

Interconnects and Interrupts in Input/Output

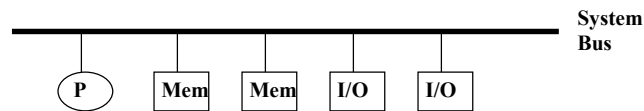
- ◆ **Last Time**
 - Cache organization, replacement
 - Virtual Memory
- ◆ **This Time**
 - Busses, Arbitration
 - Basic Input/Output Operation
 - Interrupts
 - Interrupt-Driven I/O
 - Serial and Block Device Examples
- ◆ **Reminders/Announcements**
 - HW #5 due today (start of lecture)
 - No Quiz on Thursday!
 - Final Exam 3/20, 3-6pm, Peterson 110
 - TA Review Session, Saturday 3/18? (ask the TA's, check in the newsgroup)

CS 141 Chien

1

March 14, 2000

Busses: Basic System Interconnection



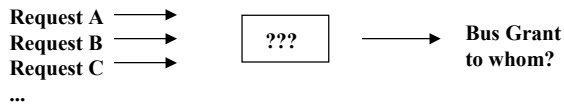
- ◆ **Bus: Electrical connection amongst multiple devices (ISA,EISA,NuBus,MBus,SCSI,PCI)**
- ◆ **Uniform Shared Interface (standard modules)**
- ◆ **Basis for Communication and Interoperability**
 - Processor and memory are just devices on a bus
- ◆ **Low Cost, Shared resource**
 - Set of wires and terminations
 - Only one device can transmit at a time

CS 141 Chien

2

March 14, 2000

Bus Arbitration



- ◆ To ensure correct operation, only one transmitter can use the bus at a time. How to decide which one? Bus Arbitration.
- ◆ What if everyone tries to use the bus at once?
 - Smoke (electrical drive in different directions)
 - Noise (no clear electrical signals)
- ◆ Arbitration goals:
 - Prevent more than one bus user at a time
 - Give all devices a fair chance at the bus

CS 141 Chien

3

March 14, 2000

Fixed Priority Arbitration



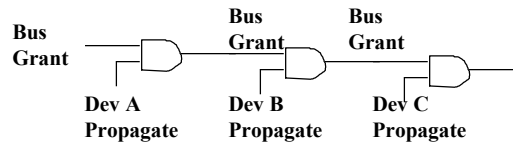
- ◆ Each device (or bus slot) given a fixed priority
- ◆ Each time, bus granted to device with highest priority
- ◆ Advantage: pretty simple
- ◆ Disadvantage:
 - Unfair, some devices may never get access
 - Scheme supports some fixed number of priority levels

CS 141 Chien

4

March 14, 2000

Daisy-Chain Arbitration



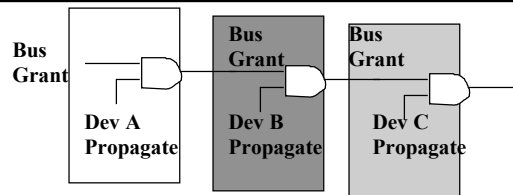
- ◆ Daisy-chain links bus grant signals in a long chain
- ◆ Each device waits for grant, uses bus, propagates the grant
- ◆ Each device gets one chance at the bus per arbitration phase
- ◆ Distributed, Fair arbitration
- ◆ Separate bus request lines start arbitration

CS 141 Chien

5

March 14, 2000

Example of Daisy Chain Arbitration



- ◆ First cycle, Device A gets the bus and uses it.
- ◆ Later cycle, Device B gets bus and uses it.
- ◆ Even later cycle, Device C gets bus and uses it.
- ◆ then, the whole thing begins again.
- ◆ Parallel wires for data and address not shown.

CS 141 Chien

6

March 14, 2000

Limitations of Busses

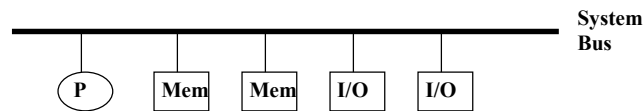
- ◆ **Propagation Delay:** signals must travel all the way to both ends (speed of light limitations)
- ◆ **Ringing:** signals must propagate up and down several times to settle
- ◆ **Loading:** increasing the number of devices can increase propagation delay and ringing
- ◆ **Inherent tradeoff** between number of devices and speeds, maximum speeds -- we're nearly there
- ◆ **MAIN PROBLEM:** limited bandwidth is shared by all the devices, it does not scale.
- ◆ **However, this is currently the dominant interconnect.**

CS 141 Chien

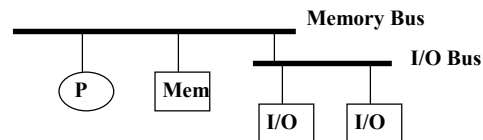
7

March 14, 2000

Basic Input/Output Architecture



- ◆ **Memory and Input/Output Busses**



- ◆ **Separate Memory and Input/Output Busses**
- ◆ **Input/Output Devices: serial and block devices**

CS 141 Chien

8

March 14, 2000

Input/Output Devices

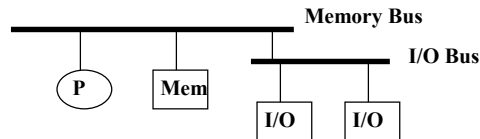
- ◆ **Types**
 - Serial devices: one word at a time (keyboard)
 - Block devices: larger chunks of data (disk)
- ◆ **Data Rates**
 - Keyboard: 10 bytes / sec
 - Mouse: 10 samples / sec ?
 - Disks: 5-15 MB/s
 - Display: 50MB/s (full rate), typically much slower
 - Ethernet: 1 MB/s, Gigabit Ethernet: 125MB/s
- ◆ **Frequency? Highly variable**
- ◆ **Note: most of these rates are extremely slow compared to modern processor speeds.**
 - Exception: high speed displays and networks

CS 141 Chien

9

March 14, 2000

A Basic Input/Output Operation



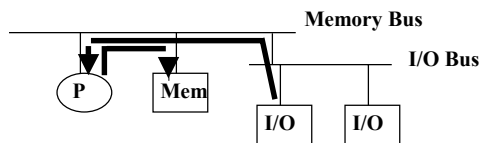
- ◆ **Processor Memory Reads/Writes to I/O area are passed through to the I/O bus**
- ◆ **Processor “polls” devices (asking if they are ready)**
- ◆ **When a pending I/O transfer is detected (data is ready), processor reads the data and places it in memory.**

CS 141 Chien

10

March 14, 2000

Input/Output Operation



- ◆ Polling involves reading the control registers of the I/O devices in turn
- ◆ Servicing the I/O involves reads of I/O data (one or many words) into registers, then writes into memory addresses
- ◆ After the I/O device has no more data the I/O transfer is complete

Summary of Basic Steps

- ◆ Processor Polls continuously for I/O operations (periodically)
- ◆ When a device is ready, processor reads the I/O data
- ◆ Processor writes I/O data into the memory
- ◆ Processor resumes other computation or I/O
- ◆ Is this efficient?

Polled Input/Output

- ◆ **Advantages**

- Simple hardware
- Everything under program control

- ◆ **Disadvantages**

- Uses processor to poll I/O devices, wasting processor cycles
- Uses processors to move the I/O data, wasting processor cycles
- All programs must poll all I/O devices, even if they're not doing I/O (someone else's I/O might be pending)

Exceptions and Interrupts

- ◆ **Exceptions cause a control transfer to an exception handler in response to an unusual event.**

- Ex. floating point overflow, memory addressing error, segmentation fault, etc.

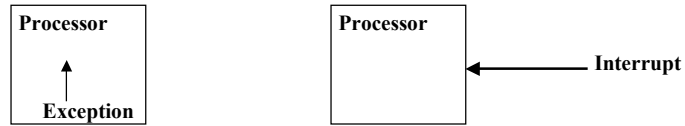
- ◆ **This control transfer looks like a “Dynamic Procedure call”**

- ◆ **Exceptions are caused by the executing instruction(s)**

- Happen “at a particular instruction”
- One handler for each exception, “vectored exceptions”

- ◆ **Handler decodes the exception based on stored status info (PC, exception register, etc.), and performs appropriate action**

Interrupts



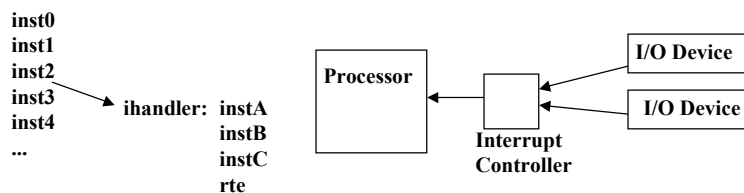
- ◆ Interrupts = “externally caused exceptions”
- ◆ Dynamic Procedure call (same as an internal exception)
- ◆ Have NOTHING to do with current execution, not synchronized with instruction stream, so delays can be tolerated.
- ◆ Interrupt -> “interrupts” what the processor is doing

CS 141 Chien

15

March 14, 2000

Interrupt Example



- ◆ External Device causes interrupt
- ◆ Executing code is stopped, dynamic procedure call
 - Delay doesn’t matter much
- ◆ State is saved (as with an exception), control is passed to the “interrupt handler”
- ◆ After servicing interrupt, control returned to previously running program

CS 141 Chien

16

March 14, 2000

Using Interrupts for Input/Output

- ◆ **Why are interrupts useful?**
 - External analogy to exceptions
 - Allow response to unusual external events without inline overhead (polling).
- ◆ **Idea: Use interrupts to reduce polling overhead for Input/output**
 - Processor initiates I/O operation
 - Device interrupts processor when its ready
 - Interrupt handler transfers data into the memory
 - Control returned to currently executing program
- ◆ **Analogous schedule for output operations**
- ◆ **Polling overhead vs. Interrupt handling cost**

CS 141 Chien

17

March 14, 2000

Advantages & Complications

- ◆ **Advantages**
 - Reduced polling time vs. interrupt cost
 - Improved program modularity (no more polling code)
 - Preserves the ability of old programs to run in new configurations
- ◆ **Complications**
 - Additional hardware required (interrupt controller, etc.)
 - How to deal with several devices?
 - What if more than one interrupt occurs at once?
- ◆ **We'll deal with these questions next.**

CS 141 Chien

18

March 14, 2000

Multiple sources of Interrupts

- ◆ **How to determine which device caused the interrupt?**

- Single device per priority
- Devices place their ID number on the address bus
- Processor polls devices to find out which caused the interrupt
- Interrupt Handler -> reads some hardware status registers to determine the source of the interrupt

- ◆ **How to handle an interrupt within an interrupt?**

- Many systems don't; disable interrupts when within an interrupt handler
- Some systems have interrupt priorities, while in an interrupt handler; only a higher priority interrupt will be handled

CS 141 Chien

19

March 14, 2000

Interrupt Handler Structure

```
inhandler: disableInts
           lw $2,device($0)
           ... do the work ...
           enableInts
           rte
```

- ◆ **Disable Interrupts**
- ◆ **Determine device causing interrupt**
- ◆ **Do appropriate processing**
- ◆ **Enable Interrupts**
- ◆ **What does the stack look like during the interrupt?**
 - stuff on stack before interrupt
 - stack frame for the interrupt handler? Special return instruction (rte)
 - when can you allocate stack frame memory? (when is it safe?)

CS 141 Chien

20

March 14, 2000

Interrupts and Response Time

Device Number	Interrupt Priority
0	3
1	2
2	2
3	1

- ◆ Lower numbers are “higher priority”
- ◆ How should we assign priorities to devices?
 - Latency is critical for some devices.
 - Ordering can have big impact on latency.
- ◆ Assume non-preemptive priority system.
 - Interrupts cannot be interrupted.
 - Pending interrupts are processed in order of priority.

CS 141 Chien

21

March 14, 2000

Guaranteed Maximum Latency

Guaranteed
Maximum (I)
Latency = MAX(service time of devices at I)
+ SUM(service time of devices with greater priority)

Device Number	Interrupt Priority	Service	Guar. Max. Latency
0	3	10	
1	2	10	
2	2	10	
3	1	10	

CS 141 Chien

22

March 14, 2000

Real-Time Systems (Embedded systems)

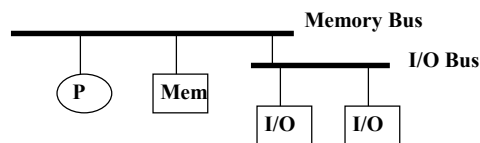
- ◆ **Timeliness of response is fundamental requirement of system design**
 - Control (cars, airplanes, manufacturing)
 - Instrumentation (measurement, data capture)
 - Tracking, Guidance (military apps)
 - Otherwise, system fails!
- ◆ **Real-Time requirements fundamentally affect the nature of system design.**

CS 141 Chien

23

March 14, 2000

Interrupt-Driven I/O (Overview)



- ◆ **Processor sets up I/O operation, continues its work**
- ◆ **Device performs I/O (a long time)**
- ◆ **Device completes, interrupts the processor**
- ◆ **Processor responds to interrupt, transfers the data**
- ◆ **I/O is complete.**
- ◆ **=> Interrupts save overhead of polling.**

CS 141 Chien

24

March 14, 2000

Summary

- ◆ Busses and Arbitration
- ◆ Basic Input/Output Devices and Architecture
- ◆ Polled Input/Output
- ◆ Interrupts as external exceptions
- ◆ Interrupt Handling
- ◆ Motivation for Interrupt-Driven Input/Output
- ◆ How Interrupt-Driven Input/Output works
- ◆ Where is there still overhead? (data movement)
 - Size of data transfers? Cost?