

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Computer Architecture
ECE 568

Part 8

Exceptions

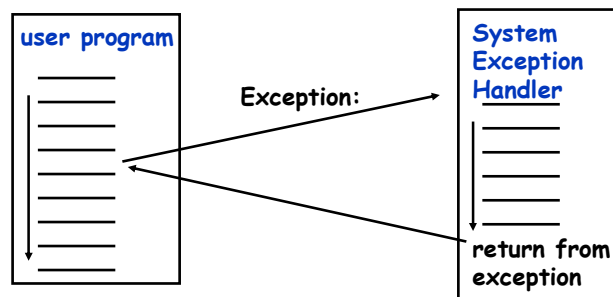
Israel Koren
Fall 2009

ECE568/Koren Part.8 .1

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Exceptions - Basics



normal control flow:
sequential, jumps, branches, calls, returns

- ◆ **Exception = unprogrammed control transfer**
 - system takes action to handle the exception
 - » must record the address of the offending instruction
 - » record any other information necessary to return afterwards
 - returns control to user
 - must save & restore user state

ECE568/Koren Part.8 .2

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Two Types of Exceptions

◆ Interrupts

- caused by external events:
 - » Network, Keyboard, Disk I/O, Timer
- asynchronous to program execution
 - » Most interrupts can be disabled for brief periods of time
- may be handled between instructions
- simply suspend and resume user program

◆ Traps

- caused by internal events
 - » exceptional conditions (overflow)
 - » errors (parity)
 - » page faults (non-resident page)
- synchronous to program execution
- condition must be remedied by the handler
- instruction may be retried and program continued or program may be aborted

Exceptions - Examples

Exception type	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminate
I/O device request	Asynchronous	Coerced	Nonmaskable	Between	Resume
Invoke operating system	Synchronous	User request	Nonmaskable	Between	Resume
Tracing instruction execution	Synchronous	User request	User maskable	Between	Resume
Breakpoint	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Resume
Floating-point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Nonmaskable	Within	Resume
Misaligned memory accesses	Synchronous	Coerced	User maskable	Within	Resume
Memory protection violations	Synchronous	Coerced	Nonmaskable	Within	Resume
Using undefined instructions	Synchronous	Coerced	Nonmaskable	Within	Terminate
Hardware malfunctions	Asynchronous	Coerced	Nonmaskable	Within	Terminate
Power failure	Asynchronous	Coerced	Nonmaskable	Within	Terminate

Exceptions in MIPS pipeline

<i>Stage</i>	<i>Possible exceptions</i>
IF	Page fault on instruction fetch; misaligned memory access; memory-protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch; misaligned memory access; memory-protection violation; memory error

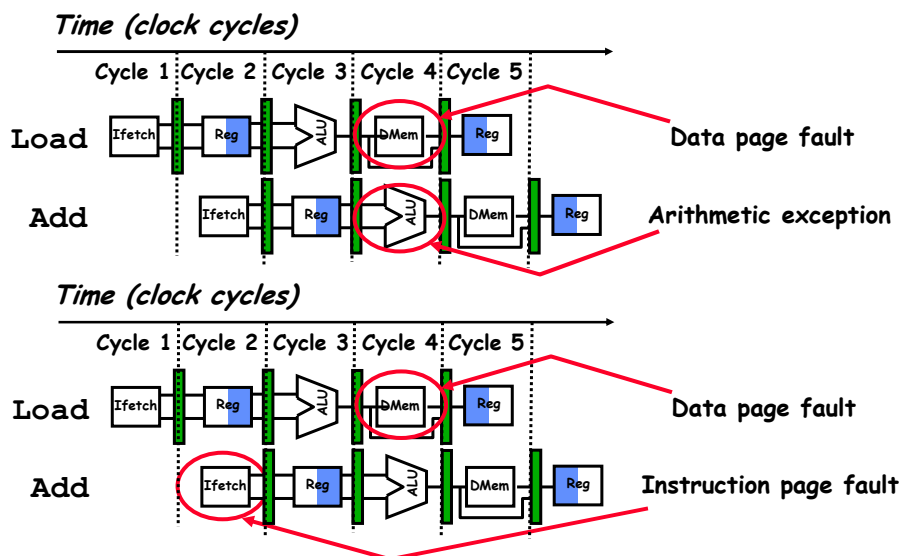
- ◆ How do we stop the pipeline? How do we restart it?
- ◆ Do we interrupt immediately or wait?
- ◆ 5 instructions, executing in 5 different pipeline stages!
 - Who caused the interrupt?

ECE568/Koren Part.8 .5

Adapted from Patterson, Katz and Kubiatiowicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Multiple exceptions



ECE568/Koren Part.8 .6

Adapted from Patterson, Katz and Kubiatiowicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Precise Interrupts/Exceptions

- ◆ Exceptions should be **Precise** or clean, i.e., the outcome should be exactly the same as in a non-pipelined machine
- ◆ **Precise** \Rightarrow state of the machine is preserved as if program executed up to the offending instruction
 - All previous instructions **completed**
 - Offending instruction and all following instructions act **as if they have not even started**
 - Same code will work on different processor implementations
 - Difficult in the presence of pipelining, out-of-order execution, ...
- ◆ **Imprecise** \Rightarrow system software has to figure out what is where and put it all back together
- ◆ Modern techniques for out-of-order execution and branch prediction help implement precise interrupts

ECE568/Koren Part.8.7

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Relationship between precise interrupts and speculation

- ◆ **Speculation: guess and check**
- ◆ **Important for branch prediction:**
 - Need to "take our best shot" at predicting branch direction
- ◆ **If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly:**
 - This is exactly the same as precise exceptions!
- ◆ **Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit***

ECE568/Koren Part.8.8

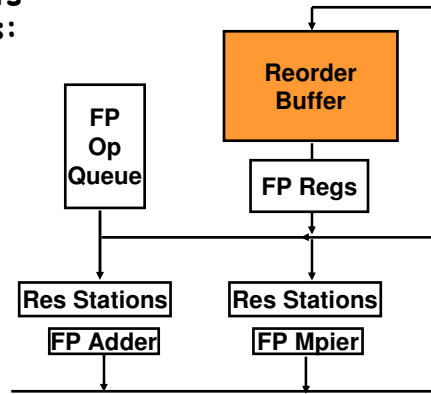
Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

HW support for precise interrupts

◆ Need HW buffer for results of uncommitted instructions:
reorder buffer

- An instruction commits when it completes its execution and all its predecessors have already committed
- Once instruction commits, result is put into register
- Therefore, easy to undo speculated instructions on mispredicted branches or *exceptions*
- Supplies operands between execution complete & commit

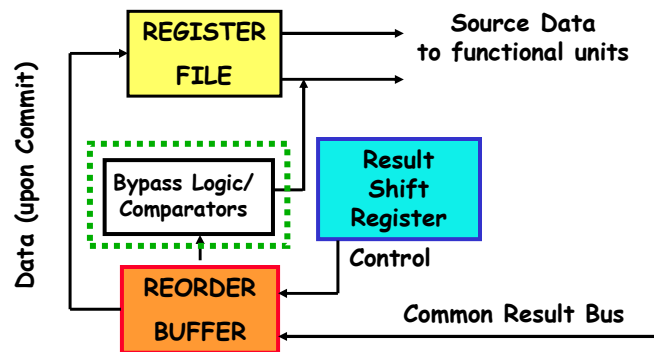


ECE568/Koren Part.8 .9

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Reorder Buffer



J. Smith & A. Pleszkun, IEEETC, May 1988 (available on the web page)

ECE568/Koren Part.8 .10

Copyright 2008 Koren UMass

Result Shift Register

PC	Instruction	Ex_Time (in FU)
6	ADDF F10,F1,F3	6
7	ADD R9,R2,R5	②

Stage	Functional unit source	Valid instr.	Tag
1		0	
②	Integer ADD	1	5
3		0	
4		0	
5	Flt. Pt. ADD	1	4
N		0	

Direction of movement ↑

Result Shift Register

Entry #	Dest. Reg.	Result	Exceptions	Valid Rslt	PC
3					
4	10			0	6
5	9			0	7
6					

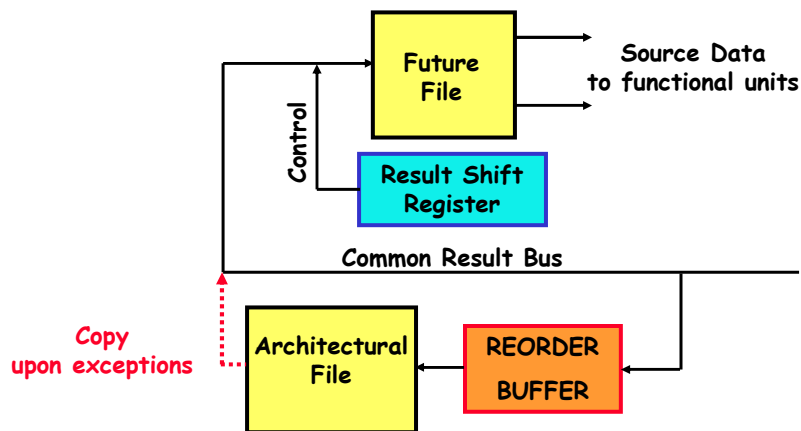
Head →

Tail →

Reorder (circular) Buffer

ECE568/Koren Part.8 .11

Future File



ECE568/Koren Part.8 .12

Copyright 2008 Koren UMass

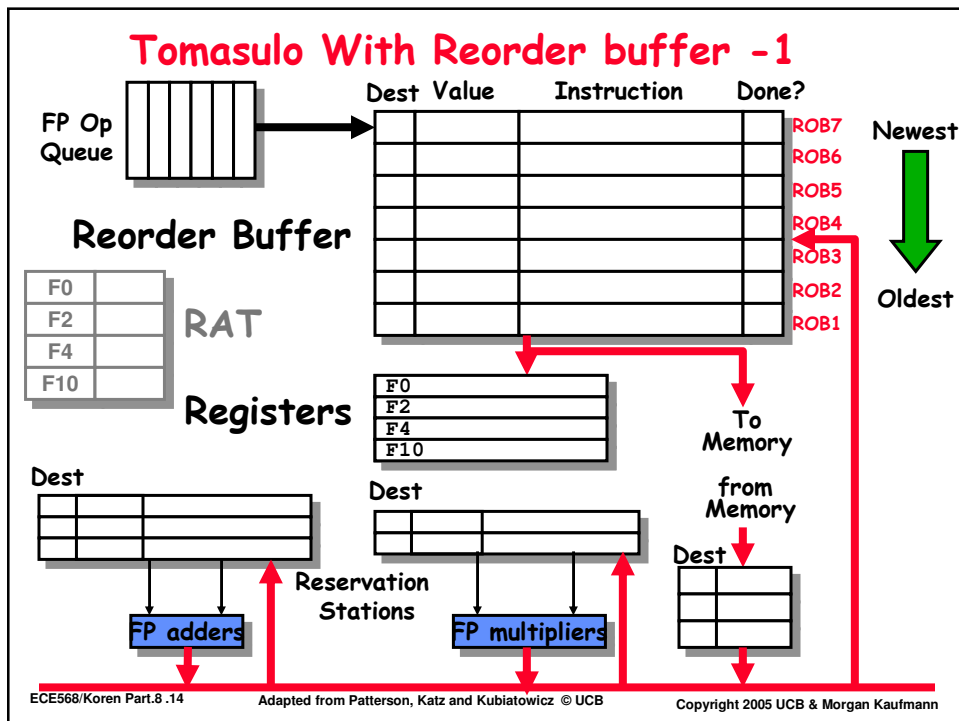
Four Steps of Speculative Tomasulo Algorithm

1. **Issue**— get instruction from FP Op Queue
 If reservation station, reorder buffer slot, and result shift register slot free, issue instr & send operands & reorder buffer no. for destination. (this stage sometimes called "dispatch")
2. **Execution**— operate on operands (EX)
 When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; this takes care of RAW. (sometimes called "issue")
3. **Write result**— finish execution (WB)
 Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available
4. **Commit**— update register with result from reorder buffer
 When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer. (sometimes called "graduation")

ECE568/Koren Part.8 .13

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann



Code Example

1. LD **F0**, 10(R2)
2. ADDD **F10**, **F4**, **F0**
3. DIVD F2, **F10**, F6
4. BNE F2, <...>
5. LD **F4**, 0(R3)
6. ADDD **F0**, **F4**, F6
7. ADDD **F0**, **F4**, F6

ECE568/Koren Part.8 .15

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

