Precise Interrupts

"unobserved system can exist in any intermediate state, upon observation system collapses to well-defined state"

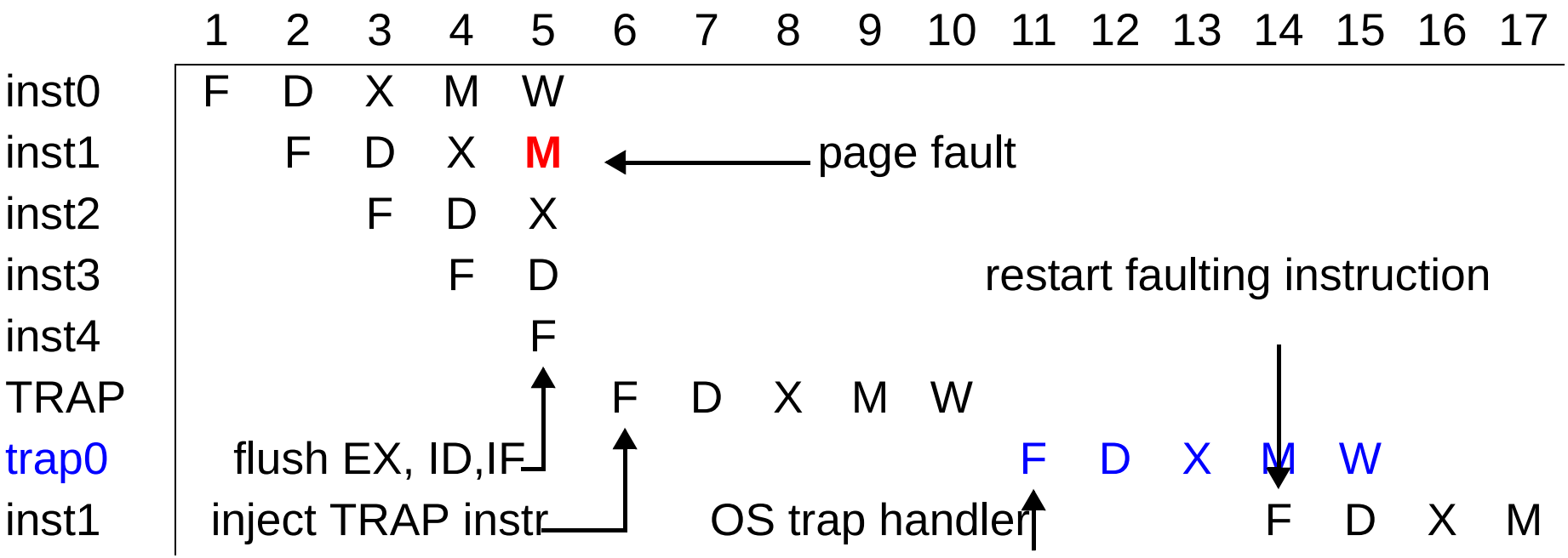
–2nd postulate of quantum mechanics

- system ⇒ processor, observation ⇒ interrupt

what is the "well-defined" state?

- von Neumann: "sequential, instruction atomic execution"
- precise state at interrupt
  - all instructions older than interrupt are complete
  - all instructions younger than interrupt haven't started
- implies interrupts are taken in program order
- necessary for VM (why?), "highly recommended" by IEEE

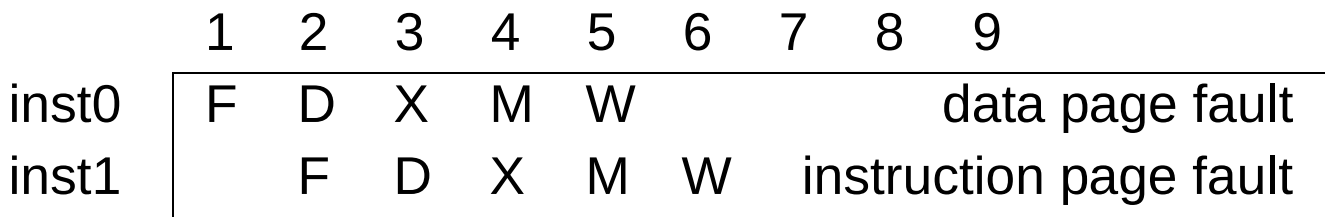
Interrupt Example: Data Page Fault



- squash (effects of) younger instructions
- inject fake TRAP instruction into IF
- from here, like a SYSCALL

More Interrupts

- interrupts can occur at different stages
  - IF, MEM: page fault, misaligned data, protection violation
  - ID: illegal/privileged instruction
  - EX: arithmetic exception



- too complicated to draw what goes on here
  - cycle2: instruction page fault, flush inst1, inject TRAP
  - c4: data page fault, flush inst0, inst1, TRAP
  - – can get into an infinite loop here (with help of OS page placement)

Posted Interrupts

posted interrupts

- set interrupt bit when condition is raised
- check interrupt bit (potentially "take" interrupt) in WB
  - + interrupts are taken in order
  - – longer latency, more complex

	1	2	3	4	5	6	7	8	9	
inst0	F	D	X	M	W					data page fault
inst1		F	D	X	M	W				instruction page fault

- what happens now?
  - c2: set inst1 bit
  - c4: set inst0 bit
  - c5: take inst0 interrupt

Interrupts and Multi-Cycle Operations

	1	2	3	4	5	6	7	8	9	10	11
`divf f0,f1,f2`	F	D	E/	E/	E/	E/	W	**div by 0 (posted)**			
`mulf f3,f4,f5`		F	D	E*	E*	W					
`addf f6,f7,f8`			F	D	E+	E+	s*	W			

multi-cycle operations + precise state = trouble

- #1: how to undo early writes?
  - e.g., must make it seem as if **`mulf`** hasn't executed
  - undo writes: future file, history file -> ugly!
- #2: how to take interrupts in-order if WB is not in-order?
  - force in-order WB

  – slow

Interrupts Are Nasty

- odd bits of state must be precise (e.g., condition codes)
- delayed branches
  - what if instruction in delay slot takes an interrupt?
- addressing modes with early-writes (e.g., auto-increment)
  - must undo write (e.g., future-file, history-file)
- some machines had precise interrupts only in integer pipe
  - sufficient for implementing VM
  - e.g., VAX/Alpha

Lucky for us, there's a nice, clean way to handle precise state

- We'll see how this is done in a couple of lectures ...

How Do We Pipeline x86?

The x86 ISA has some really nasty instructions - how did Intel ever figure out how to build a pipelined x86 microprocessor?

Solution: at runtime, “crack” x86 instructions (macro-ops) into RISC-like micro-ops

- First used in P6 (Pentium Pro)
- Used in all subsequent x86 processors, including those from AMD

What are the potential challenges for implementing this solution?

Summary

- principles of pipelining
  - pipeline depth: clock rate vs. number of stalls (CPI)
- hazards
  - structural
  - data (RAW, WAR, WAW)
  - control
- multi-cycle operations
  - structural hazards, WAW hazards
- interrupts
  - precise state

next up: dynamic ILP (parts of H&P chapter 2)