CS305
Computer Architecture

Handling Exceptions

Bhaskaran Raman
Room 406, KR Building
Department of CSE, IIT Bombay

http://www.cse.iitb.ac.in/~br

What are Exceptions?

- Exception: unforeseen or unintentional condition during program execution
- Internal exception: cause is within a program
  - E.g. division by 0, arithmetic overflow, misaligned memory, invalid opcode
- External exception: cause is outside program
  - E.g. from I/O devices, from timer
  - Also called interrupts
- Exceptions can be ‘intentional’: to provide ‘features’
  - E.g. breakpoints to debug, virtual memory, copy on write

Handling Exceptions

EPC

Special register (not part of reg-file): takes on value of PC causing exception

Cause

Special register (not part of reg-file) to indicate cause of exception

When exception occurs:
EPC = PC causing exception
Cause: set as per cause
PC = **exception/interrupt handler**
0x8000 0180 for MIPS
0x8000 0080 in SPIM

Can have:
PC = N x (cause code)
Called **vectored interrupts**

Exception/interrupt handler: part of OS

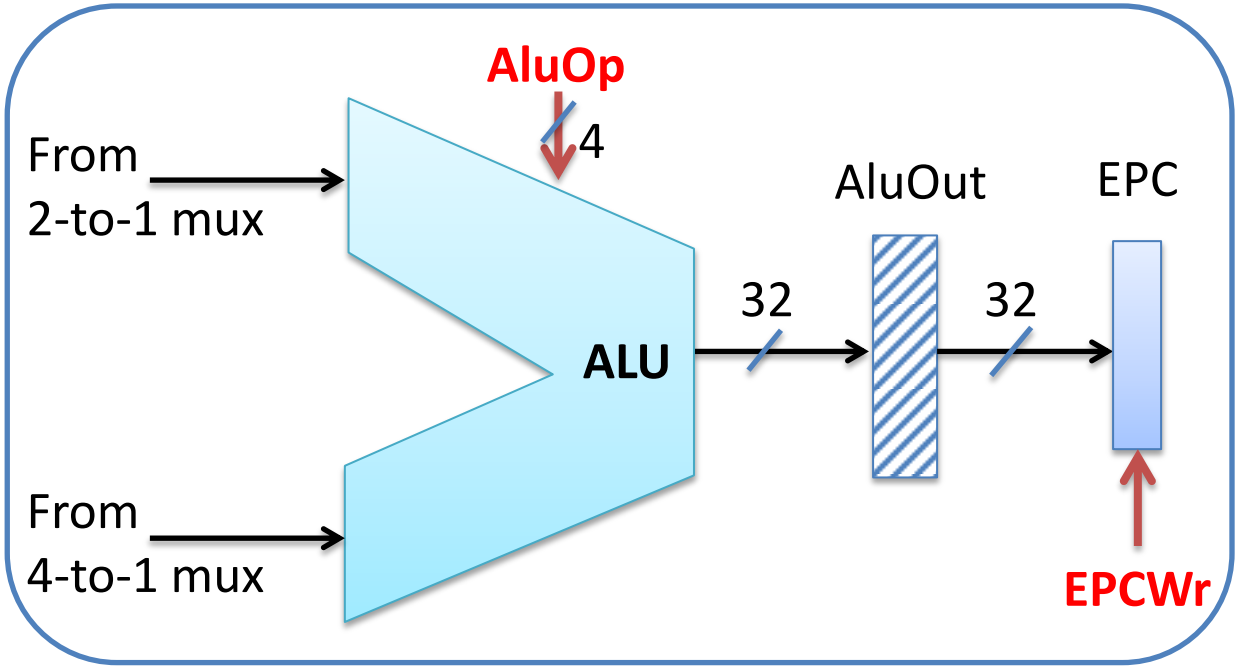
Implementing Exceptions in the Multi-Cycle MIPS

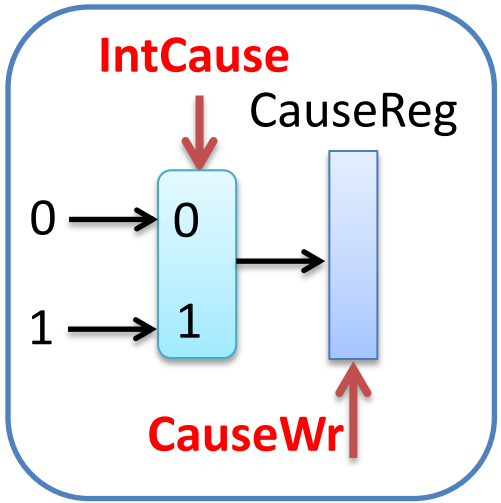
- Consider two kinds of exceptions (for simplicity):
  - Arithmetic overflow: cause=0
  - Unknown opcode: cause=1

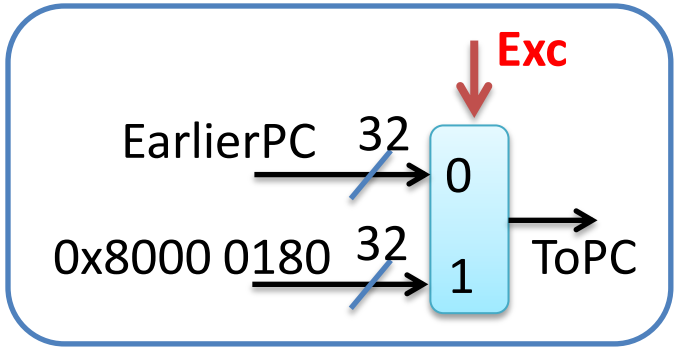
Implementing Exceptions: The Execution Plan

Stage X exception happens:
Stage (X+1) Exception:
(a) EPC = (PC+4) – 4
(b) Cause = <cause>
(c) PC = 0x8000 0180

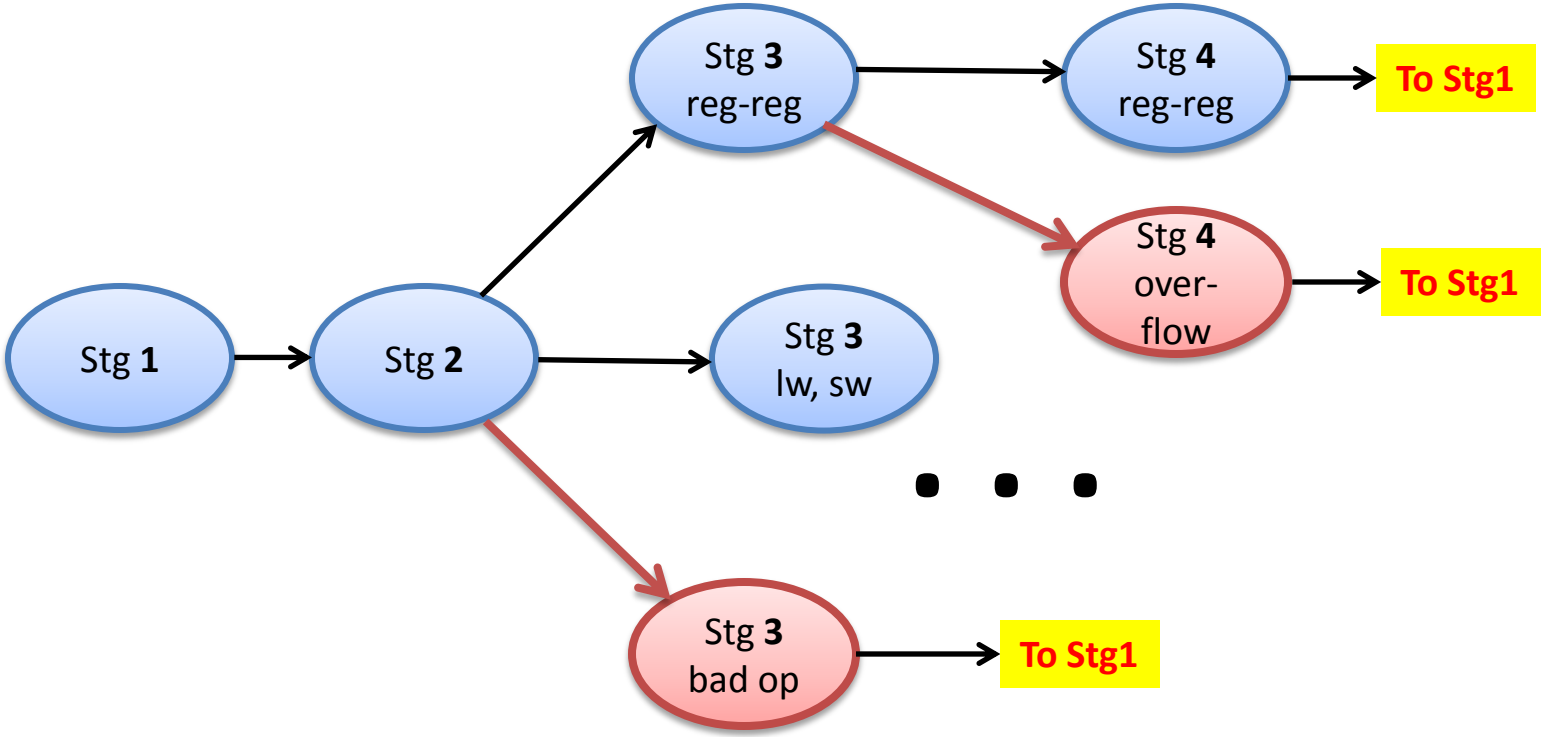
Implementing Exceptions: Data Path Enhancements







Implementing Exceptions: State Machine Changes



Implementing Exceptions: Control Lines in New States

RegWr: disabled

ALUIp1Ctl: Choose PC

ALUIp2Ctl2: Choose 4

ALUOp2: Choose sub

MemRd, MemWr: disabled

IRWr: disabled

PCSrc: x

PCWr: enabled

Exc: 1=0x8000 0180

EPCWr, CauseWr: enabled

IntCause: 0 (stg4 overflow), 1 (stg3 bad opcode)

Returning from Exception Handler

- Cannot use $ra !
- Need special registers
- MIPS has 2 OS registers: $k0, $k1
  - To be used only by OS

Complexity in Control

- Control can become very complex
- E.g. IA32 architecture (Intel)
  - Hundreds of instructions
  - Impossible to *draw* a state diagram, and be convinced that its correct!
  - Control is too complex!
  - Complex instructions: e.g. MOVS copies string from one memory location to another
- Exception handling ➔ control becomes more complex

Handling Complexity: Micro-Code

- Use a micro-program to generate control lines!
  - Use micro-instructions: simple, branching, sub-routines
- A simple processor within a complex processor
- Additional layer of abstraction
  - CS is all about abstraction

Summary

- Exceptions: unexpected in normal program execution
  - Will happen, need to handle
- Hardware implementation can be enhanced to handle exceptions
- Large or complex instruction set ➔ complex control
  - Exception handling adds to complexity
  - Solution: add a layer of abstraction, use micro-coded control