

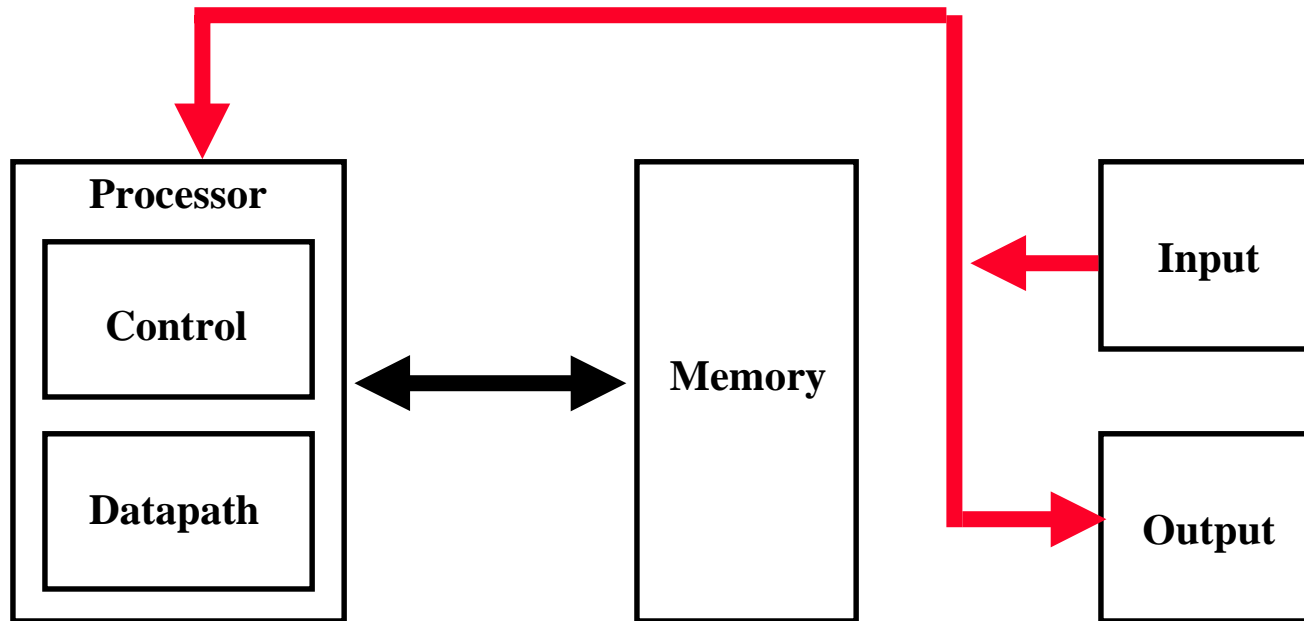
CPS 104
Computer Organization and Programming
Lecture 23: I/O, Interrupts and Exceptions

Nov. 22, 1999

Dietolf (Dee) Ramm

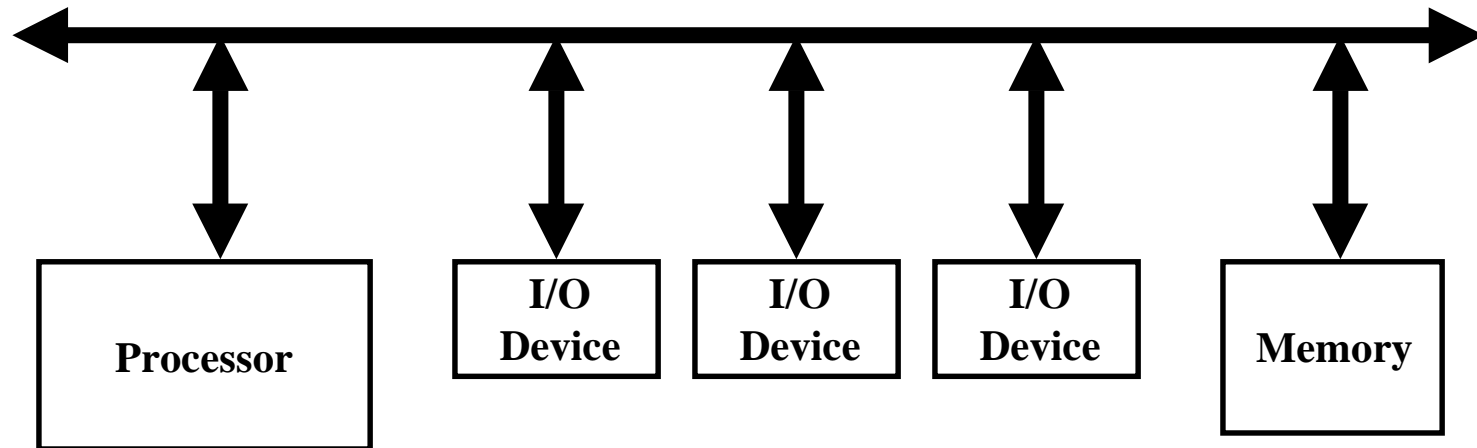
<http://www.cs.duke.edu/~dr/cps104.html>

Buses: Connecting I/O to Processor and Memory



- A bus is a shared communication link
- It uses one set of wires to connect multiple subsystems

Advantages of Buses



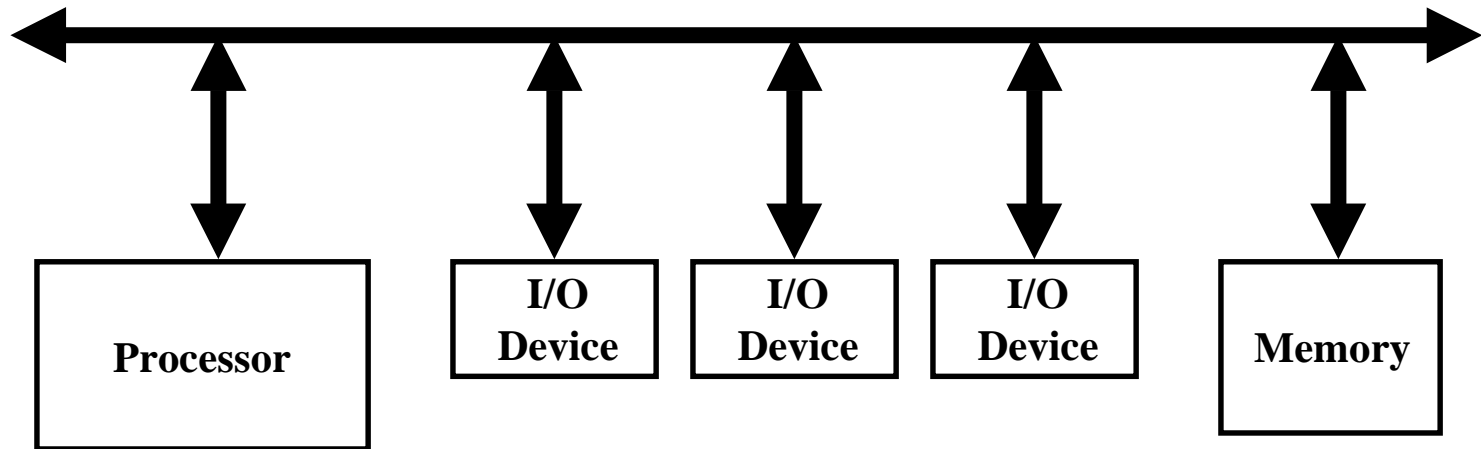
- **Versatility:**

- * New devices can be added easily
- * Peripherals can be moved between computer systems that use the same bus standard

- **Low Cost:**

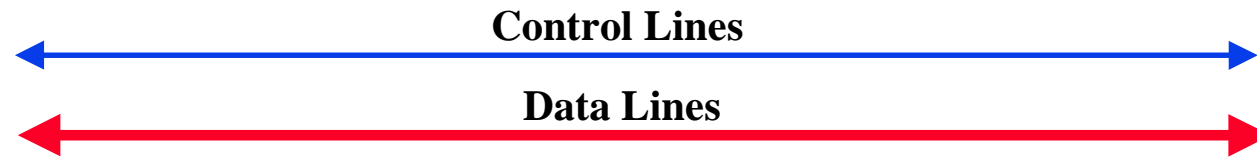
- * A single set of wires is shared in multiple ways

Disadvantages of Buses



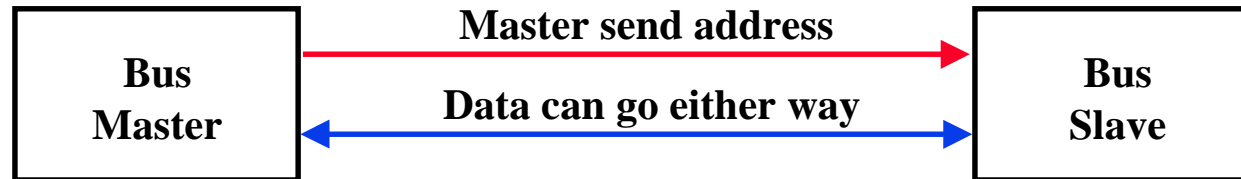
- **The bus creates a communication bottleneck**
 - * **Bus bandwidth can limit the maximum I/O throughput**
- **The maximum bus speed is largely limited by:**
 - * **The length of the bus**
 - * **The number of devices on the bus**
 - * **The need to support a range of devices with:**
 - **Widely varying latencies**
 - **Widely varying data transfer rates**

The General Organization of a Bus



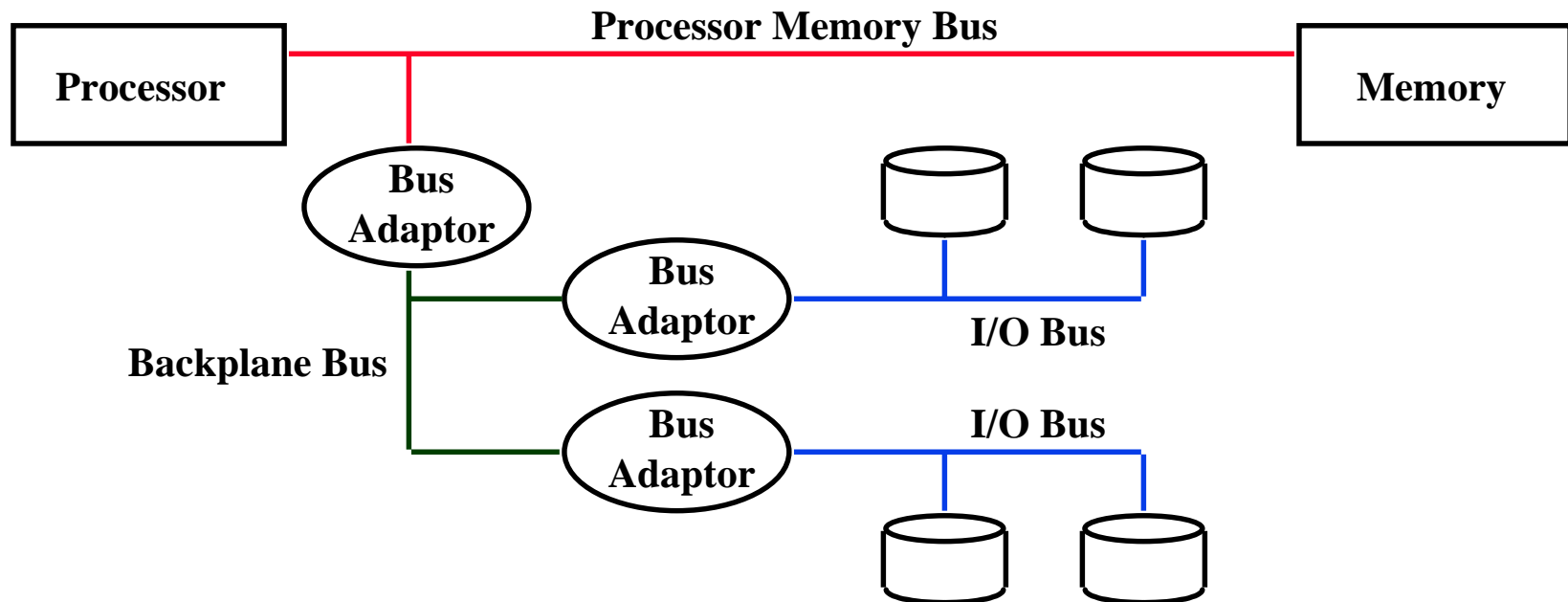
- **Control lines:**
 - * Signal requests and acknowledgments
 - * Indicate what type of information is on the data lines
- **Data lines** carry information between the source and the destination:
 - * Data and Addresses
 - * Complex commands

Master versus Slave



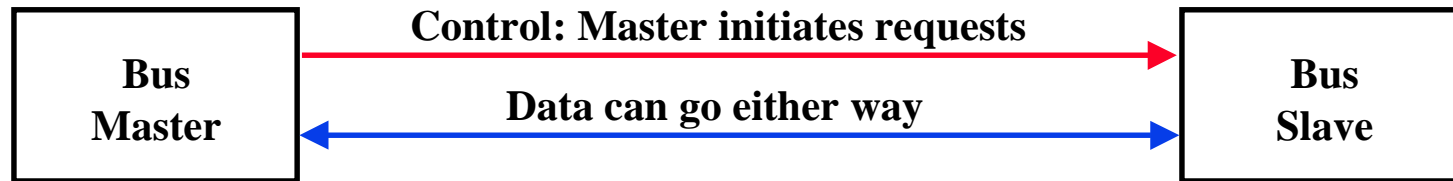
- **A bus transaction includes two parts:**
 - * **Sending the address**
 - * **Receiving or sending the data**
- **Master is the one who starts the bus transaction by:**
 - * **Sending the address**
- **Slave is the one who responds to the address by:**
 - * **Sending data to the master if the master ask for data**
 - * **Receiving data from the master if the master wants to send data**

A Three-Bus System



- **A small number of backplane buses tap into the processor-memory bus**
 - * **Processor-memory bus is used for processor memory traffic**
 - * **I/O buses are connected to the backplane bus**
- **Advantage: loading on the processor bus is greatly reduced**

Obtaining Access to the Bus

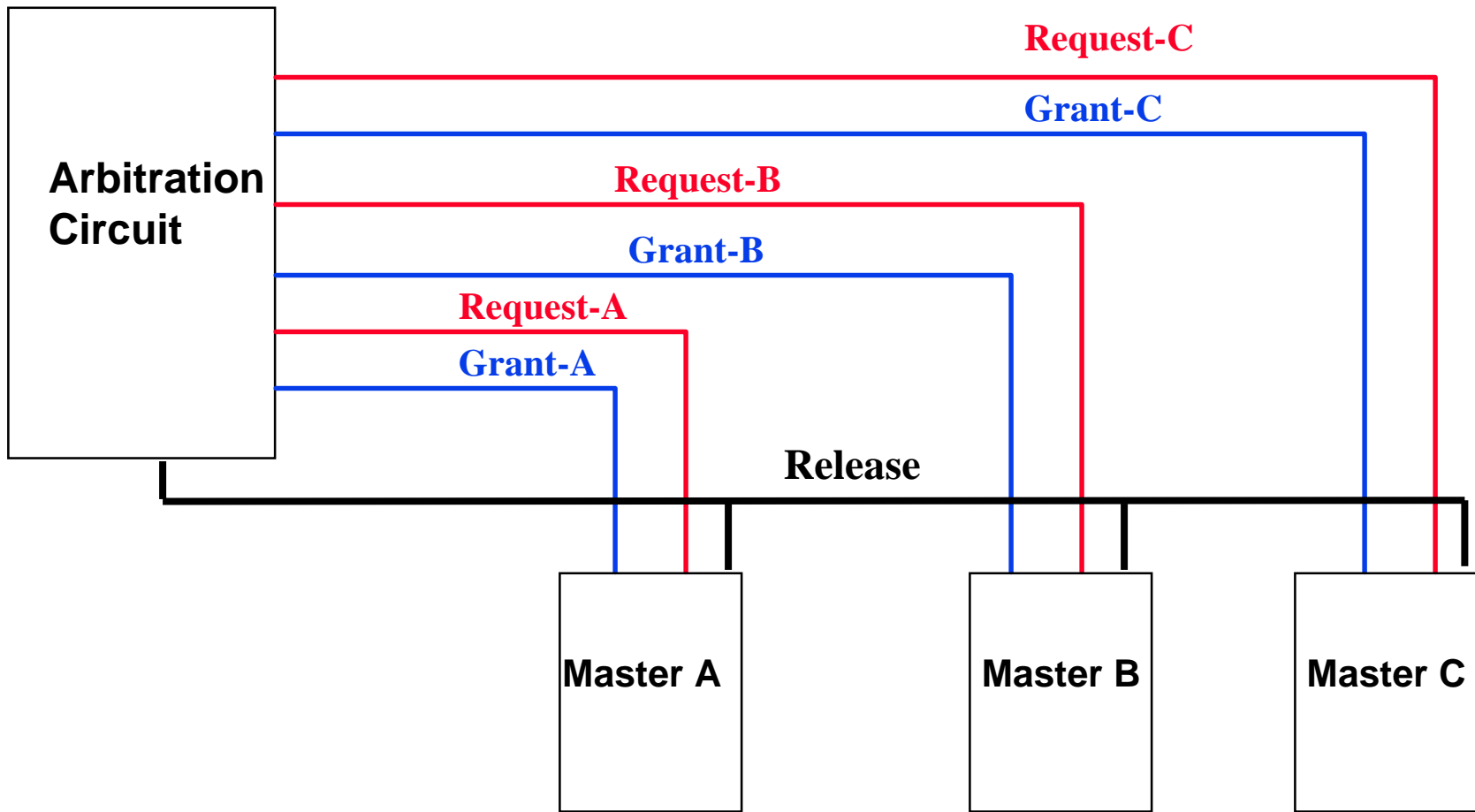


- One of the most important issues in bus design:
 - * How is the bus reserved by a devices that wishes to use it?
- Chaos is avoided by a master-slave arrangement:
 - * Only the bus master can control access to the bus:
It initiates and controls all bus requests
 - * A slave responds to read and write requests
- The simplest system:
 - * Processor is the only bus master
 - * All bus requests must be controlled by the processor
 - * Major drawback: the processor is involved in every transaction

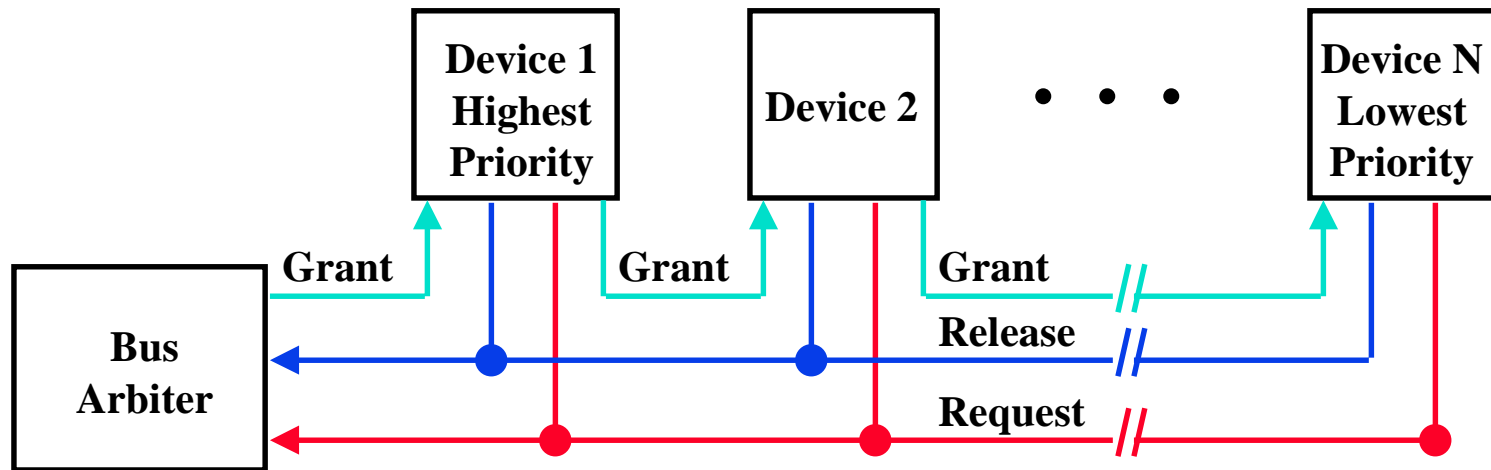
Multiple Potential Bus Masters: the Need for Arbitration

- **Bus arbitration scheme:**
 - * A bus master wanting to use the bus asserts the bus request
 - * A bus master cannot use the bus until its request is granted
 - * A bus master must signal to the arbiter after finish using the bus
- **Bus arbitration schemes usually try to balance two factors:**
 - * **Bus priority:** the highest priority device should be serviced first
 - * **Fairness:** Even the lowest priority device should never be completely locked out from the bus
- **Bus arbitration schemes can be divided into four broad classes:**
 - * **Distributed arbitration by self-selection:** each device wanting the bus places a code indicating its identity on the bus.
 - * **Distributed arbitration by collision detection:** Ethernet uses this.
 - * **Daisy chain arbitration:** single device with all request lines.
 - * **Centralized, parallel arbitration:** see next-next slide

Centralized Bus Arbitration



The Daisy Chain Bus Arbitration Scheme



- Advantage: simple

- Disadvantages:

- * Cannot assure fairness:

- A low-priority device may be locked out indefinitely

- * The use of the daisy chain grant signal also limits the bus speed

Summary of Bus Options

<i>Option</i>	<i>High performance</i>	<i>Low cost</i>
Bus width	Separate address & data lines	Multiplex address & data lines
Data width	Wider is faster (e.g., 64 bits)	Narrower is cheaper (e.g., 8 bits)
Transfer size	Multiple words has less bus overhead	Single-word transfer is simpler
Bus masters	Multiple (requires arbitration)	Single master (no arbitration)
Clocking	Synchronous	Asynchronous

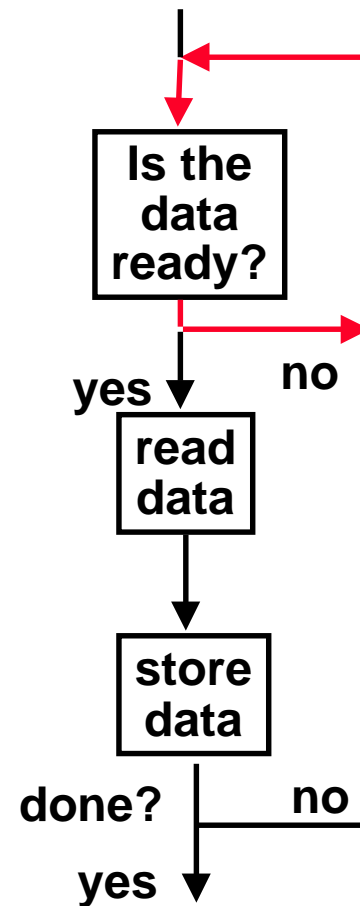
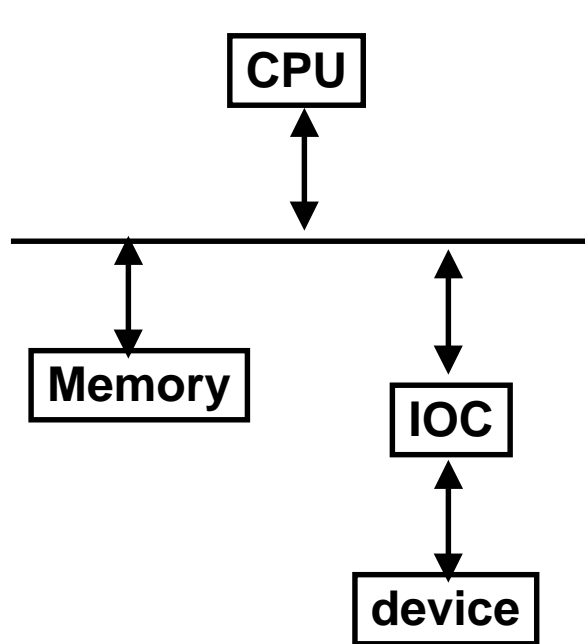
1993 Backplane/IO Bus Survey

Bus	SBus	TurboChannel	MicroChannel	PCI
Originator	Sun	DEC	IBM	Intel
Clock Rate (MHz)	16-25	12.5-25	async	33-66
Addressing	Virtual	Physical	Physical	Physical
Data Sizes (bits)	8,16,32	8,16,24,32	8,16,24,32,64	8,16,24,32,64
Master	Multi	Single	Multi	Multi
Arbitration	Central	Central	Central	Central
32 bit read (MB/s)	33	25	20	33
Peak (MB/s)	89	84	75	111 (222)
Max Power (W)	16	26	13	25

OS and I/O Systems Communication Requirements

- **The Operating System must be able to prevent:**
 - * **The user program from communicating with the I/O device directly**
- **If user programs could perform I/O directly:**
 - * **Protection to the shared I/O resources could not be provided**
- **Three types of communication are required:**
 - * **The OS must be able to give commands to the I/O devices**
 - * **The I/O device must be able to notify the OS when the I/O device has completed an operation or has encountered an error**
 - * **Data must be transferred between memory and an I/O device**

Polling: Programmed I/O



**busy wait loop
not an efficient
way to use the CPU
unless the device
is very fast!**

**but checks for I/O
completion can be
dispersed among
computation
intensive code**

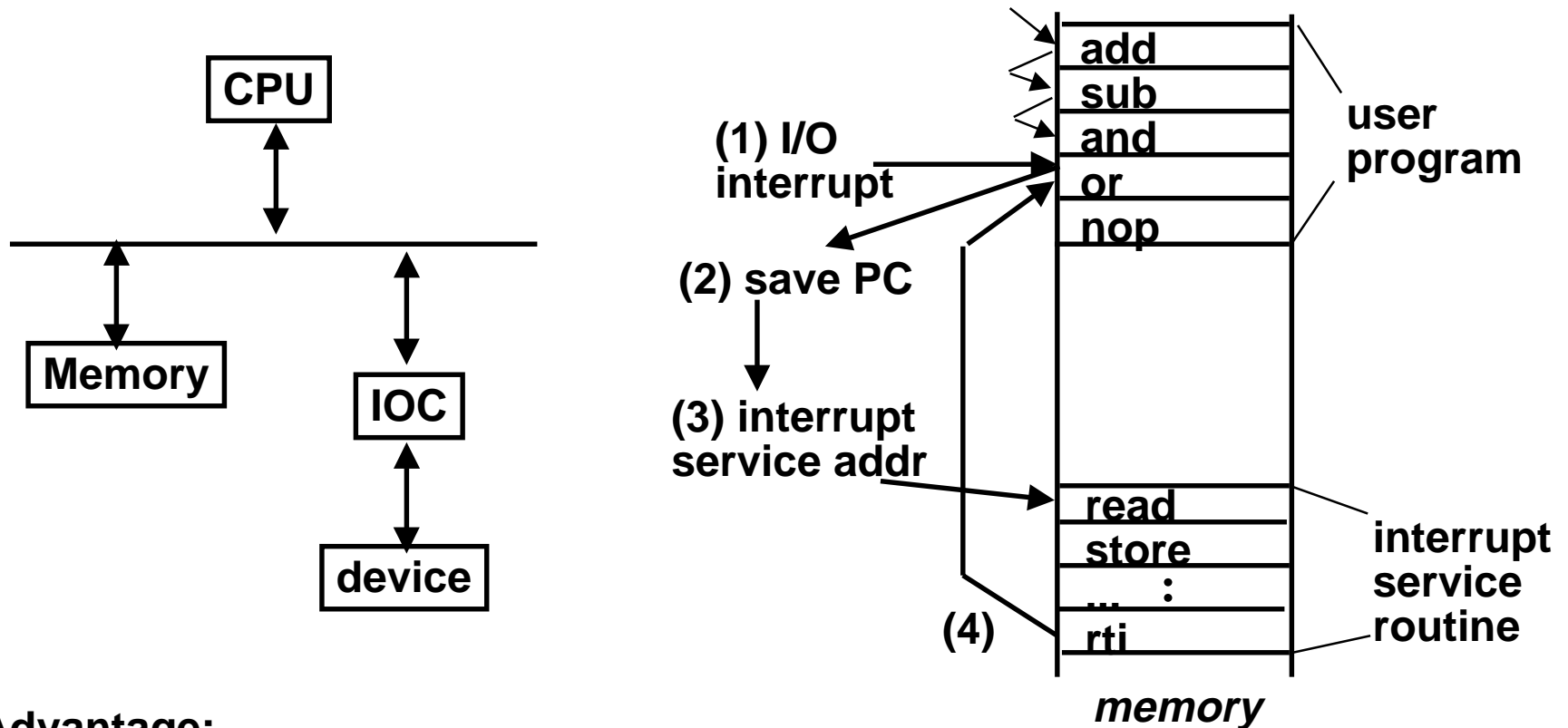
- **Advantage:**

- * **Simple: the processor is totally in control and does all the work**

- **Disadvantage:**

- * **Polling overhead can consume a lot of CPU time**

Interrupt Driven Data Transfer



- **Advantage:**

- * User program progress is only halted during actual transfer

- **Disadvantage, special hardware is needed to:**

- * Cause an interrupt (I/O device)
- * Detect an interrupt (processor)
- * Save the proper states to resume after the interrupt (processor)

I/O Interrupt

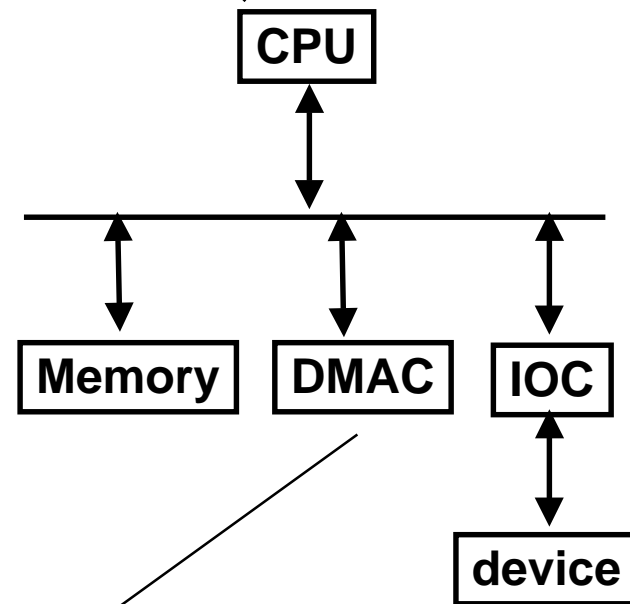
- **An I/O interrupt is just like the exceptions except:**
 - * **An I/O interrupt is asynchronous**
 - * **Further information needs to be conveyed**
- **An I/O interrupt is asynchronous with respect to instruction execution:**
 - * **I/O interrupt is not associated with any instruction**
 - * **I/O interrupt does not prevent any instruction from completion**
 - **You can pick your own convenient point to take an interrupt**
- **I/O interrupt is more complicated than exception:**
 - * **Needs to convey the identity of the device generating the interrupt**
 - * **Interrupt requests can have different urgencies:**
 - **Interrupt request needs to be prioritized**

Delegating I/O Responsibility from the CPU: DMA

- **Direct Memory Access (DMA):**

- * External to the CPU
- * Act as a master on the bus
- * Transfer blocks of data to or from memory without CPU intervention

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".



DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.

Summary:

- **Disk I/O Benchmarks: I/O rate vs. Data rate vs. latency**
- **Three Components of Disk Access Time:**
 - * **Seek Time: advertised to be 8 to 12 ms. May be lower in real life.**
 - * **Rotational Latency: 4.1 ms at 7200 RPM and 8.3 ms at 3600 RPM**
 - * **Transfer Time: 2 to 12 MB per second**
- **Three types of buses: Processor-memory, I/O, Back-plane**
 - * **performance Vs. cost**
- **Bus arbitration schemes: simplicity Vs. fairness**
- **I/O device notifying the operating system:**
 - * **Polling: it can waste a lot of processor time**
 - * **I/O interrupt: similar to exception except it is asynchronous**
- **Delegating I/O responsibility from the CPU: DMA, or even IOP**

Interrupts and Exceptions

Interrupts Exceptions and Traps

- **Interrupts, Exceptions and Traps** are asynchronous changes in the control flow. Interrupts and Exceptions can be viewed as asynchronous (unscheduled) procedure calls.
- Interrupts and exceptions are designed to provide:
 - ◆ **protection mechanisms:** Error handling, TLB management.
 - ◆ **efficiency:** Overlap I/O and execution, . . .
 - ◆ **Illusion of parallelism:** Timesharing, multithreading
 - ◆ **Ability to handle Asynchronous external events:** Network connections, keyboard input, DMA I/O, . . .

Interrupts and Exceptions

- **Exception:** a change in execution caused by a condition that occurs **within** the processor.
 - ◆ segmentation fault (access outside program boundaries, illegal access, . . .)
 - ◆ bus error
 - ◆ divide by 0
 - ◆ overflow
 - ◆ page fault (virtual memory...)
- **Interrupt:** a change in execution caused by an external event
 - ◆ devices: disk, network, keyboard, etc.
 - ◆ clock for timesharing (multitasking)
 - ◆ These are useful events, must do something when they occur.
- **Trap:** a user-requested exception
 - ◆ Operating system call (syscall)
 - ◆ Breakpoints (debugging mode)

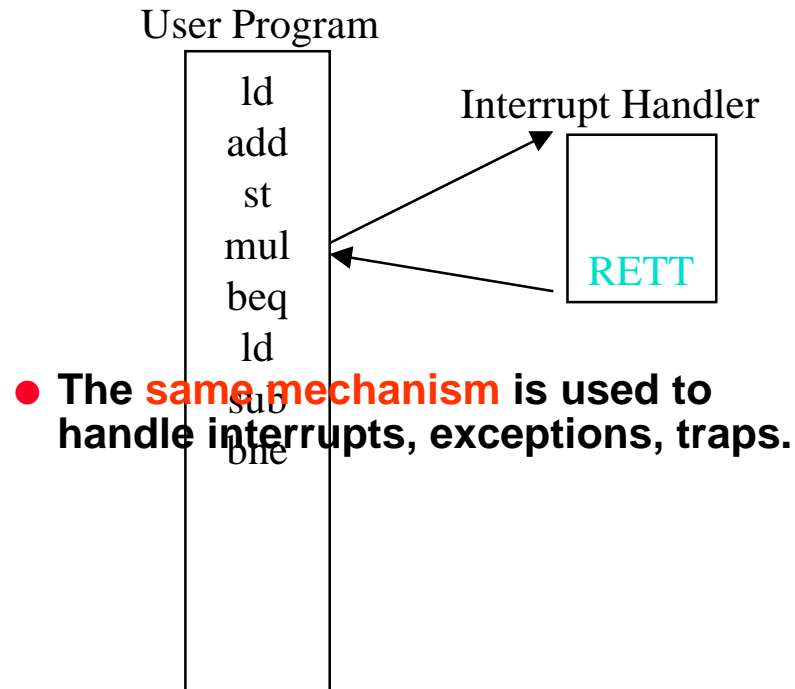
An Execution Context

- The state of the CPU associated with a thread of control (process)
 - * general purpose registers (integer and floating point)
 - * status registers (e.g., condition codes)
 - * program counter, stack pointer
- Need to be able to switch between contexts
 - * **timesharing**: sharing the machine among many processes
 - * better utilization of machine (overlap I/O of one process with computation of another)
 - * different **modes** (Kernel v.s. user)
- Maintained by operating system

Context Switches

- **Save current execution context**
 - * **Save registers and program counter**
 - * **information about the context (e.g., ready, blocked)**
- **Restore other context**
- **Need data structures in kernel to support this**
 - * **process control block**
- **Why do we context switch?**
 - * **Timesharing: HW clock tick**
 - * **I/O begin and/or end**
- **How do we know these events occur?**
 - * **Interrupts...**

Handling an Exception



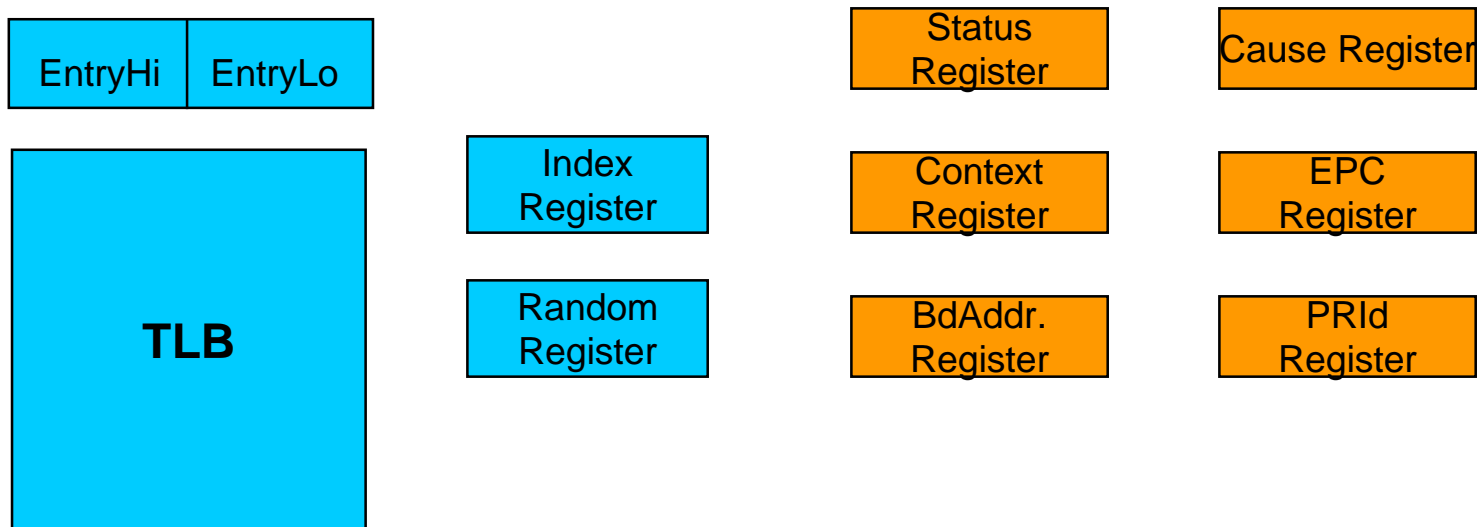
- Invoke specific kernel routine based on type of interrupt
 - * interrupt/exception handler
- Must determine what caused interrupt
 - * **Cause register** holds information on what caused the interrupt
 - * $PC \leq \text{interrupt_handler}$
- Vectored Interrupts
 - * $PC = \text{interrupt_table}[i]$
- Clear the interrupt
- Return from interrupt (RETT) to different process (e.g, context switch)

Execution Mode

- What if interrupt occurs while in interrupt handler?
 - * *Problem:* Could lose information for one interrupt clear of interrupt #1, clears both #1 and #2
 - * *Solution:* **disable interrupts**
- Disabling interrupts is a protected operation
 - * Only the kernel can execute it
 - * user v.s. kernel mode
 - * **mode bit in CPU status register**
- Other protected operations
 - * installing interrupt handlers
 - * manipulating CPU state (saving/restoring status registers)
 - * Updating TLB, Changing page table, changing page protection.
- Changing modes
 - * interrupts
 - * **system calls (syscall instruction)**

Interrupts and exceptions in the MIPS 2000

- The MIPS has a special coprocessor **CP0**, that is dedicated to Exception handling.



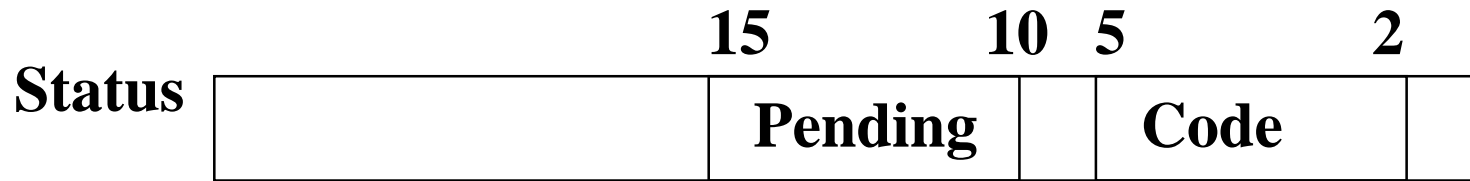
MIPS-2000 CP0

- **EPC** (Exception Program Counter): a 32-bit register used to hold the address of the affected instruction
 - * register 14 of coprocessor 0
- **Cause**: a register used to record the cause of the exception.
 - * In the MIPS architecture this register is 32 bits, though some bits are currently unused.
- **BadVAddr**: register contains memory address at which memory reference occurred
 - * when memory access exception occurs
 - * register 8 of coprocessor 0
- **Status**: interrupt mask and enable bits
 - * register 12 of coprocessor 0
- **PRId**: Processor Revision Identifier register

Additions to MIPS ISA

- Special coprocessor instruction:
 - ◆ `mfc0 rt, rd` # register `rd` in CP0 is loaded into register `rt`
 - ◆ `mtc0 rt, rd` # register `rt` is loaded into CP0 register `rd`
 - ◆ `rfe` # Return from exception, restore Interrupt mask and status register, (allow change in execution mode).
- TLB special instructions:
 - ◆ `tlbp` # The index register is loaded with the address of the TLB entry whose contents match the contents of `EntryHi` and `EntryLo` registers.
 - ◆ `tlbr` # Read Indexed TLB entry: The `EntryHi` and `EntryLo` registers are loaded with the content of the TLB pointed at by the TLB `Index` register
 - ◆ `tlbwi` # Write Indexed TLB entry: The `EntryHi` and `EntryLo` registers are stored in the TLB entry pointed at by the TLB `Index` register
 - ◆ `tlbwr` # Write random TLB entry: The `EntryHi` and `EntryLo` registers are stored in the TLB entry pointed at by the TLB `random` register

Details of Cause register



- **Pending interrupt** 5 hardware levels: bit set if interrupt occurs but not yet serviced
 - * handles cases when more than one interrupt occurs at same time, or records interrupt requests when interrupts disabled
- **Exception Code** encodes reasons for interrupt
 - * 0 (INT) => external interrupt
 - * 4 (ADDRL) => address error exception (load or instr fetch)
 - * 5 (ADDRS) => address error exception (store)
 - * 6 (IBUS) => bus error on instruction fetch
 - * 7 (DBUS) => bus error on data fetch
 - * 8 (Syscall) => Syscall exception
 - * 9 (BKPT) => Breakpoint exception
 - * 10 (RI) => Reserved Instruction exception
 - * 12 (OVF) => Arithmetic overflow exception

How are Exceptions Handled?

- **Machine must save the address of the offending instruction in the EPC (Exception Program Counter)**
- **Then transfer control to the Operating System (OS) at some specified address**
- **OS performs some action in response**
- **OS terminates program or returns (eventually) using EPC**
 - * **could return to different program (context switch)**
- **Exceptions are an “unscheduled procedure call” to a fixed location!**

How are Exceptions Handled?

- 2 types of exceptions in our current implementation
 - * undefined instruction
 - * arithmetic overflow
- Which Event caused Exception?
 - * Option 1 (used by MIPS): **Cause register** contains reason
 - * Option 2 Vectored interrupts: cause is part of the address.
 - addresses separated by 32 instructions
 - E.g.,
 - *

<u>Exception Type</u>	<u>Exception Vector Address</u>
Undefined instruction	C0 00 00 00
Arithmetic overflow	C0 00 00 20

 - Jump to appropriate address

What happens to Instruction with Exception?

- **Some problems could occur in the way the exceptions are handled.**
- **Example:**
 - * **in the case of arithmetic overflow, the instruction causing the overflow completes writing its result, because the overflow branch is in the state when the write completes.**
 - * **However, the architecture may define the instruction as having no effect if the instruction causes an exception; MIPS specifies this.**
- **Virtual memory exceptions must prevent handler from changing the machine state.**
- **This aspect of handling exceptions becomes complex and potentially limits performance => why it is hard**