

Precise Exceptions and Out-of-Order Execution

Samira Khan

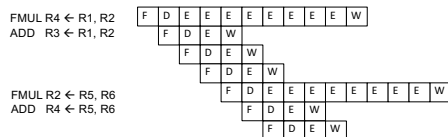
Multi-Cycle Execution

- Not all instructions take the same amount of time for "execution"
- Idea: **Have multiple different functional units that take different number of cycles**
 - Can be pipelined or not pipelined
 - Can let independent instructions start execution on a different functional unit before a previous long-latency instruction finishes execution

2

ISSUES IN PIPELINING: MULTI-CYCLE EXECUTE

- Instructions can take different number of cycles in EXECUTE stage
 - Integer ADD versus FP Multiply



- What is wrong with this picture?
 - What if FMUL incurs an exception?
 - Sequential semantics of the ISA NOT preserved!

3

The Von Neumann Model/Architecture

- Also called *stored program computer* (instructions in memory). Two key properties:
 - **Stored program**
 - Instructions stored in a linear memory array
 - **Memory is unified** between instructions and data
 - The interpretation of a stored value depends on the control signals
 - **Sequential instruction processing**
 - One instruction processed (fetched, executed, and completed) at a time
 - **Program counter (instruction pointer)** identifies the current instr.
 - **Program counter is advanced sequentially** except for control transfer instructions

4

HANDLING EXCEPTIONS IN PIPELINING

- Exceptions versus interrupts
- Cause
 - Exceptions: internal to the running thread
 - Interrupts: external to the running thread
- When to Handle
 - Exceptions: when detected (and known to be non-speculative)
 - Interrupts: when convenient
 - Except for very high priority ones
 - Power failure
 - Machine check
- Priority: process (exception), depends (interrupt)
- Handling Context: process (exception), system (interrupt)

5

PRECISE EXCEPTIONS/INTERRUPTS

- The architectural state should be consistent when the exception/interrupt is ready to be handled
1. All previous instructions should be completely retired.
 2. No later instruction should be retired.
- Retire = commit = finish execution and update arch. state

6

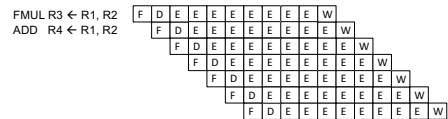
WHY DO WE WANT PRECISE EXCEPTIONS?

- Aid software debugging
- Enable (easy) recovery from exceptions, e.g. page faults
- Enable (easily) restartable processes

7

ENSURING PRECISE EXCEPTIONS IN PIPELINING

- Idea: Make each operation take the same amount of time

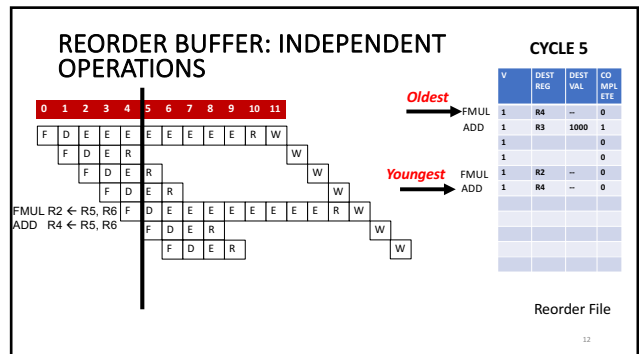
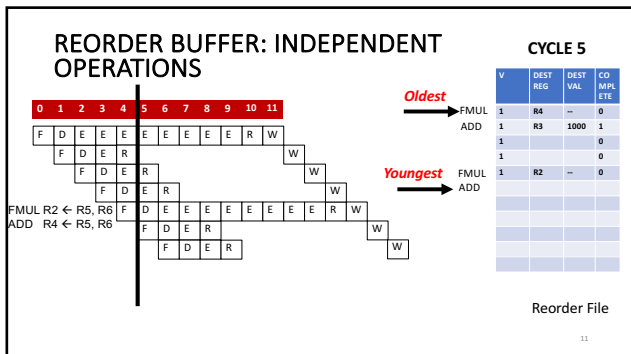
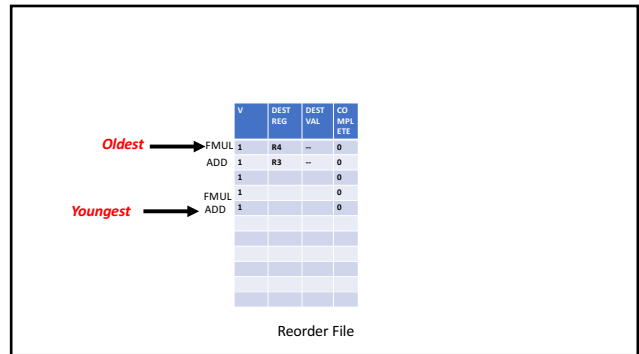
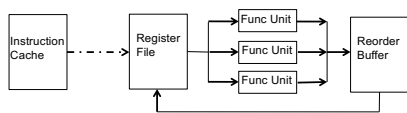


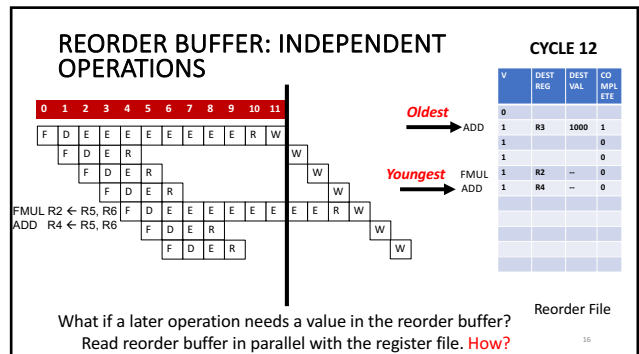
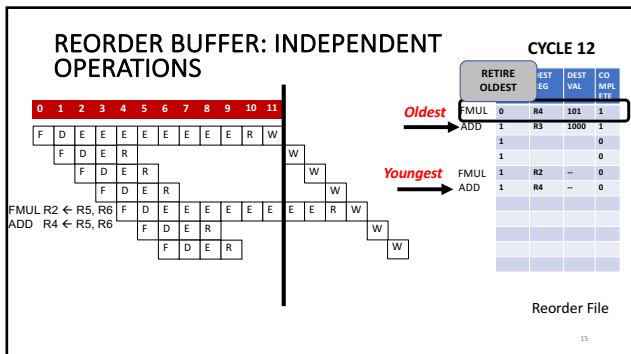
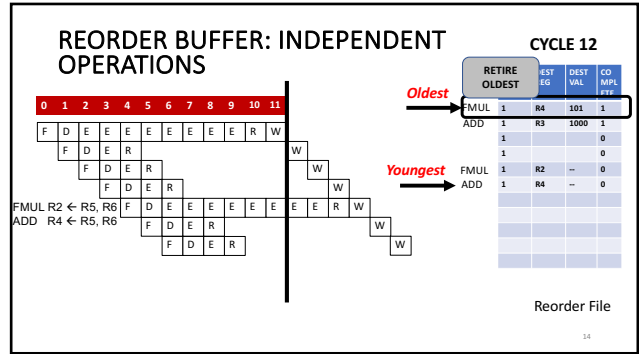
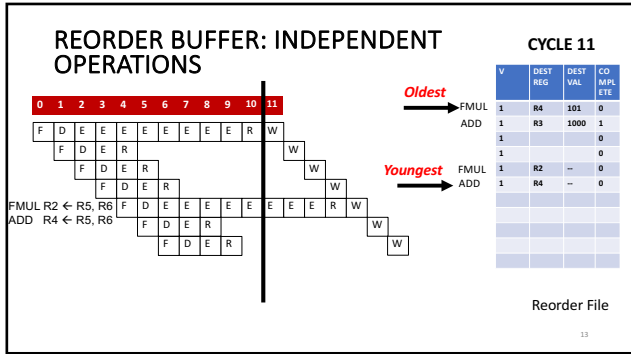
- Downside
 - What about memory operations?
 - Each functional unit takes 500 cycles?

8

SOLUTION: REORDER BUFFER (ROB)

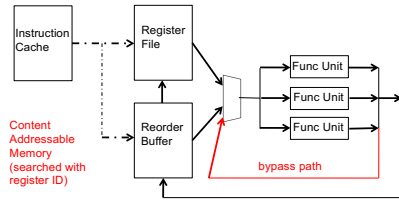
- Idea: Complete instructions out-of-order, but reorder them before making results visible to architectural state
- When instruction is decoded it reserves an entry in the ROB
- When instruction completes, it writes result into ROB entry
- When instruction oldest in ROB and it has completed, its result moved to reg. file or memory





REORDER BUFFER: HOW TO ACCESS?

- A register value can be in the register file, reorder buffer, (or bypass paths)



17

Search for Register Value

	VAL	V
R1	1	1
R2		0
R3		0
R4		0
R5	5	1
R6	6	1
R7	8	1
R8	8	1
R9	9	1
R10	10	1
R11	11	0

	V	DEST REG	DEST VAL	CO MPL ETC
	0			
→ Oldest	1	R3	1000	1
	1			0
	1			0
→ Youngest	1	R2	--	0
	1	R4	--	0

SIMPLIFYING REORDER BUFFER ACCESS

- Idea: Use indirection
- Access register file first
 - If register not valid, register file stores the ID of the reorder buffer entry that contains (or will contain) the value of the register
 - Mapping of the register to a ROB entry
- Access reorder buffer next
- What is in a reorder buffer entry?

V	DestRegID	DestRegVal	StoreAddr	StoreData	BranchTarget	PC/IP	Control/valid bits
---	-----------	------------	-----------	-----------	--------------	-------	--------------------

- Can it be simplified further?

19

Search for Register Value

	VAL	REG	V
R1	1		1
R2		5	0
R3		2	0
R4		6	0
R5	5		1
R6	6		1
R7	8		1
R8	8		1
R9	9		1
R10	10		1
R11	11		1

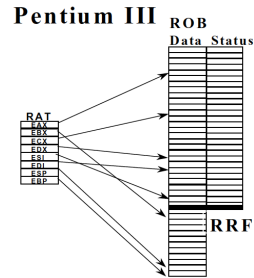
	V	DEST REG	DEST VAL	CO MPL ETC
	0			
→ Oldest	1	R3	1000	1
	1			0
	1			0
→ Youngest	1	R2	--	0
	1	R4	--	0

REORDER BUFFER PROS AND CONS

- Pro
 - Conceptually simple for supporting precise exceptions
- Con
 - Reorder buffer needs to be accessed to get the results that are yet to be written to the register file
 - CAM or indirection → increased latency and complexity

23

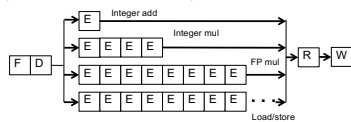
Reorder Buffer in Intel Pentium III



Boggs et al., "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2001.

In-Order Pipeline with Reorder Buffer

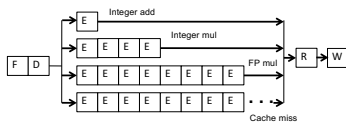
- **Decode (D):** Access regfile/ROB, allocate entry in ROB, check if instruction can execute, if so **dispatch** instruction
- **Execute (E):** Instructions can complete out-of-order
- **Completion (R):** Write result to **reorder buffer**
- **Retirement/Commit (W):** Check for exceptions; if none, write result to architectural register file or memory; else, flush pipeline and start from exception handler
- **In-order dispatch/execution, out-of-order completion, in-order retirement**



23

Out-of-Order Execution (Dynamic Instruction Scheduling)

AN IN-ORDER PIPELINE



- Problem: A true data dependency stalls dispatch of younger instructions into functional (execution) units
- Dispatch: Act of sending an instruction to a functional unit

25

CAN WE DO BETTER?

- What do the following two pieces of code have in common (with respect to execution in the previous design)?

```

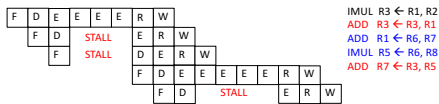
IMUL R3 ← R1, R2      LD R3 ← R1 (0)
ADD R3 ← R3, R1       ADD R3 ← R3, R1
ADD R1 ← R6, R7       ADD R1 ← R6, R7
IMUL R5 ← R6, R8      IMUL R5 ← R6, R8
ADD R7 ← R9, R9       ADD R7 ← R9, R9
    
```

- Answer: First ADD stalls the whole pipeline!
 - ADD cannot dispatch because its source registers unavailable
 - Later independent instructions cannot get executed
- How are the above code portions different?
 - Answer: Load latency is variable (unknown until runtime)
 - What does this affect? Think compiler vs. microarchitecture

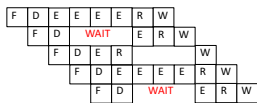
26

IN-ORDER VS. OUT-OF-ORDER DISPATCH

- In order dispatch + precise exceptions:



- Out-of-order dispatch + precise exceptions:



- 16 vs. 12 cycles

27

PREVENTING DISPATCH STALLS

- Any way to prevent dispatch stalls?
 - Dataflow: fetch and "fire" an instruction when its inputs are ready
 - Problem: in-order dispatch (scheduling, or execution)
 - Solution: out-of-order dispatch (scheduling, or execution)

28

TOMASULO'S ALGORITHM

- OoO with register renaming invented by Robert Tomasulo
 - Used in IBM 360/91 Floating Point Units
 - Read: Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," IBM Journal of R&D, Jan. 1967.
- What is the major difference today?
 - Precise exceptions: IBM 360/91 did NOT have this
 - Patt, Hwu, Shebanow, "HPS, a new microarchitecture: rationale and introduction," MICRO 1985.
 - Patt et al., "Critical issues regarding HPS, a high performance microarchitecture," MICRO 1985.
- Variants are used in most high-performance processors
 - Initially in Intel Pentium Pro, AMD K5
 - Alpha 21264, MIPS R10000, IBM POWER5, IBM z196, Oracle UltraSPARC T4, ARM Cortex A15

29

- These slides are not covered in the class
- These are for students who want to know more

- What is the insight of OOO execution?

OUT-OF-ORDER EXECUTION (DYNAMIC SCHEDULING)

- Idea: Move the dependent instructions out of the way of independent ones (s.t. independent ones can execute)
 - Rest areas for dependent instructions: Reservation stations
- Monitor the source "values" of each instruction in the resting area
- When all source "values" of an instruction are available, "fire" (i.e. dispatch) the instruction
 - Instructions dispatched in *dataflow* (not control-flow) order
- Benefit:
 - *Latency tolerance*: Allows independent instructions to execute and complete in the presence of a long latency operation

32

The Von Neumann Model/Architecture

- Also called *stored program computer* (instructions in memory). Two key properties:
 - **Memory is unified** between instructions and data
 - The interpretation of a stored value depends on the control signals
 - When is a value interpreted as an instruction?
- **Sequential instruction processing**
 - One instruction processed (fetched, executed, and completed) at a time
 - **Program counter (instruction pointer)** identifies the current instr.
 - **Program counter is advanced sequentially** except for control transfer instructions

33

The Dataflow Model (of a Computer)

- Von Neumann model: An instruction is fetched and executed in **control flow order**
 - As specified by the **instruction pointer**
 - Sequential unless explicit control flow instruction
- Dataflow model: An instruction is fetched and executed in **data flow order**
 - i.e., when its operands are ready
 - i.e., there is **no instruction pointer**
 - Instruction ordering specified by data flow dependence
 - Each instruction specifies "who" should receive the result
 - An instruction can "fire" whenever all operands are received
 - Potentially many instructions can execute at the same time
 - Inherently more parallel!

34

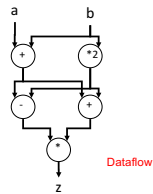
Von Neumann vs Dataflow

- Consider a Von Neumann program
 - What is the significance of the program order?
 - What is the significance of the storage locations?

```

v <= a + b;
w <= b * z;
x <= v - w;
y <= v + w;
z <= x * y
    
```

Sequential



Dataflow

- Which model is more natural to you as a programmer?

35

Data Flow Advantages/Disadvantages

- Advantages
 - Very good at exploiting **irregular parallelism**
 - Only real dependencies constrain processing
- Disadvantages
 - Debugging difficult (no precise state)
 - Interrupt/exception handling is difficult (what is precise state semantics?)
 - Too much parallelism? (Parallelism control needed)
 - High bookkeeping overhead (tag matching, data storage)
 - Memory locality is not exploited

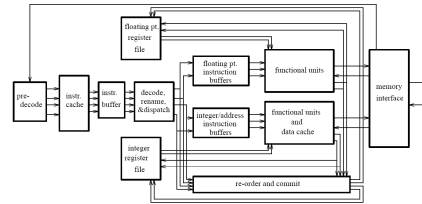
36

OOO EXECUTION: RESTRICTED DATAFLOW

- An out-of-order engine dynamically builds the dataflow graph of a piece of the program
 - which piece?
- The dataflow graph is limited to the instruction window
 - Instruction window: all decoded but not yet retired instructions
- Can we do it for the whole program?
- Why would we like to?
- In other words, how can we have a large instruction window?

37

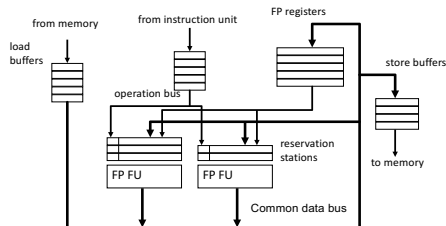
GENERAL ORGANIZATION OF AN OOO PROCESSOR



• Smith and Sohi, "The Microarchitecture of Superscalar Processors," Proc. IEEE, Dec. 1995.

38

TOMASULO'S MACHINE: IBM 360/91



39