Credits: Dr. Asanovic (MIT), Dr. Yousif (Intel), Dr. Park (Virginia Tech), Dr. Patterson (UCB)

Advanced Pipelining: Precise Exceptions

Handout 09

October 14, 2004

Shoukat Ali

shoukat@umr.edu

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

1

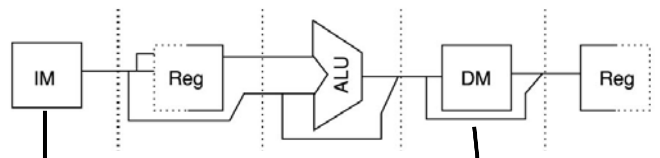
Interrupts

- also called traps, faults, exceptions
- it is a **signal** to the CPU to interrupt instruction stream processing
- who can **raise** an interrupt?

2

Who Can Raise an Interrupt?

- IF and MEM stages



memory raises an interrupt if:

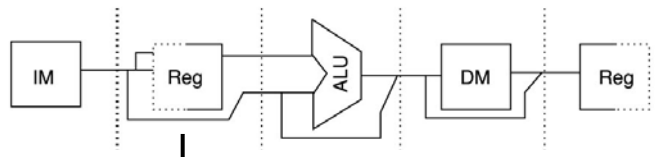
1) the requested block is not in memory, called a “page fault”

2) memory access is misaligned

3) access is to a protected area of memory

Who Can Raise an Interrupt?

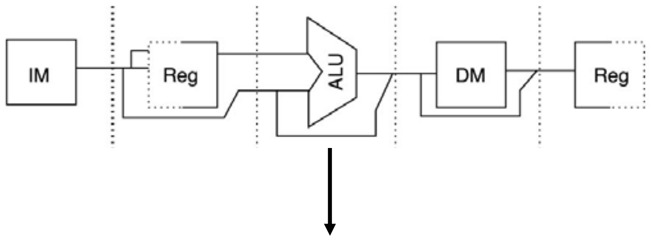
- ID stage



decoding logic raises an interrupt if instruction is unimplemented, i.e., if instruction opcode (or funct) are not recognized

Who Can Raise an Interrupt?

- EXE stage

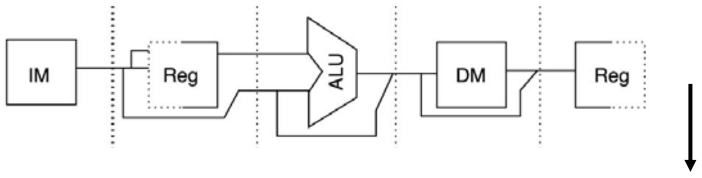


ALU unit raises an interrupt if the result overflows

5

Who Can Raise an Interrupt?

- control unit, AFTER an instruction completes AND if a “breakpoint” is set



breakpoint, requested by user for debugging

6

Who Can Raise an Interrupt?

- I/O device, e.g., at a printer jam
- power unit, e.g., at a power failure

7

Interrupts – Some Types

- synchronous: an interrupt that occurs at the same instant whenever the program is executed with same data and memory allocation
  - an overflow interrupt, page fault
- asynchronous: one that does not depend on the CPU and memory
  - caused by external devices, printers, keyboards, mice

8

Interrupts – Some Types

- within: an interrupt that occurs in the middle of an instruction, and must be serviced before the instruction can be completed
  - instruction must be stopped and restarted after servicing interrupt
  - offending instruction: an instruction that raises an interrupt
- between: one that can be serviced after completing the current instruction, i.e., between instructions (printer jam)
- other types exist, page A-41

9

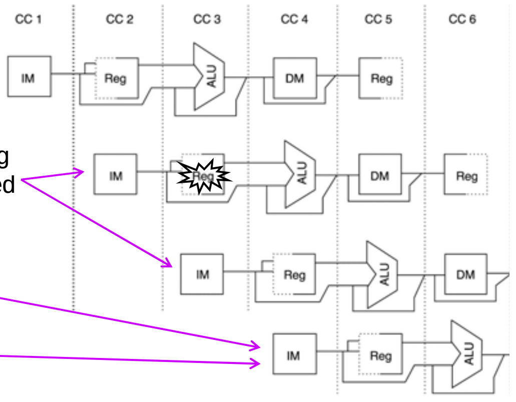
What to Do When An Interrupt is Raised?

4. let all instructions before offending instruction complete

3. turn off all writes for the offending instruction and the instructions issued after it

1. stop fetching further instructions from the program

2. force a “trap” instruction into the pipeline on the next IF



- trap instruction will change the PC so that an “interrupt service routine (ISR)” can be started
  - ISR will save PC of the offending inst so that it can be restarted
  - ISR will service interrupt
  - ISR will reload the addr of offending instruction into the PC

10

Precise Interrupts

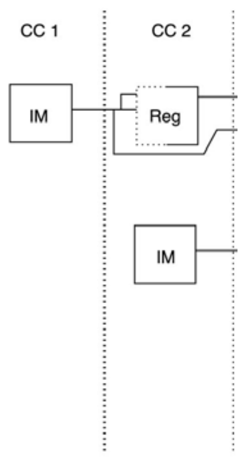
- **precise interrupt:** an interrupt that can be handled in the manner described on the previous slide
  - that is, the program is resumed after the interrupt “as if interrupt never happened”
  - the processor state after the interrupt service is exactly as it was before the offending instruction

11

Pipelining Makes Interrupts Servicing Harder

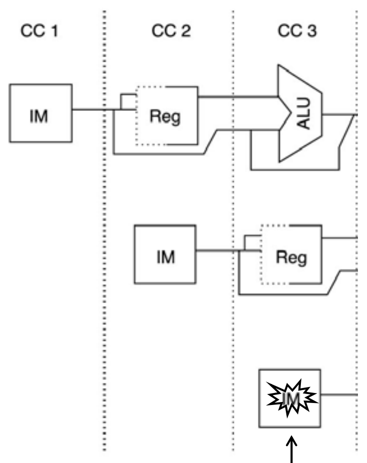
12

Pipelining Makes Interrupts Servicing Harder



13

Pipelining Makes Interrupts Servicing Harder

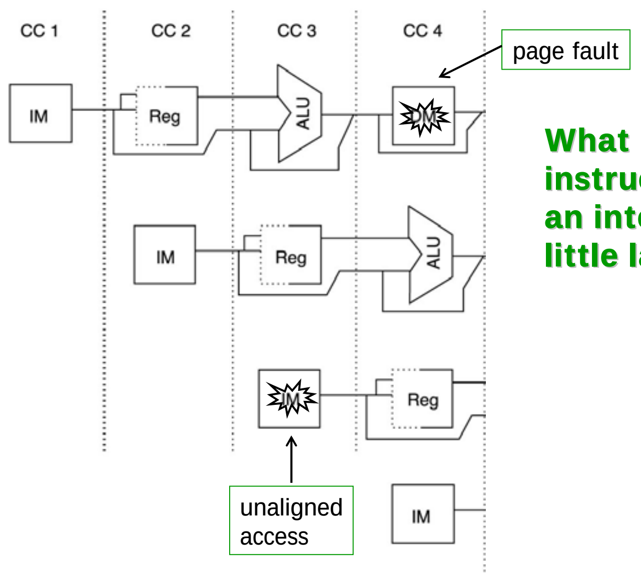


unaligned access

Should we service this interrupt?

14

Pipelining Makes Interrupts Servicing Harder



What if an earlier instruction raises an interrupt a little later?

15

Pipelining Makes Interrupts Servicing Harder

- key idea: there can be **multiple interrupts** raised in different cycles
- so which interrupt should be serviced first?
    - *first raised, first served*: should the system service interrupts in the order they are raised?
    - *earliest instruction interrupt first*: should the system service interrupts from earlier instruction first?

16

Interrupts Must Be Serviced In Un-Pipelined Order

- *definition:* un-pipelined order is the order in which instructions would execute if there were no pipelining

- interrupts must be serviced in the un-pipelined order
  - *all exceptions on inst x MUST be serviced before any exception on instr (x+1) is serviced*

- system should not service interrupts on the *first raised, first served* basis
  - *first raised, first served* will lead to interrupts being serviced out of *un-pipelined order*

17

Ensuring Interrupts Are Serviced In Unpipelined Order

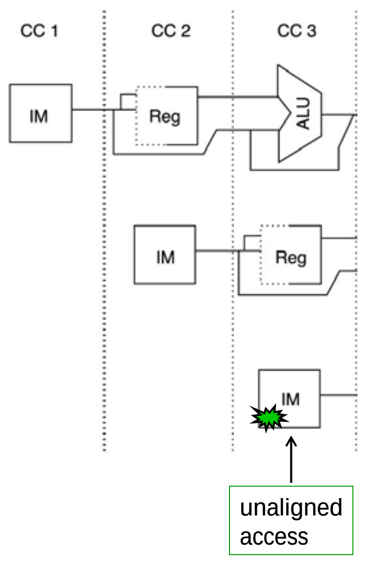
- solution: CPU services instruction x's interrupt only when x is in its WB stage
  - if any stage of x raises an interrupt
    - x does not notify the CPU immediately
    - instead, x POSTS the message "instruction x needs service" in the next pipeline register

  - this *posting* travels to stage 5

  - every clock cycle, CPU checks pipeline register MEM/WB to see if any request for interrupt service has been posted
    - if yes, it performs steps given on slide "Stopping and Restarting a Pipeline"
    - if no, instruction is *committed*, i.e., it can write register file

18

Servicing Interrupts Only In WB Stage



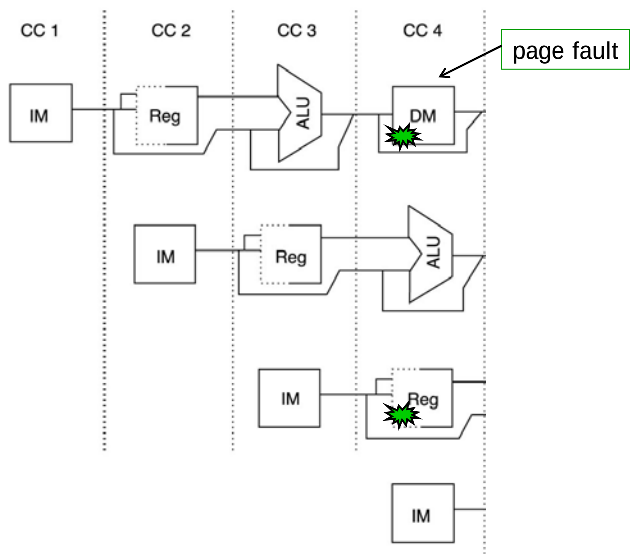
Should we service this interrupt?

No. just post it.

Posting shown here with a "small cornered flash."

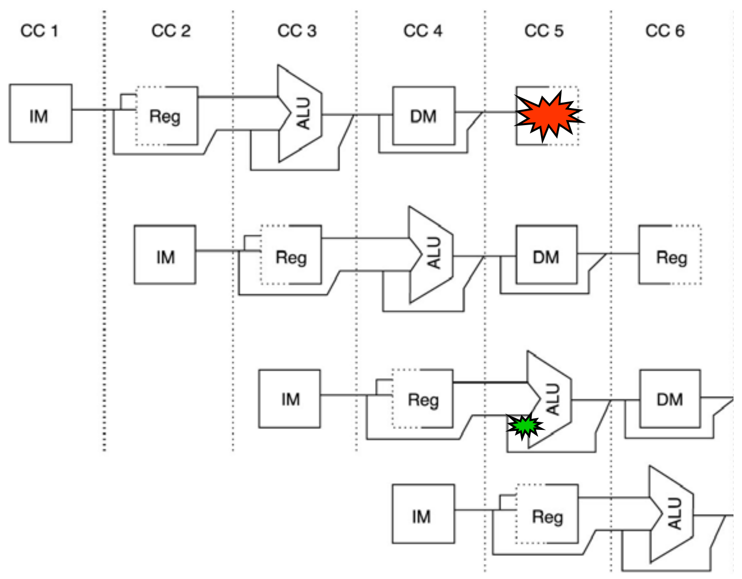
19

Servicing Interrupts Only In WB Stage



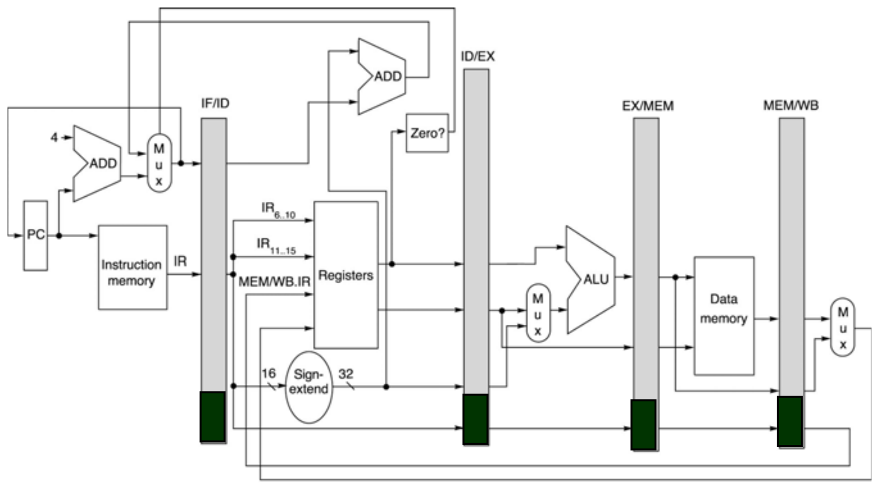
20

Servicing Interrupts Only In WB Stage



21

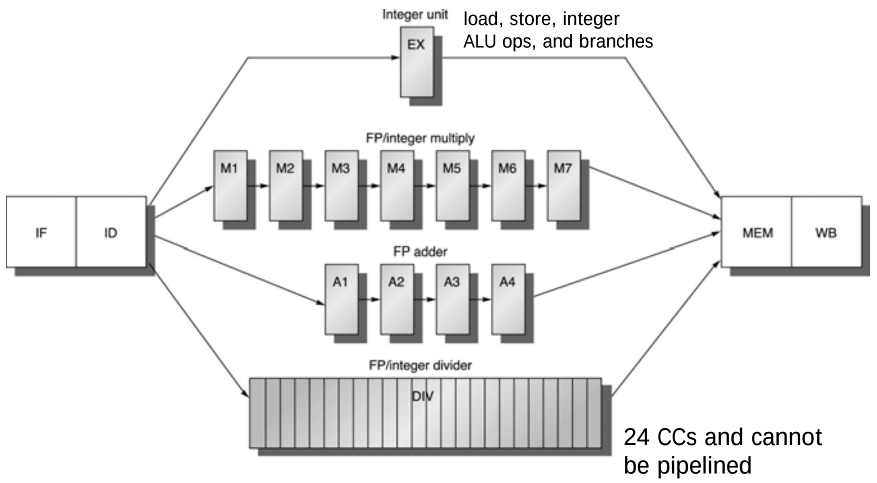
Pipeline Registers Must Carry Interrupt Postings



22

MIPS Floating Point Pipeline

- key fact: the ***execution stage*** for a FP operation is longer than that for an integer operation



24 CCs and cannot be pipelined

Interrupt Problems: MIPS FP Pipeline

- instructions can now complete out of order
- "service in WB" does not guarantee precise interrupts any more!
- example

mul	**IF**	**ID**	**EX1**	**EX2**	**EX3**	**EX4**	**EX5**	**EX6**	**EX7**	**MEM**	**WB**
add		**IF**	**ID**	**EX1**	**EX2**	**EX3**	**EX4**	**MEM**	**WB**		

- what if mul raises an exception after the add completed?
  - by the time we stop the pipeline, add has already modified processor state (i.e., registers)
  - state after restarting of mul will not be same as before mul
- solution: maintain a set of extra registers called "future file"
  - if x completes before an earlier instruction, x's result is written in future file
  - when all of x's predecessors complete, x is committed