

计算机组成与系统结构

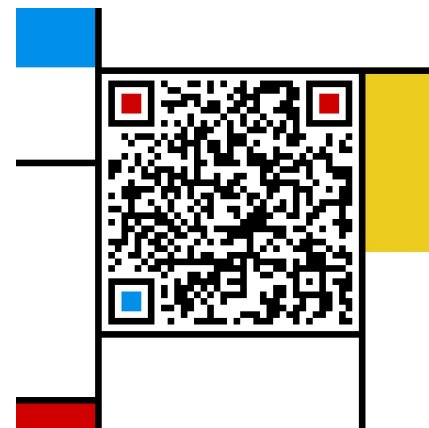
Computer Organization & System Architecture

Huang Kejie (黄科杰) 百人计划研究员

Office: 玉泉校区老生仪楼 304

Email address: huangkejie@zju.edu.cn

HP: 17706443800




Types of Data Hazards

Consider executing a sequence of


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

type of instructions


Data-dependence

$$\begin{array}{ll} r_3 \leftarrow r_1 \text{ op } r_2 & \text{Read-after-Write} \\ r_5 \leftarrow r_3 \text{ op } r_4 & \text{(RAW) hazard} \end{array}$$


Anti-dependence

$$\begin{array}{ll} r_3 \leftarrow r_1 \text{ op } r_2 & \text{Write-after-Read} \\ r_1 \leftarrow r_4 \text{ op } r_5 & \text{(WAR) hazard} \end{array}$$


Output-dependence

$$\begin{array}{ll} r_3 \leftarrow r_1 \text{ op } r_2 & \text{Write-after-Write} \\ r_3 \leftarrow r_6 \text{ op } r_7 & \text{(WAW) hazard} \end{array}$$


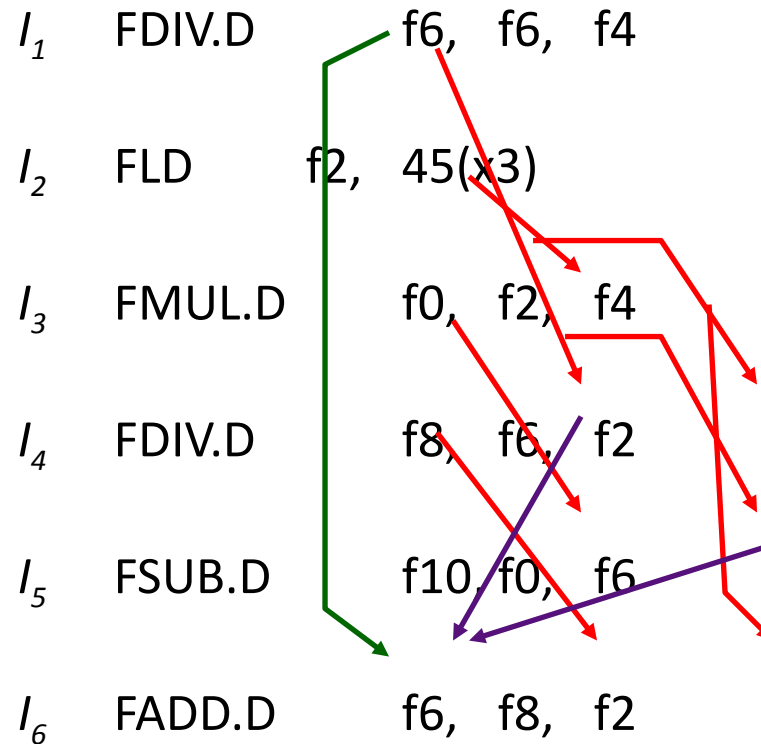
Review: Three Strategies for Data Hazards

- ***Interlock***
 - Wait for hazard to clear by holding dependent instruction in issue stage
- ***Bypass***
 - Resolve hazard earlier by bypassing value as soon as available
- ***Speculate***
 - Guess on value, correct if wrong

Register vs. Memory Dependence

- Data hazards due to register operands can be determined at the decode stage, but data hazards due to memory operands can be determined only after computing the effective address
- Store: $M[r1 + disp1] \leftarrow r2$
- Load: $r3 \leftarrow M[r4 + disp2]$
- Does $(r1 + disp1) = (r4 + disp2)$?

Data Hazards: An Example



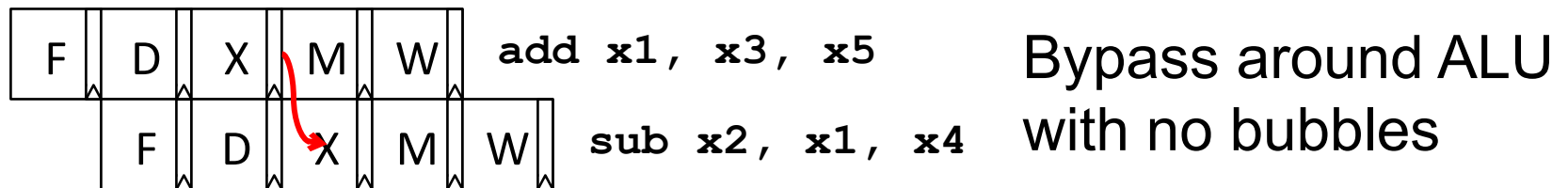
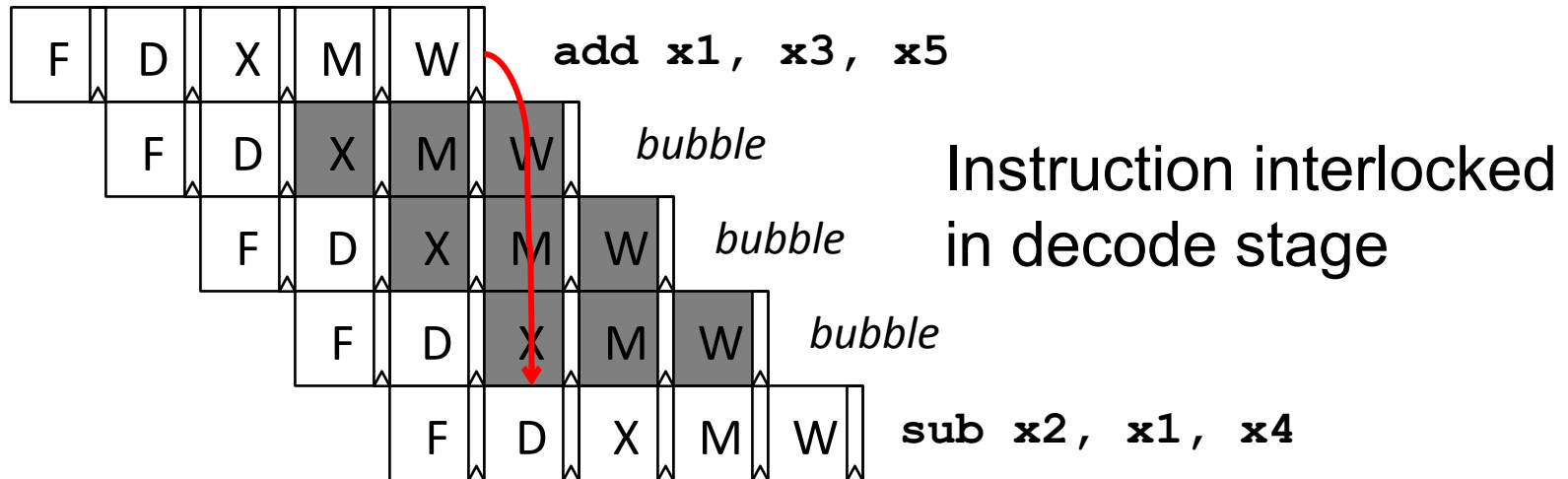
RAW Hazards

WAR Hazards

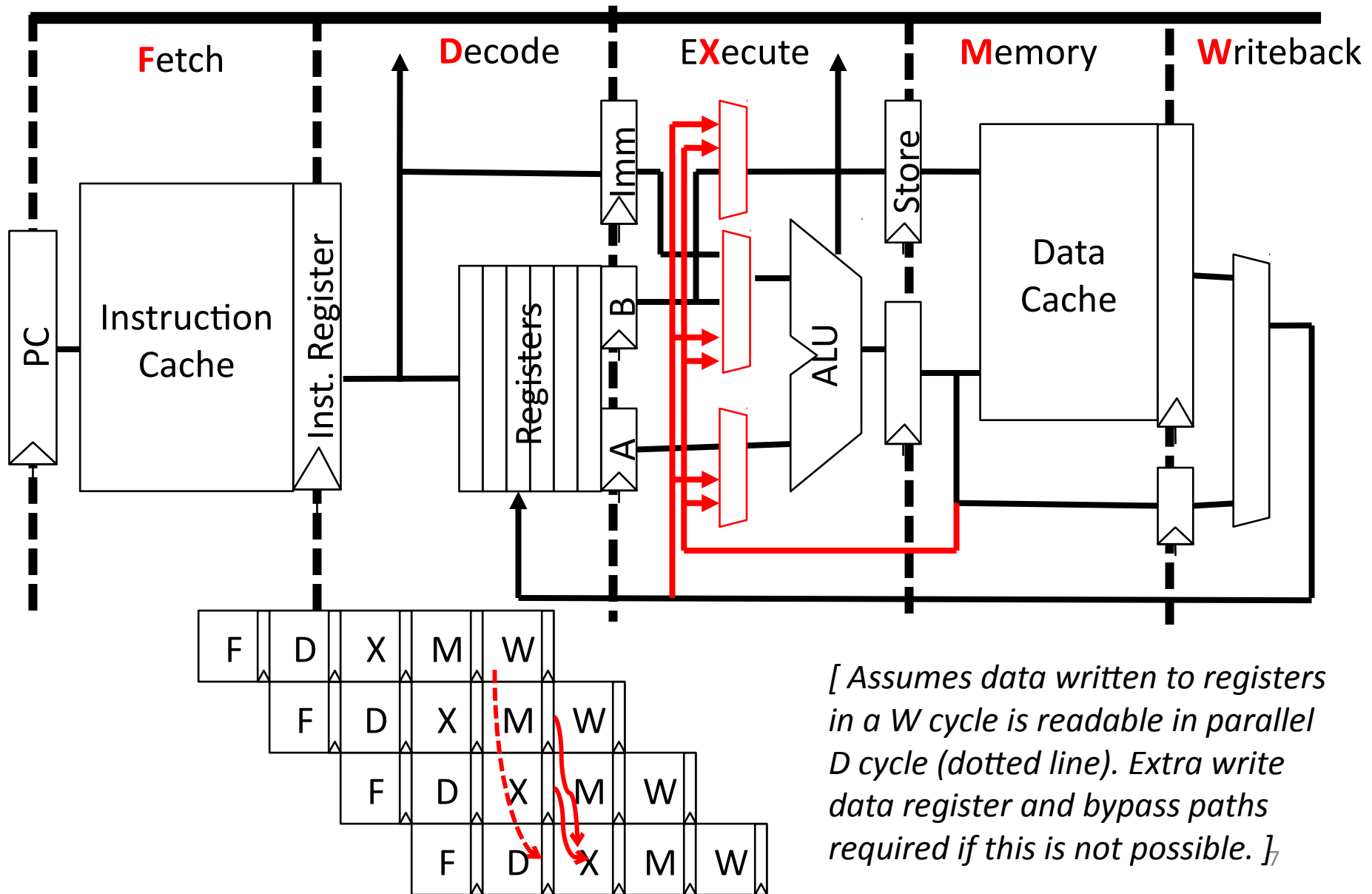
WAW Hazards

Interlocking Versus Bypassing

`add x1, x3, x5`
`sub x2, x1, x4`



Fully Bypassed Data Path



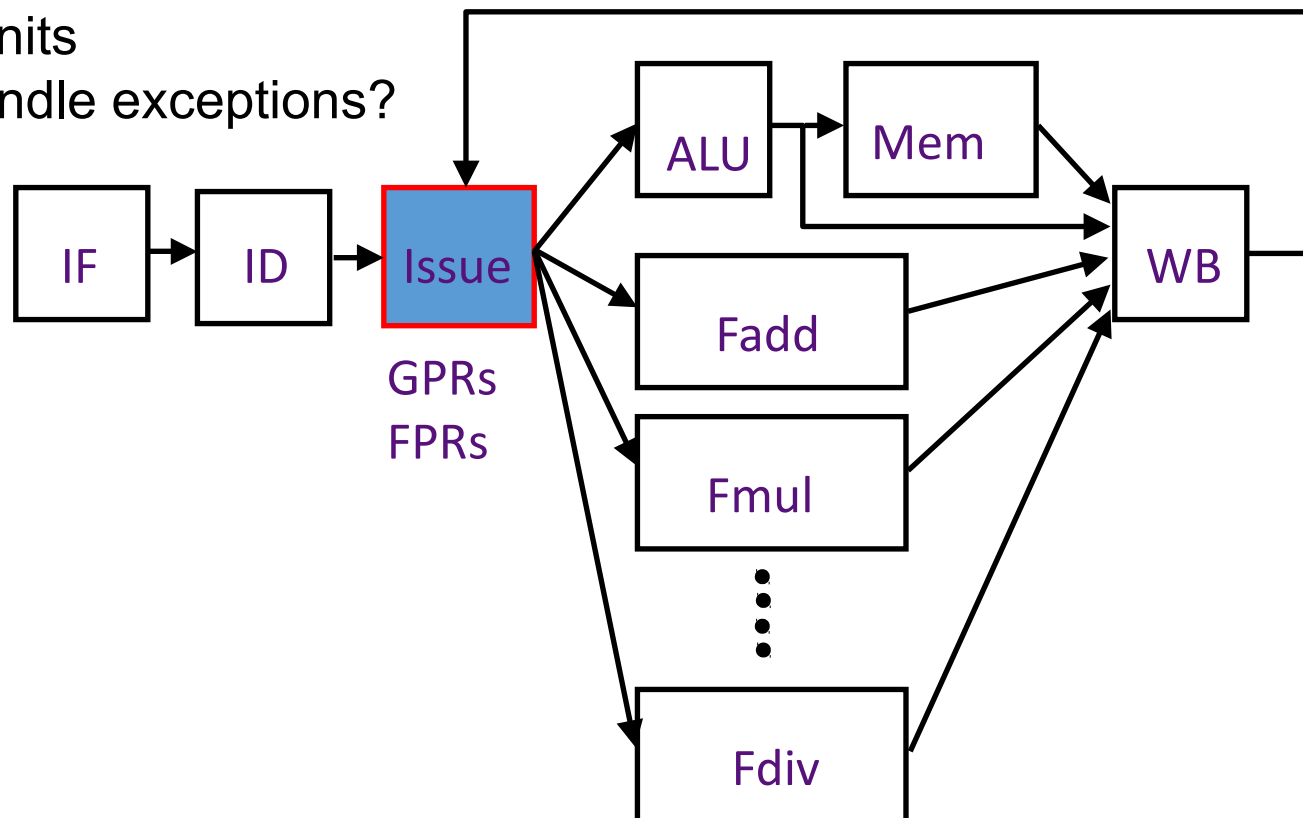
Complex Pipelining: Motivation

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

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

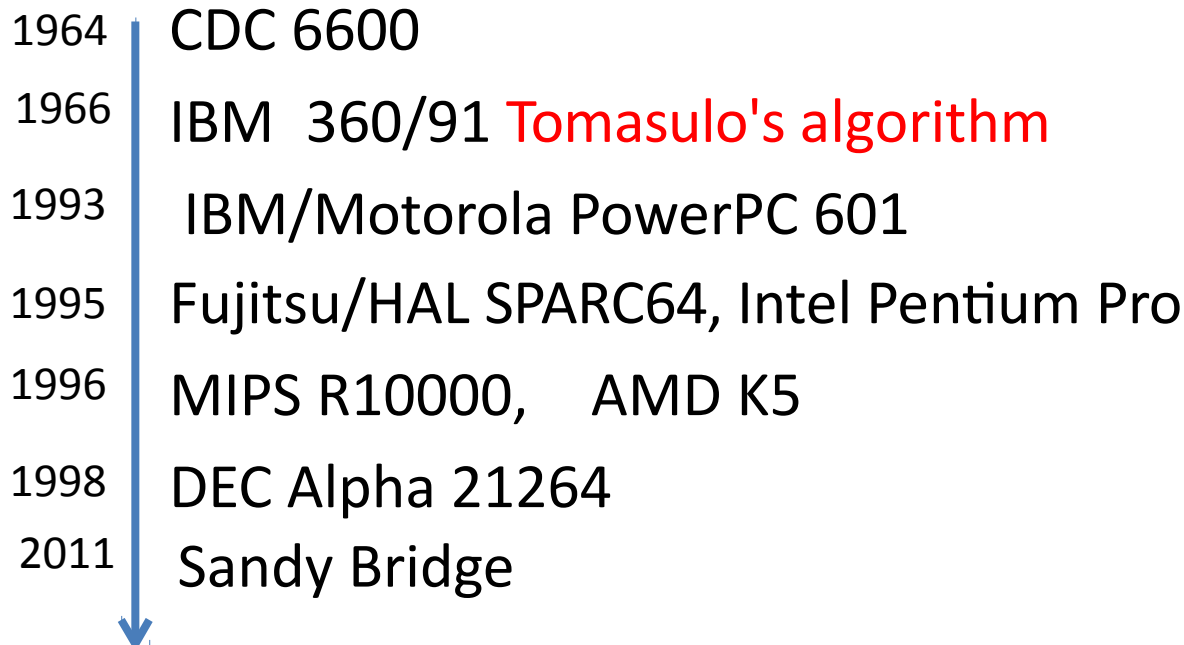
Issues in Complex Pipeline Control

- Structural conflicts at the execution stage if some FPU or memory unit is not pipelined and takes more than one cycle
- Structural conflicts at the write-back stage due to variable latencies of different functional units
- Out-of-order write hazards due to variable latencies of different functional units
- How to handle exceptions?



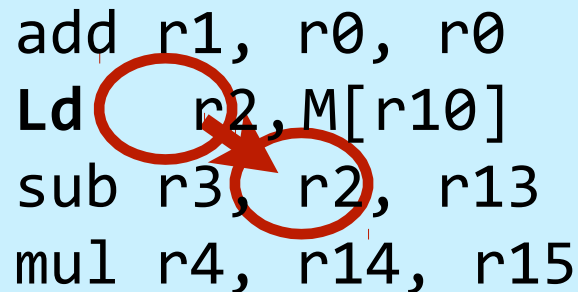
What is OOO?

- OOO execution is a type of processing where the instructions can begin execution as soon as operands are ready
- Instructions are issued in order however execution proceeds out of order
- Evolution



Why Out-of-Order?

- Great for ...
 - tolerating variable latencies
 - finding ILP in code (instruction-level parallelism)
 - complex method for fine-grain data **prefetching**
 - plays nicely with poor compilers and lazily written code
- Downsides
 - complicated
 - expensive (area & power)
- Performance! (and easy to program!)



```
add r1, r0, r0
Ld r2, M[r10]
sub r3, r2, r13
mul r4, r14, r15
```

OoO is widely used in industry

- Intel Xeon/i-series (10-100W)
- ARM Cortex mobile chips (1W)
- Sun/Oracle Niagara UltraSPARC
- Intel Atom
- Play Station

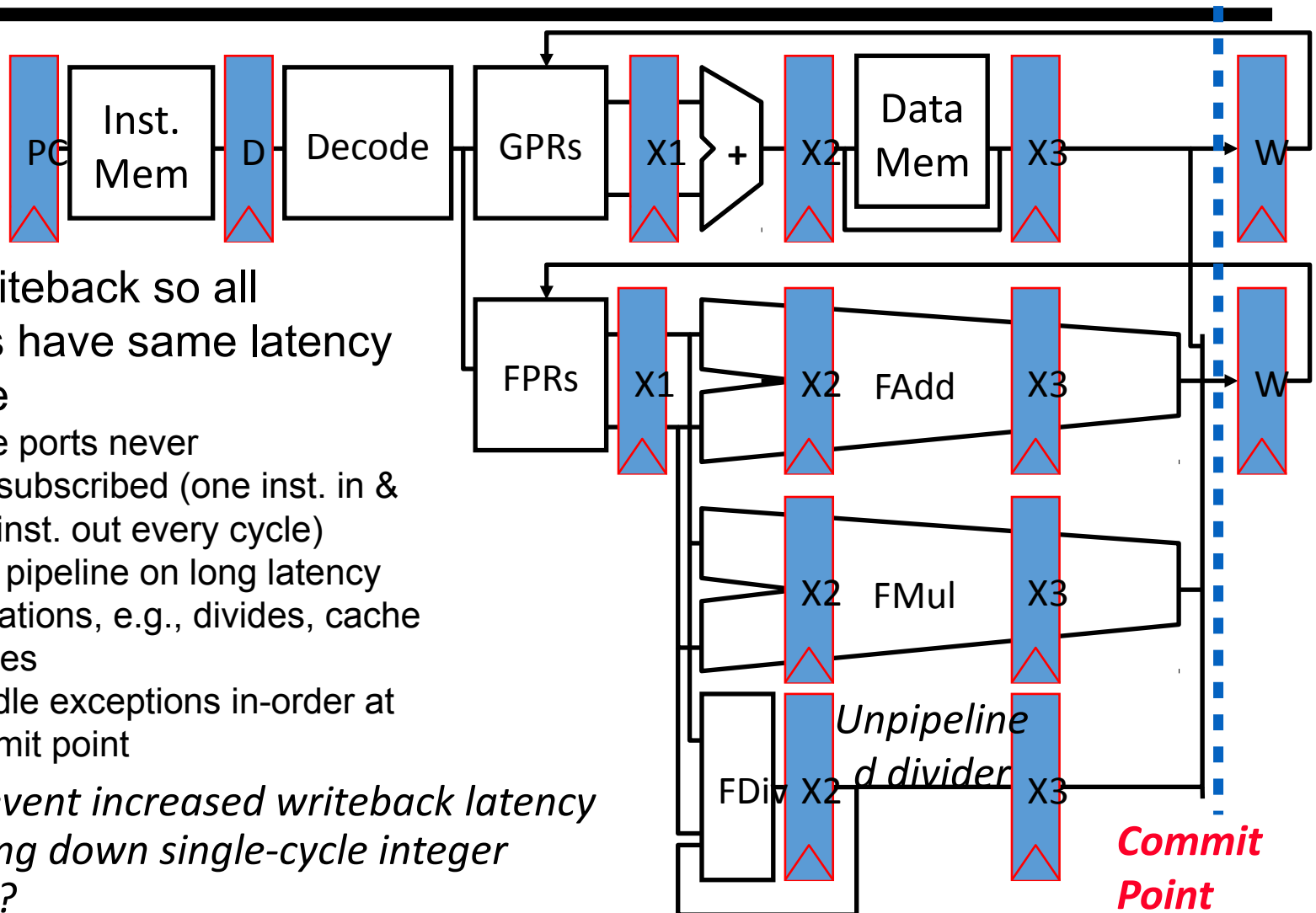


Recap: Complex In-Order Pipeline

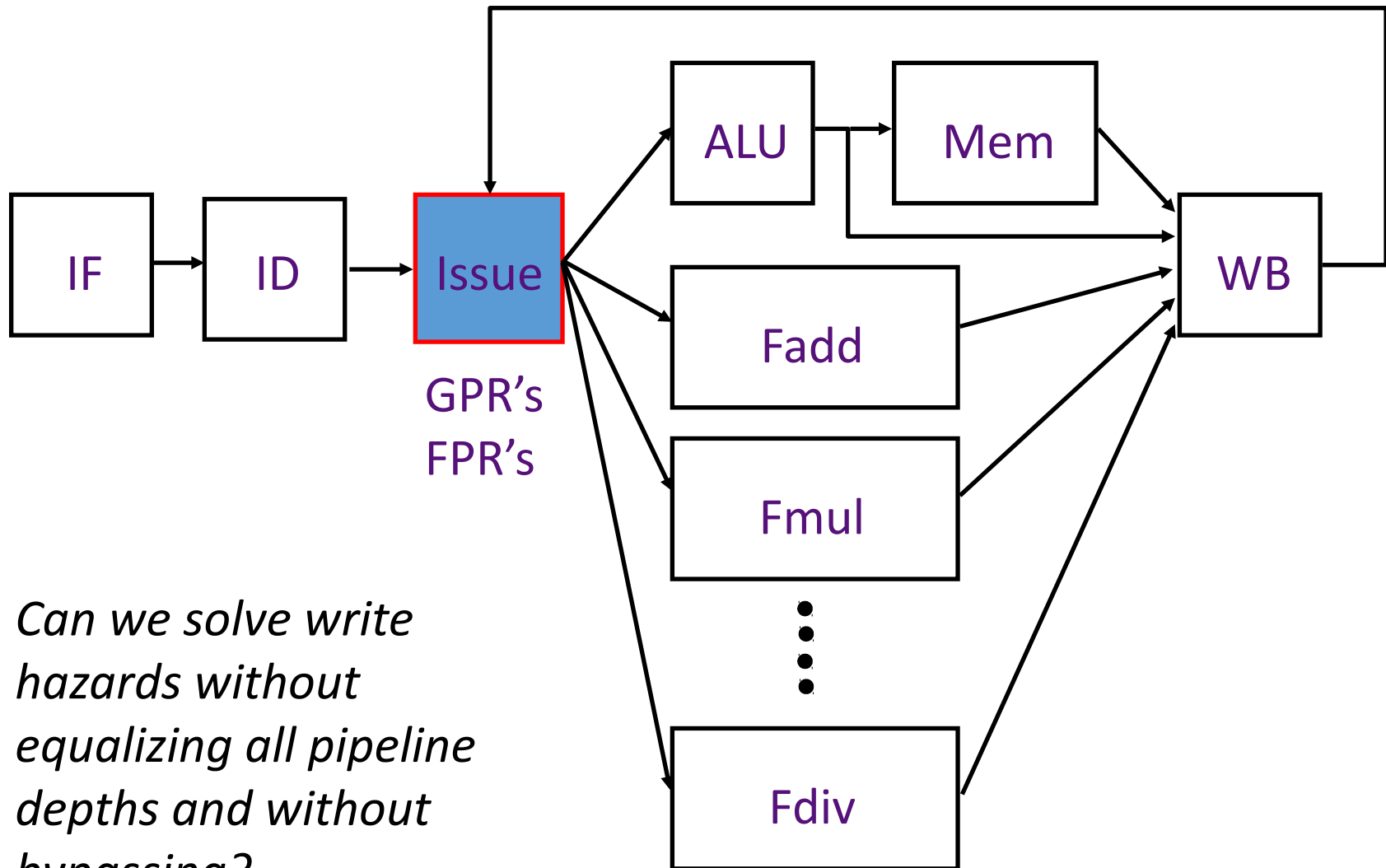
- Delay writeback so all operations have same latency to W stage
 - Write ports never oversubscribed (one inst. in & one inst. out every cycle)
 - Stall pipeline on long latency operations, e.g., divides, cache misses
 - Handle exceptions in-order at commit point

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

Bypassing

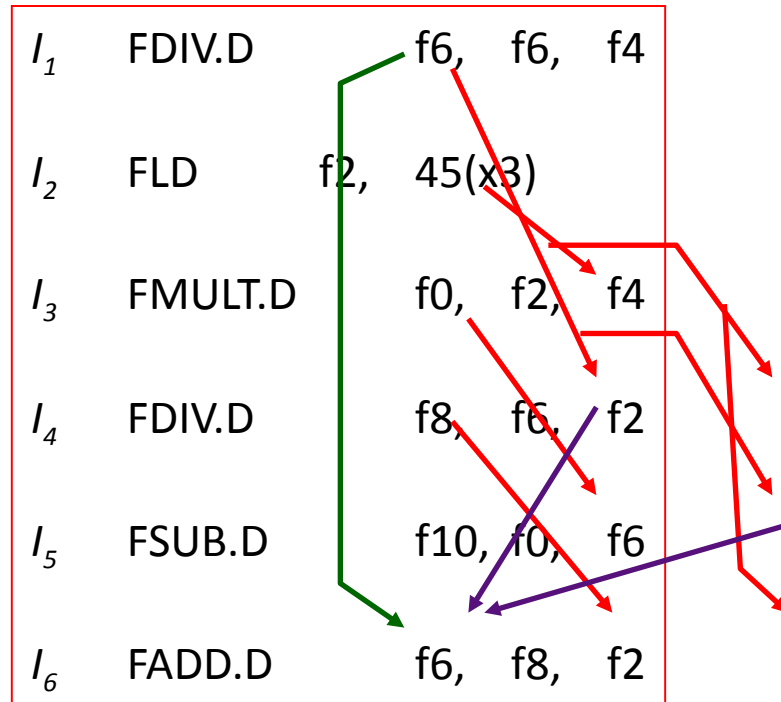


Complex Pipeline



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

Instruction Scheduling



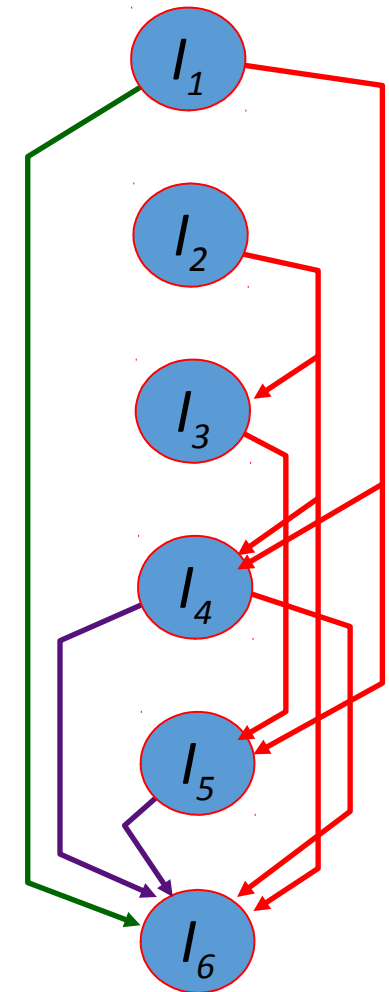
Valid orderings:

in-order I_1 I_2 I_3 I_4 I_5 I_6

out-of-order I_2 I_1 I_3 I_4 I_5 I_6

out-of-order I_1 I_2 I_3 I_5 I_4 I_6

out-of-order



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? (RAW?)
 - Is it safe to write the destination? (WAR?WAW?)
 - Is there a structural conflict at the WB stage?

A Data Structure for Correct Issues

Keeps track of the status of Functional Units

Name	Busy	Op	Dest	Src1	Src2
Int					
Mem					
Add1					
Add2					
Add3					
Mult1					
Mult2					
Div					

The instruction i at the Issue stage consults this table

FU available? check the busy column

RAW? search the dest column for i 's sources

WAR? search the source columns for i 's destination

WAW? search the dest column for i 's destination

*An entry is added to the table if no hazard is detected;
An entry is removed from the table after Write-Back*

Simplifying the Data Structure

Assuming In-order Issue

- Suppose the instruction is not dispatched by the Issue stage if a RAW hazard exists or the required FU is busy, and that operands are latched by functional unit on issue:

- Can the dispatched instruction cause a
 - WAR hazard ?

NO: Operands read at issue

- WAW hazard ?

YES: Out-of-order completion

Simplifying the Data Structure ...

- No WAR hazard
 - →no need to keep src1 and src2
- The Issue stage does not dispatch an instruction in case of a WAW hazard
 - → a register name can occur at most once in the dest column
- WP[reg#] : a bit-vector to record the registers for which writes are pending
 - These bits are set by the Issue stage and cleared by the WB stage
 - → Each pipeline stage in the FU's must carry the register destination field and a flag to indicate if it is valid

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 by Issue stage and cleared by WB stage
- Issue checks the instruction (opcode dest src1 src2)
 - against the scoreboard (Busy & WP) to dispatch
 - **FU available?** **Busy[FU#]**
 - **RAW?** **WP[src1] or WP[src2]**
 - **WAR?** *cannot arise*
 - **WAW?** **WP[dest]**

Scoreboard Dynamics

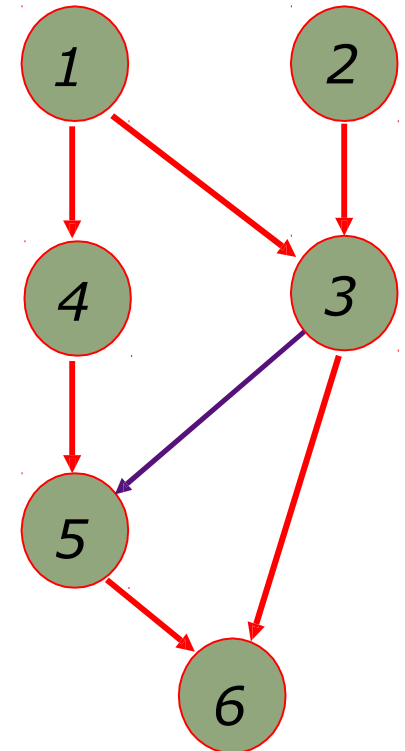
	Functional Unit Status								Registers Reserved for Writes	
	Int(1)	Add(1)	Mult(3)			Div(4)		WB		
t0	I_1		f6				f6			
t1	I_2	f2		f6			f6	f2		
t2			f6		f2		f6	f2		I_2
t3	I_3	f0				f6		f6, f0		
t4		f0				f6		f6, f0		I_1
t5	I_4		f0, f8					f0, f8		
t6			f8			f0		f0, f8		I_3
t7	I_5	f10			f8			f8, f10		
t8			f8, f10			f8, f10			I_5	
t9			f8			f8		I_4		
t10	I_6	f6					f6			
t11	I_1	FDIV.D	f6, f6	f6, f4		f6			I_6	
	I_2	FLD	f2, 45(x3)							
	I_3	FMULT.D	f0, f2, f4							
	I_4	FDIV.D	f8, f6, f2							
	I_5	FSUB.D	f10, f0, f6							
	I_6	FADD.D	f6, f0, f2							

In-order Issue

<i>in-order comp</i>	1	2			<u>1</u>	<u>2</u>	3	4		<u>3</u>	5	<u>4</u>	6	<u>5</u>	<u>6</u>
<i>out-of-order comp</i>	1	2	<u>2</u>	3	<u>1</u>	4	<u>3</u>	5	<u>5</u>	<u>4</u>	6	<u>6</u>			

In-Order Issue Limitations: an example

1	LD	F2,	34(R2)	<i>latency</i> 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

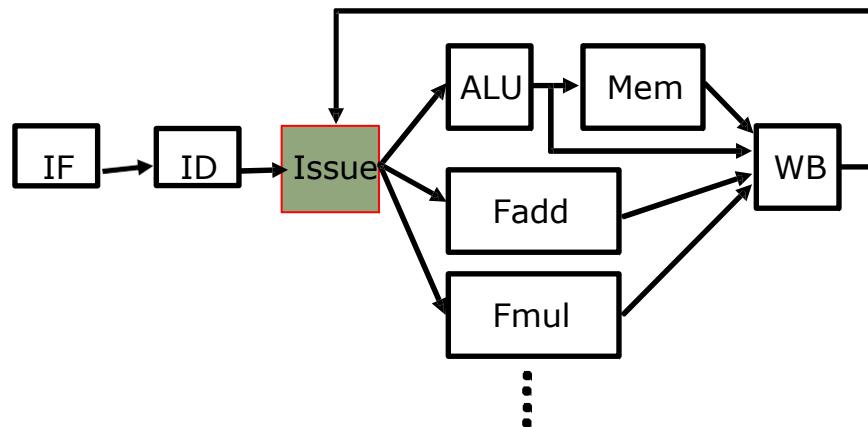


In-order: 1 (2,1) 2 3 4 4 3 5 . . . 5 6 6

In-order issue restriction prevents instruction 4 from being dispatched

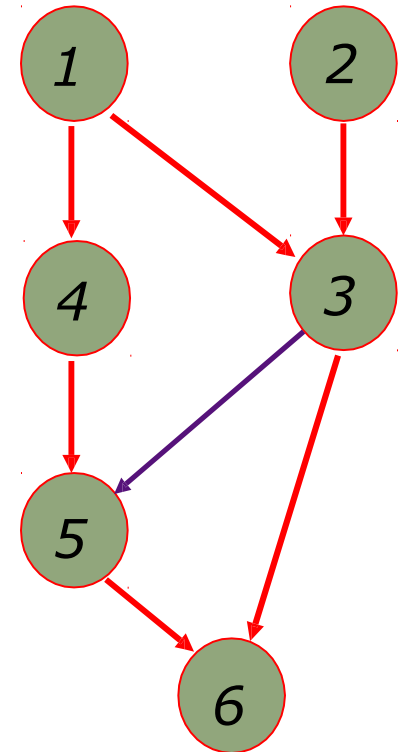
Out-of-Order Issue

- 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.
 - Note: WAR possible again because issue is out-of-order (WAR not possible with in-order issue and latching of input operands at functional unit)
- 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

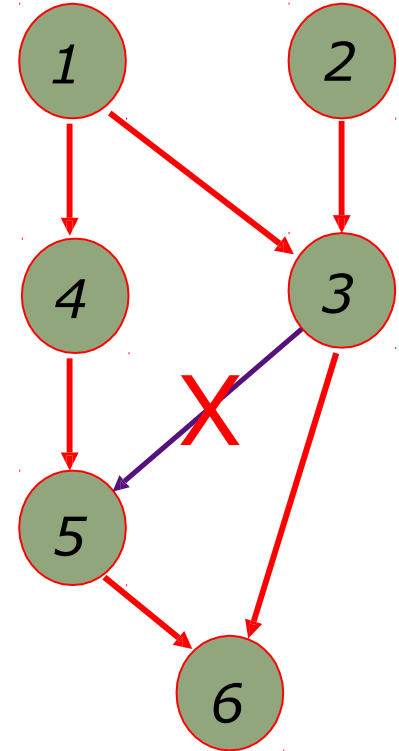
Out-of-order dispatch by itself does not provide any significant performance improvement !

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

Issue Limitations: In-Order and Out-of-Order

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



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

Out-of-order:

	1	(2, <u>1</u>)	4	<u>4</u>	5	.	.	<u>2</u>	(3, <u>5</u>)	<u>3</u>	6	.	.	<u>5</u>	6	<u>6</u>
						.	6									

Any antidependence can be eliminated by renaming.

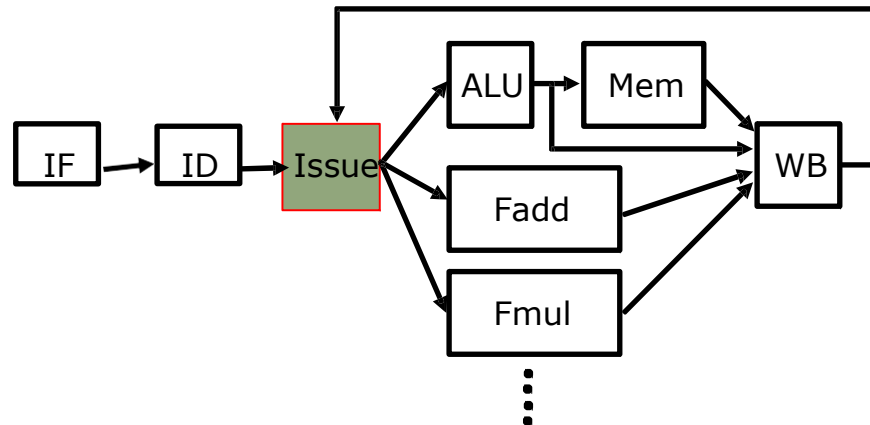
(renaming \Rightarrow additional storage)

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



Renaming & Out-of-order Issue An example

Renaming table

	p
F1	
F2	
F3	
F4	
F5	
F6	

Reorder buffer

data	Ins#	use	exec	op	p1
src1	p2			src2	

data / t_i

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

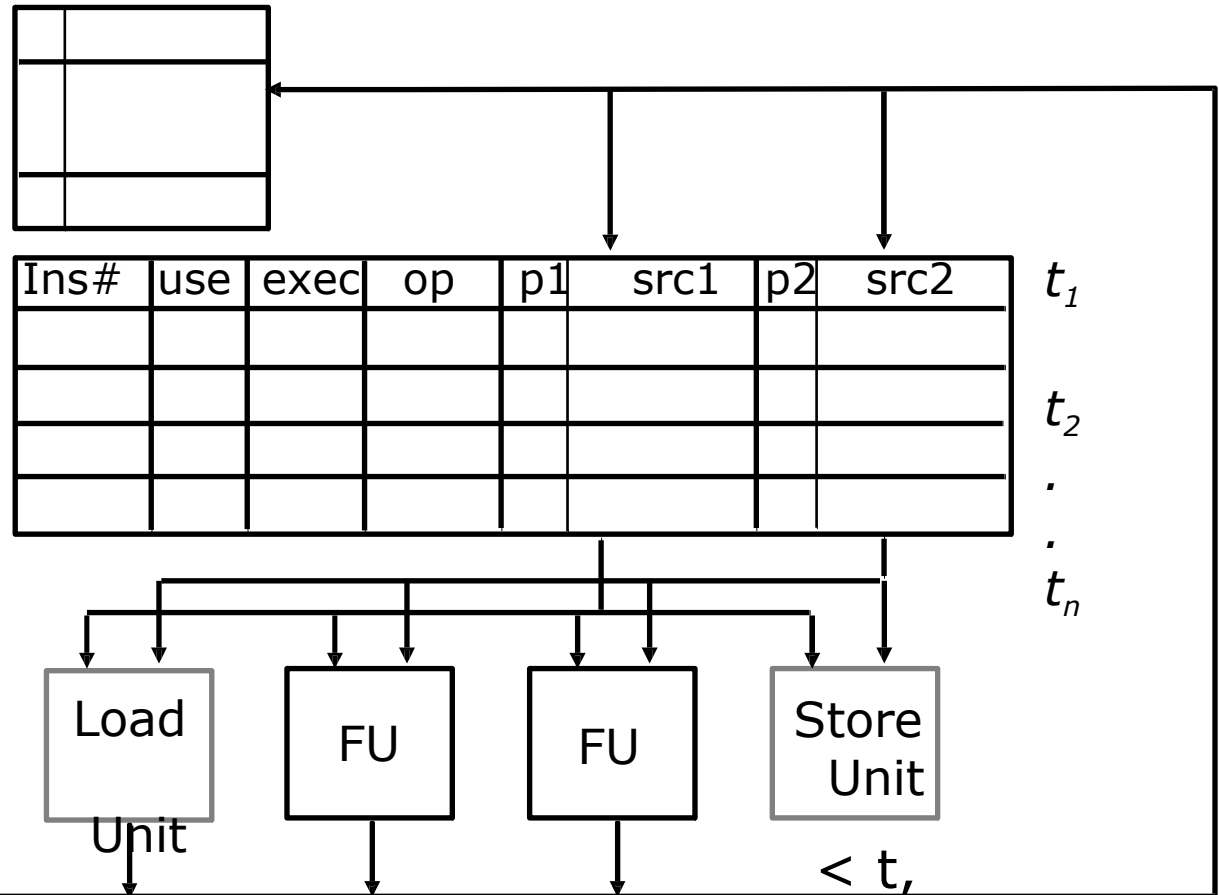
- When are names in sources replaced by data?
Whenever an FU produces data
- When can a name be reused?

Data-Driven Execution

*Renaming
table &
reg file*

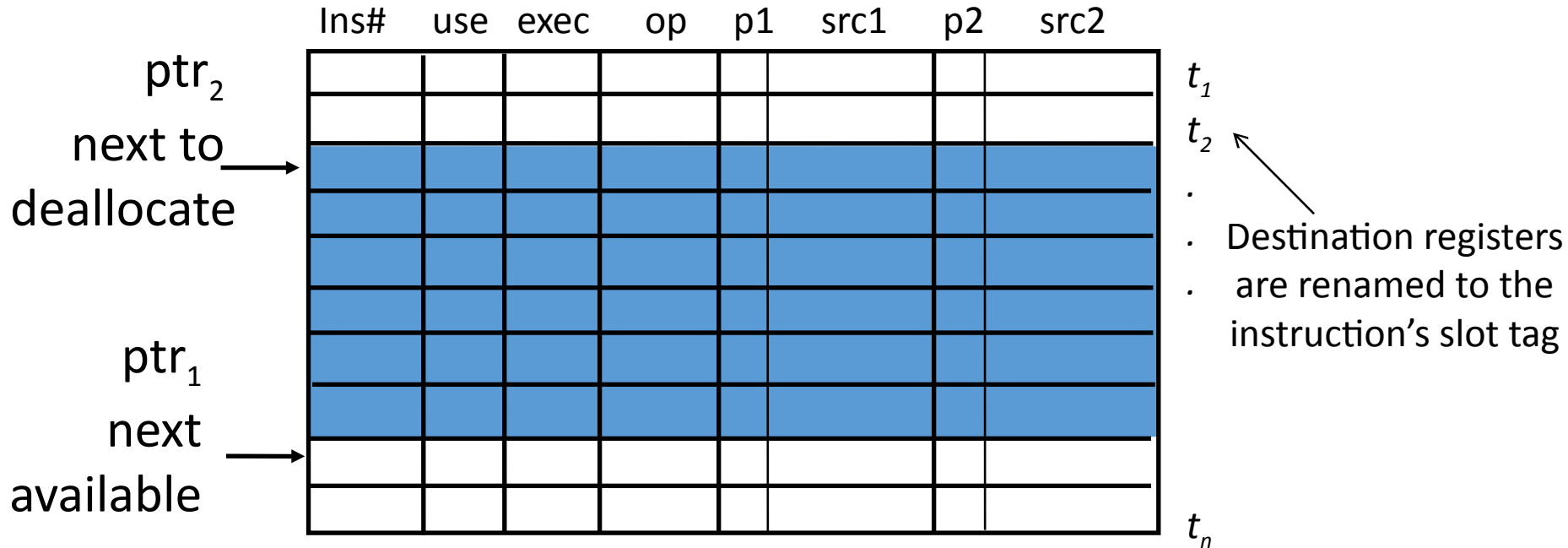
*Reorder
buffer*

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

Reorder Buffer Management



ROB managed circularly

- “exec” bit is set when instruction begins execution
- When an instruction completes its “use” bit is marked free
- ptr_2 is incremented only if the “use” bit is marked free

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

	p	data
f1		
f2		v1
f3		
f4		t5
f5		
f6		t3
f7		
f8		v4

v1

data / t_i

Reorder buffer

Ins#	use	exec	op	p1	src1	p2	src2
1	0	0	LD				
2	0	0	LD				
3	1	0	MUL	0	v2	1	v1
4	0	0	SUB	1	v1	1	v1
5	1	0	DIV	1	v1	0	v4

t_1

t_2

t_3

t_4

t_5

.

.

1 FLD	f2, 34(x2)
2 FLD	f4, 45(x3)
3 FMULT.D	f6, f4, f2
4 FSUB.D	f8, f2, f2
5 FDIV.D	f4, f2, f8
6 FADD.D	f10, f6, f4

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