



Modern Computer Architecture

Lecture5 Instruction Level Parallelism (I)

Hongbin Sun

国家集成电路人才培养基地

Xi'an Jiaotong University

Review: Pipeline Performance

- **Pipeline CPI = Ideal pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls**
 - **Ideal pipeline CPI**: measure of the maximum performance attainable by the implementation
 - **Structural hazards**: HW cannot support this combination of instructions
 - **Data hazards**: Instruction depends on result of prior instruction still in the pipeline
 - **Control hazards**: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches, jumps, exceptions)


Review: Types of Data Hazards

Consider executing a sequence of

$$r_k \leftarrow (r_i) \text{ op } (r_j)$$


type of instructions

Data-dependence

$$\begin{array}{l} r_3 \leftarrow (r_1) \text{ op } (r_2) \\ r_5 \leftarrow (r_3) \text{ op } (r_4) \end{array}$$



Read-after-Write
(RAW) hazard

Anti-dependence

$$\begin{array}{l} r_3 \leftarrow (r_1) \text{ op } (r_2) \\ r_1 \leftarrow (r_4) \text{ op } (r_5) \end{array}$$


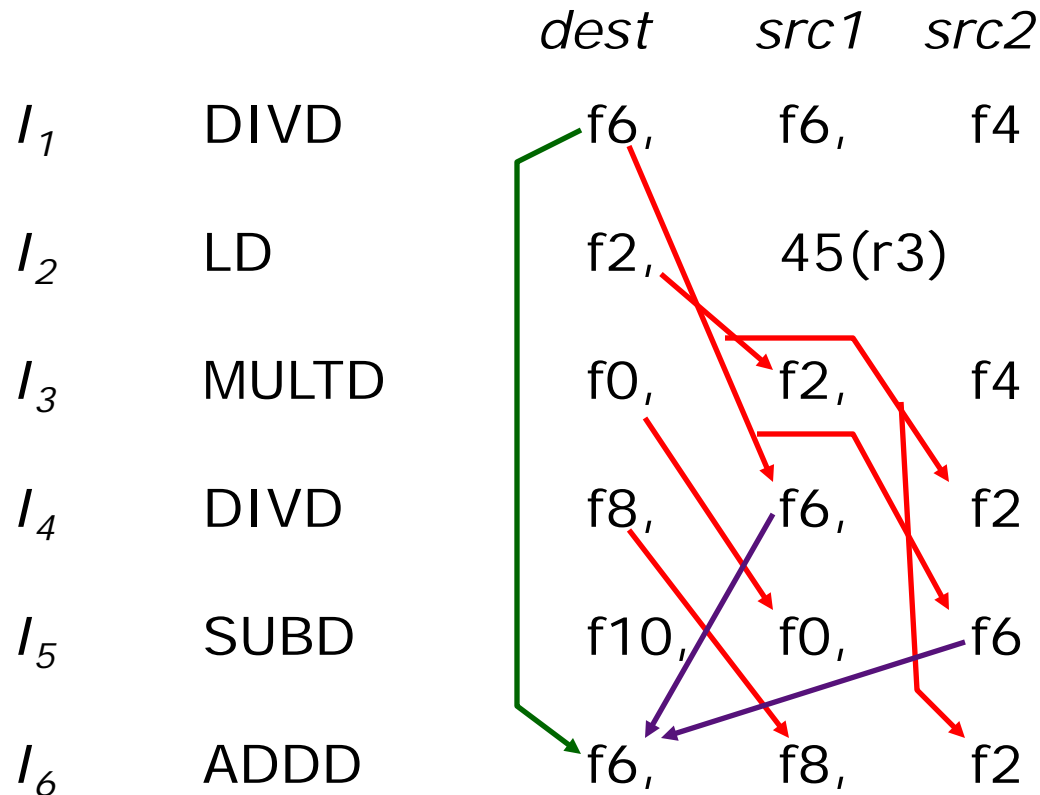
Write-after-Read
(WAR) hazard

Output-dependence

$$\begin{array}{l} r_3 \leftarrow (r_1) \text{ op } (r_2) \\ r_3 \leftarrow (r_6) \text{ op } (r_7) \end{array}$$


Write-after-Write
(WAW) hazard

Data Hazards: An Example

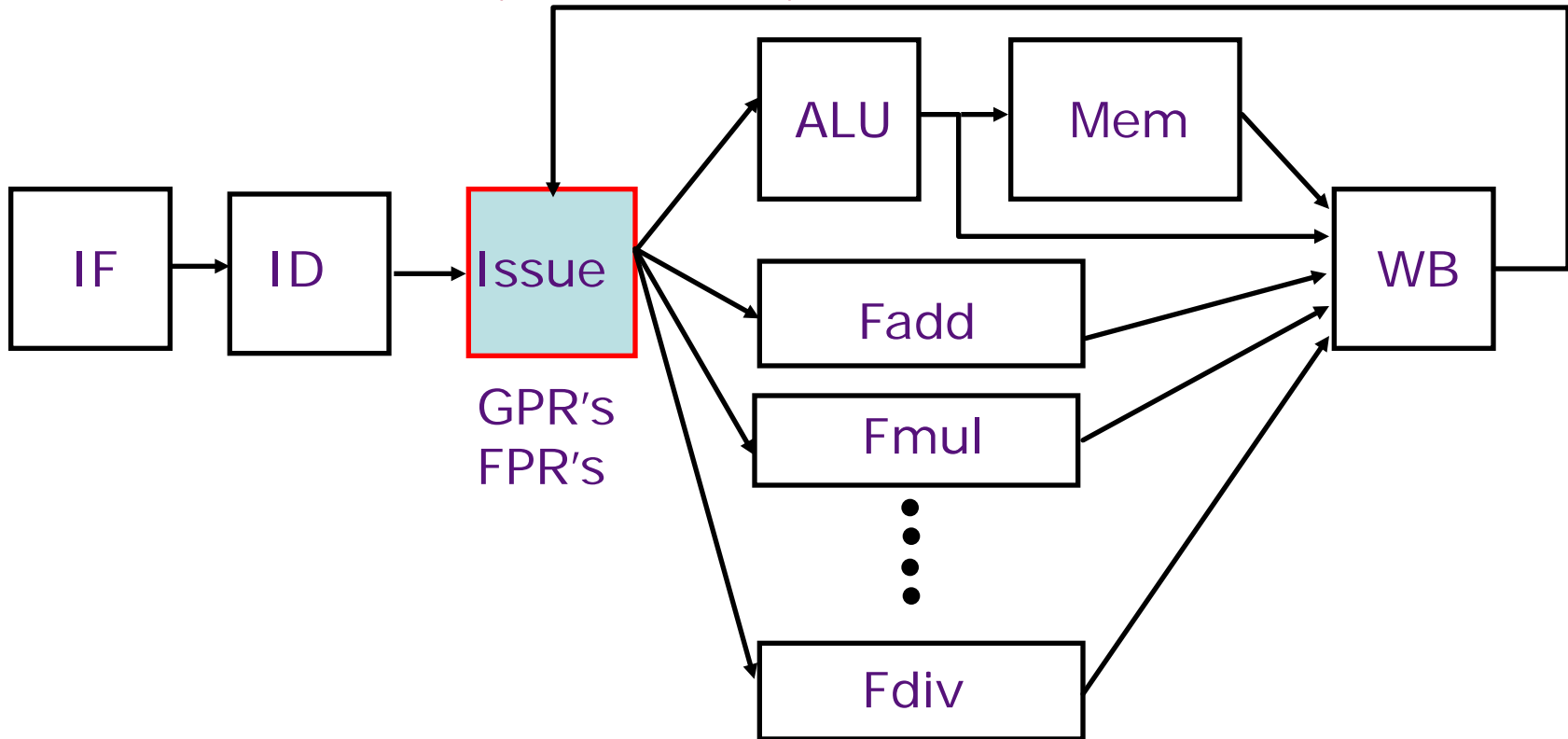


RAW Hazards

WAR Hazards

WAW Hazards

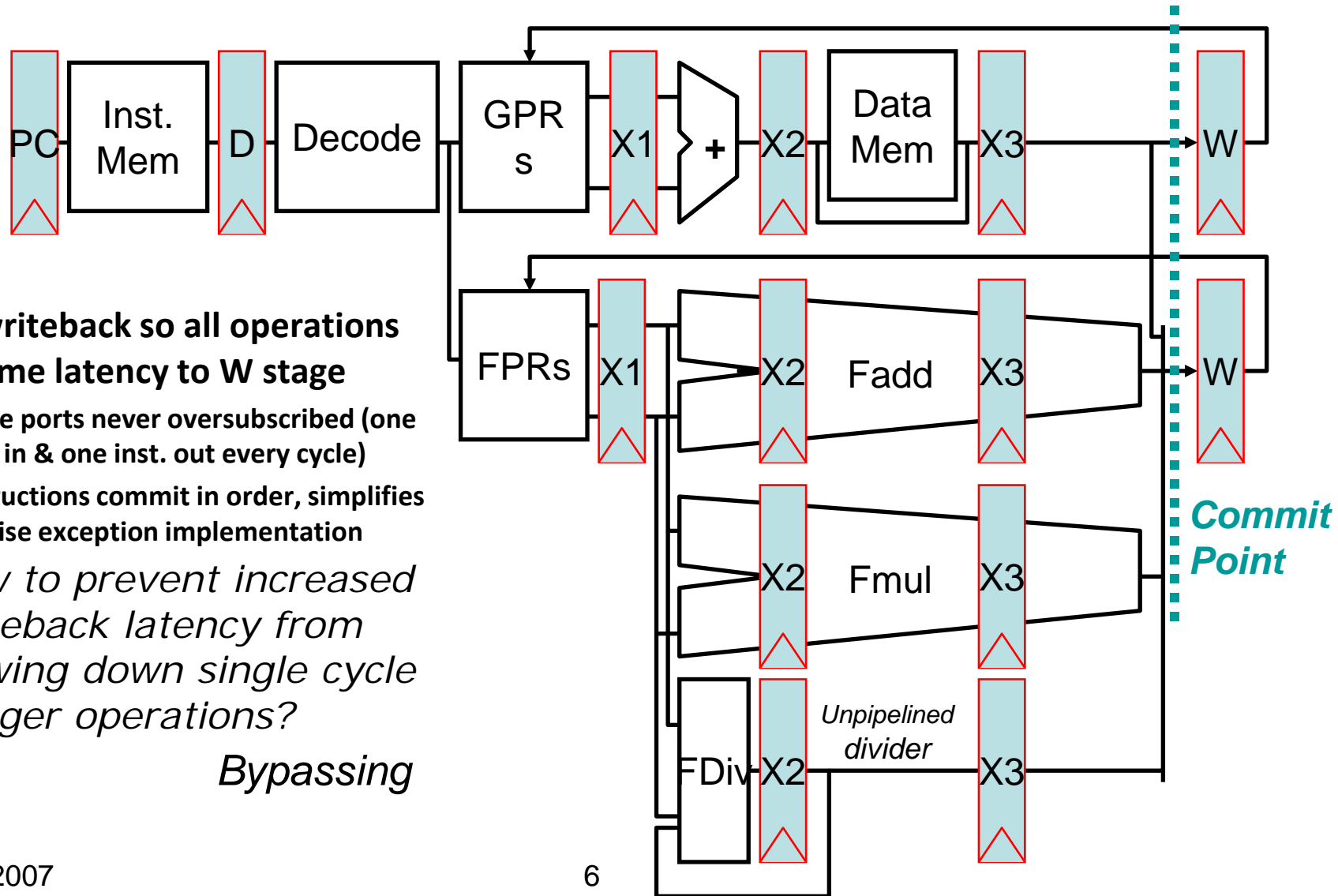
Complex Pipelining



Pipelining becomes complex when we want high performance in the presence of:

- Long latency or partially pipelined floating-point units
- Multiple function and memory units
- Memory systems with variable access time
- Precise exceptions

Complex In-Order Pipeline



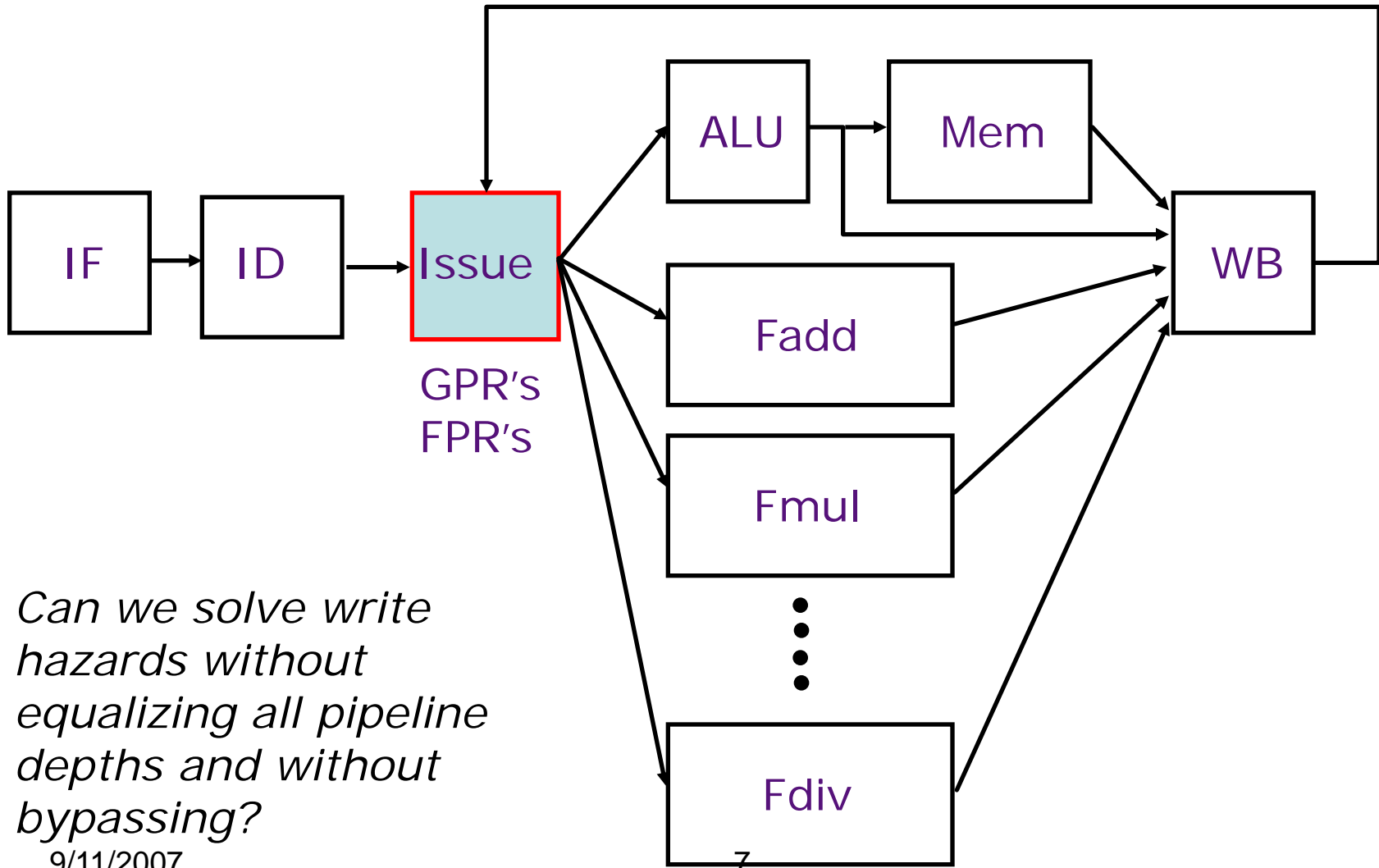
- **Delay writeback so all operations have same latency to W stage**

- Write ports never oversubscribed (one inst. in & one inst. out every cycle)
- Instructions commit in order, simplifies precise exception implementation

How to prevent increased writeback latency from slowing down single cycle integer operations?

Bypassing

Complex Pipeline



Can we solve write hazards without equalizing all pipeline depths and without bypassing?

When is it Safe to Issue an Instruction?

Suppose a data structure keeps track of all the instructions in all the functional units

The following checks need to be made before the Issue stage can dispatch an instruction

- Is the required function unit available?
- Is the input data available? \Rightarrow RAW?
- Is it safe to write the destination? \Rightarrow WAR? WAW?
- Is there a structural conflict at the WB stage?

Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU's availability.
(FU = Int, Add, Mult, Div)

These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which
writes are pending.

These bits are set to true by the Issue stage and set to
false by the WB stage

Issue checks the instruction (opcode dest src1 src2)
against the scoreboard (Busy & WP) to dispatch

FU available?

RAW?

WAR?

WAW?

Busy[FU#]

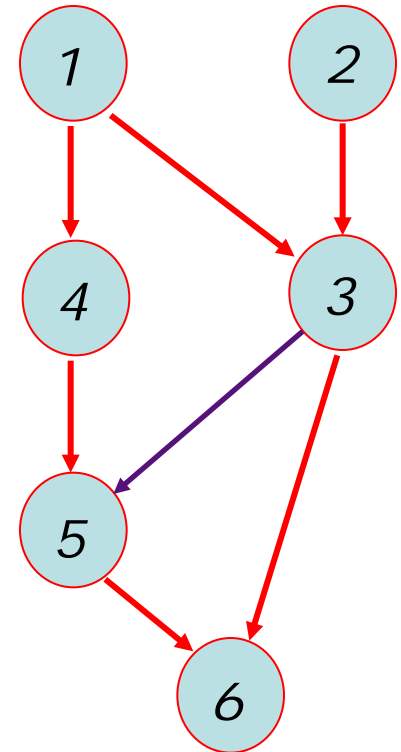
WP[src1] or WP[src2]

cannot arise

WP[dest]

In-Order Issue Limitations: *an example*

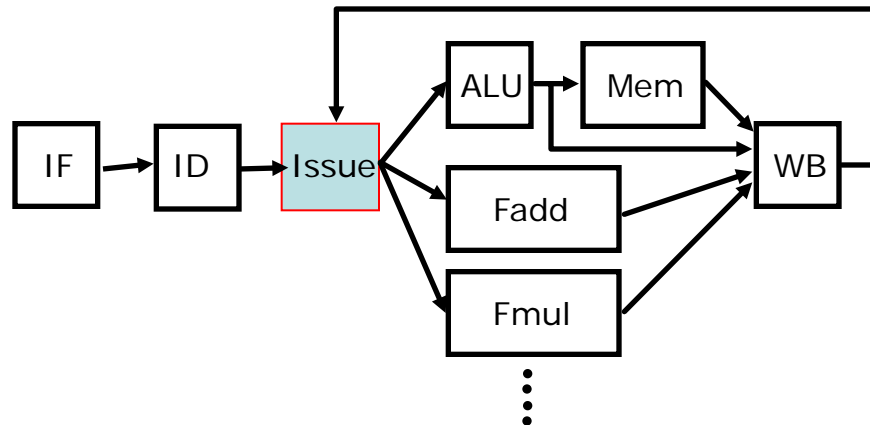
					<i>latency</i>
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		<i>long</i>
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4,	F2,	F8	4
6	ADDD	F10,	F6,	F4	1



In-order: 1 (2,1) 2 3 4 4 3 5 . . . 5 6 6

In-order restriction prevents instruction 4 from being dispatched

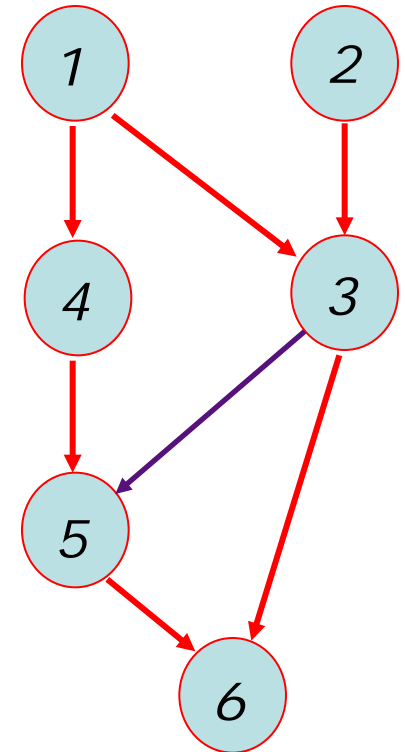
Out-of-Order Issue



- Issue stage buffer holds multiple instructions waiting to issue.
- Decode adds next instruction to buffer if there is space and the instruction does not cause a WAR or WAW hazard.
- Any instruction in buffer whose RAW hazards are satisfied can be issued (*for now at most one dispatch per cycle*). On a write back (WB), new instructions may get enabled.

In-Order Issue Limitations: *an example*

					<i>latency</i>
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		<i>long</i>
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4,	F2,	F8	4
6	ADDD	F10,	F6,	F4	1



In-order: 1 (2,1) 2 3 4 4 3 5 . . . 5 6 6

Out-of-order: 1 (2,1) 4 4 2 3 . . 3 5 . . . 5 6 6

Out-of-order execution did not allow any significant improvement!



How many instructions can be in the pipeline?

Which features of an ISA limit the number of instructions in the pipeline?

Number of Registers

Which features of a program limit the number of instructions in the pipeline?

Control transfers

Out-of-order dispatch by itself does not provide any significant performance improvement !

Overcoming the Lack of Register Names

Floating Point pipelines often cannot be kept filled with small number of registers.

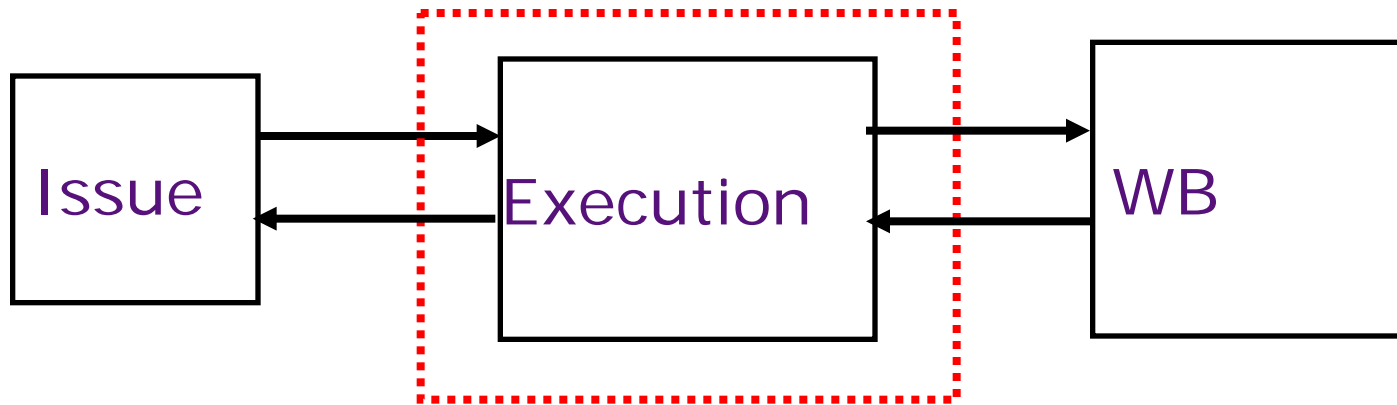
IBM 360 had only 4 Floating Point Registers

Can a microarchitecture use more registers than specified by the ISA without loss of ISA compatibility ?

Robert Tomasulo of IBM suggested an ingenious solution in 1967 based on on-the-fly *register renaming*

Little's Law

$$\text{Throughput (T)} = \text{Number in Flight (N)} / \text{Latency (L)}$$



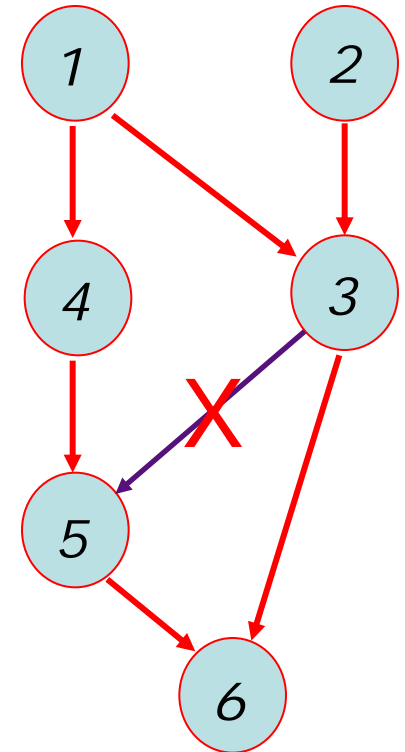
Example:

- 4 floating point registers
 - 8 cycles per floating point operation
- ⇒ maximum of ½ issue per cycle!

排队理论（Theory of Queues）中：在一个稳定的系统中，长时间观察到的平均顾客数量 L ，等于，长时间观察到的有效到达速率 λ 与平均每个顾客在系统中花费的时间之乘积，即 $L = \lambda W$ 。

Instruction-level Parallelism via Renaming

					<i>latency</i>
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		<i>long</i>
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4' ,	F2,	F8	4
6	ADDD	F10,	F6,	F4'	1



In-order: 1 (2,1) 2 3 4 4 3 5 . . . 5 6 6

Out-of-order: 1 (2,1) 4 4 5 . . . 2 (3,5) 3 6 6

Any antidependence can be eliminated by renaming.

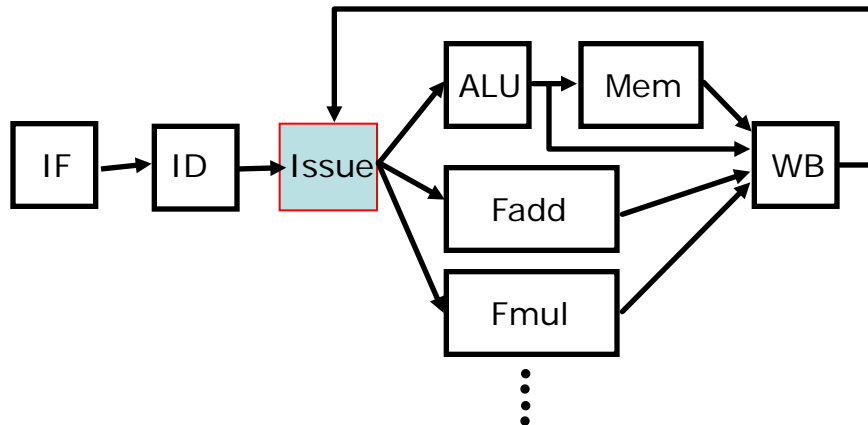
(renaming \Rightarrow additional storage)

9/11/2007

Can it be done in hardware?

yes!

Register Renaming



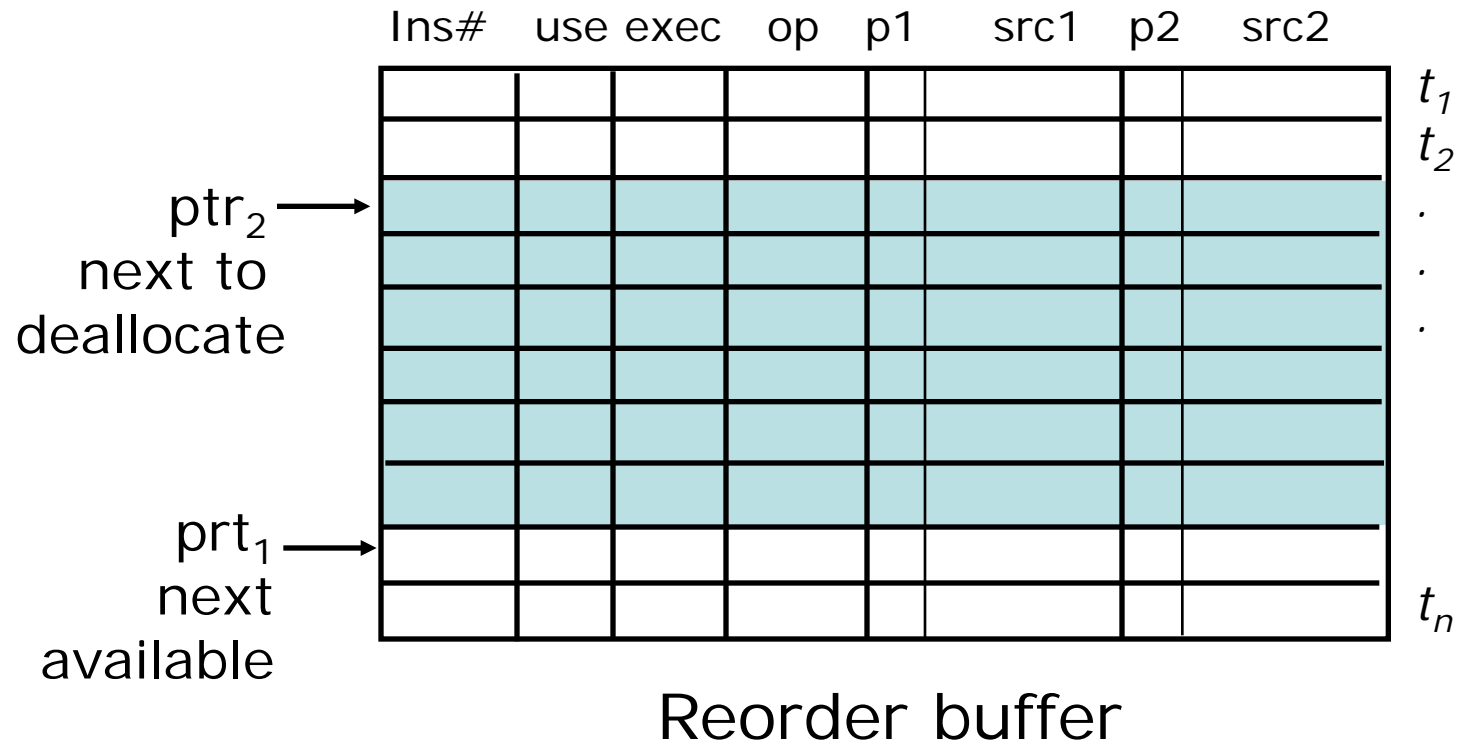
- Decode does register renaming and adds instructions to the issue stage reorder buffer (ROB)

⇒ renaming makes WAR or WAW hazards impossible

- Any instruction in ROB whose RAW hazards have been satisfied can be dispatched.

⇒ Out-of-order or dataflow execution

Dataflow execution



Instruction slot is candidate for execution when:

- It holds a valid instruction ("use" bit is set)
- It has not already started execution ("exec" bit is clear)
- Both operands are available (p1 and p2 are set)

Renaming & Out-of-order Issue

An example

Renaming table

v1

	p	data
F1		
F2		v1
F3		
F4		t5
F5		
F6		t3
F7		
F8		t4

data / t_i

Reorder buffer

Ins#	use	exec	op	p1	src1	p2	src2	
1	0	0	LD					t ₁
2	0	0	LD					t ₂
3	1	0	MUL	0	t2	1	v1	t ₃
4	0	0	SUB	1	v1	1	v1	t ₄
5	1	0	DIV	1	v1	0	t4	t ₅
								.
								.

1	LD	F2,	34(R2)
2	LD	F4,	45(R3)
3	MULTD	F6,	F4, F2
4	SUBD	F8,	F2, F2
5	DIVD	F4,	F2, F8
6	ADDD	F10,	F6, F4

9/11/2007

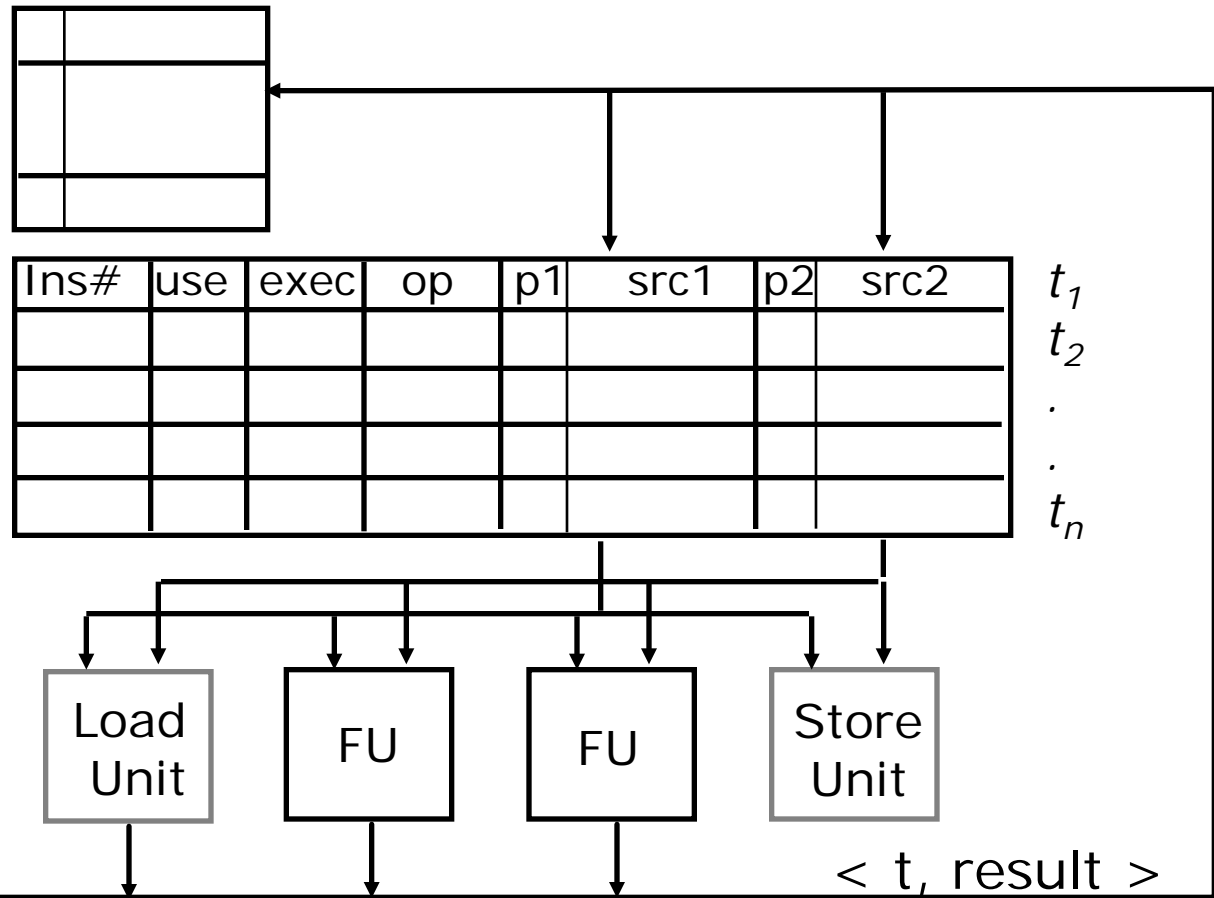
- When are names in sources replaced by data?
Whenever an FU produces data
- When can a name be reused?
Whenever an instruction completes

Data-Driven Execution

*Renaming
table &
reg file*

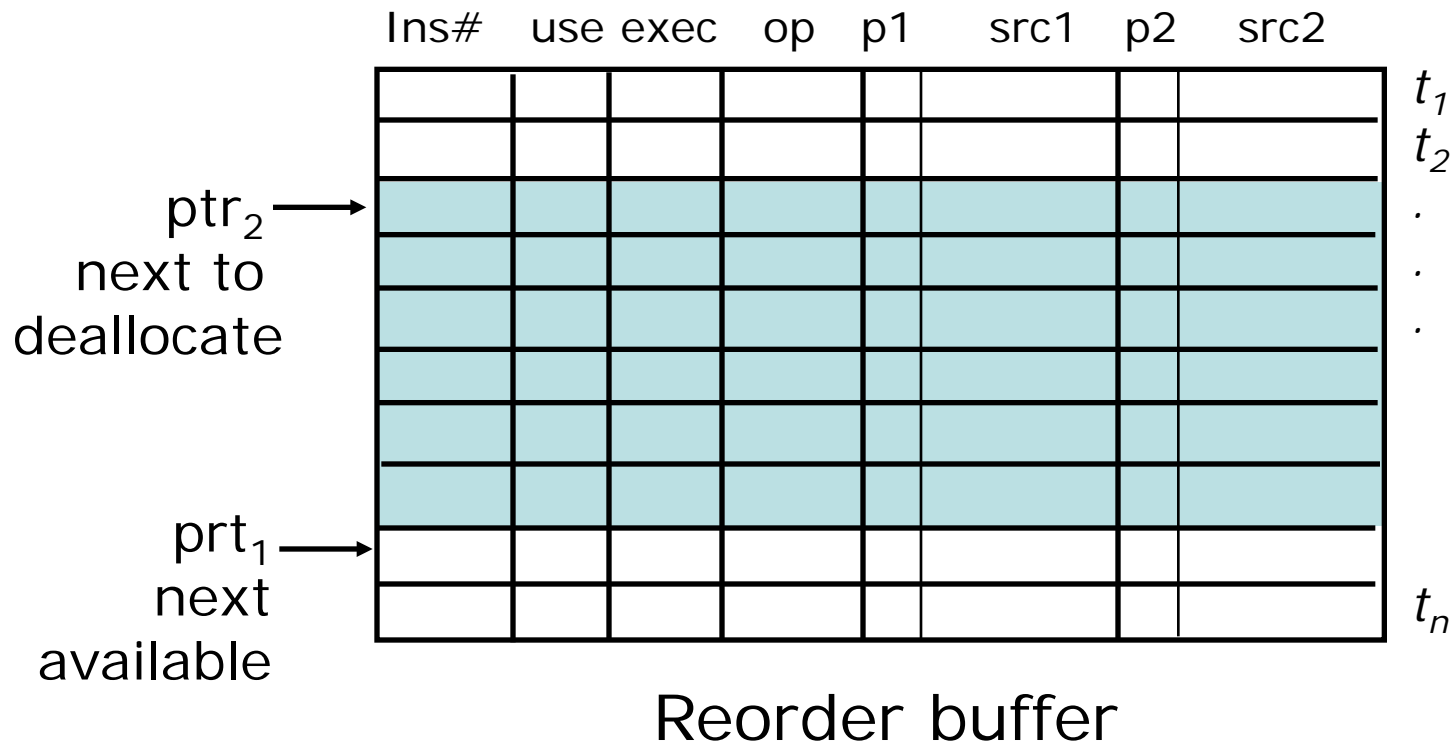
*Reorder
buffer*

Replacing the
tag by its value
is an expensive
operation



- Instruction template (i.e., tag t) is allocated by the Decode stage, which also stores the tag in the reg file
- When an instruction completes, its tag is deallocated

Simplifying Allocation/Deallocation

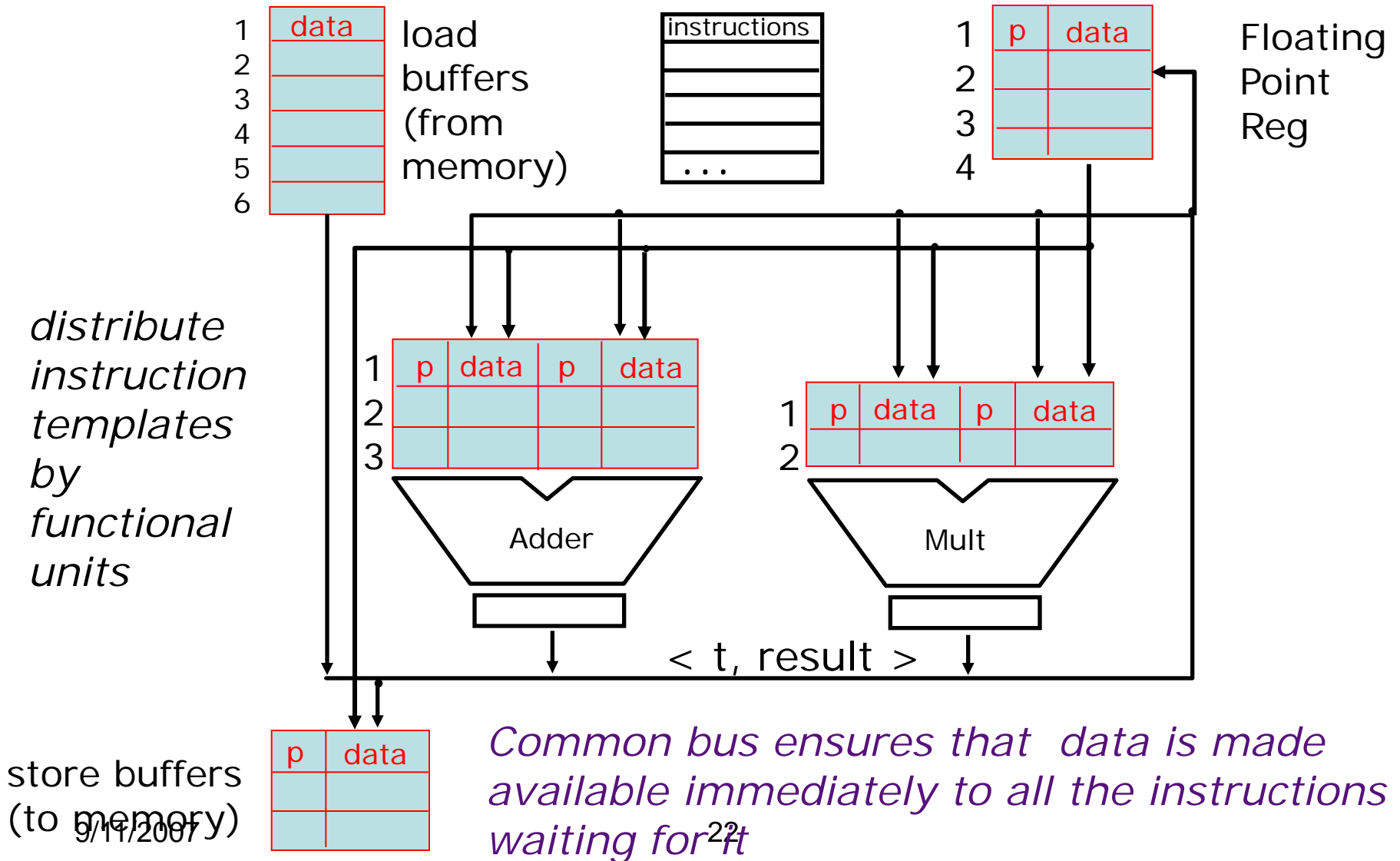


Instruction buffer is managed circularly

- "exec" bit is set when instruction begins execution
- When an instruction completes, its "use" bit is marked free
- ptr₂ is incremented only if the "use" bit is marked free

IBM 360/91 Floating Point Unit

R. M. Tomasulo, 1967



Effectiveness?

Renaming and Out-of-order execution was first implemented in 1969 in IBM 360/91 but did not show up in the subsequent models until mid-Nineties.

Why ?

Reasons

1. Effective on a very small class of programs
2. Memory latency a much bigger problem
3. Exceptions not precise!

One more problem needed to be solved

Control transfers

Precise Interrupts


It must appear as if an interrupt is taken between two instructions (say I_i and I_{i+1})

- the effect of all instructions up to and including I_i is totally complete
- no effect of any instruction after I_i has taken place

The interrupt handler either aborts the program or restarts it at I_{i+1} .

Out-of-order Completion

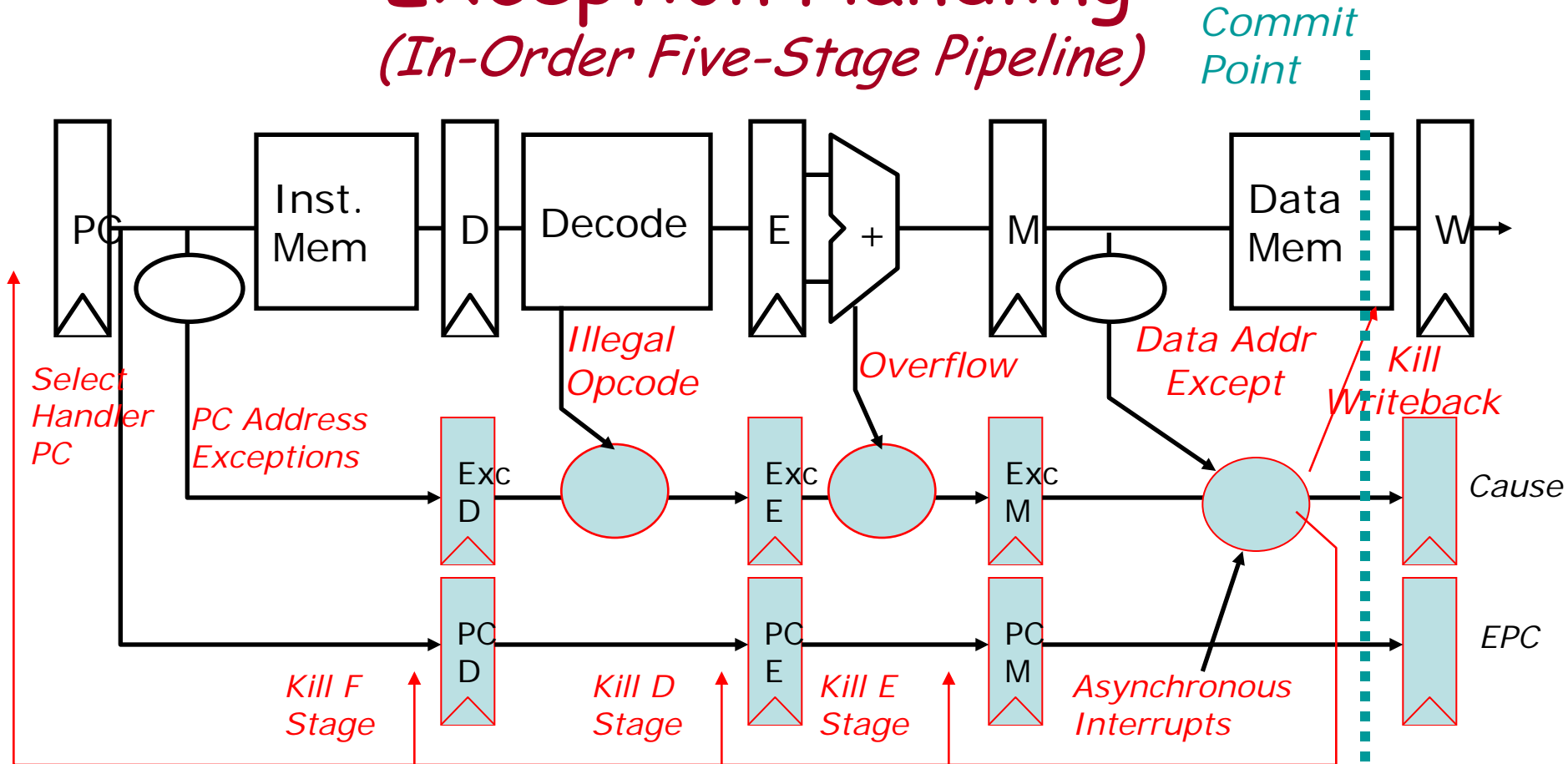
out-of-order comp 1 2 2 3 1 4 3 5 5 4 6 6

Consider interrupts  *restore f2* *restore f10*

25

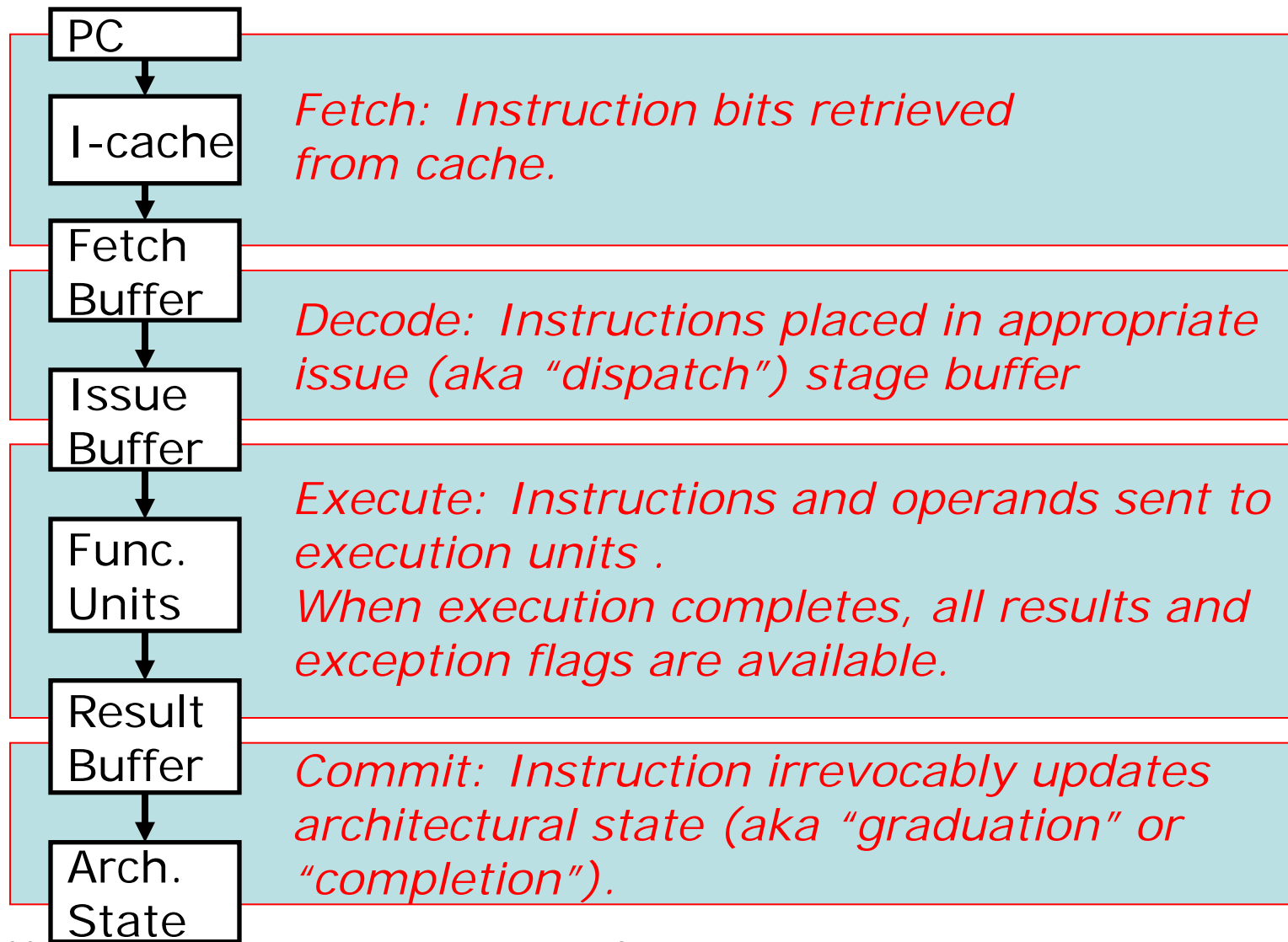
Exception Handling

(In-Order Five-Stage Pipeline)

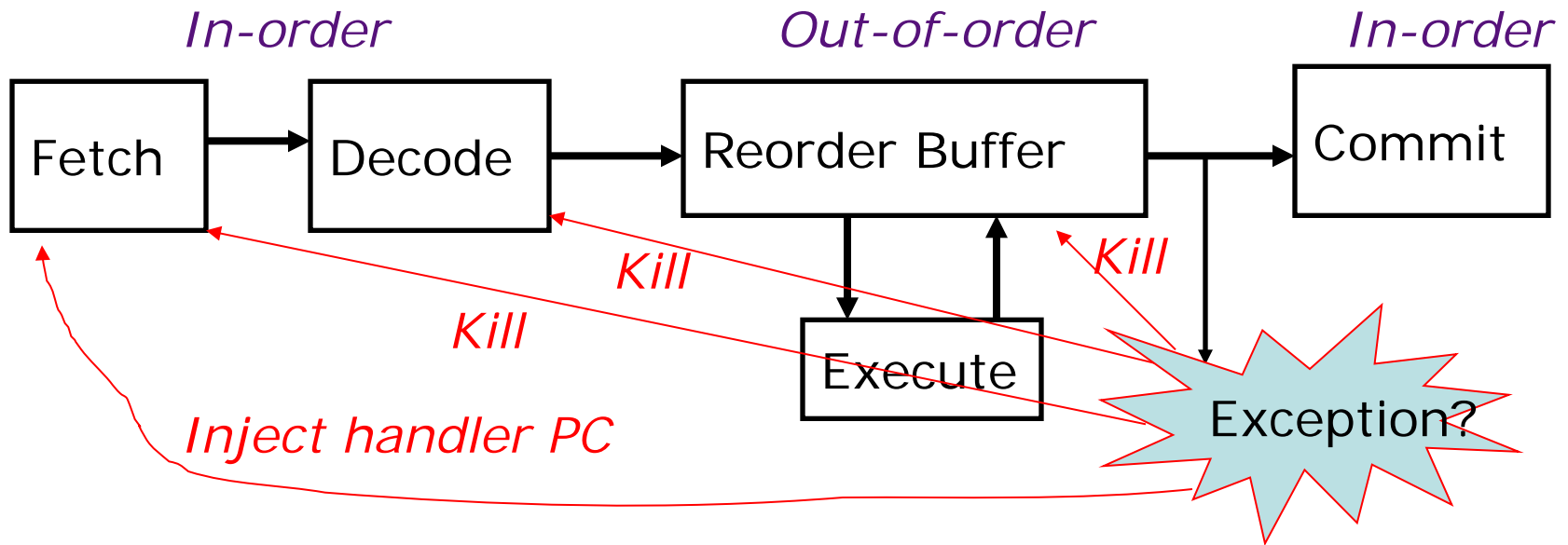


- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

Phases of Instruction Execution



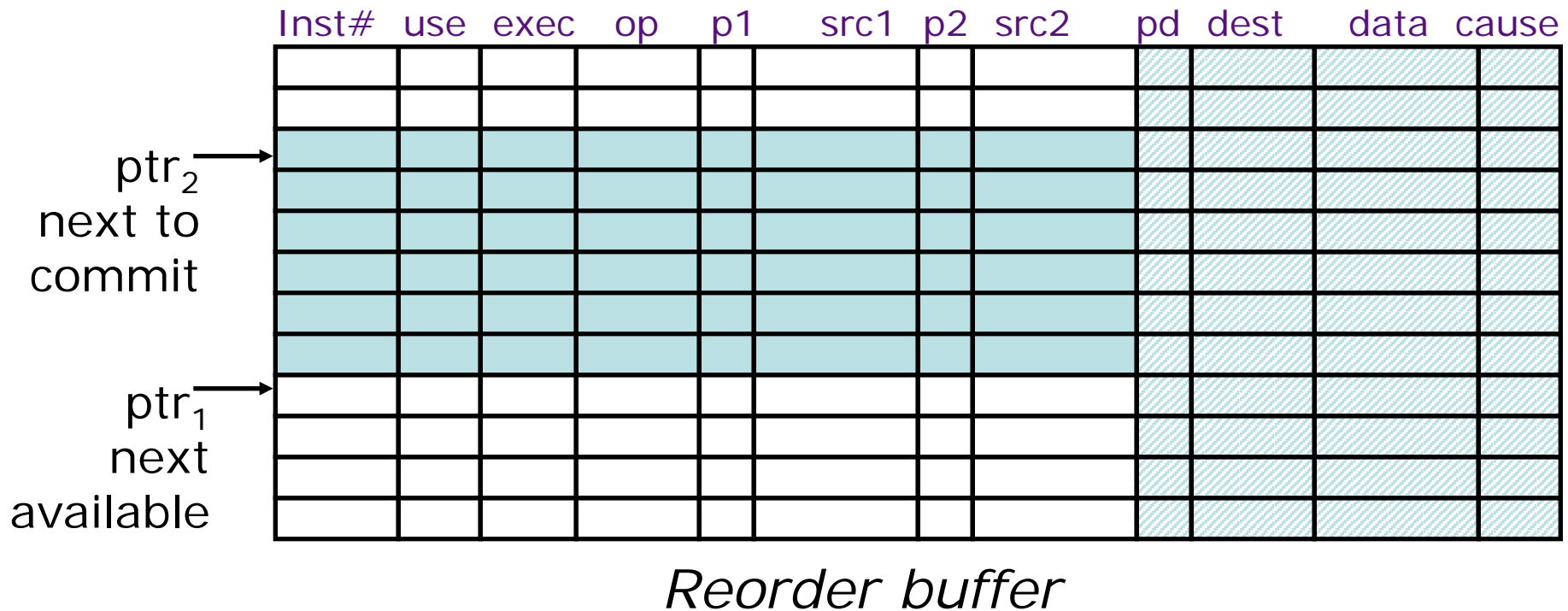
In-Order Commit for Precise Exceptions



- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order (\Rightarrow out-of-order completion)
- *Commit* (write-back to architectural state, i.e., regfile & memory) is in-order

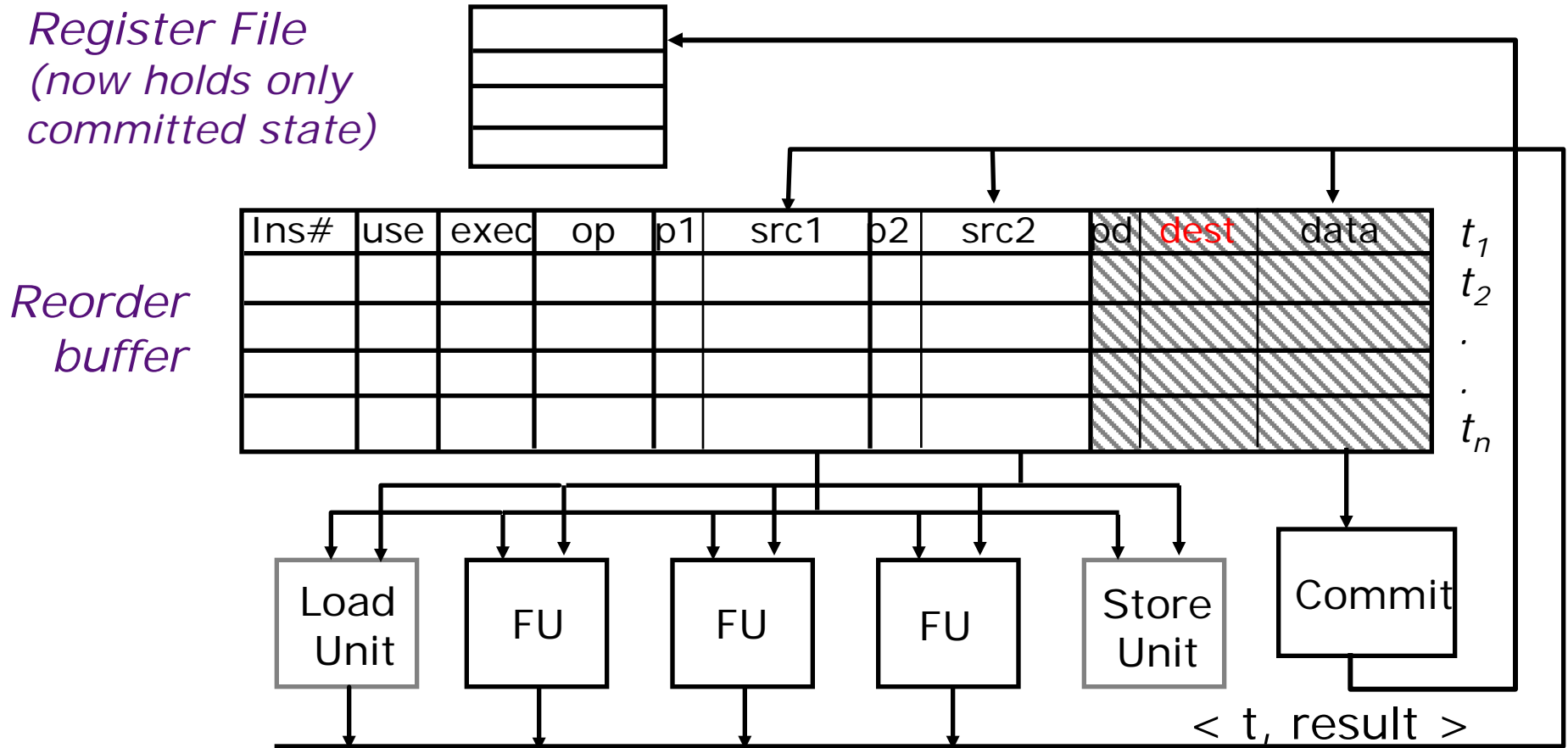
Temporary storage needed to hold results before commit (shadow registers and store buffers)

Extensions for Precise Exceptions



- add <pd, dest, data, cause> fields in the instruction template
- commit instructions to reg file and memory in program order \Rightarrow buffers can be maintained circularly
- on exception, clear reorder buffer by resetting $\text{ptr}_1 = \text{ptr}_2$
(stores must wait for commit before updating memory)

Rollback and Renaming

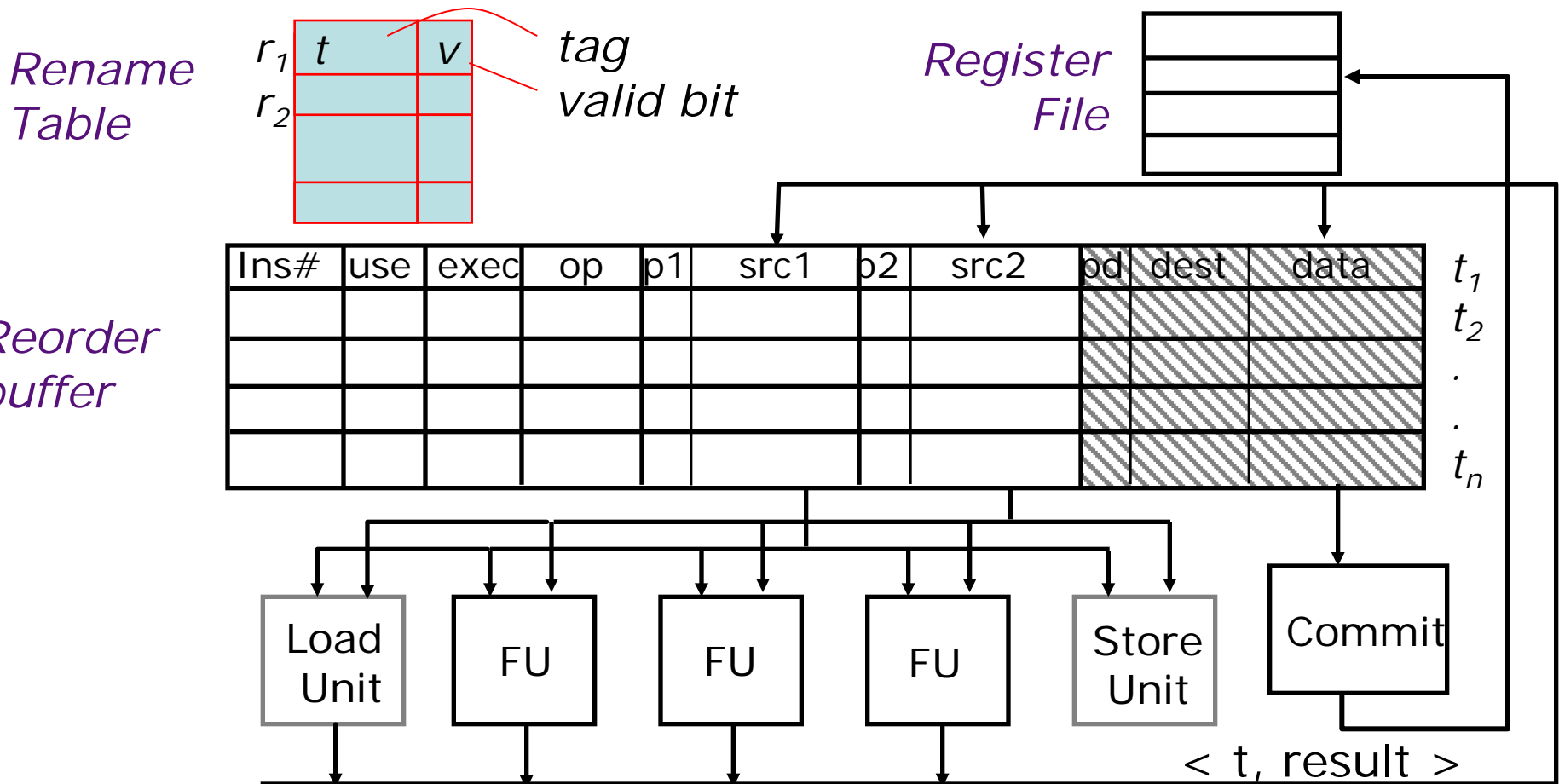


Register file does not contain renaming tags any more.

How does the decode stage find the tag of a source register?

Search the "dest" field in the reorder buffer

Renaming Table



Renaming table is a cache to speed up register name look up.
It needs to be cleared after each exception taken.

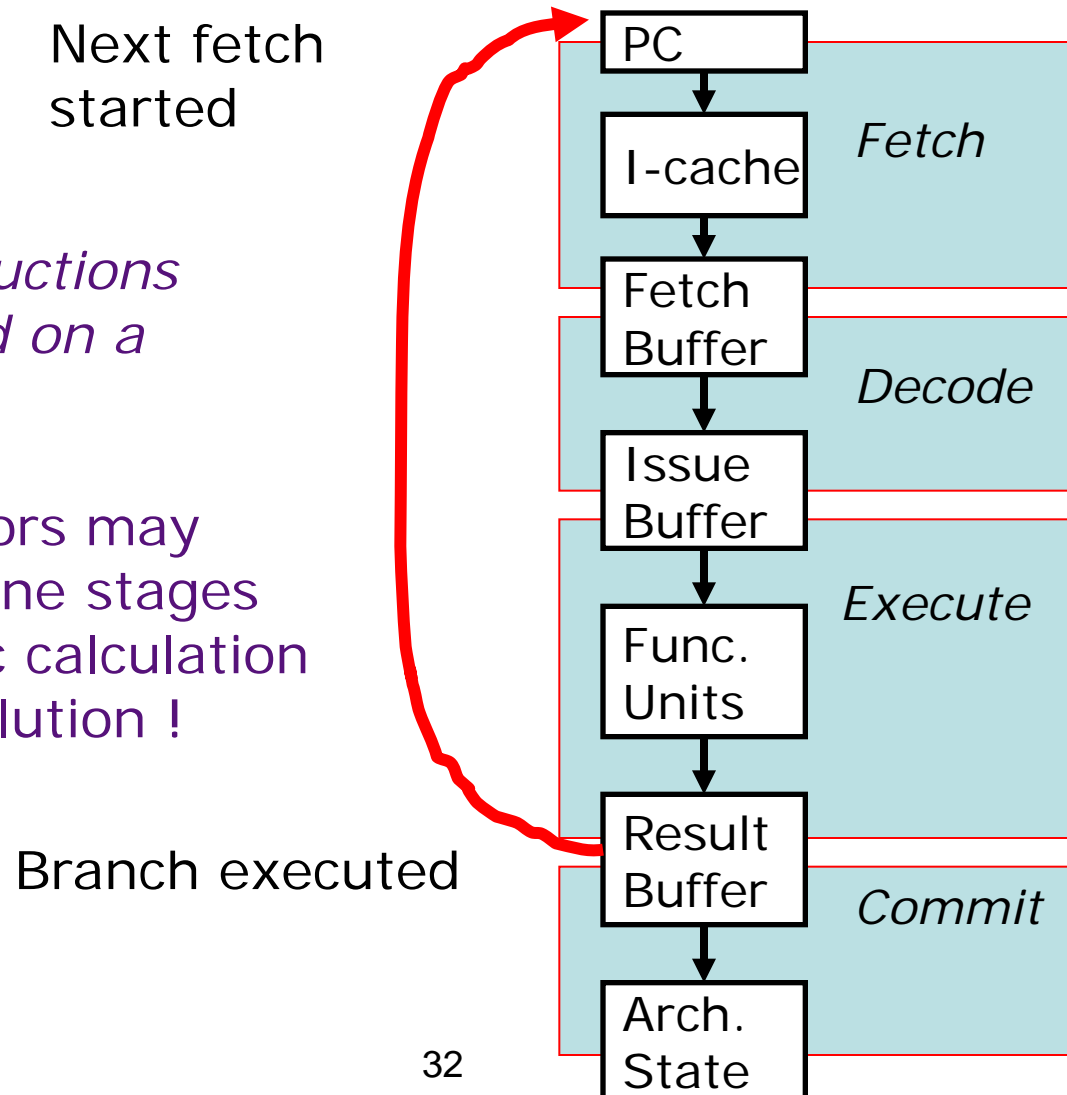
When else are valid bits cleared?

Control transfers

Branch Penalty

How many instructions need to be killed on a misprediction?

Modern processors may have > 10 pipeline stages between next pc calculation and branch resolution !



Average Run-Length between Branches

Average dynamic instruction mix from SPEC92:

	SPECint92	SPECfp92
ALU	39 %	13 %
FPU Add		20 %
FPU Mult		13 %
load	26 %	23 %
store	9 %	9 %
branch	16 %	8 %
other	10 %	12 %

SPECint92: *compress, eqntott, espresso, gcc, li*
SPECfp92: *doduc, ear, hydro2d, mdjdp2, su2cor*

What is the average *run length* between branches?

next lecture: Branch prediction & Speculative execution

Paper Discussion: B5000 vs IBM 360

- **IBM set foundations for ISAs since 1960s**
 - 8-bit byte
 - Byte-addressable memory (as opposed to word-addressable memory)
 - 32-bit words
 - Two's complement arithmetic (but not the first processor)
 - 32-bit (SP) / 64-bit (DP) Floating Point format and registers
 - Commercial use of microcoded CPUs
 - Binary compatibility / computer family
- **B5000 very different model: HLL only, stack, Segmented VM**
- **IBM paper made case for ISAs good for microcoded processors \Rightarrow leading to CISC**