

# **Computer Architecture**

## **Superscalar I**

Ting-Jung Chang

NYCU CS

# Recap: Pipeline Hazards

- **Structural Hazard:** An instruction in the pipeline may need a resource being used by another instruction in the pipeline
- **Data Hazard:** An instruction depends on a data value produced by an earlier instruction
- **Control Hazard:** Whether an instruction should be executed depends on a control decision made by an earlier instruction

# Quick Check

- Which of the following statements correctly describe characteristics of pipelined processor architectures?
  - A. Pipelining reduces the execution time (latency) of individual instructions.
  - B. Pipelining can achieve a lower average CPI compared to a multi-cycle processor.
  - C. Full bypassing resolves all data hazards, though it is costly.
  - D. Pipeline registers are part of the RISC-V architectural state.

# Recap: Memory Hierarchy



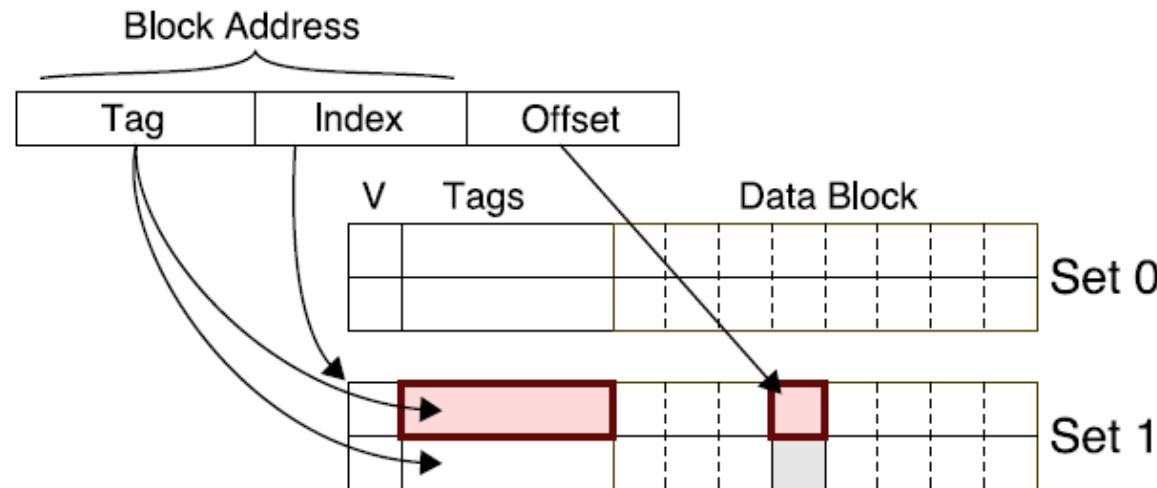
- Capacity: Register << SRAM << DRAM
- Latency: Register << SRAM << DRAM
- Bandwidth: on-chip >> off-chip
- On a data access:
  - if data is in fast memory → low-latency access to SRAM
  - if data is not in fast memory → long-latency access to DRAM
- Memory hierarchies only work if the small, fast memory actually stores data that is reused by the processor → **spatial and temporal locality!**

# Recap: Classifying Caches



- **Block Placement:** Where can a block be placed in the cache?
- **Block Identification:** How a block is found if it is in the cache?
- **Block Replacement:** Which block should be replaced on a miss?
- **Write Strategy:** What happens on a write?

# Recap: Categorizing Misses



- **Compulsory** – first-reference to a block, occur even with infinite cache
- **Capacity** – cache is too small to hold all data needed by program, occur even under perfect replacement policy (loop over 5 cache lines)
- **Conflict** – misses that occur because of collisions due to less than full associativity (loop over 3 cache lines)

# Quick Check

- Which of the following statements correctly describe characteristics of cache memory?
  - A. The memory hierarchy is built to trade off speed, cost, and capacity.
  - B. Direct-mapped caches generally have higher miss rates than set-associative caches of the same size.
  - C. Fully associative caches avoid conflict misses with minimal hardware cost.
  - D. Write-back caches simplify hardware but generate more memory traffic than write-through.

# 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

# Agenda

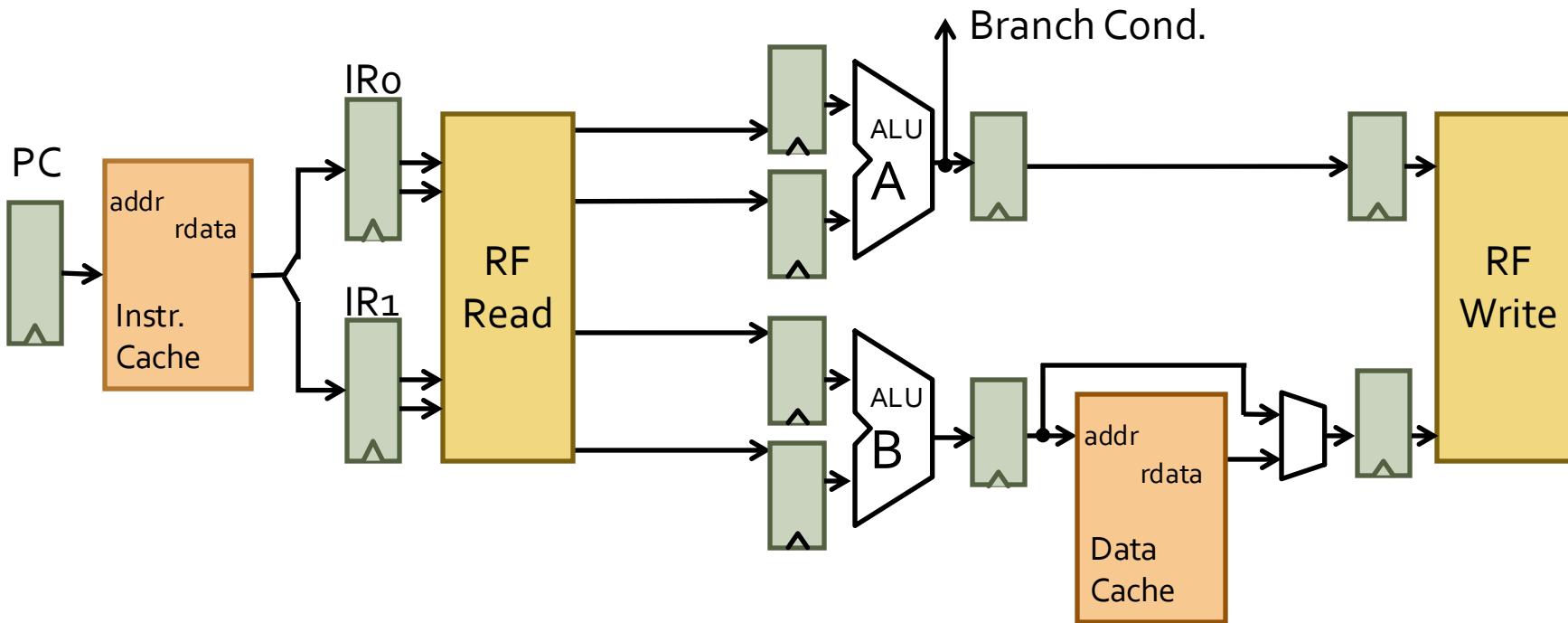
- Superscalar Processors
- Interrupts
- Out-of-Order Processors

# Introduction to Superscalar Processor

- Processors studied so far are fundamentally limited to  $CPI \geq 1$
- Superscalar processors enable  $CPI < 1$  ( $IPC > 1$ ) by executing multiple instructions in parallel
  - Instruction-level parallelism!
- Can have both in-order and out-of-order superscalar processors
  - We will start with in-order

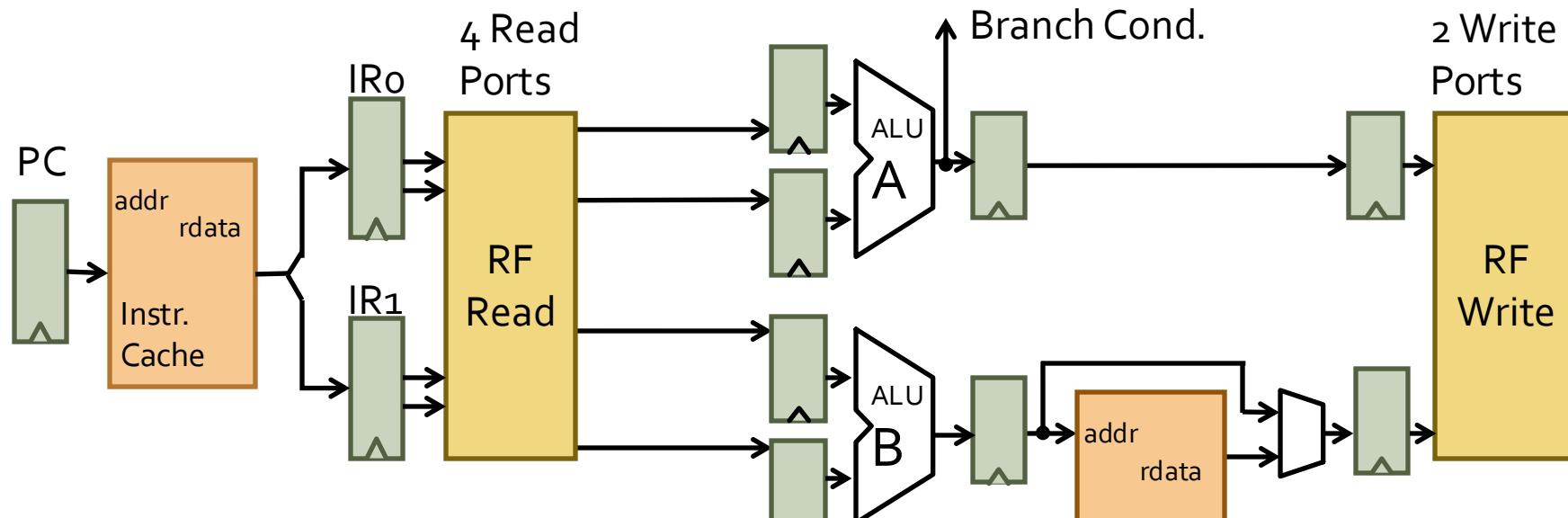
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

# Baseline 2-Way In-Order Superscalar Processor



Pipe A: Integer Ops., Branches  
Pipe B: Integer Ops., Memory

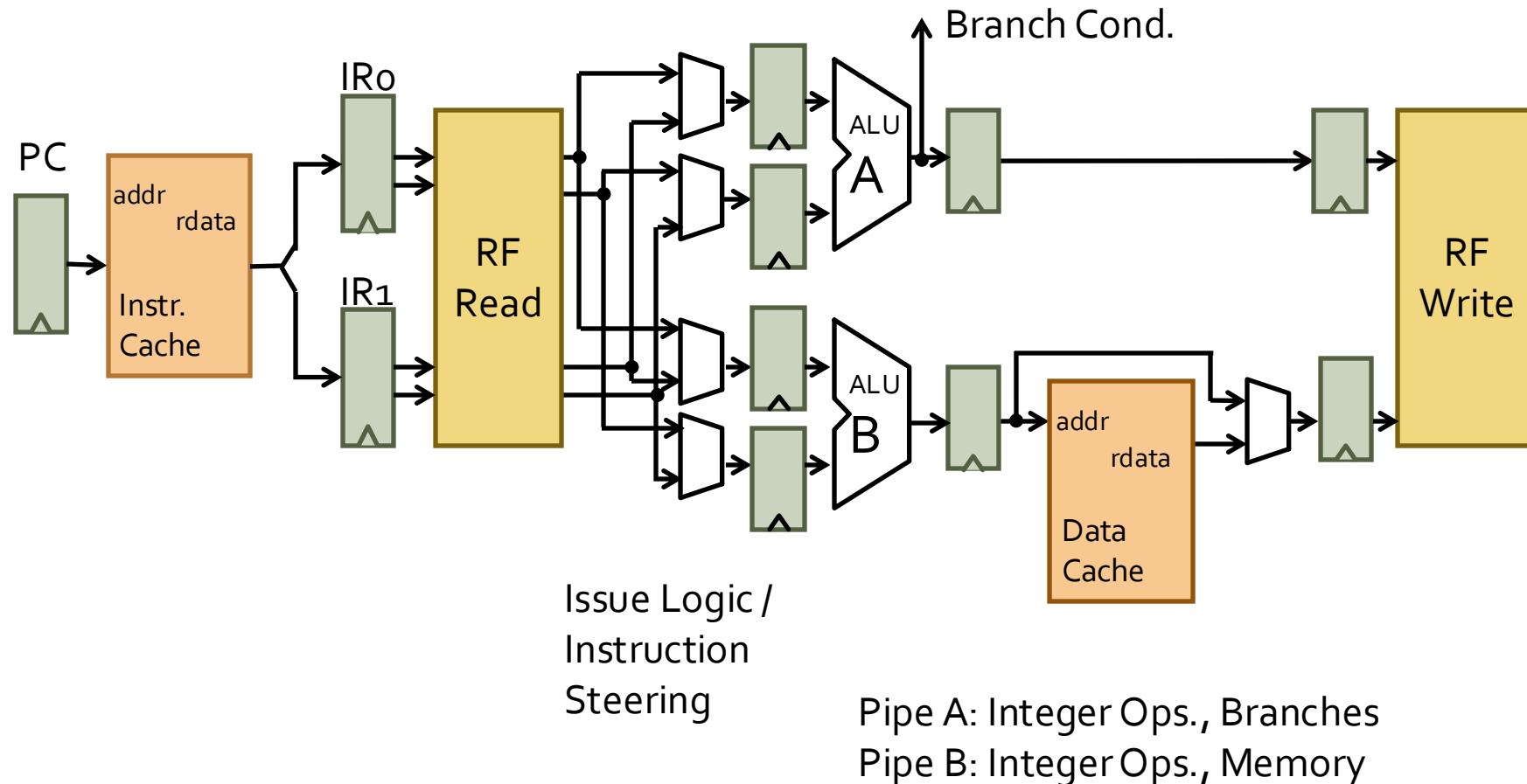
# Baseline 2-Way In-Order Superscalar Processor



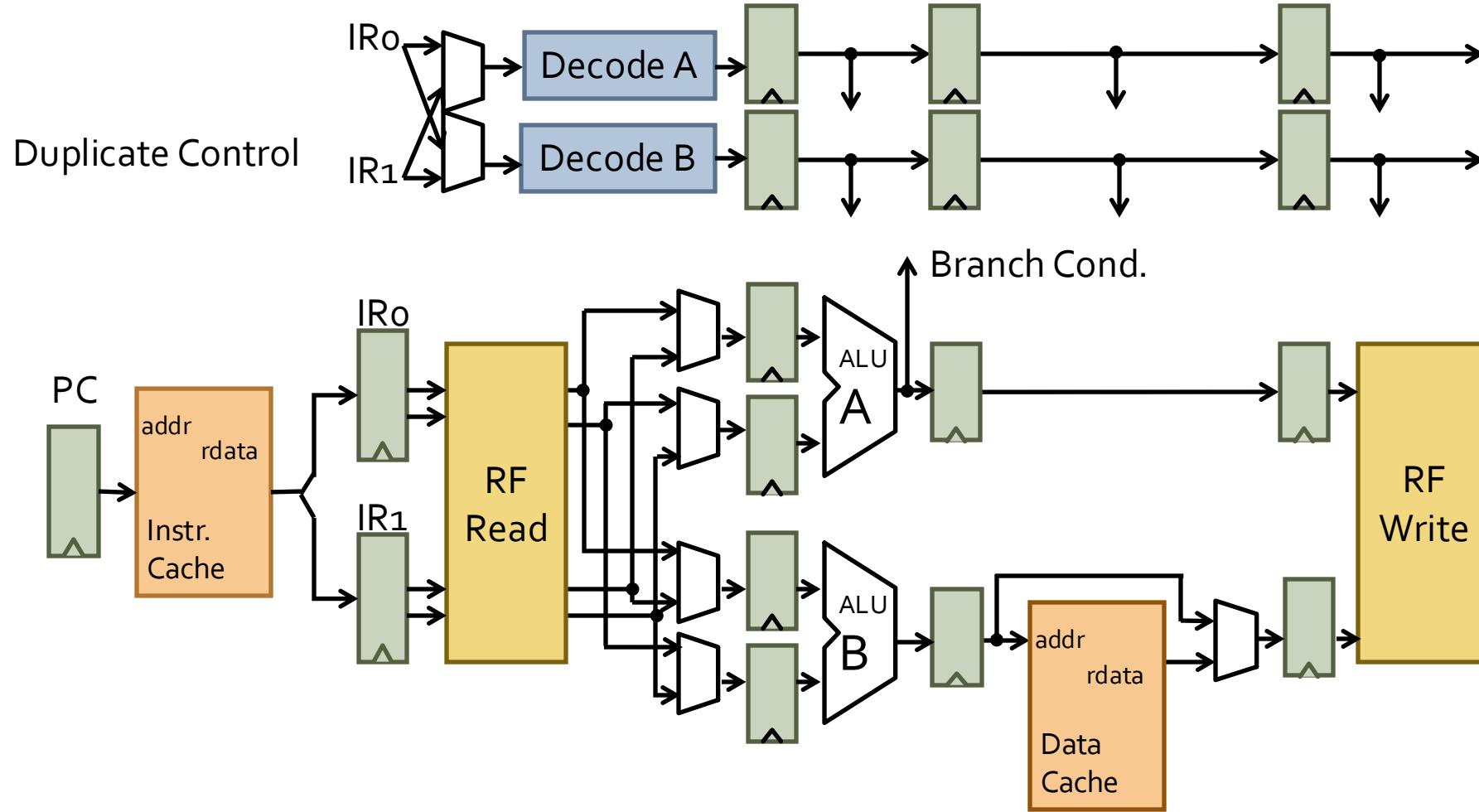
Fetch 2 Instructions at  
the same time

Pipe A: Integer Ops., Branches  
Pipe B: Integer Ops., Memory

# Baseline 2-Way In-Order Superscalar Processor



# Baseline 2-Way In-Order Superscalar Processor



# Issue Logic Pipeline Diagrams

OpA	F	D	A0	A1	W		
OpB	F	D	B0	B1	W		
OpC		F	D	A0	A1	W	
OpD		F	D	B0	B1	W	
OpE			F	D	A0	A1	W
OpF			F	D	B0	B1	W



CPI = 0.5 (IPC = 2)

Double Issue Pipeline  
Can have two instructions  
in same stage at same time

addi	F	D	A0	A1	W		
lw	F	D	B0	B1	W		
lw	F	D	B0	B1	W		
addi	F	D	A0	A1	W		
lw		F	D	B0	B1	W	
lw		F	D	D	B0	B1	W

Instruction Issue Logic swaps from natural position

Structural hazard

# Dual Issue Data Hazards

## No Bypassing:

addi x1,x1,1	F	D	A0	A1	W			
addi x3,x4,1	F	D	B0	B1	W			
addi x5,x6,1	F	D	A0	A1	W			
addi x7,x5,1	F	D	D	D	D	A0	A1	W

## Full Bypassing:

addi x1,x1,1	F	D	A0	A1	W	
addi x3,x4,1	F	D	B0	B1	W	WAR Hazard Possible?
addi x5,x6,1	F	D	A0	A1	W	
addi x7,x5,1	F	D	D	A0	A1	W

# Fetch Logic and Alignment

Cyc	Addr	Instr
0	0x000	OpA
0	0x004	OpB
1	0x008	OpC
1	0x00C	J 0x100
...		
2	0x100	OpD
2	0x104	J 0x204
...		
3	0x204	OpE
3	0x208	J 0x30C
...		
4	0x30C	OpF
4	0x310	OpG
5	0x314	OpH

0x000	0	0	1	1
...				
0x100	2	2		
...				
0x200		3	3	
...				
0x300				4
0x310	4	5		

Fetching across cache lines is  
very hard. May need extra ports.

# Fetch Logic and Alignment

Cyc	Addr	Instr	Ideal, No Alignment Constraints						
0	0x000	OpA	OpA	F	D	A0	A1	W	
0	0x004	OpB	OpB	F	D	B0	B1	W	
1	0x008	OpC	OpC	F	D	B0	B1	W	
1	0x00C	J 0x100	J	F	D	A0	A1	W	
...			OpD		F	D	B0	B1	W
2	0x100	OpD	J		F	D	A0	A1	W
2	0x104	J 0x204	OpE		F	D	B0	B1	W
...			J		F	D	A0	A1	W
3	0x204	OpE	OpF		F	D	B0	B1	W
3	0x208	J 0x30C	J		F	D	A0	A1	W
...			OpG		F	D	A0	A1	W
4	0x30C	OpF	OpH		F	D	B0	B1	W
4	0x310	OpG			F	D	A0	A1	W
5	0x314	OpH							

# With Alignment Constraints

Cyc	Addr	Instr
?	0x000	OpA
?	0x004	OpB
?	0x008	OpC
?	0x00C	J 0x100
...		
?	0x100	OpD
?	0x104	J 0x204
...		
?	0x204	OpE
?	0x208	J 0x30C
...		
?	0x30C	OpF
?	0x310	OpG
?	0x314	OpH

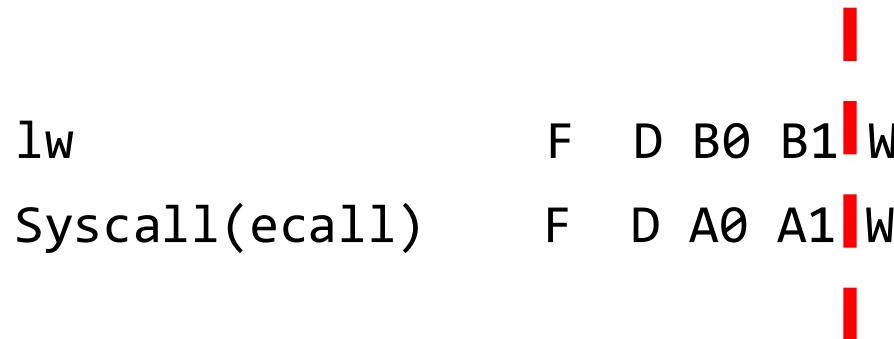
0x000	0	0	1	1
...				
0x100	2	2		
...				
0x200	3	3	4	4
...				
0x300			5	5
0x310	6	6		

# With Alignment Constraints

Cyc	Addr	Instr	F	D	A0	A1	W
0	0x000	OpA					
0	0x004	OpB					
1	0x008	OpC					
1	0x00C	J 0x100					
2	0x100	OpD					
2	0x104	J 0x204					
3	0x200	?			-	-	-
3	0x204	OpE					
4	0x208	J 0x30C					
4	0x20C	?			-	-	-
5	0x308	?			-	-	-
5	0x30C	OpF					
6	0x310	OpG					
6	0x314	OpH					

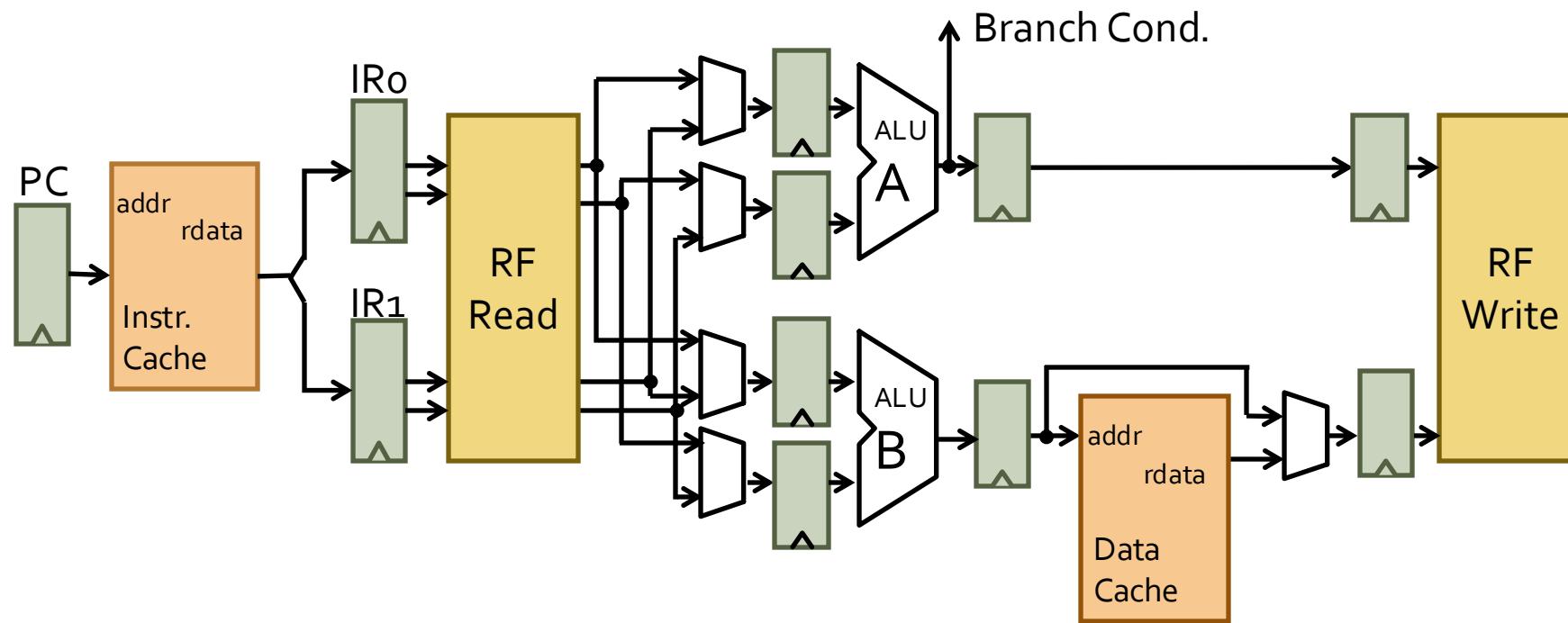
# Precise Exceptions and Superscalars

- Similar to tracking program order for data dependencies, we need to track order for exceptions

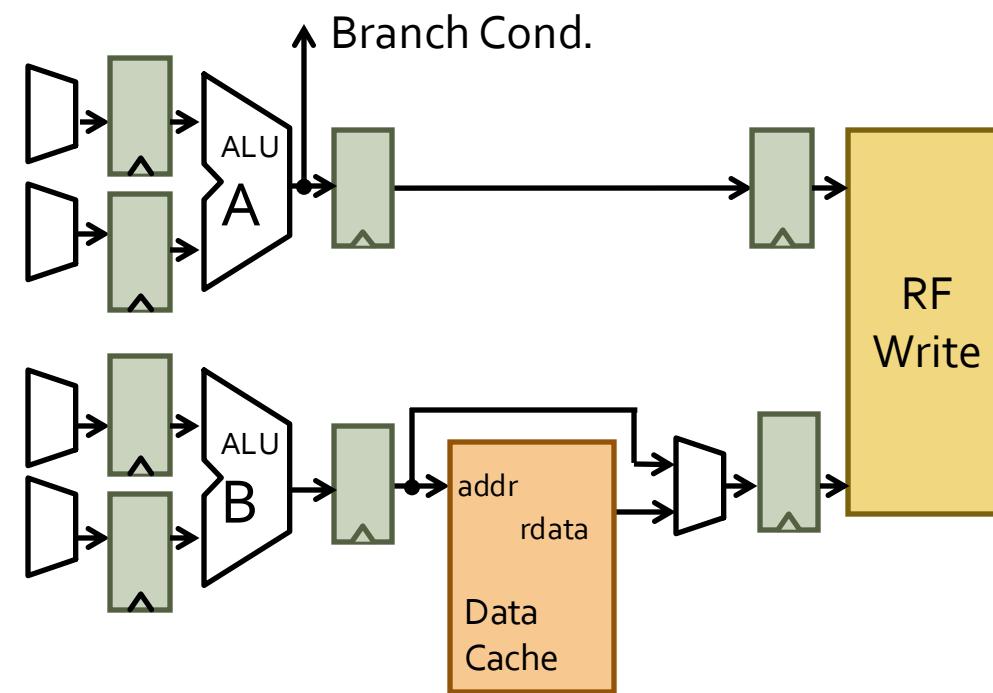


l<sub>w</sub> is in B pipeline but commits first in logical order!

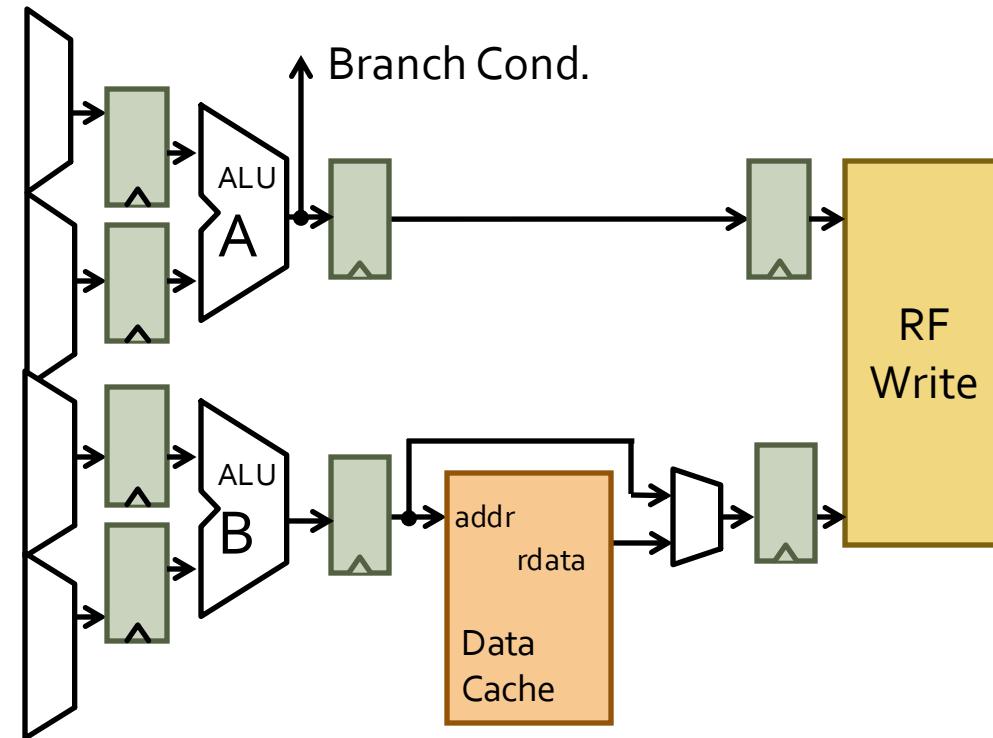
# Bypassing in Superscalar Pipelines



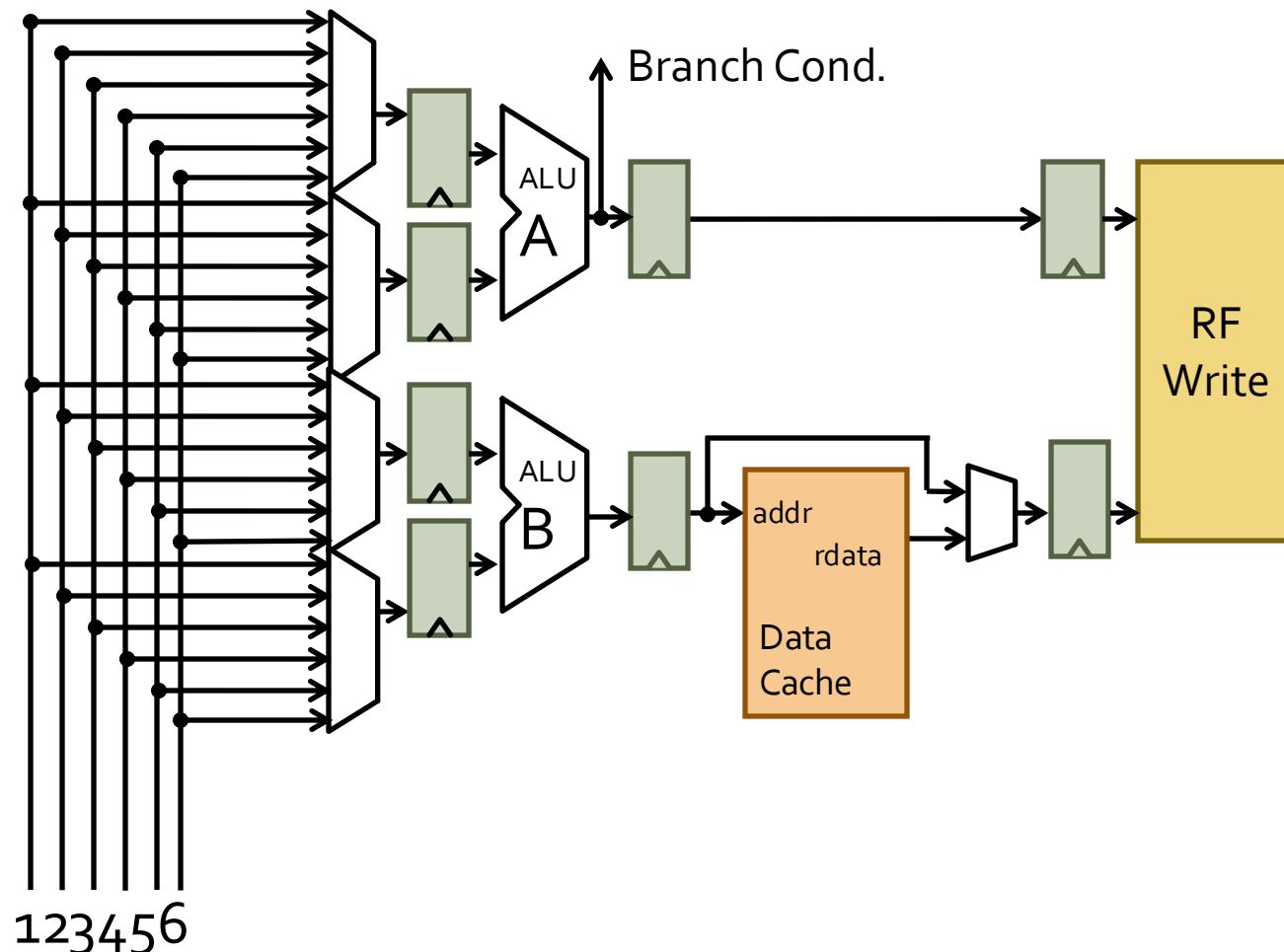
# Bypassing in Superscalar Pipelines



# Bypassing in Superscalar Pipelines



# Bypassing in Superscalar Pipelines



# Breaking Decode and Issue Stage

- Bypass network can become very complex
- Can motivate breaking Decode and Issue stage

D = decode, possibly resolve structural hazards

I = register file read, bypassing, issue/steer Instructions to proper unit

OpA F D I A0 A1 W

OpB F D I B0 B1 W

OpC F D I A0 A1 W

OpD F D I B0 B1 W

Good decision?

# Superscalars Multiply Branch Cost

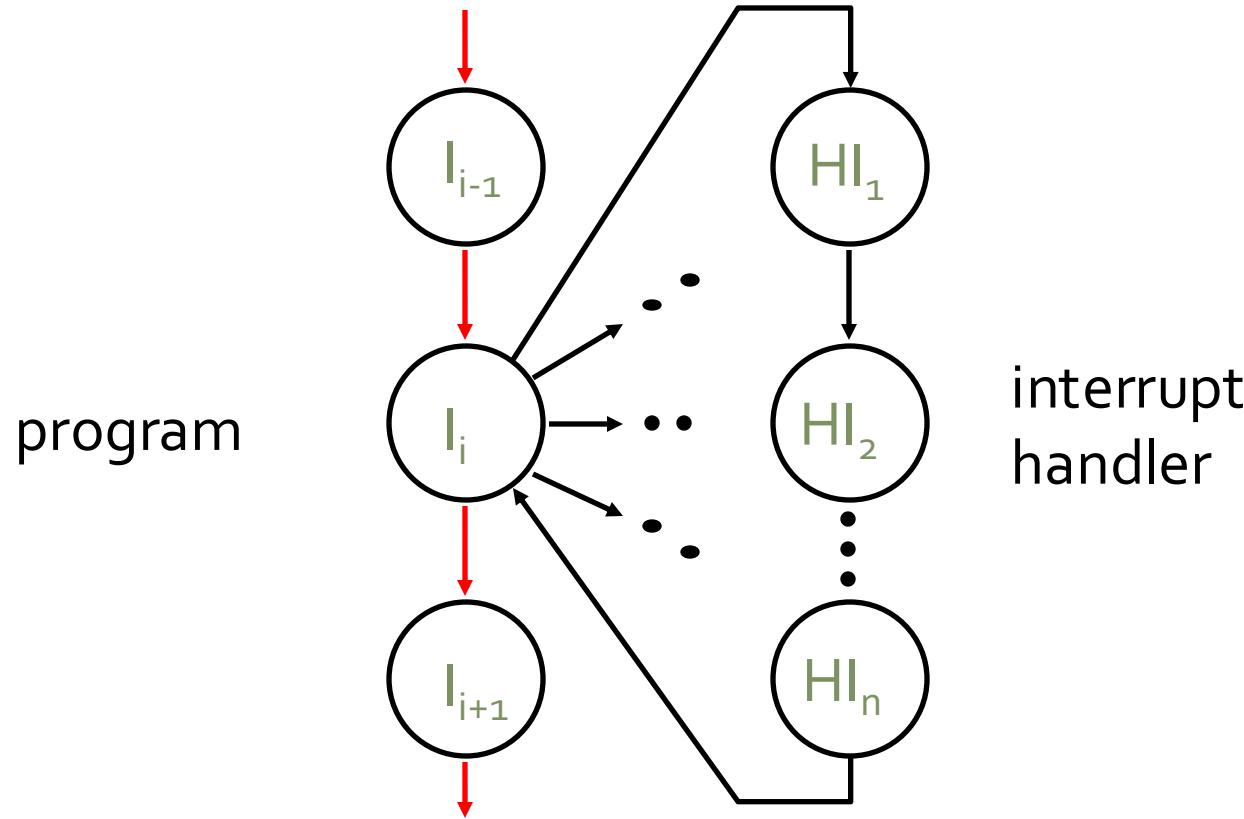
beq	F	D	I	A0	A1	W
OpA	F	D	I	B0	-	-
OpB		F	D	I	-	-
OpC		F	D	I	-	-
OpD			F	D	-	-
OpE			F	D	-	-
OpF				F	-	-
OpG				F	-	-
OpH				F	D	I
OpI				F	D	I
				B0	B1	W

# Agenda

- Superscalar
- Interrupts
- Out-of-Order Processors

# Interrupts:

## altering the normal flow of control



An external or internal event that needs to be processed by another (system) program. The event is usually unexpected or rare from program's point of view.

# Causes of Exceptions

Interrupt: an **event** that requests the attention of the processor

- Asynchronous: an **external event**
  - input/output device service request
  - timer expiration
  - power disruptions, hardware failure
- Synchronous: an **internal exception** (*a.k.a. exceptions/trap*)
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception
  - misaligned memory access
  - virtual memory exceptions: page faults, TLB misses, protection violations
  - software exceptions: ecall instruction, e.g., jumps into kernel

# Asynchronous Interrupts:

invoking the interrupt handler

- An I/O device requests attention by asserting one of the **prioritized interrupt request lines**
- When the processor decides to process the interrupt
  - It stops the current program at instruction  $I_i$ , completing all the instructions up to  $I_{i-1}$  (a **precise interrupt**)
  - It saves the PC of instruction  $I_i$  in a special register (mepc)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode

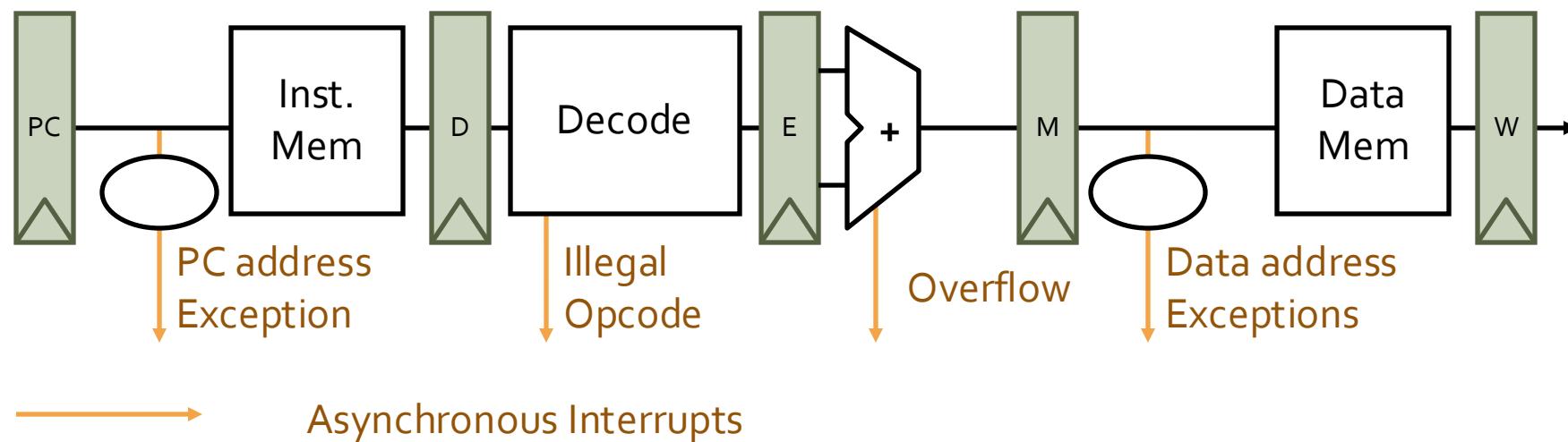
# Interrupt Handler

- Saves mepc before re-enabling interrupts to allow nested interrupts
  - need an instruction to move mepc into GPRs
  - need a way to mask further interrupts at least until mepc can be saved
- Needs to read a **status register** (mcause, and possibly mtval) that indicates the cause of the interrupt
- Uses the special instruction mret (machine return from exception) to resume user code, this:
  - enables interrupts
  - restores the processor to the previous privilege mode
  - restores hardware status and control state

# Synchronous Interrupts

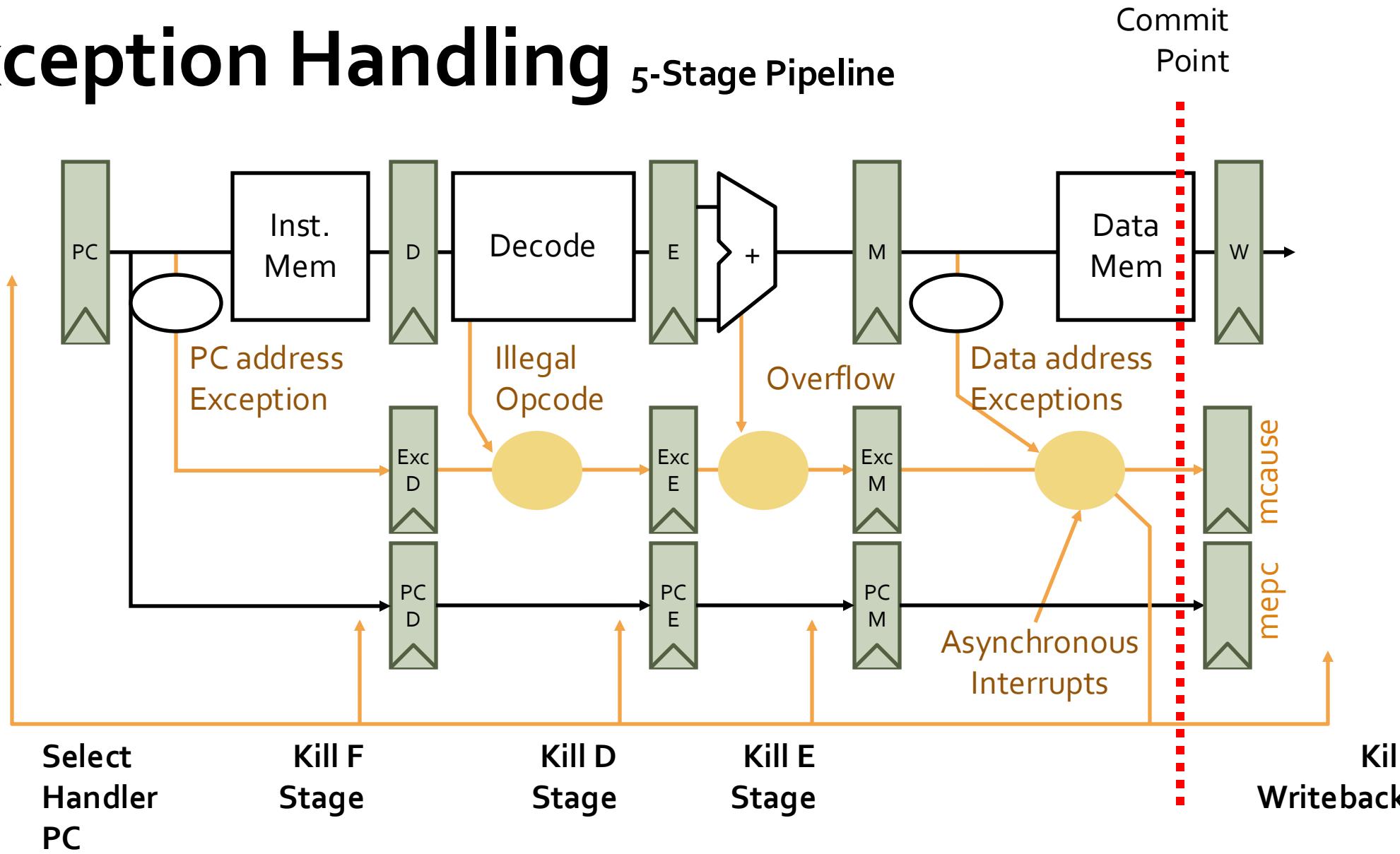
- A synchronous interrupt (exception) is caused by a **particular instruction**
- In general, the instruction cannot be completed and needs to be **restarted** after the exception has been handled
  - requires undoing the effect of one or more partially executed instructions
- In the case of a system call trap, the instruction is considered to have been completed
  - ecall is an instruction that traps to a higher privilege level (usually supervisor or machine mode)
  - Handler resumes at instruction after system call

# Exception Handling 5-Stage Pipeline



- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

# Exception Handling 5-Stage Pipeline



# Exception Handling

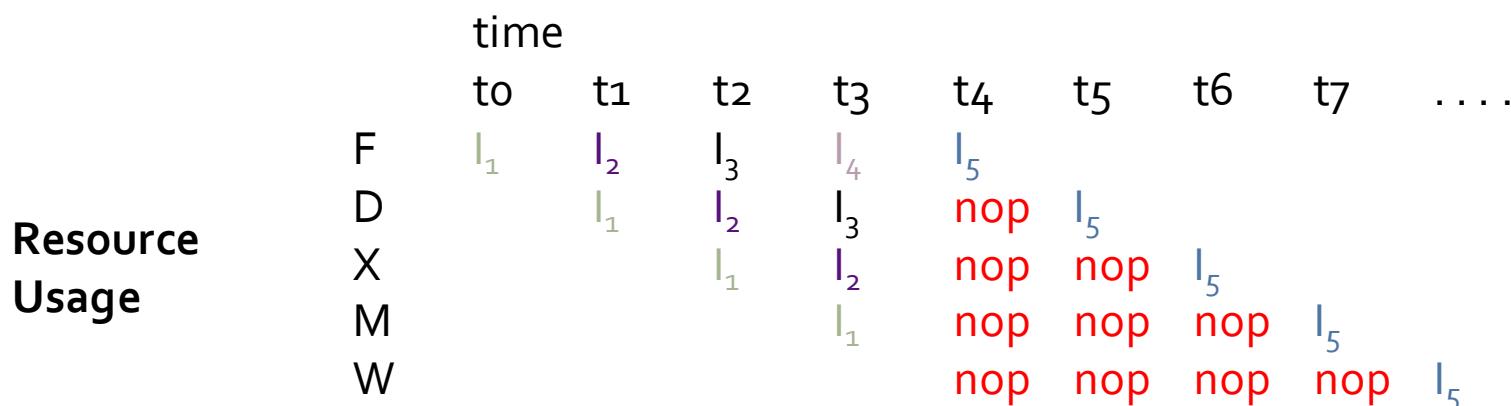
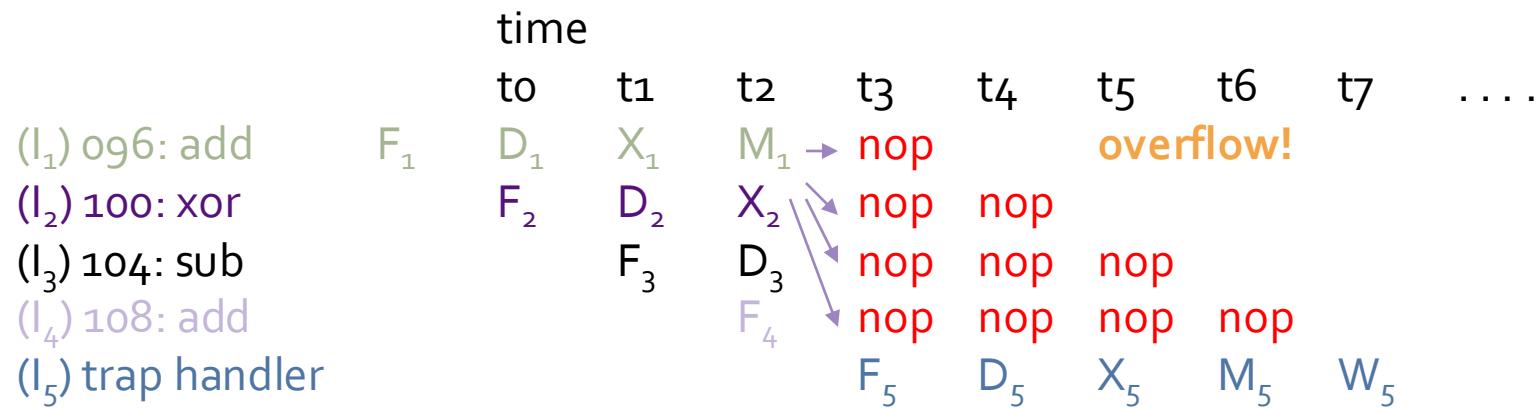
5-Stage Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions **for a given instruction**
- Inject external interrupts at commit point
- If exception at commit: update mcause and mepc registers, kill all stages, inject handler PC into fetch stage

# Speculating on Exceptions

- Prediction mechanism
  - Exceptions are rare, so simply predicting no exceptions is very accurate!
- Check prediction mechanism
  - Exceptions detected at end of instruction execution pipeline, special hardware for various exception types
- Recovery mechanism
  - Only write architectural state at commit point, so can throw away partially executed instructions after exception
  - Launch exception handler after flushing pipeline
- Bypassing allows use of uncommitted instruction results by following instructions

# Exception Pipeline Diagram



# Agenda

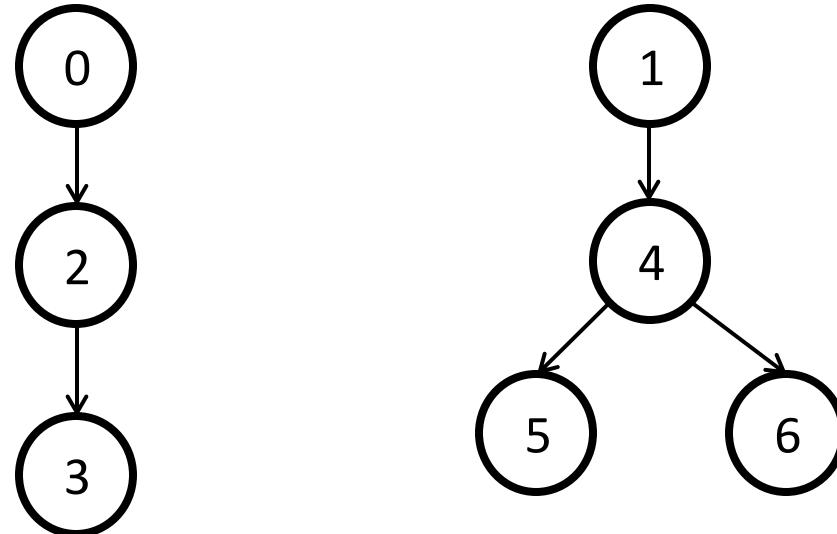
- Superscalar
- Interrupts
- Out-of-Order Processors

# Out-Of-Order (OOO) Introduction

Name	Frontend	Issue	Writeback	Commit	
I <sub>4</sub>	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I <sub>2</sub> O <sub>2</sub>	IO	IO	000	000	Scoreboard
I <sub>2</sub> OI	IO	IO	000	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO <sub>3</sub>	IO	000	000	000	Scoreboard and Issue Queue
IO <sub>2</sub> I	IO	000	000	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

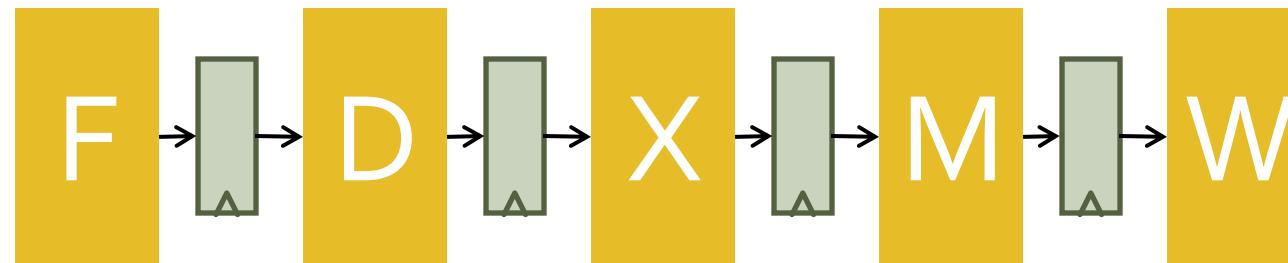
# OOO Motivating Code Sequence

```
0 mul x1, x2, x3  
1 addi x11, x10, 1  
2 mul x5, x1, x4  
3 mul x7, x5, x6  
4 addi x12, x11, 1  
5 addi x13, x12, 1  
6 addi x14, x12, 2
```

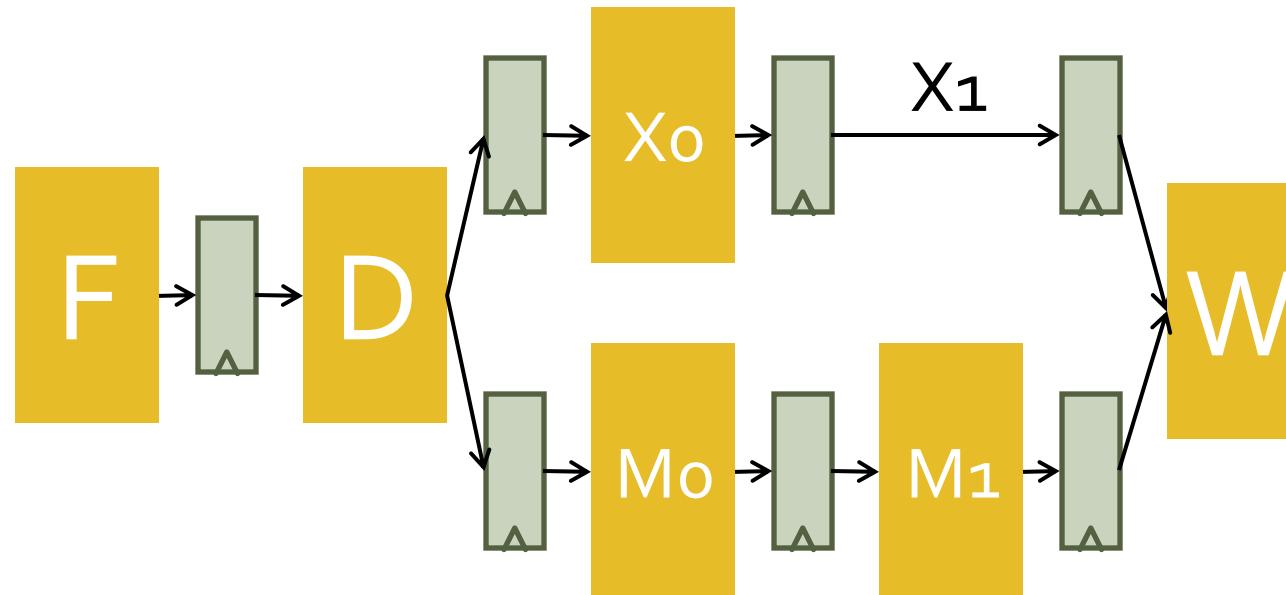


- Two independent sequences of instructions enable flexibility in terms of how instructions are scheduled in total order
- We can schedule statically in software or dynamically in hardware

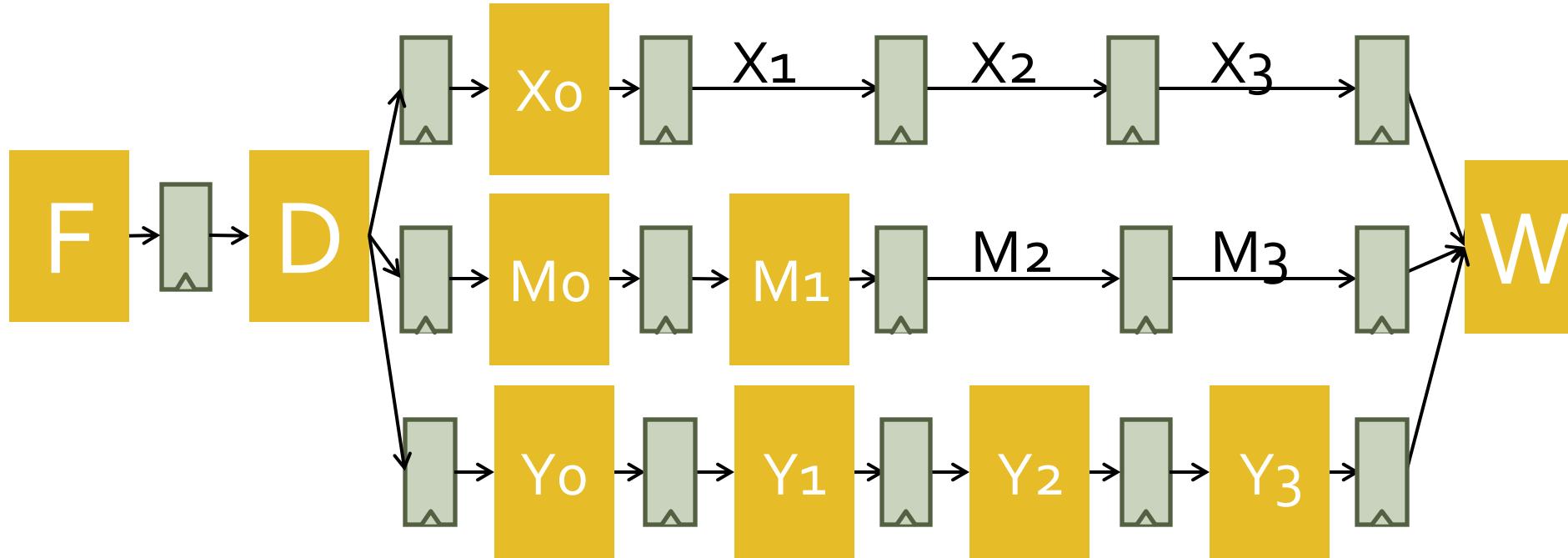
# I4: In-Order Front-End, Issue, Writeback, Commit



# I4: In-Order Front-End, Issue, Writeback, Commit

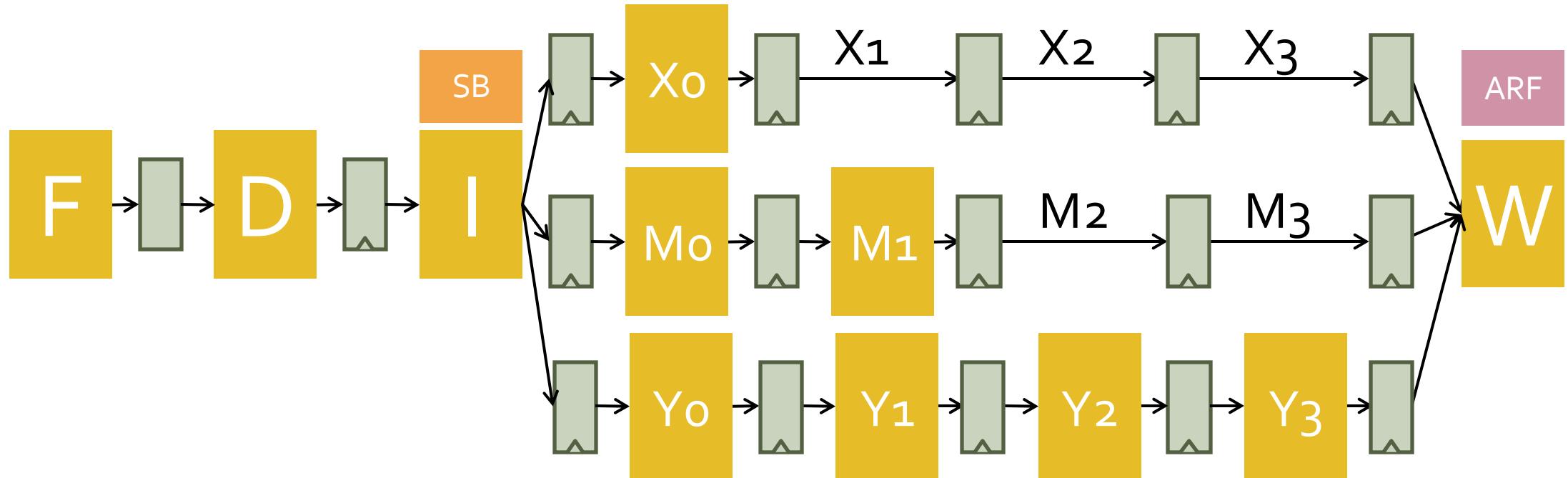


# I4: In-Order Front-End, Issue, Writeback, Commit (4-stage MUL)



To avoid increasing CPI, needs full bypassing which can be expensive. To help cycle time, add Issue stage where register file read and instruction “issued” to Functional Unit

# I4: In-Order Front-End, Issue, Writeback, Commit (4-stage MUL)



ARF

SB

R

R/W

W

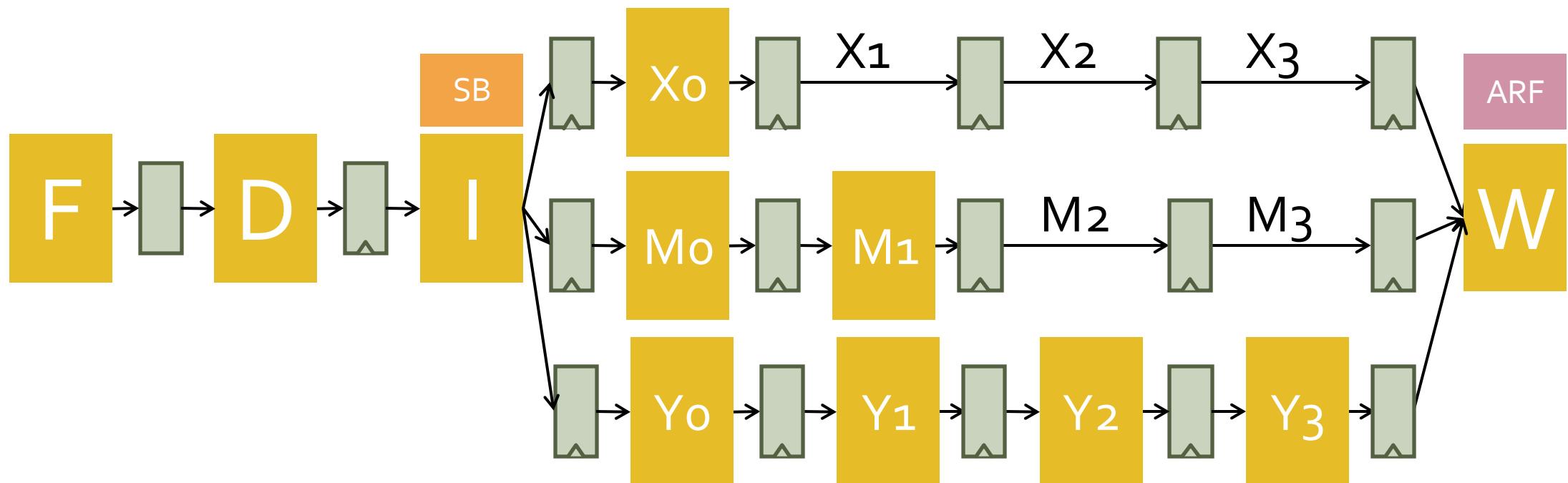
W

# Basic Scoreboard

	P	F	4	3	2	1	Data Avail.
x1			1				→
x2				→			→
x3				→			→
...							
x31				→			→

**P:** Pending, Write to Destination in flight  
**F:** Which functional unit is writing register  
**Data Avail.:** Where is the write data in the functional unit pipeline

- A 1 in the Data Avail. field of column 1 means that the result data is in stage 1 of functional unit F.
- The F and Data Avail. fields can be used to determine when and where to bypass.
- A 1 in column o means that the functional unit is in the Writeback stage during that cycle.
- The bits in the Data Avail. field shift right every cycle.



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

```

0 mul    x1, x2, x3
1 addi   x11,x10,1
2 mul    x5, x1, x4
3 mul    x7, x5, x6
4 addi   x12,x11,1
5 addi   x13,x12,1
6 addi   x14,x12,2

```

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	mul	x1, x2, x3	F	D	I	Y0	Y1	Y2	Y3	W											
1	addi	x11, x10, 1		F	D	I	X0	X1	X2	X3	W										
2	mul	x5, x1, x4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	mul	x7, x5, x6			F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W				
4	addi	x12, x11, 1				F	F	F	D	D	D	I	X0	X1	X2	X3	W				
5	addi	x13, x12, 1					F	F	F	D	I	X0	X1	X2	X3	W					
6	addi	x14, x12, 2							F	D	I	X0	X1	X2	X3	W					

Cyc D I                  4 3 2 1 0                  Dest Regs

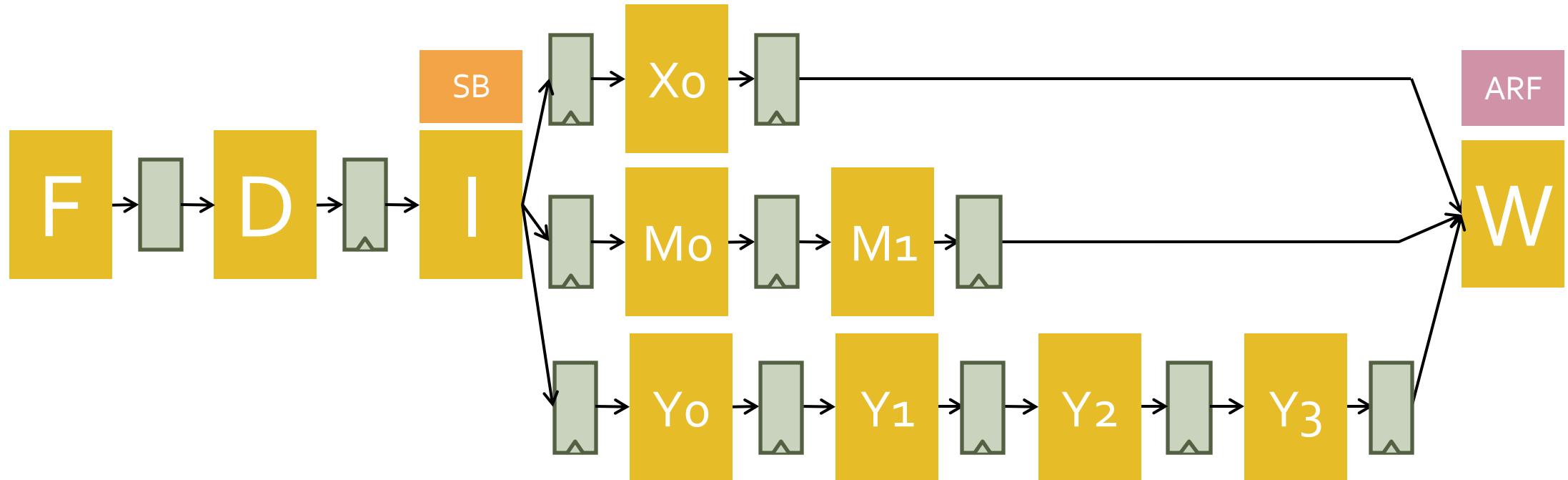
1	0																			
2	1	0																		
3	2	1				1														
4						1	1													
5							1	1												
6	3	2					1	1												
7						1		1	1											
8							1		1											
9								1												
10	4	3						1												
11	5	4				1			1											
12	6	5					1	1												
13	6						1	1	1											
14							1	1	1	1										
15								1	1	1	1									
16								1	1	1										
17									1	1										
18										1										

RED Indicates if we look at F Field, we can bypass on this cycle

What does the scoreboard look like at cycle 7?



# I<sub>2</sub>O<sub>2</sub>: In-order Frontend/Issue, Out-of-order Writeback/Commit



ARF

SB

R

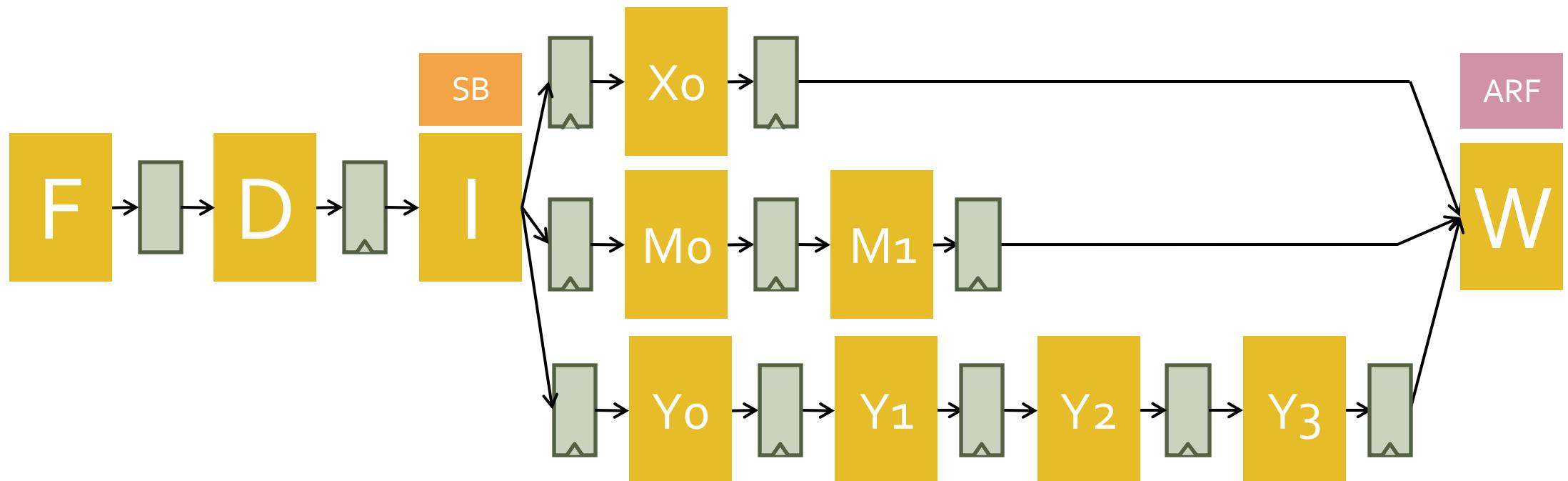
R/W

W

W

# I<sub>2</sub>O<sub>2</sub> Scoreboard

- Similar to I<sub>4</sub>, but we can now use it to track structural hazards on Writeback port
- Set bit in Data Avail. according to length of pipeline
- Architecture conservatively stalls to avoid WAW hazards by stalling in Issue therefore current scoreboard sufficient. More complicated scoreboard needed for processing WAW Hazards




---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

0	mul	x1, x2, x3
1	addi	x11, x10, 1
2	mul	x5, x1, x4
3	mul	x7, x5, x6
4	addi	x12, x11, 1
5	addi	x13, x12, 1
6	addi	x14, x12, 2

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	mul	x1, x2, x3	F	D	I	Y0	Y1	Y2	Y3	W											
1	addi	x11, x10, 1		F	D	I	X0	W													
2	mul	x5, x1, x4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	mul	x7, x5, x6			F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W				
4	addi	x12, x11, 1				F	F	F	D	D	D	D	I	X0	W						
5	addi	x13, x12, 1					F	F	F	F	D	I	X0	W							
6	addi	x14, x12, 2							F	D	I	I	X0	W							

Cyc D I                  4 3 2 1 0                  Dest Regs

1	0																			
2	1 0																			
3	2 1		1	□																
4			1	1																
5			1	1																
6	3 2			1																
7			1	1																
8			1																	
9			1																	
10	4 3				1															
11	5 4		1	□	1															
12	6 5		1	□	1															
13			1	1	1															
14	6		□	1	1															
15				1	1															
16				1																
17																				
18																				

RED Indicates if we look at F Field, we can bypass on this cycle

What does the scoreboard look like at cycle 7?

Writes with two cycle latency. Structural Hazard

# Early Commit Point?

0	mul	x1, x2, x3	F D I Y0 Y1 Y2 Y3 /
1	addi	x11, x10, 1	F D I X0 W /
2	mul	x5, x1, x4	F D I I I /
3	mul	x7, x5, x6	F D D D /
4	addi	x12, x11, 1	F F F /
5	addi	x13, x12, 1	/
6	addi	x14, x12, 2	

- Limits certain types of exceptions

# Recap: Out-Of-Order (OOO)

Name	Frontend	Issue	Writeback	Commit	
I <sub>4</sub>	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I <sub>2</sub> O <sub>2</sub>	IO	IO	000	000	Scoreboard
I <sub>2</sub> OI	IO	IO	000	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO <sub>3</sub>	IO	000	000	000	Scoreboard and Issue Queue
IO <sub>2</sub> I	IO	000	000	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

# Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)
  - David Wentzlaff (Princeton)
- MIT material derived from course 6.823
- UCB material derived from course CS252
- Cornell material derived from course ECE 4750
- Princeton material derived from course ECE 475