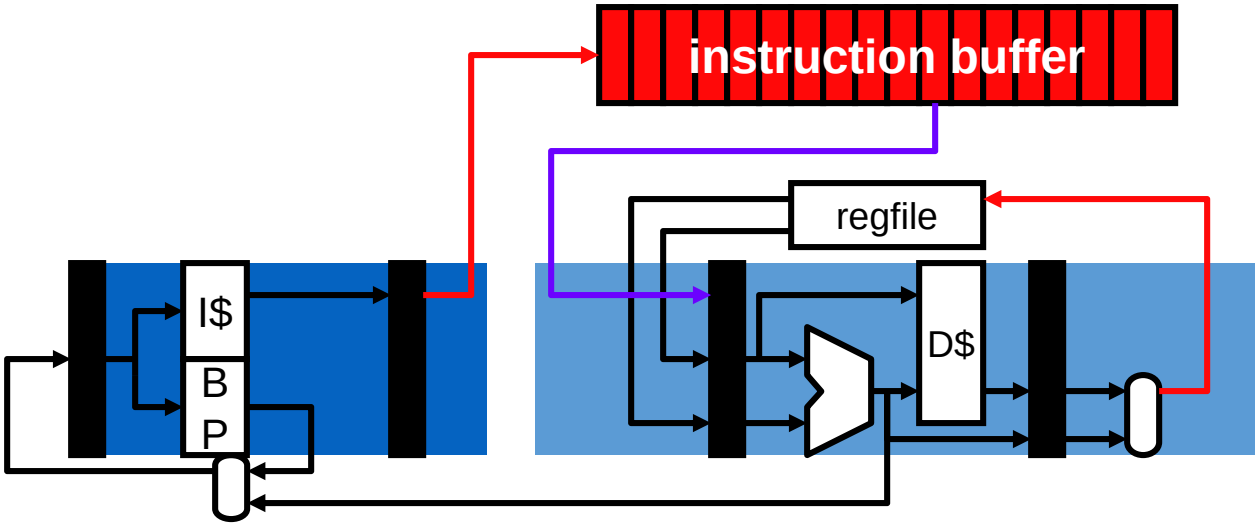
Precise State

- Speculative execution requires
  - Abort & restart at every branch (covered later)
  - Abort & restart at every load (covered later)
- Synchronous (exception and trap) events require
  - Abort & restart at every load, store, divide, …
- Asynchronous (hardware) interrupts require
  - Abort & restart at every ??
- Real world: bite the bullet
  - Implement abort & restart at every instruction
  - Called **Precise State**

Precise State Implementation Options

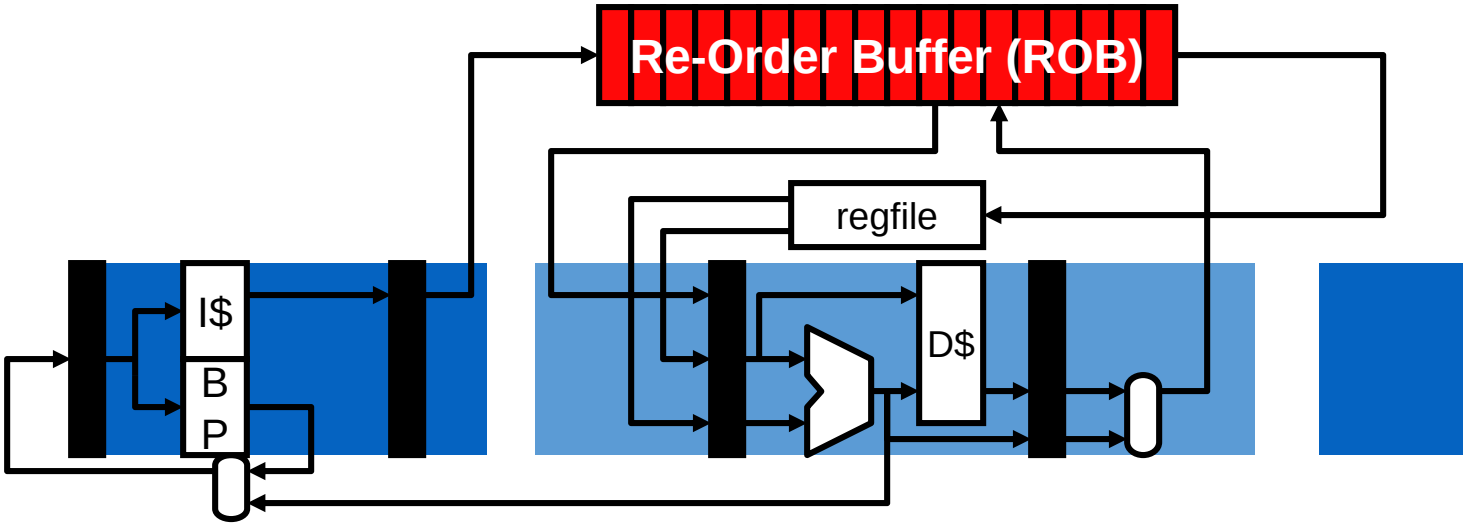
- Imprecise state: ignore the problem!
  - Makes page faults (any restartable exceptions) difficult
  - Makes speculative execution practically impossible
  - ✗ Bad idea!
- Force in-order completion (W): stall pipe if necessary
  - Slow (takes away benefit of Out-of-Order)
  - ✗ Bad idea!
- Keep track of precise state in hardware
  - Reset current state from precise state when needed

The Problem with Precise State



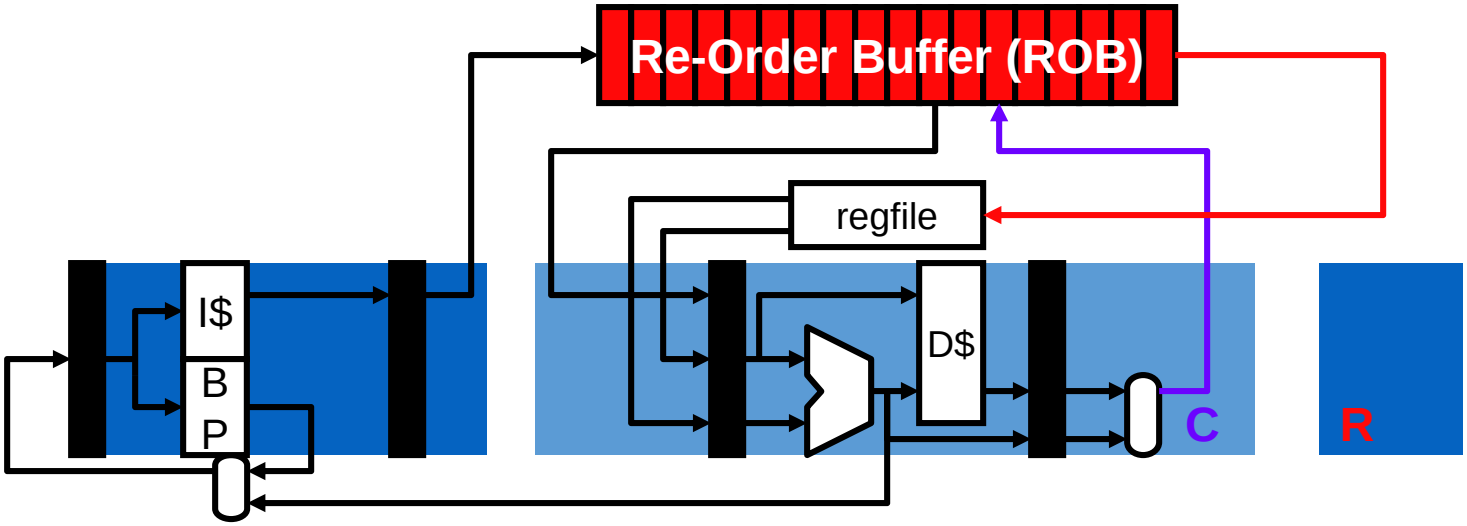
- Problem: writeback combines two functions
  - Forward values to younger instructions: out-of-order is OK
  - Write values to registers: needs to be in order
- Solution: split writeback into two stages
  - Similar to solution as for out-of-order dispatch and issue

Re-Order Buffer (ROB)



- Instruction buffer → *Re-Order Buffer (ROB)*
  - Buffer completed results en route to register file
  - Can be merged with RS or separate (common today)
- Split writeback (W) into two stages
  - Why is there no latch between W1 and W2?

Complete and Retire

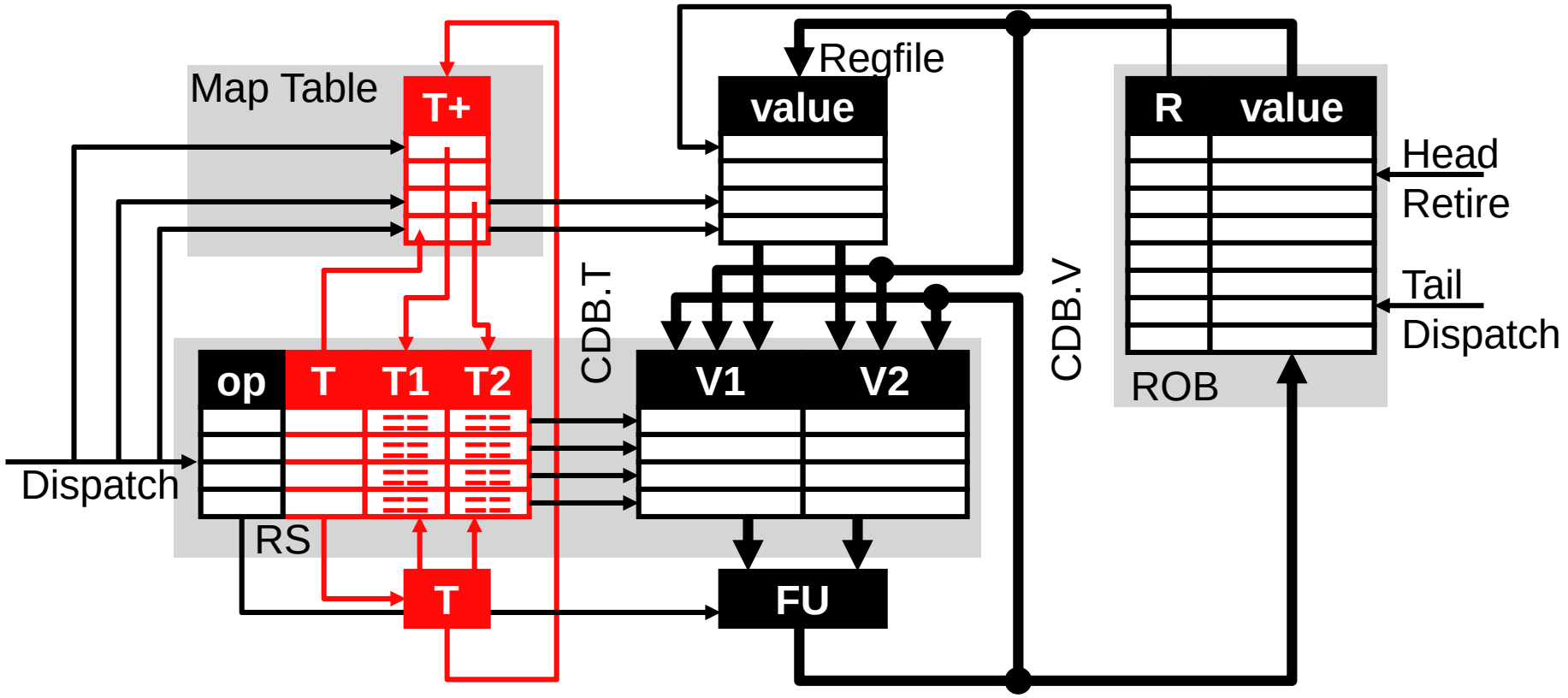


- Complete (**C**): instructions write results into ROB
  - Out-of-order: don't block younger instructions
- *Retire* (**R**): a.k.a. *commit*, graduate
  - ROB writes results to register file
  - In-order: stall back-propagates to younger instructions

P6 (Pentium Pro) Structures

- P6: Start with Tomasulo's algorithm… add ROB
- ROB (separate from RS)
  - **head**, **tail**: pointers maintain sequential order
  - **R**: instruction output register
  - **V**: instruction output value
- Tags are different
  - Tomasulo: RS# → P6: ROB#
- Map Table is different
  - **T+**: tag + "ready-in-ROB" bit
  - T==0 → Value is ready in register file
  - T==0+ → Value is ready in the ROB
  - T!=0 → Value is not ready

P6 Data Structures (1/2)



P6 Data Structures (2/2)

ROB							
ht	#	Insn	R	V	S	X	C
	1	`f1 = ldf (r1)`					
	2	`f2 = mulf f0,f1`					
	3	`stf f2,(r1)`					
	4	`r1 = addi r1,4`					
	5	`f1 = ldf (r1)`					
	6	`f2 = mulf f0,f1`					
	7	`stf f2,(r1)`					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	

CDB	
T	V

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

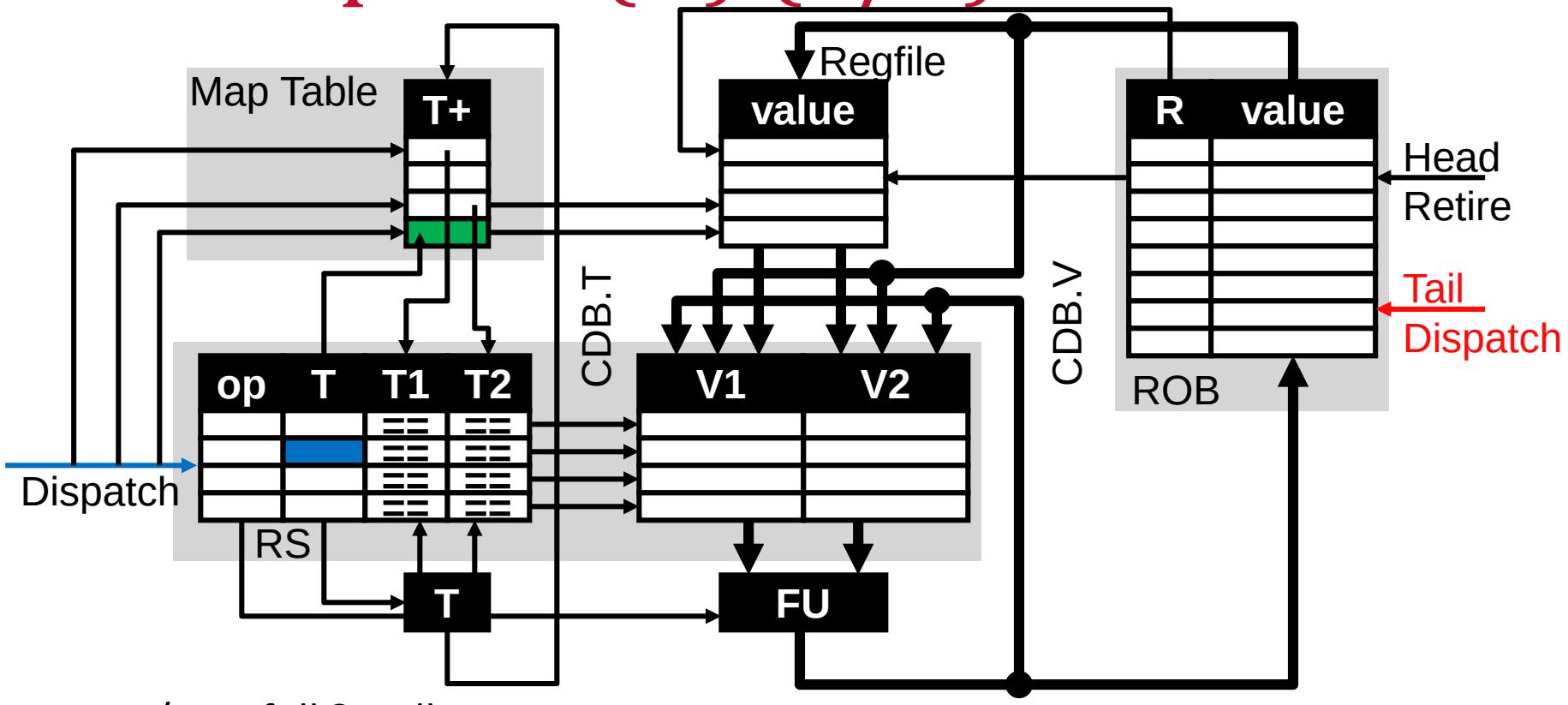
P6 Pipeline

- New pipeline structure: F, **D**, S, **X**, **C**, **R**
  - **D (dispatch)**
    - Structural hazard (ROB/RS) ? **stall**
    - Allocate ROB/RS
    - Set RS tag to ROB#
    - Set Map Table entry to ROB# and clear "ready-in-ROB" bit
    - Read ready registers into RS (from either ROB or Regfile)
  - **X (execute)**
    - Free RS entry
      - No need to wait for W, because tag is from ROB instead of RS

P6 Pipeline

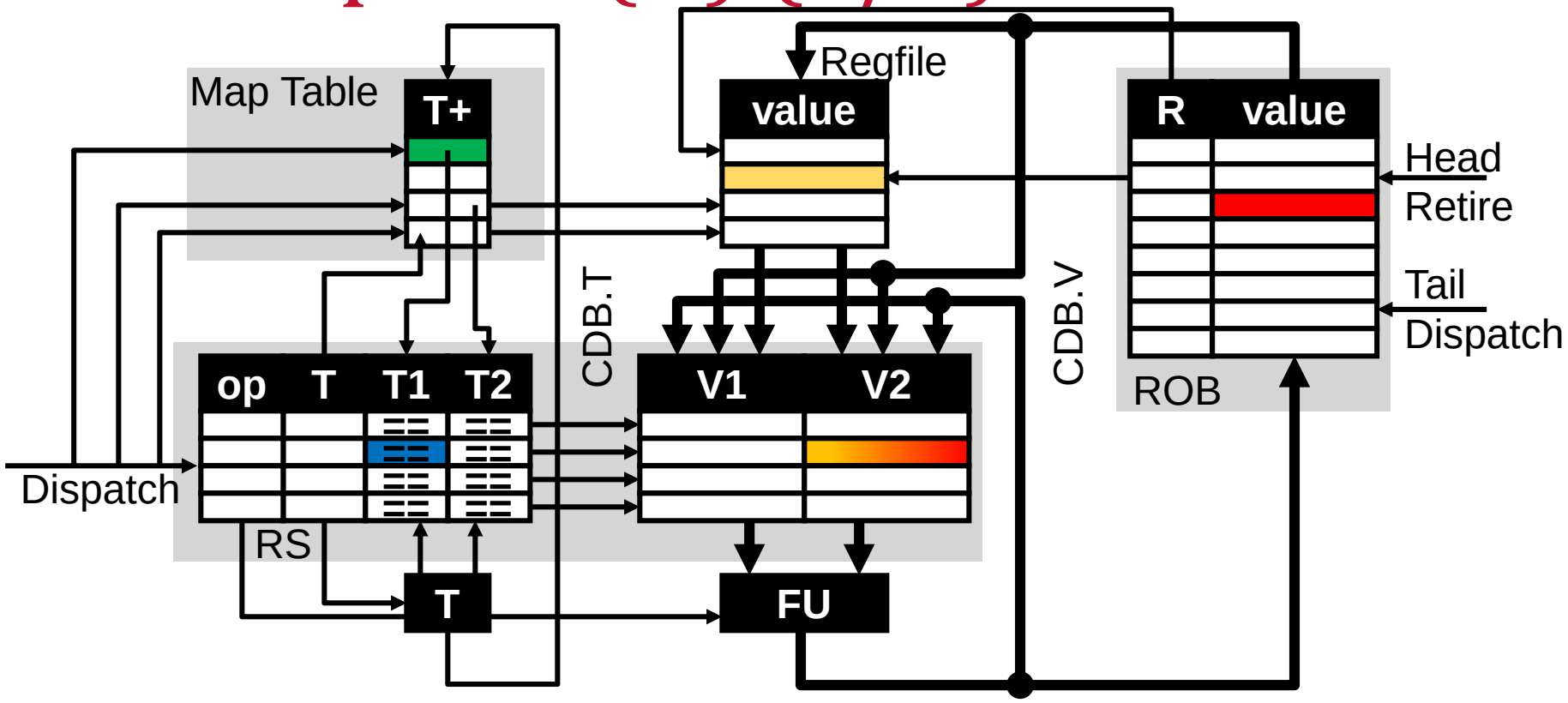
- **C (complete)**
  - Structural hazard (CDB)? **wait**
  - Write value into ROB entry
  - If Map Table has same entry, set "ready-in-ROB" bit (+)
- **R (retire)**
  - Instruction at ROB head not complete ? **stall**
  - Handle any exceptions
    - Some go before instruction (branch mispredict, page fault) – why?
    - Some go after instruction (e.g., trap) – why?
  - Copy Value of instruction at ROB head to Regfile
  - Free ROB entry

P6 Dispatch (D) (1/2)



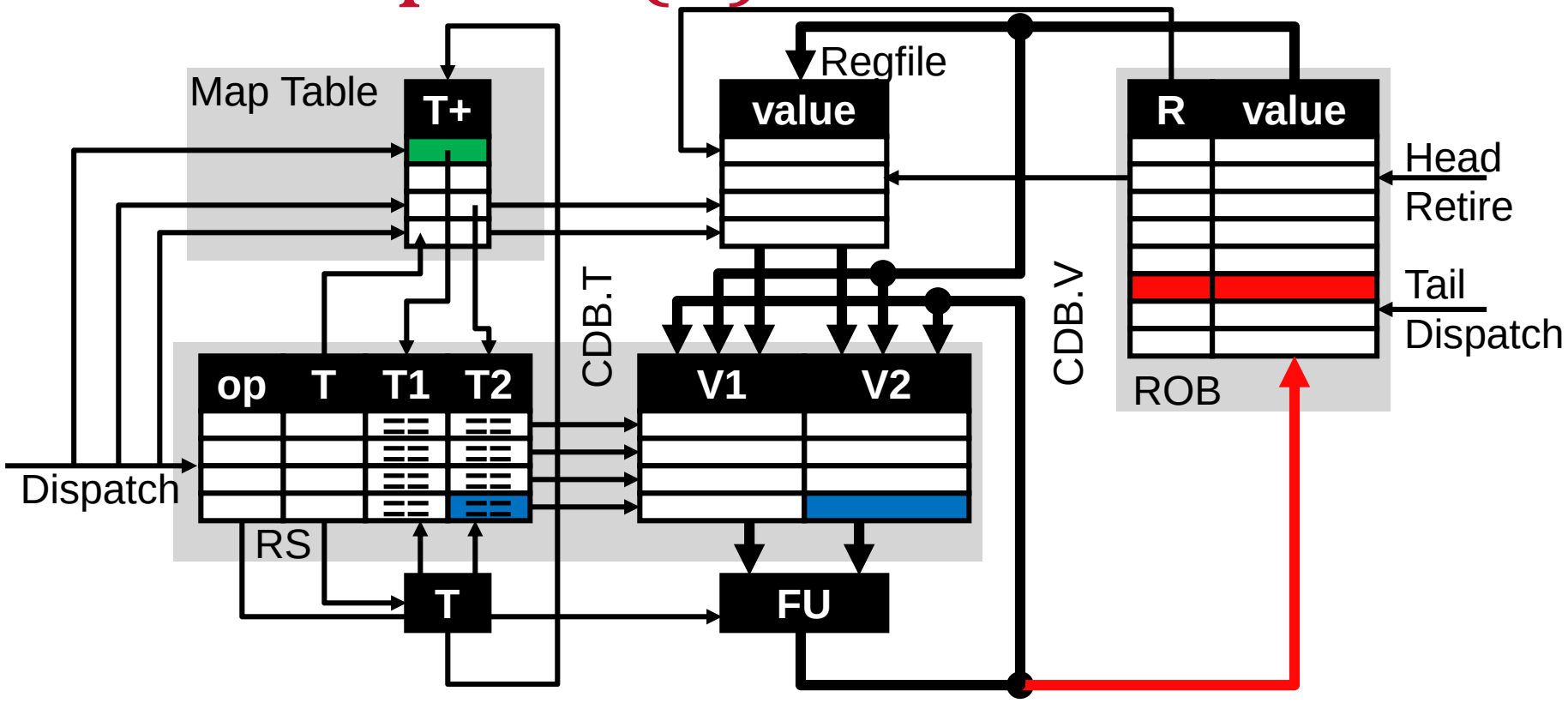
- RS/ROB full ? stall
- Allocate ROB entry
- Allocate RS entry, assign ROB# to RS output tag
- Map Table entry set to ROB#, clear “ready-in-ROB”

P6 Dispatch (D) (2/2)



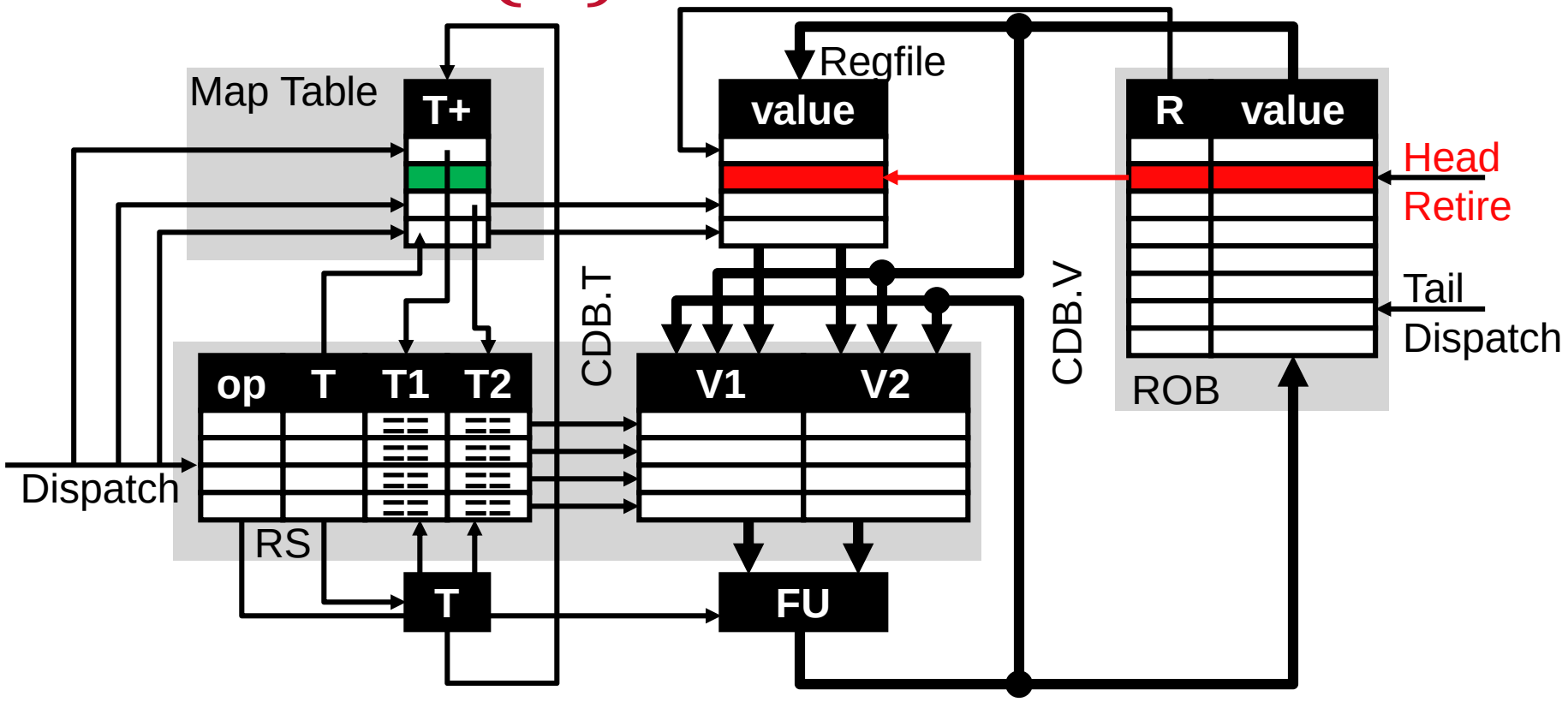
- Read tags for register inputs from Map Table
  - Tag==0 → value from Regfile
  - Tag==0+ → value from ROB
  - Tag!=0 → Map Table tag to RS

P6 Complete (C)



- CDB busy ? stall : broadcast <value,tag> on CDB
- Result → ROB
- if MapTable entry matches tag (T) → "ready-in-ROB" bit
- If RS T1 or T2 matches, write CDB.V into RS slot

P6 Retire (R)



- ROB head not complete ? stall : free ROB entry
  - Write ROB head result to Regfile
  - if MapTable entry matches tag (T), clear the entry

P6: Cycle 1

ROB							
ht	#	Insn	R	V	S	X	C
ht	1	f1 = ldf (r1)	f1				
	2	f2 = mulf f0,f1					
	3	stf f2,(r1)					
	4	r1 = addi r1,4					
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1
f2	
r1	

CDB	
T	V

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

set ROB# tag

allocate

P6: Cycle 2

ROB							
ht	#	Insn	R	V	S	X	C
h	1	f1 = ldf (r1)	f1		c2		
t	2	f2 = mulf f0,f1	f2				
	3	stf f2,(r1)					
	4	r1 = addi r1,4					
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1
f2	ROB#2
r1	

CDB	
T	V

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

set ROB# tag

allocate

P6: Cycle 3

ROB							
ht	#	Insn	R	V	S	X	C
h	1	f1 = ldf (r1)	f1		c2	c3	
	2	f2 = mulf f0,f1	f2				
t	3	stf f2,(r1)					
	4	r1 = addi r1,4					
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1
f2	ROB#2
r1	

CDB	
T	V

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

free
allocate

P6: Cycle 4

ROB							
ht	#	Insn	R	V	S	X	C
h	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2		c4		
	3	stf f2,(r1)					
t	4	r1 = addi r1,4	r1				
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

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

CDB	
T	V
ROB#1	[f1]

`ldf` finished

1. set "ready-in-ROB" bit
2. write result to ROB
3. CDB broadcast

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

allocate

ROB#1 ready
grab CDB.V

P6: Cycle 5

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
h	2	f2 = mulf f0,f1	f2		c4	c5	
	3	stf f2,(r1)					
	4	r1 = addi r1,4	r1		c5		
t	5	f1 = ldf (r1)	f1				
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#2
r1	ROB#4

CDB	
T	V

ldf retires
1. write ROB result to regfile

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	yes	add	ROB#4			[r1]	
2	LD	yes	ldf	ROB#5		ROB#4		
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	no						
5	FP2	no						

allocate

free

P6: Cycle 6

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
h	2	f2 = mulf f0,f1	f2		c4	c5+	
	3	stf f2,(r1)					
	4	r1 = addi r1,4	r1		c5	c6	
	5	f1 = ldf (r1)	f1				
t	6	f2 = mulf f0,f1	f2				
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#6
r1	ROB#4

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	ROB#5		ROB#4		
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	
5	FP2	no						

free

allocate

P6: Cycle 7

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
h	2	f2 = mulf f0,f1	f2		c4	c5+	
	3	stf f2,(r1)					
	4	r1 = addi r1,4	r1	[r1]	c5	c6	c7
	5	f1 = ldf (r1)	f1		c7		
t	6	f2 = mulf f0,f1	f2				
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#4	[r1]

stall Dispatch (no free STore RS)

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	ROB#5		ROB#4		CDB.V
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	
5	FP2	no						

ROB#4 ready
grab CDB.V

P6: Cycle 8

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
h	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
	3	stf f2,(r1)			c8		
	4	r1 = addi r1,4	r1	[r1]	c5	c6	c7
	5	f1 = ldf (r1)	f1		c7	c8	
t	6	f2 = mulf f0,f1	f2				
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#2	[f2]

addi stall Retire (in-order retire)

ROB#2 invalid in MapTable
don't set "ready-in-ROB"

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#3	ROB#2		[f2]	[r1]
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	
5	FP2	no						

ROB#2 ready
grab CDB.V

P6: Cycle 9

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
h	3	stf f2,(r1)			c8	c9	
	4	r1 = addi r1,4	r1	[r1]	c5	c6	c7
	5	f1 = ldf (r1)	f1	[f1]	c7	c8	c9
	6	f2 = mulf f0,f1	f2		c9		
t	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#5	[f1]

retire mulf

all pipe stages active at once!

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	CDB.V
5	FP2	no						

free → re-allocate

ROB#5 ready

grab CDB.V

P6: Cycle 10

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
h	3	stf f2,(r1)			c8	c9	c10
	4	r1 = addi r1,4	r1	[r1]	c5	c6	c7
	5	f1 = ldf (r1)	f1	[f1]	c7	c8	c9
	6	f2 = mulf f0,f1	f2		c9	c10	
t	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	no						
5	FP2	no						

free

P6: Cycle 11

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5	c8
	3	stf f2,(r1)			c8	c9	c10
h	4	r1 = addi r1,4	r1	[r1]	c5	c6	c7
	5	f1 = ldf (r1)	f1	[f1]	c7	c8	c9
	6	f2 = mulf f0,f1	f2		c9	c10	
t	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

retire stf

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	no						
5	FP2	no						

Precise State in P6

- Point of ROB is maintaining precise state
  - How does that work?
    1. Wait until last good insn. retires, first bad insn. at ROB head
    2. Zero (0) contents of ROB, RS, and Map Table
    3. Start over
  - Works because zero (0) means the right thing…
    - 0 in ROB/RS → entry is empty
    - Tag == 0 in Map Table → register is in Regfile
  - …and because writes to regfile and D$ take place at R
- Example: page fault in first `stf`
  - Next slide…

P6: Cycle 9 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
h	3	stf f2,(r1)			c8	c9	
	4	r1 = addi r1,4	r1	[r1]	c5	c6	c7
	5	f1 = ldf (r1)	f1	[f1]	c7	c8	c9
	6	f2 = mulf f0,f1	f2		c9		
t	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#5	[f1]

PAGE FAULT

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	CDB.V
5	FP2	no						

P6: Cycle 10 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
	3	stf f2,(r1)					
	4	r1 = addi r1,4					
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	

CDB	
T	V

faulting insn at ROB head?
CLEAR EVERYTHING
set fetch PC to fault handler

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

P6: Cycle 11 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
ht	3	stf f2,(r1)					
	4	r1 = addi r1,4					
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	

CDB	
T	V

PF handler done?
CLEAR EVERYTHING
`iret` fetch PC to faulting insn.

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#3			[f4]	[r1]
4	FP1	no						
5	FP2	no						

P6: Cycle 12 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	f1 = ldf (r1)	f1	[f1]	c2	c3	c4
	2	f2 = mulf f0,f1	f2	[f2]	c4	c5+	c8
h	3	stf f2,(r1)			c12		
t	4	r1 = addi r1,4	r1				
	5	f1 = ldf (r1)					
	6	f2 = mulf f0,f1					
	7	stf f2,(r1)					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	ROB#4

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	yes	addi	ROB#4			[r1]	
2	LD	no						
3	ST	yes	stf	ROB#3			[f4]	[r1]
4	FP1	no						
5	FP2	no						

P6 Performance

- What is the cost of precise state?
  - + In general: same performance as "plain" Tomasulo
    - ROB is not a performance device
    - Maybe a little better (RS freed earlier → fewer structural hazards)
  - – Unless ROB is too small
    - In which case ROB structural hazards become a problem
  - – Rules of thumb for ROB size
    - At least N (pipe width) * number of pipe stages between D and R
    - At least N * $t_{hit\text{-}L2}$
    - Can add a factor of 2 to both if you want
    - What is the rationale behind these?