Exceptions
Input/Output
Announcements

• HW 8 due tomorrow

• Final exam: Thursday May 19, 7pm @ Olin 255, 100mins
  – More details to follow next Tuesday
    • Review session, office hours, etc.
    • Sample exam will be posted soon
  – Inform instructor about schedule conflict ASAP

• Final grading scheme
  – Median (●)
  – Std. Deviation (σ) = min(8.0, σ)
Review: Virtual Address and Page Table

- **VPN** analogous to the index for a cache
- Page table stored in MM
- Special CPU register
- Indicating that a page contains $2^{12} = 4$ KB
- Page table entries (PTEs)
Review: Translation Lookaside Buffer (TLB)

- Small cache of recently accessed PTE (typically 16-512 entries, fully associative)

```
<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Tag: virtual page number</th>
<th>Data: physical page number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Accessed only on a TLB miss

Physical memory

Page table

Virtual page number

Valid Dirty Ref

Physical page or disk address

Disk storage
Exceptions and Interrupts

- **Useful methods for signaling the CPU that some event has occurred that requires action**
  - In response, the CPU may *suspend* the running program in order to handle the exception/interrupt

- **Exceptions** are used to handle conditions that arise when executing instructions on the processor
  - Detected by the processor itself

- **Interrupts** are used to handle (asynchronous) events external to the processor
  - I/O device request, external error or malfunction
Why are Exceptions Useful?

- **Allow user programs to get service from the OS**
  - A system call creates an exception that kicks out the user program and transfers control to exception handler, which will invoke a proper OS service routine.

- **Handle unexpected events**
  - Overflow, divide-by-zero, invalid opcode, memory protection violation, etc.

- **Handle page faults**
Pipeline with Exception Handling
Pipeline with Exception Handling

When an Exception signal is raised:

- The control unit (CU) sets the *Cause* of the exception and *Exception PC* (with the address of the *faulting instruction*)

- All instructions before the exception complete

- The *faulting instruction* (that causes the exception), and any behind it in the pipeline, are turned into NOPs

- The PC of the first instruction in the exception handler code is loaded into the PC register
Instruction Page Fault

[SUB R5,R5,R7] not in main memory

LW R4,0(R1)
ADD R1,R2,R3
Instruction Page Fault

[SUB R5,R5,R7]  <TLB miss>

LW  R4,0(R1)  ADD R1,R2,R3

Lecture 25: 10
Instruction Page Fault

[SUB R5,R5,R7]

<stall>

<access page table>

<LW and ADD have completed>
Instruction Page Fault

[SUB R5, R5, R7]

<page fault>
Instruction Page Fault

[SUB R5,R5,R7]

<page fault>
Instruction Page Fault

[SUB R5, R5, R7]
&page fault
Instruction Page Fault

[SUB R5, R5, R7]
&page fault>
Instruction Page Fault

1st instruction in exception handler
Enabling Program Restart and Calling OS

The exception handler then takes the following actions:

- **Saves the program state** into memory so this program can later be restored when the exception has been handled.

- **Reads the Cause register** and determines the appropriate service of the OS to call.
A Recap: Handling a Page Fault

ADD R1,R2,R3
LW  R4,0(R1)
SUB R5,R5,R7

instruction page boundary

ADD R1,R2,R3
LW  R4,0(R1)
SUB R5,R5,R7

TLB miss
Page table miss

CPU generates a page fault exception

OS brings in the page, updates page tables and TLB, restores program state, and resumes the program

exception handler saves program state and transfers control to the OS
Computer with Input/Output

Processor

L1 Inst Cache
+ Inst TLB

L1 Data Cache
+ Data TLB

L2 Cache

Main Memory

Input/Output

disk, keyboard, graphics, network, etc

interconnect (e.g., bus)

Input

Output
Input/Output Devices

- I/O devices are the media to allow computer systems to interact with the outside world.
Example Server System with I/O

DDR = “Double Data Rate” (memory interface standard)

DIMM = “Dual Inline Memory Module” (circuit board with DRAMs on both sides)

Serial ATA = “Serial Advanced Technology Attachment” (interface standard for storage device)

PCIe = "Peripheral Component Interconnect Express" (high-speed bus standard)
I/O Controller

• An I/O controller manages one or more peripheral devices
  – Function: coordinates data transfers between the device(s) and the rest of computer system
  – Interface: contains a set of special registers for communication with the processor
    • Command registers
      – Tells the device to do something
      – Written by CPU/OS
    • Status registers (read by processor/OS)
      – Indicates the status of the device (ready, busy, error)
      – Read by CPU/OS
    • Data input/output registers
Accessing I/O Devices

• How do we get a command/data to the right device?

• Dedicated I/O instructions
  – Separate Load/Store instructions to access I/O registers
  – Only the OS can use these instructions

• Memory-mapped I/O
  – Portion of the physical address space is assigned to I/O device registers
  – Only the OS can access these addresses
  – Each I/O device register has a unique memory address
Data Transfer Between I/O and Memory

• **Programmed I/O (PIO)**
  – Processor completely arbitrates transfer of data from device to memory
    • Typically much less efficient than DMA

• **Direct Memory Access (DMA)**
  – I/O device transfers data directly to main memory
  – Processor/OS sets up the transfer through I/O commands
  – And then can do something else, like run another program
Informing the Processor

• A device needs to inform the processor when an I/O operation is completed

• Polling
  – Processor periodically reads the Status Register, which indicates when an operation is done

• Interrupt-driven I/O
  – I/O device sends an interrupt to the processor when the operation is done
  – More efficient than polling
Let’s Pull Some Pieces Together

- Page fault occurs
- Exception handler gets loaded
- Exception handler takes action
- OS sets up disk transfer
- OS schedules another program
- Data is read from disk and transferred to main memory
- Disk controller interrupts processor
- Second program is interrupted
- First program can run again
Data Page Fault Occurs in Program A

INSTR

Lecture 25: 27
Exception Handler Gets Loaded

1st instruction in exception handler

Exception PC
Exception Cause

Control Signals

sign bit

Exception

PC

EL

F_m … F_0

D_IN

Data TLB

L1 Data

$
Exception Handler Takes Action

• Saves program A state

• Reads the Cause register and determines that a page fault occurred

• Calls the appropriate part of the OS
OS Sets Up Disk Transfer using DMA

- **Intel Xeon 5300 processor**
- **Front Side Bus (1333 MHz, 10.5 GB/sec)**
- **FB DDR2 667 (5.3 GB/sec)**
- **Main memory DIMMs**
- **Serial ATA (300 MB/sec)**
- **Disk**
- **LPC (1 MB/sec)**
- **USB 2.0 (60 MB/sec)**
- **Keyboard, mouse, ...**
- **Parallel ATA (100 MB/sec)**
- **PCIe x16 (or 2 PCIe x8) (4 GB/sec)**
- **PCIe x4 (1 GB/sec)**
- **PCI-X bus (1 GB/sec)**
- **NB-SB interface**
- ** memory controller**
- **SATA controller**

"read 4K bytes from disk starting at addr X to memory starting at addr Y"
OS Schedules Another Program

- While waiting for the disk read to complete for program A, the OS scheduler may run a different task (program B).

- It loads the processor with the state (PC, PTR, and RF) of program B.
OS Switches from Program A to B

• What do we do about the PC?
  – Save

• What do we do about the registers?
  – Save

• What do we do about the Page Table Register?
  – Save

• What do we do about the TLBs?
  – Invalidate all TLB entries

• What do we do about the caches?
OS Switches from Program A to B

Page Table Register

CU

Adder

RF

LD
SA
SB
DR

D_in

Exception PC

Exception Cause

EL

L1 Data

Fm \ldots F_0

D_IN

Data TLB

MW

MD

Inst TLB

Decoder

IF/ID

PCJ

PC

+2

L1 Inst

$
Page is Read from Disk

-now running program B-

Intel Xeon 5300 processor

Front Side Bus (1333 MHz, 10.5 GB/sec)

Main memory DIMMs

FB DDR2 667 (5.3 GB/sec)

Serial ATA (300 MB/sec)

Disk

LPC (1 MB/sec)

USB 2.0 (60 MB/sec)

Keyboard, mouse, ...

“read 4K bytes starting at addr X”

Intel Xeon 5300 processor

PCIe x16 (or 2 PCIe x8)

(4 GB/sec)

PCIe x4 (1 GB/sec)

PCIe x4 (1 GB/sec)

PCI-X bus (1 GB/sec)

PCI-X bus (1 GB/sec)

Parallel ATA (100 MB/sec)

CD/DVD

memory controller

FSB interface

NB-SB interface

SATA controller
Page is Read from Disk

(now running program B)

Intel Xeon 5300 processor

Front Side Bus (1333 MHz, 10.5 GB/sec)

FB DDR2 667 (5.3 GB/sec)

PCIe x16 (or 2 PCIe x8) (4 GB/sec)

Main memory DIMMs

Serial ATA (300 MB/sec)

SATA controller

data returned to disk controller

Disk

USB 2.0 (60 MB/sec)

PCIe x4 (1 GB/sec)

PCIe x4 (1 GB/sec)

Parallel ATA (100 MB/sec)

PCI-X bus (1 GB/sec)

PCI-X bus (1 GB/sec)

FSB interface

memory controller

NB-SB interface

CD/DVD

Keyboard, mouse, ...
DMA Transfer of Page to Memory

(data written to memory location Y (via DMA))

Intel Xeon 5300 processor

Front Side Bus (1333 MHz, 10.5 GB/sec)

FB DDR2 667 (5.3 GB/sec)

PCIe x16 (or 2 PCIe x8) (4 GB/sec)

Main memory DIMMs

Serial ATA (300 MB/sec)

PCIe x4 (1 GB/sec)

Disk

PCIe x4 (1 GB/sec)

LPC (1 MB/sec)

PCI-X bus (1 GB/sec)

Keyboard, mouse, ...

USB 2.0 (60 MB/sec)

PCI-X bus (1 GB/sec)

CD/DVD

Parallel ATA (100 MB/sec)

SATA controller

NB-SB interface

memory controller

FSB interface

(data transferred to memory controller)

(now running program B)
I/O Controller Interrupts Processor

(now running program B)

interrupt
“I/O complete”

Intel Xeon 5300 processor

Front Side Bus (1333 MHz, 10.5 GB/sec)

Intel Xeon 5300 processor

FB DDR2 667 (5.3 GB/sec)

PCIe x16 (or 2 PCIe x8) (4 GB/sec)

Main memory DIMMs

Disk

Disk

Serial ATA (300 MB/sec)

LPC (1 MB/sec)

USB 2.0 (60 MB/sec)

Keyboard, mouse, ...

Parallel ATA (100 MB/sec)

CD/DVD

FSB interface

NB-SB interface

memory controller

SATA controller

PCle x4 (1 GB/sec)

PCle x4 (1 GB/sec)

PCI-X bus (1 GB/sec)

PCI-X bus (1 GB/sec)
Program B is Interrupted

The state of program B is saved by the interrupt handler
Program A Can Now Run Again

- Page fault was handled, so OS marks program A as runnable

- If OS scheduler chooses to run program A, it loads its state (PC of the LW, PTR, and registers)

- Key point: Processor was free to do other work during the long I/O transfer time
  - **DMA**: Processor did not have to directly handle data transfers from device to memory
  - **Interrupt-driven I/O**: Processor did not have to poll the device to see when the I/O operation completed
Building a Complete Computer
Next Time

More on Final Exam
Advanced Topics