EE 660: Computer Architecture Superscalar Pipelines

Yao Zheng

Department of Electrical Engineering

University of Hawai‘i at Mānoa

UNIVERSITY of HAWAI‘I®
MĀNOA

Based on the slides of Prof. David Wentzlaff

Agenda

- Instruction-Level Parallelism
- Fetch Logic and Alignment
- Interrupts and Exceptions
- Interrupts and Bypassing

Agenda

- Instruction-Level Parallelism
- Fetch Logic and Alignment
- Interrupts and Exceptions
- Interrupts and Bypassing

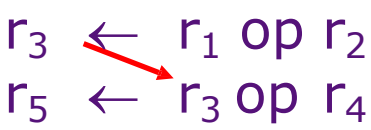
Types of Data Hazards

Consider executing a sequence of

$r_k \leftarrow r_i \;\; op \;\; r_j$

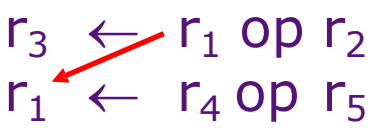
type of instructions

Data-dependence



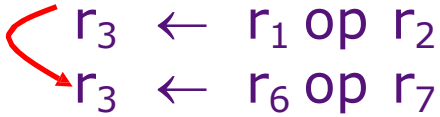
Read-after-Write (RAW) hazard

Anti-dependence



Write-after-Read (WAR) hazard

Output-dependence

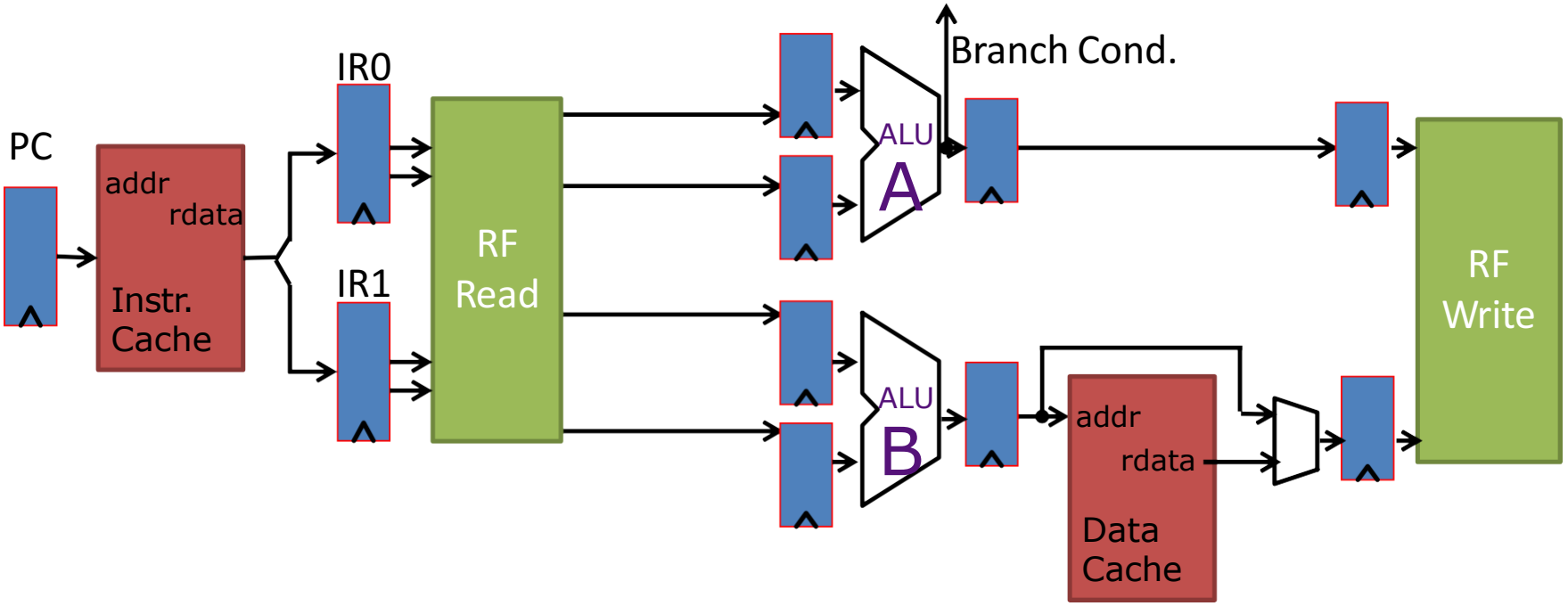


Write-after-Write (WAW) hazard

Introduction to Superscalar Processor

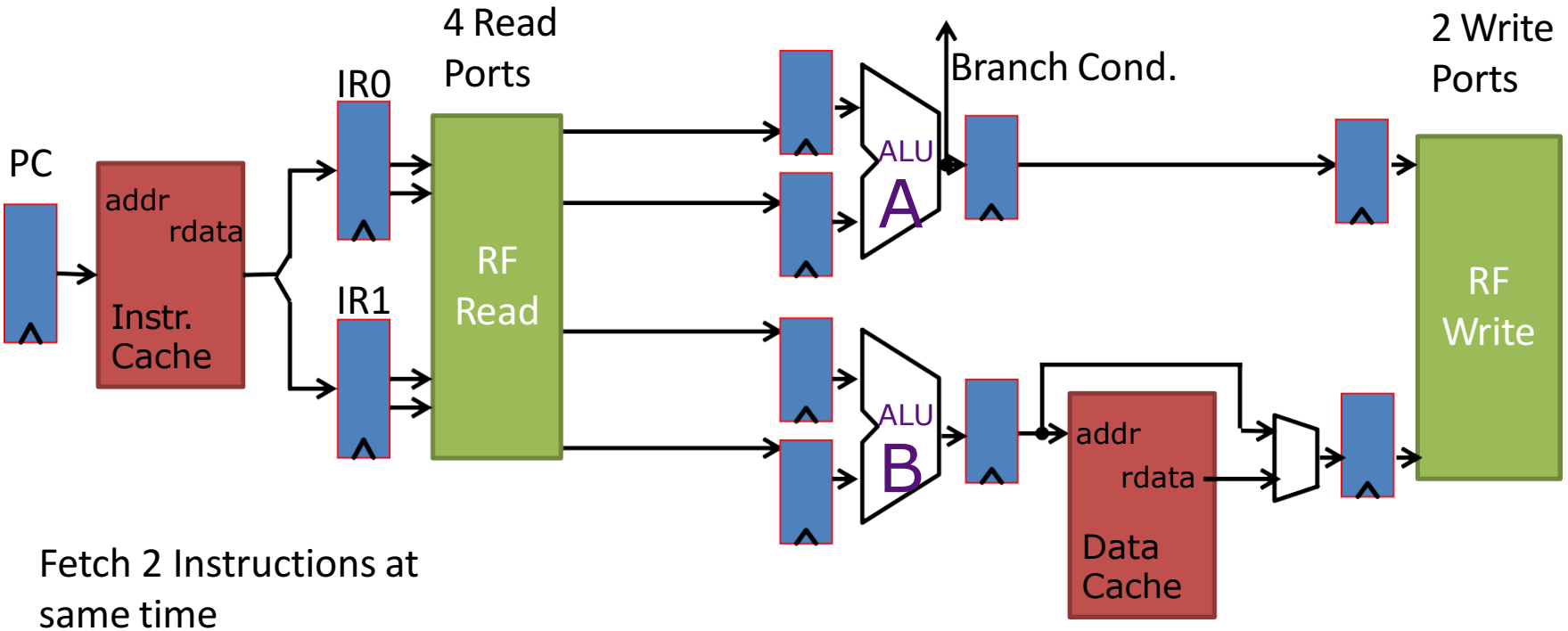
- Processors studied so far are fundamentally limited to CPI >= 1
- Superscalar processors enable CPI < 1 (IPC > 1) by executing multiple instructions in parallel
- Can have both in-order and out-of-order superscalar processors. We will start with in-order.

Baseline 2-Way In-Order Superscalar Processor



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

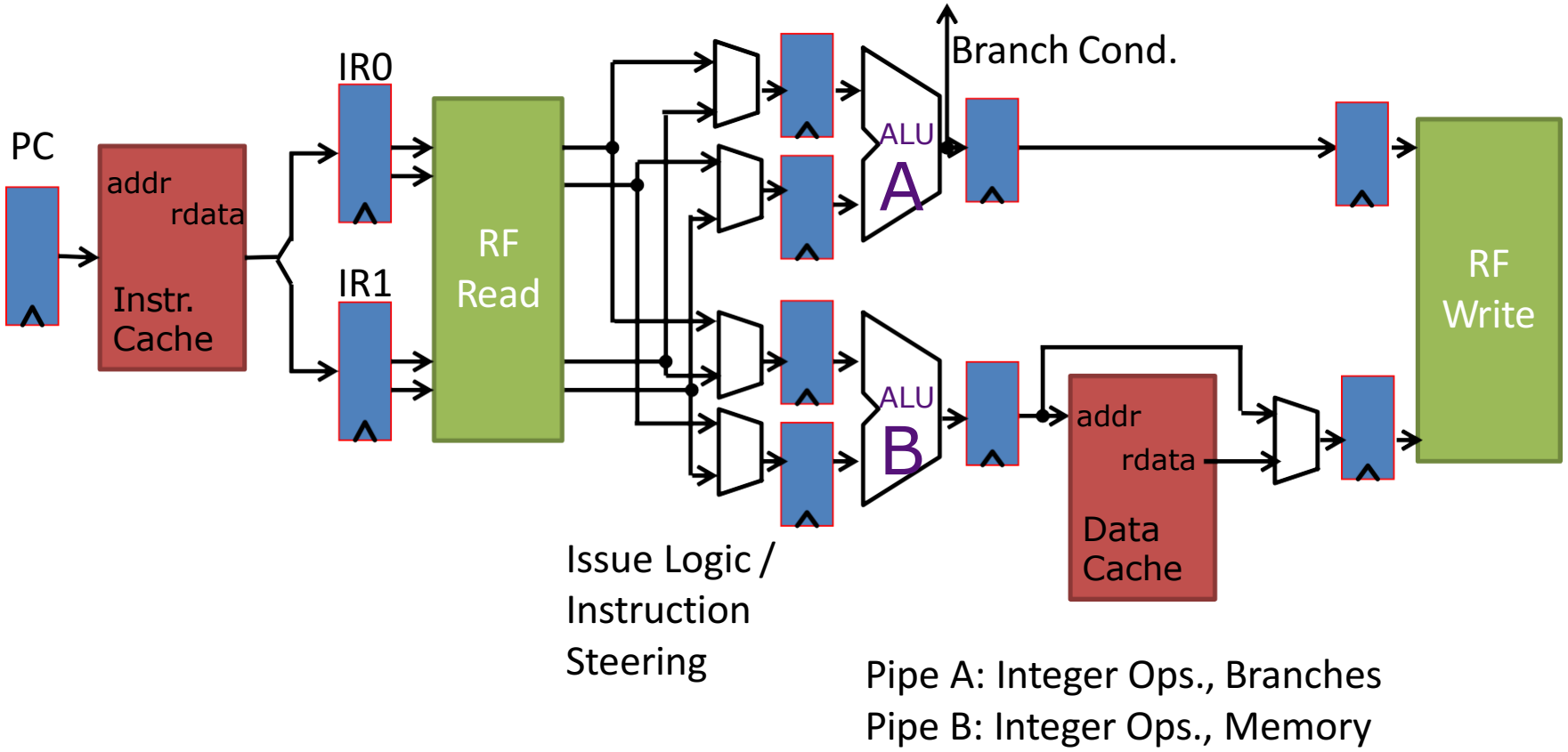
Baseline 2-Way In-Order Superscalar Processor



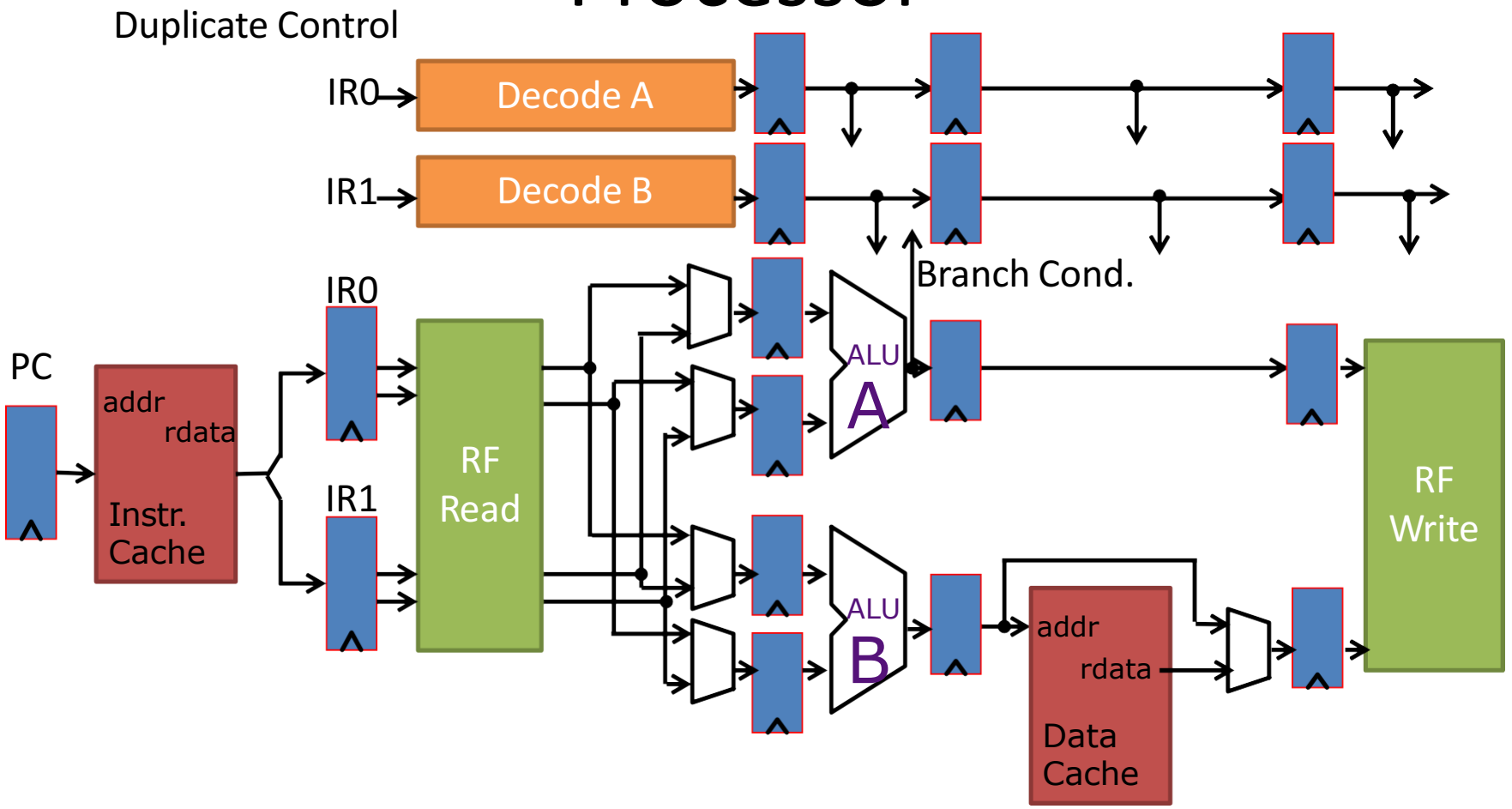
Fetch 2 Instructions at same time

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

Baseline 2-Way In-Order Superscalar Processor

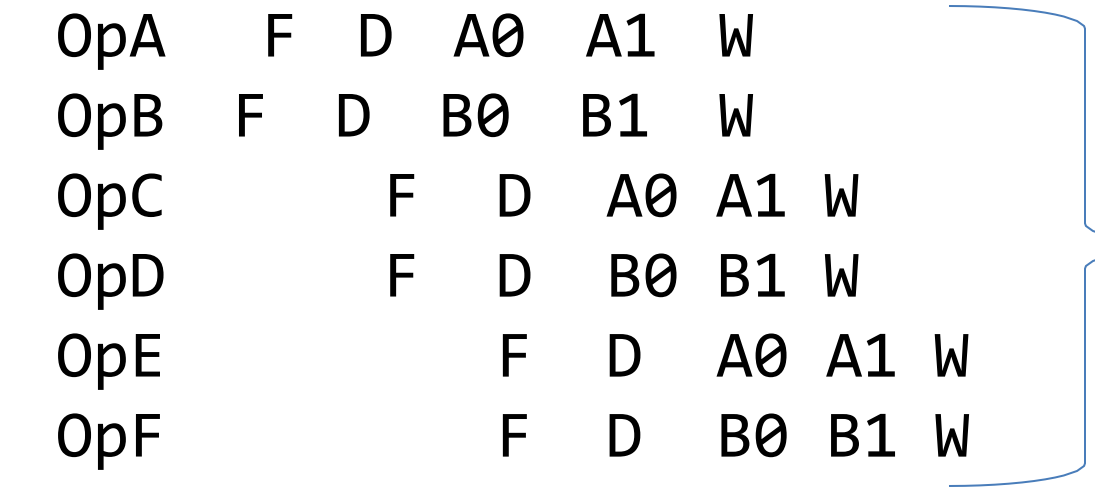


Baseline 2-Way In-Order Superscalar Processor



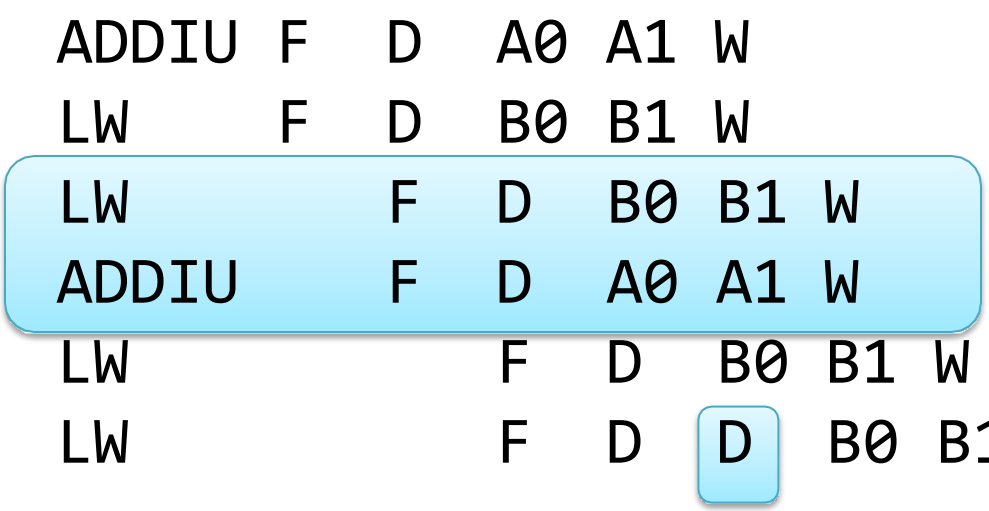
Pipe A: Integer Ops., Branches
Pipe B: Integer Ops., Memory

Issue Logic Pipeline Diagrams



CPI = 0.5 (IPC = 2)

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



Instruction Issue Logic swaps from natural position

Structural Hazard

Dual Issue Data Hazards

No Bypassing:

```
ADDIU R1,R1,1  F  D  A0 A1 W
ADDIU R3,R4,1  F  D  B0 B1 W
ADDIU R5,R6,1     F  D  A0 A1 W
ADDIU R7,R5,1     F  D  D  D  D  A0 A1 W
```

Full Bypassing:

```
ADDIU R1,R1,1  F  D  A0 A1 W
ADDIU R3,R4,1  F  D  B0 B1 W
ADDIU R5,R6,1     F  D  A0 A1 W
ADDIU R7,R5,1     F  D  D  A0 A1 W
```

Dual Issue Data Hazards

```
Order Matters:
ADDIU R1,R1,1   F  D  A0 A1 W
ADDIU R3,R4,1   F  D  B0 B1 W
ADDIU R7,R5,1      F  D  A0 A1 W
ADDIU R5,R6,1      F  D  B0 B1 W
```

WAR Hazard Possible?

Agenda

- Instruction-Level Parallelism
- Fetch Logic and Alignment
- Interrupts and Exceptions
- Interrupts and Bypassing

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

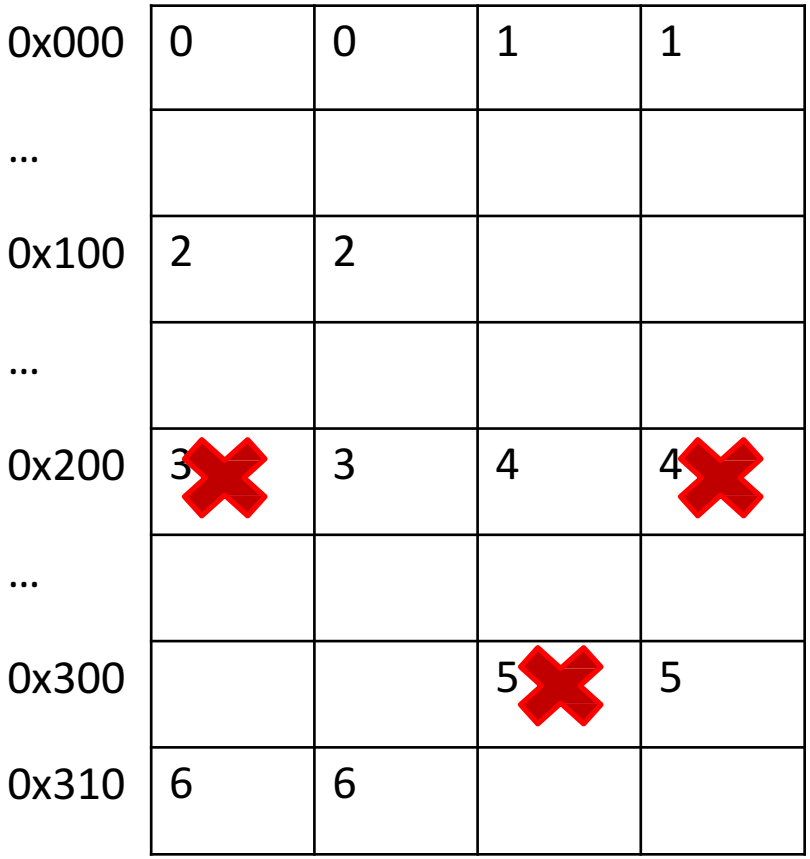
```
Cyc Addr Instr
0   0x000 OpA
0   0x004 OpB
1   0x008 OpC
1   0x00C J 0x100
…
2   0x100 OpD
2   0x104 J 0x204
…
3   0x204 OpE
3   0x208 J 0x30C
…
4   0x30C OpF
4   0x310 OpG
5   0x314 OpH
```

Ideal, No Alignment Constraints

```
OpA F  D  A0 A1 W
OpB F  D  B0 B1 W
OpC    F  D  B0 B1 W
J      F  D  A0 A1 W
OpD       F  D  B0 B1 W
J         F  D  A0 A1 W
OpE          F  D  B0 B1 W
J            F  D  A0 A1 W
OpF             F  D  A0 A1 W
OpG             F  D  B0 B1 W
OpH                F  D  A0 A1 W
```

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
```



With Alignment Constraints

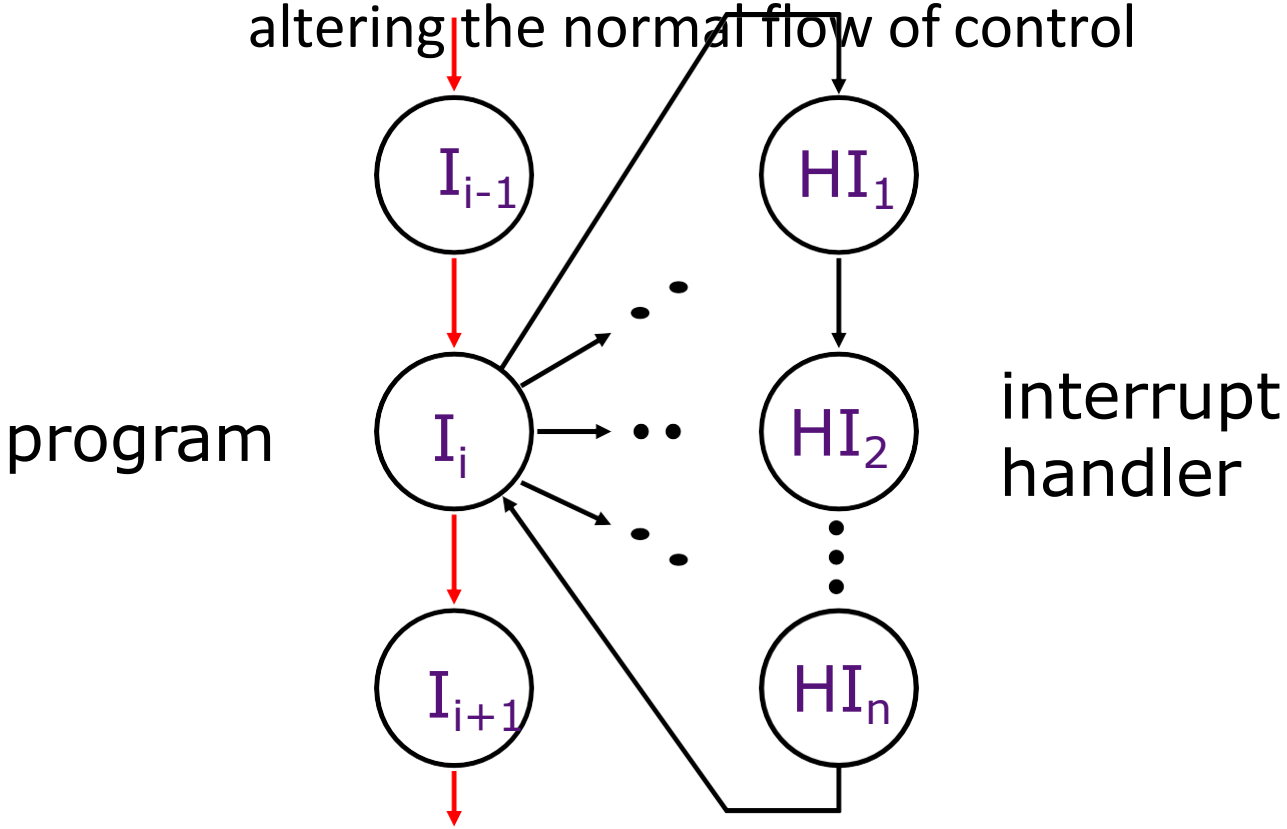
```
Cyc Addr Instr
1   0x000 OpA        F  D  A0 A1 W
1   0x004 OpB        F  D  B0 B1 W
2   0x008 OpC           F  D  B0 B1 W
2   0x00C J 0x100       F  D  A0 A1 W
3   0x100 OpD              F  D  B0 B1 W
3   0x104 J 0x204          F  D  A0 A1 W
4   0x200 ?                   F  -  -  -  -
4   0x204 OpE                 F  D  A0 A1 W
5   0x208 J 0x30C                F  D  A0 A1 W
5   0x20C ?                      F  -  -  -  -
6   0x308 ?                         F  -  -  -  -
6   0x30C OpF                       F  D  A0 A1 W
7   0x310 OpG                          F  D  A0 A1 W
7   0x314 OpH                          F  D  B0 B1 W
```

Agenda

- Instruction-Level Parallelism
- Fetch Logic and Alignment
- Interrupts and Exceptions
- Interrupts and Bypassing

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 Interrupts

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:* system calls, 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 (EPC)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode

Interrupt Handler

- Saves EPC before re-enabling interrupts to allow nested interrupts $\Rightarrow$
  - need an instruction to move EPC into GPRs
  - need a way to mask further interrupts at least until EPC can be saved
- Needs to read a *status register* that indicates the cause of the interrupt
- Uses a special indirect jump instruction RFE (*return-from-exception*) to resume user code, this:
  - enables interrupts
  - restores the processor to the user mode
  - restores hardware status and control state

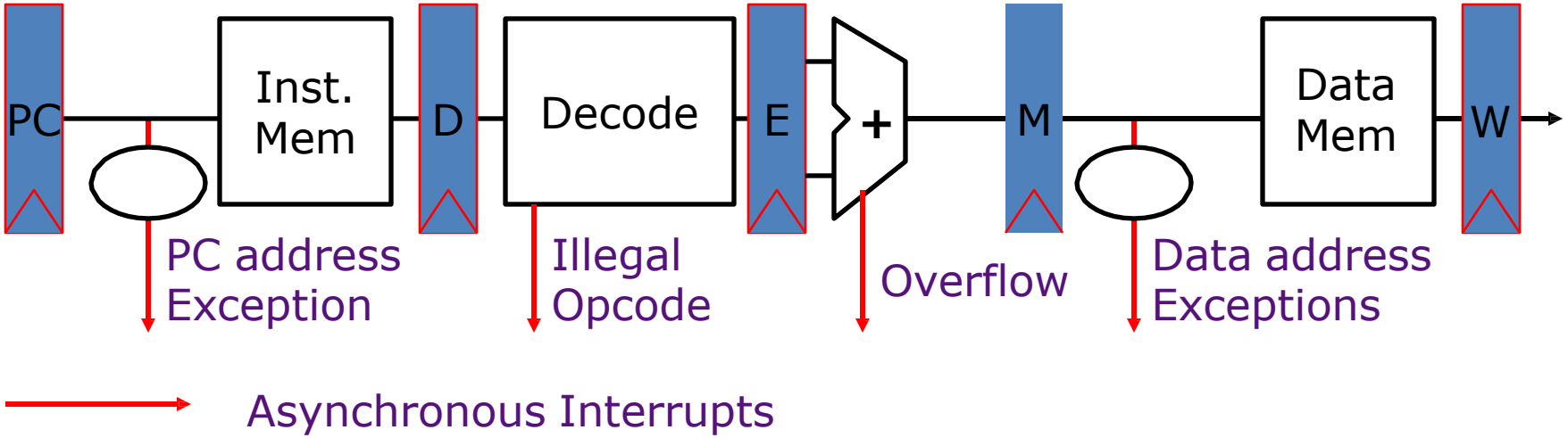
Synchronous Interrupts

- 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

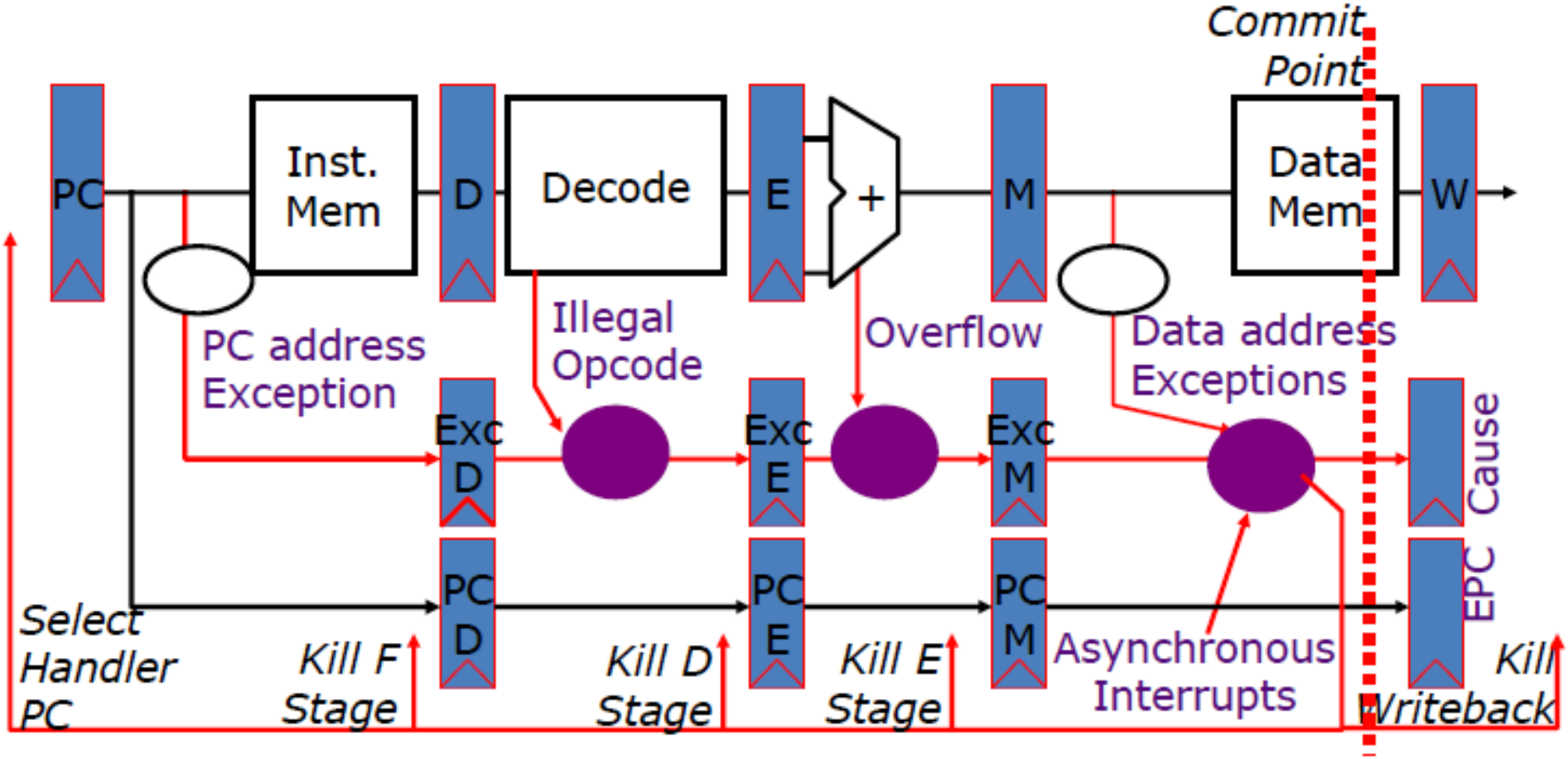
- In the case of a system call trap, the instruction is considered to have been completed
  - syscall is a special jump instruction involving a change to privileged kernel mode
  - Handler resumes at instruction after system call

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

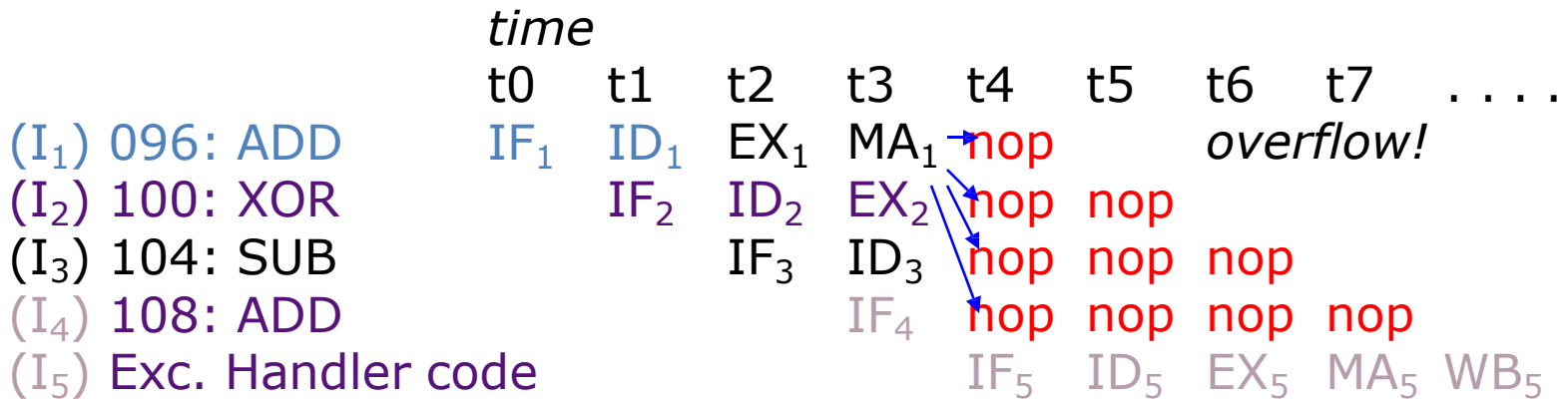
- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler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

Speculating on Exceptions

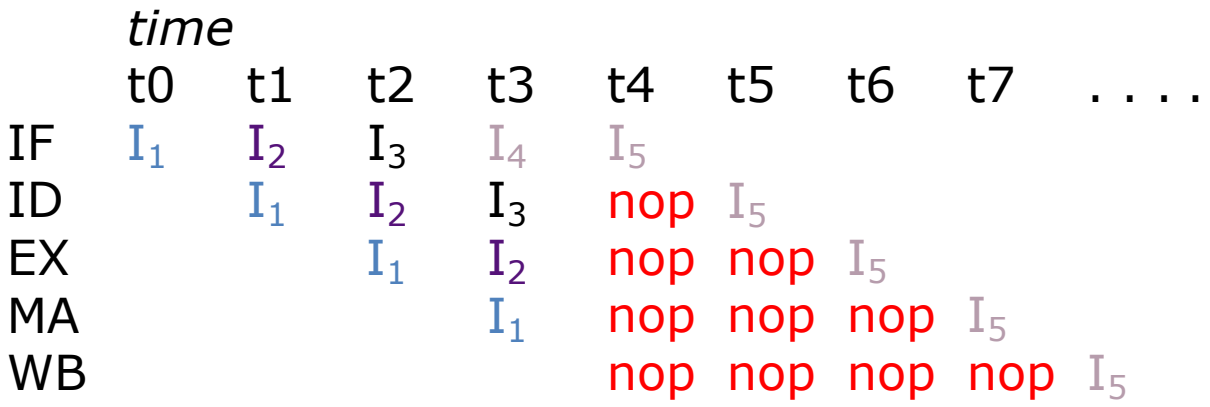
- Prediction mechanism
  - Exceptions are rare, so simply predicting no exceptions is very accurate!
- Check prediction mechanism
  - Exceptions detected at end of instruction execution pipeline, special hardware for various exception types
- Recovery mechanism
  - Only write architectural state at commit 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



Resource Usage

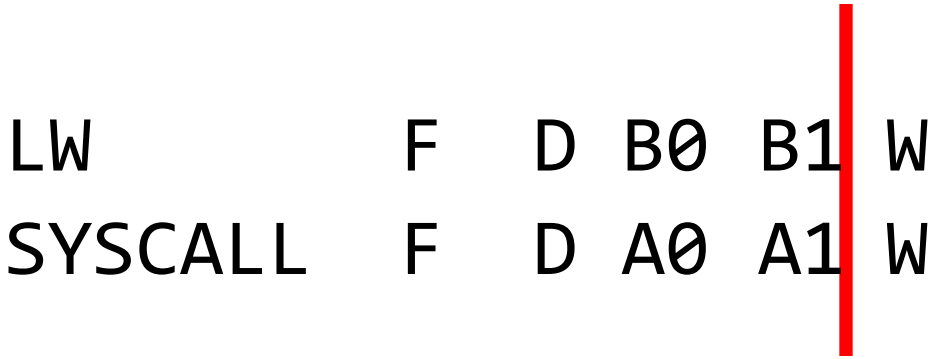


Agenda

- Instruction-Level Parallelism
- Fetch Logic and Alignment
- Interrupts and Exceptions
- Interrupts and Bypassing

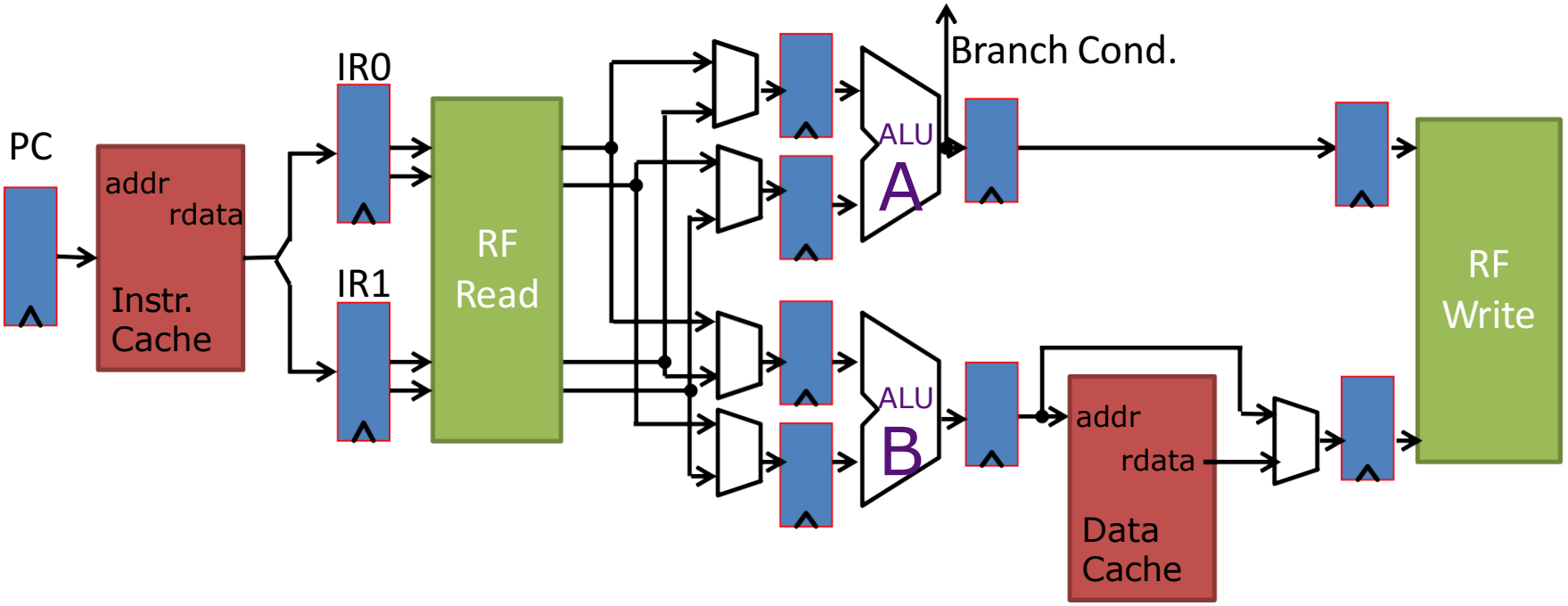
Precise Exceptions and Superscalars

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

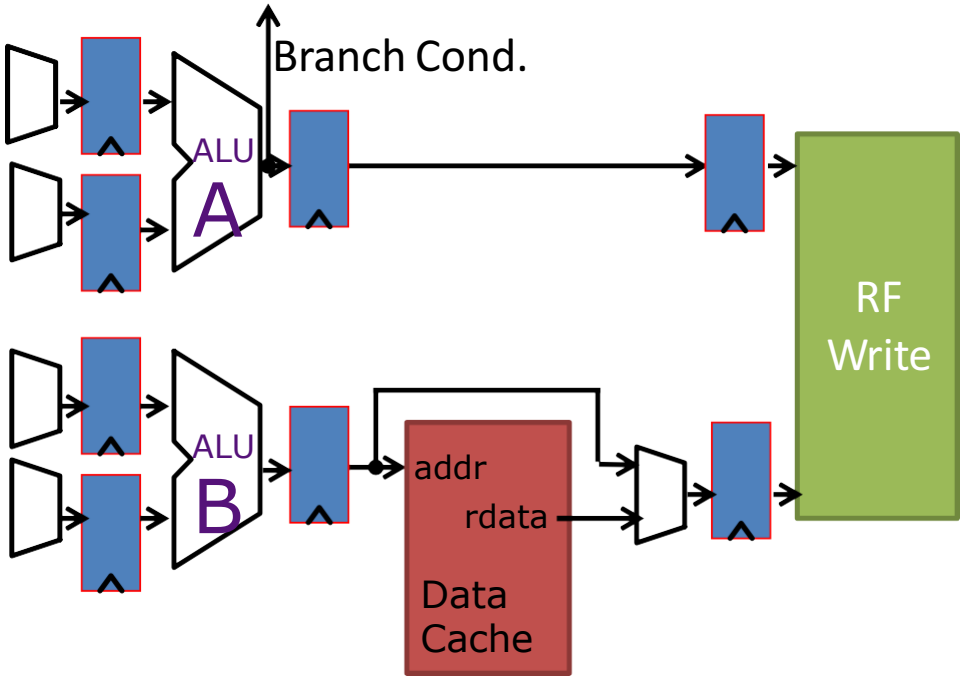


LW is in B pipeline, but commits first in logical order!

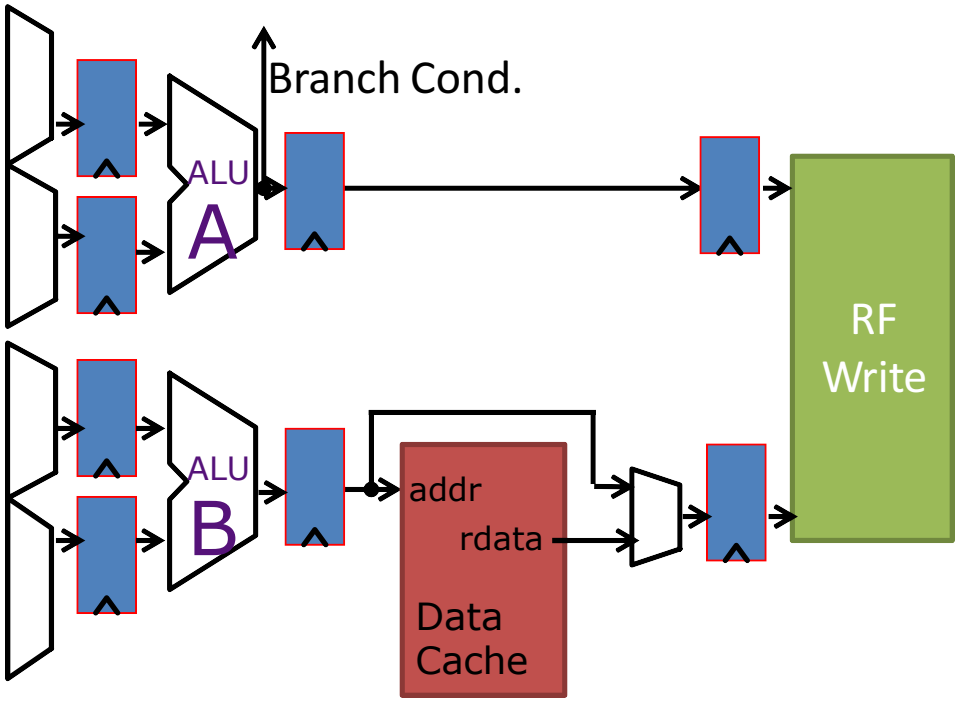
Bypassing in Superscalar Pipelines



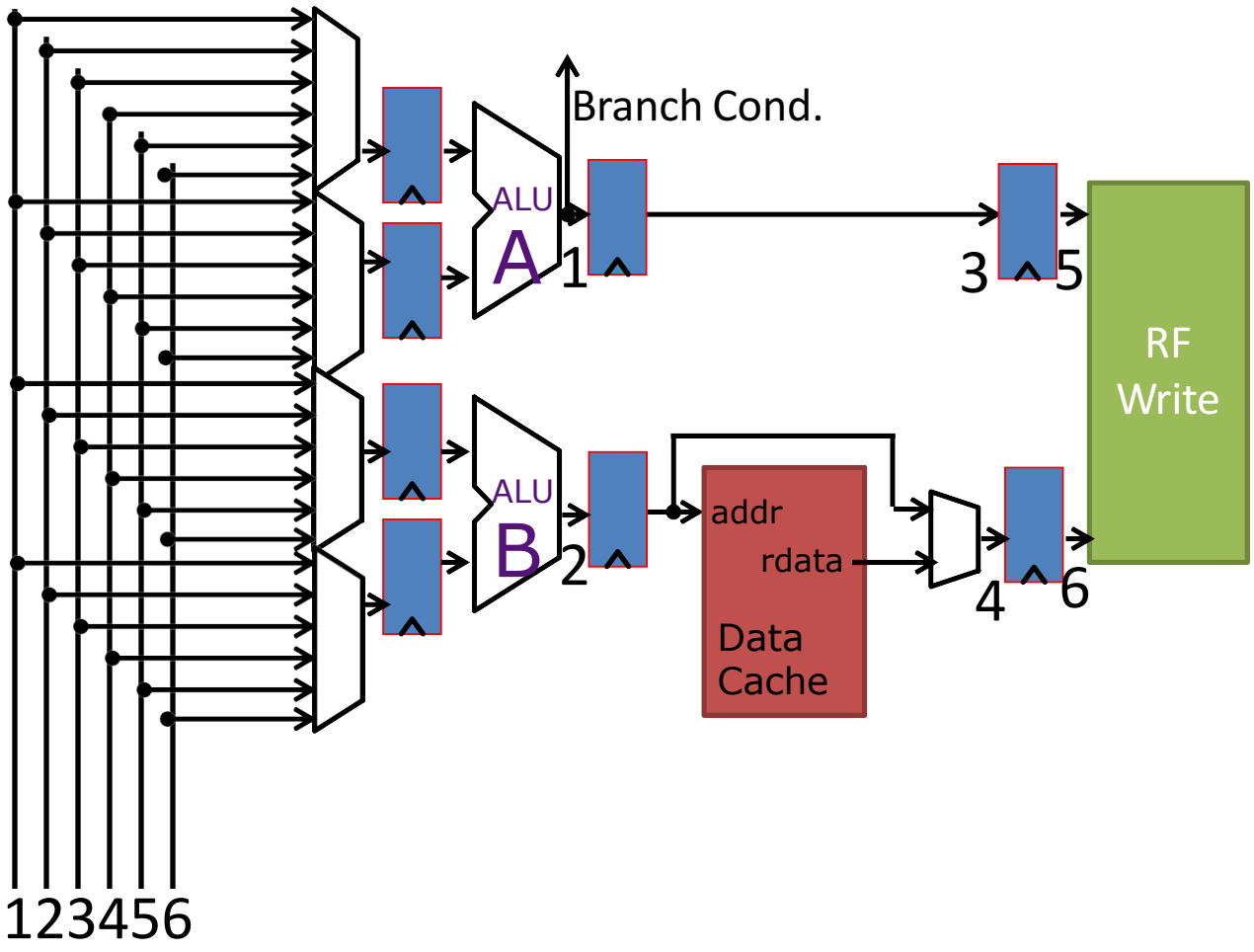
Bypassing in Superscalar Pipelines



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
```

Superscalars Multiply Branch Cost

```
BEQZ   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  A0 A1 W
OpI                   F  D  I  B0 B1 W
```

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)

- MIT material derived from course 6.823
- UCB material derived from course CS252 & CS152
- Cornell material derived from course ECE 4750