Exceptions

CSE 410, Spring 2004

Computer Systems

http://www.cs.washington.edu/education/courses/410/04sp/

Reading and References

- Reading
  - » Section 6.7*, Computer Organization and Design, Patterson and Hennessy*
- Reference
  - » Chapter 5, *See MIPS Run*, D. Sweetman

Exceptions and Interrupts

- Many things can happen while executing the assembled instructions
  - » External events (I/O device interrupt)
  - » Memory Translation exceptions
  - » Unusual floating point values
  - » Program errors (eg, invalid instruction)
  - » Data integrity failure
  - » System calls

Exceptions

- An *exception* is an internal event
  - » The unexpected or unusual condition was caused by something the program did
  - » examples include
    - arithmetic overflows, floating point problems
    - syscalls
  - » If you ran the program again, the exception would (probably) happen again at the same point in the program's execution

Exception/Pipelining Interface

- Suppose an `add` instruction overflows, causing an overflow exception
- Instructions after the `add` are already in the pipeline
  - » The partially computed instructions must be *flushed*
- Exception must be caught before register contents have changed

“Precise” Exceptions

- A pipelined CPU always has several instructions in various phases of completion
- When an exception occurs, the CPU will record the location of the *exception victim*
- With Precise Exceptions
  - » All preceding instructions are completed
  - » All work on the victim and following is erased

Interrupts

- An *interrupt* is an external event
  - » The unexpected condition was not directly caused by the program
  - » An I/O device request is an example
  - » If you ran the program again, the interrupt would probably *not* happen at the same point
  - » Interrupts are another type of exception, caused by an external event

What should happen?

- These events result in a *change in the flow of control*
- Normally, the next instruction executed is the one following the current instruction
- When one of these events takes place, something else happens
  - » The system must respond to the event
  - » The response depends on the type of event

Exception Handling

1. The CPU saves the address of the offending instruction in a register
2. Makes the reason for the exception known
   - Set the value of the *status register*, or
   - Use *vectored interrupts* to do step 3
3. Transfers control to the operating system
4. Operating system decides what to do

Exceptions example

```
        .data
big:       .word 0x7FFFFFFF
kernelref: .word 0x80000000
        .text
main:
        la     $t0,big           # a valid aligned address
        lw     $t1,1($t0)        # err - unaligned load
        lw     $t0,kernelref     # kernel area address
        sw     $t1,0($t0)        # err - bad address
        lw     $t0,big           # big number
        lw     $t1,big           # another big number
        add    $t2,$t0,$t1       # err - arithmetic overflow
        j      $ra
```

Exception Example results

Console

```
Exception 4  [Unaligned address in inst/data fetch]  occurred and ignored
Exception 7  [Bad address in data/stack read]  occurred and ignored
Exception 12  [Arithmetic overflow]  occurred and ignored
```

“trap.handler” is our OS

```
.ktext 0x80000080
.set noat
# Because we are running in the kernel, we can use
# $k0/$k1 without saving their old values.
move $k1 $at  # Save $at
.set at
sw $v0 s1     # Not re-entrant and we can't trust $sp
sw $a0 s2
mfc0 $k0 $13  # Cause
sgt $v0 $k0 0x44 # ignore interrupt exceptions
bgtz $v0 ret
. . .
```

$k0, $k1

- Note that the trap handler uses $k0 and $k1 to get itself started
- Those are the only registers that it knows are not being used by the user program
- An exception or interrupt may happen at any time
- So the value of $k0 and $k1 will change while your program is executing

Frequent Exceptions

- Syscall
  - » user program call to the operating system for service
- Translation buffer missing entry
  - » memory event, likely response is memory allocation
- Interrupt
  - » device input / output event