

INPUT/OUTPUT (I/O) SUBSYSTEMS

- Overview of I/O performance measurement and analysis
- Processor interface issues
- Buses
- Types and characteristics of I/O devices
 - ▷ Hard disk storage
 - ▷ Network interfaces
- I/O system design

MOTIVATION FOR STUDYING I/O

- CPU performance improves by 50% to 100% per year
- I/O systems' performance improvements are limited by physics (in some cases)
 - ▷ Mechanical delays (disk drives):
Latency improvement is of order 5% per year
 - ▷ Electrical and optical phenomena (dispersion, attenuation, crosstalk):
Improvement is 5% to 25% per year
- Amdahl's law implies that, sooner or later, most of the latency will be due to the part that is hardest to improve
 - ▷ Given: 10% of instructions perform I/O, CPU is 10x faster
 - ▷ Improvement is only 5x \Rightarrow lose 50% of improvement
- **I/O bottleneck lowers the value of CPU improvements**
 - ▷ As technology evolves, a diminishing fraction of total latency is due to the CPU

I/O PERFORMANCE METRICS

- **Bandwidth** (bits or bytes per second):
 - ▷ Peak
 - ▷ Sustained
 - ▷ Useful for buses and networks
- **Throughput** (I/O processes per second)
 - ▷ Useful for file serving and transaction processing
- **Latency** = total time for an I/O process from start to finish
 - ▷ Most important to users
 - Latency too great \Rightarrow user loses train of thought
 - Latency
 - = controller time + wait time + $\frac{\text{no. bytes}}{\text{bandwidth}}$ + CPU time
 - overlap

PROCESSOR INTERFACE ISSUES

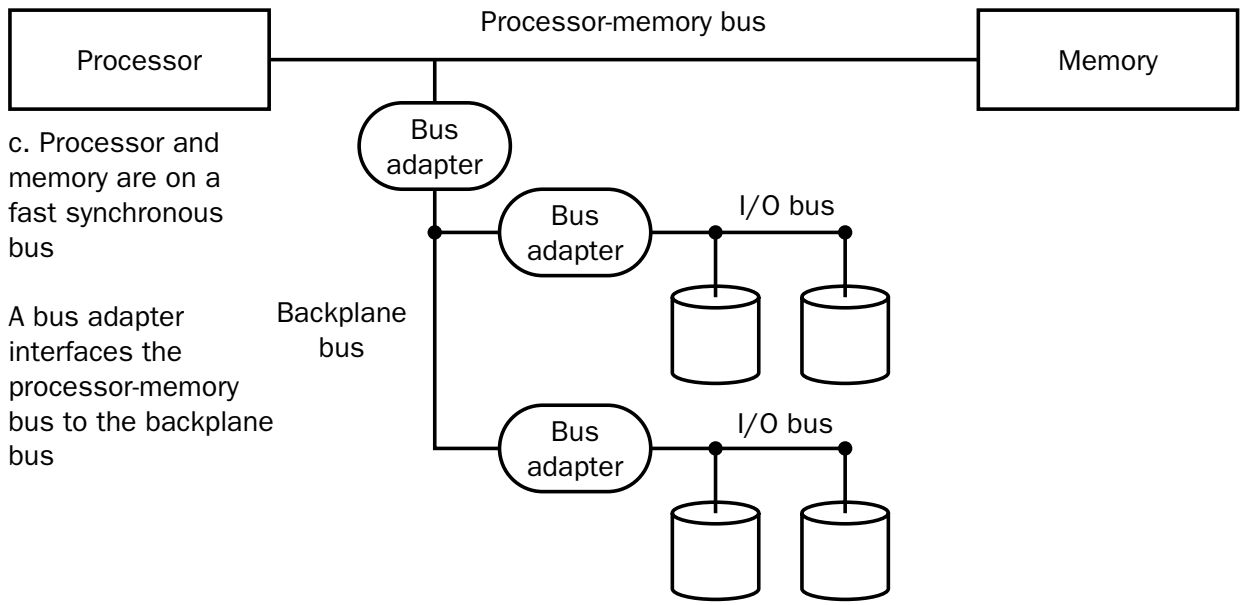
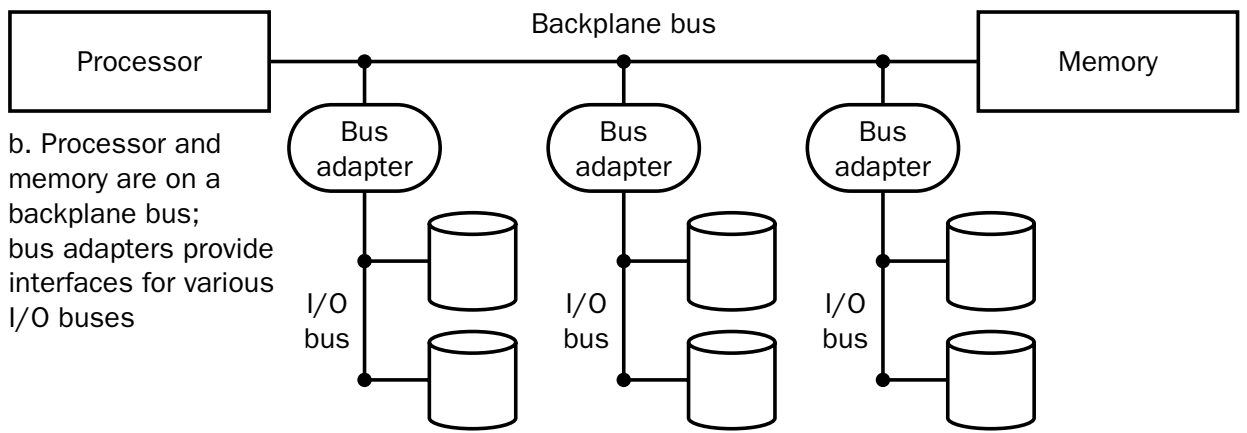
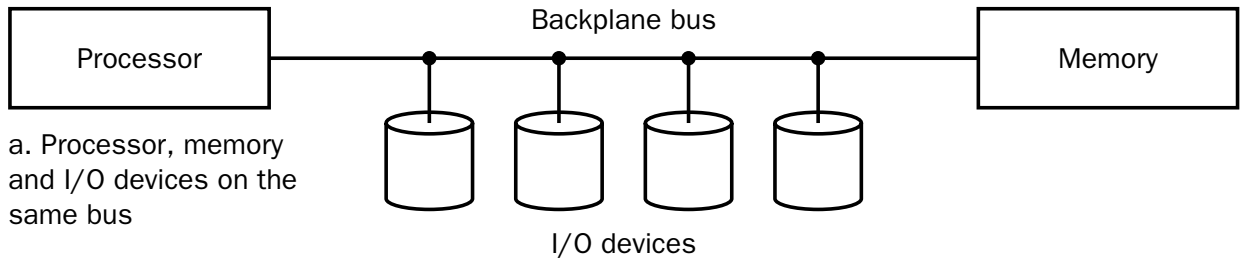
- Interconnections
 - ▷ Buses
- Processor interface
 - ▷ Interrupts
 - ▷ Memory-mapped I/O
- I/O control structures
 - ▷ Polling
 - ▷ Interrupts
 - ▷ DMA
 - ▷ I/O controllers
 - ▷ I/O processors
- Capacity, access time, bandwidth, cost

BUSES

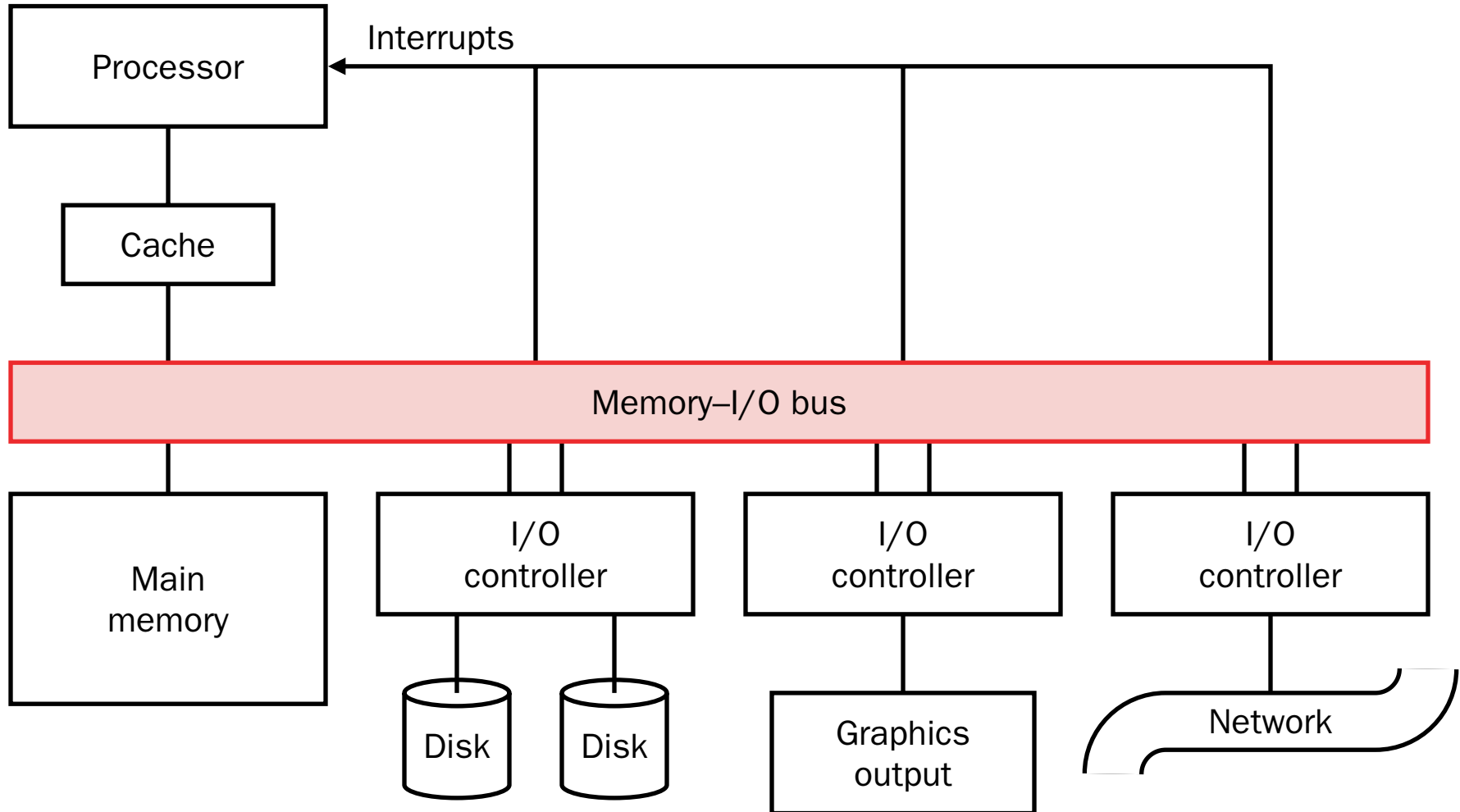
- **Bus:** A communication link shared by multiple subsystems
 - ▷ Physically: Parallel conductors (traces on die or PC board; cable)
 - ▷ Advantages:
 - Low cost (compared to point-to-point wiring)
 - Versatility of interconnections
 - ▷ Disadvantages:
 - Electrical problems \Rightarrow short length
 - ◇ Bus skew
 - ◇ Dispersion
 - ◇ Crosstalk
 - Shared resource \Rightarrow contention
 - ▷ Organization:
 - Control lines to signal & acknowledge requests
 - Data lines to carry addresses, data or commands

PROCESSOR-I/O INTERFACE BUS TYPES

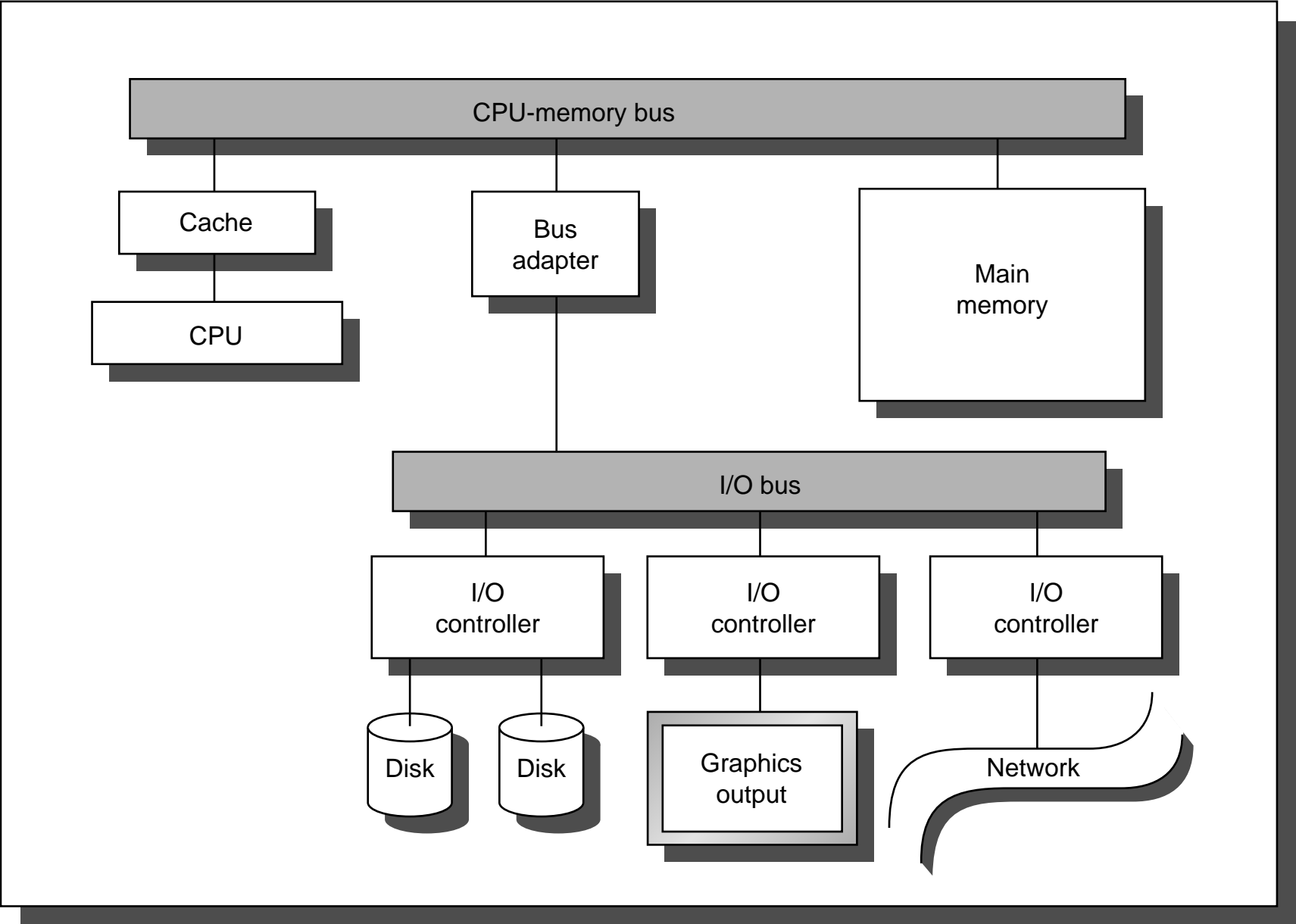
- Backplane bus
 - ▷ Processor, memory and I/O devices coexist on the same bus
 - ▷ In olden times, often built into the backplane of a computer
 - An interconnection structure that was part of the chassis
 - ▷ Processor architecture includes explicit I/O instructions (IN, OUT)
 - ▷ Standard backplane buses: VMEbus, Multibus, NuBus, PCI, ISA (Industry Standard Architecture) bus
- I/O bus
 - ▷ Examples: IDE, SCSI



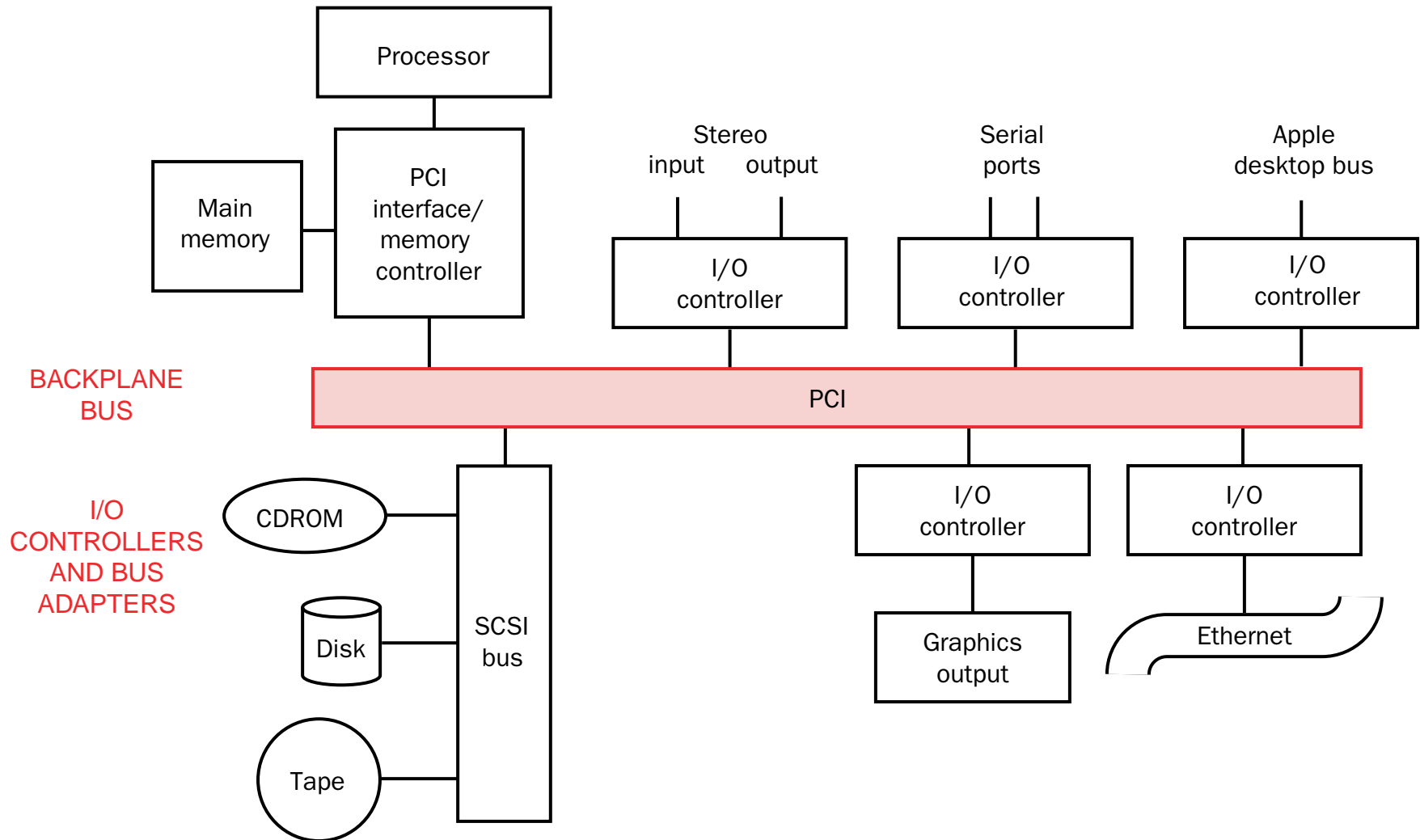
I/O SYSTEM USING ONLY A BACKPLANE BUS



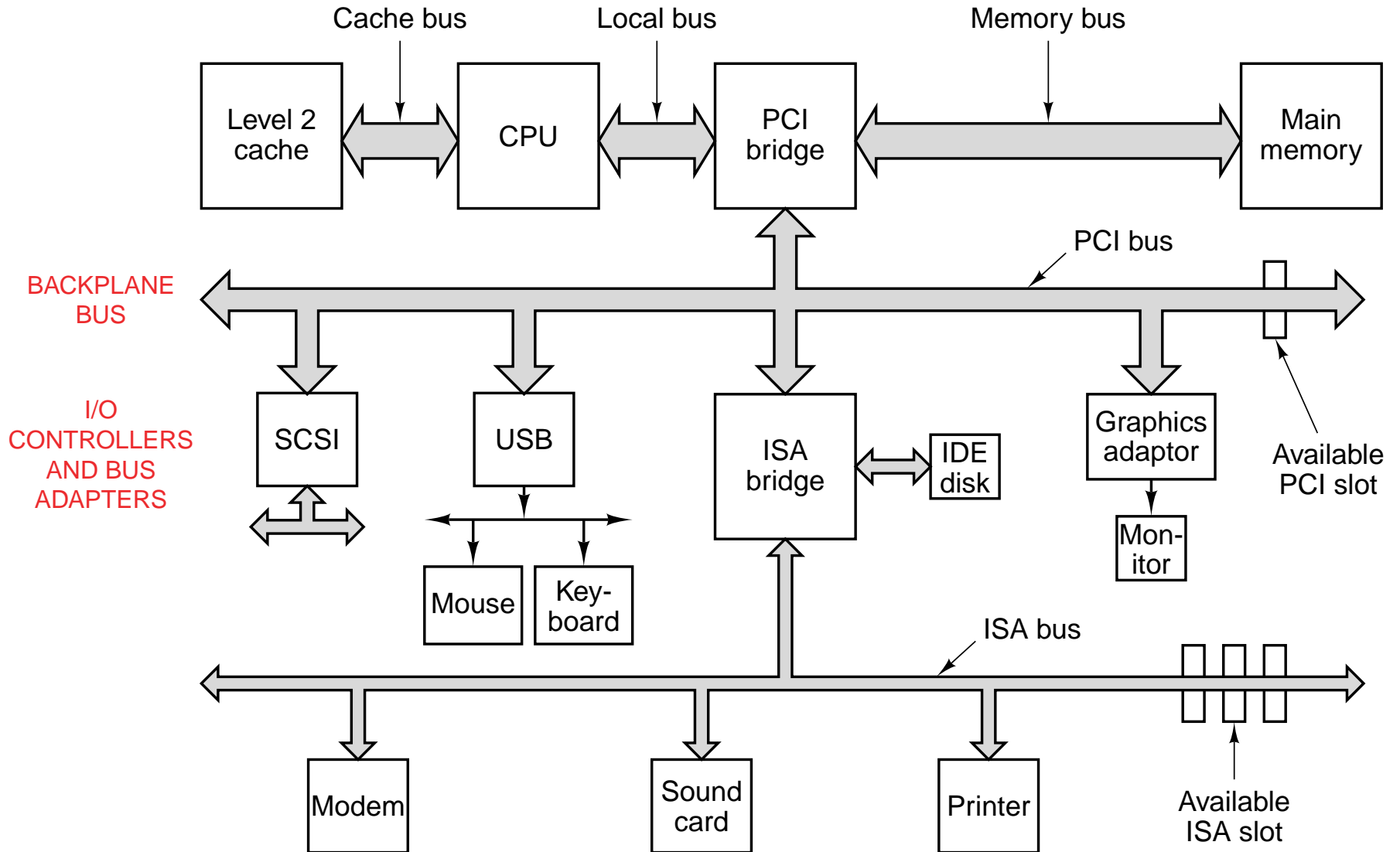
I/O SYSTEM USING AN I/O BUS



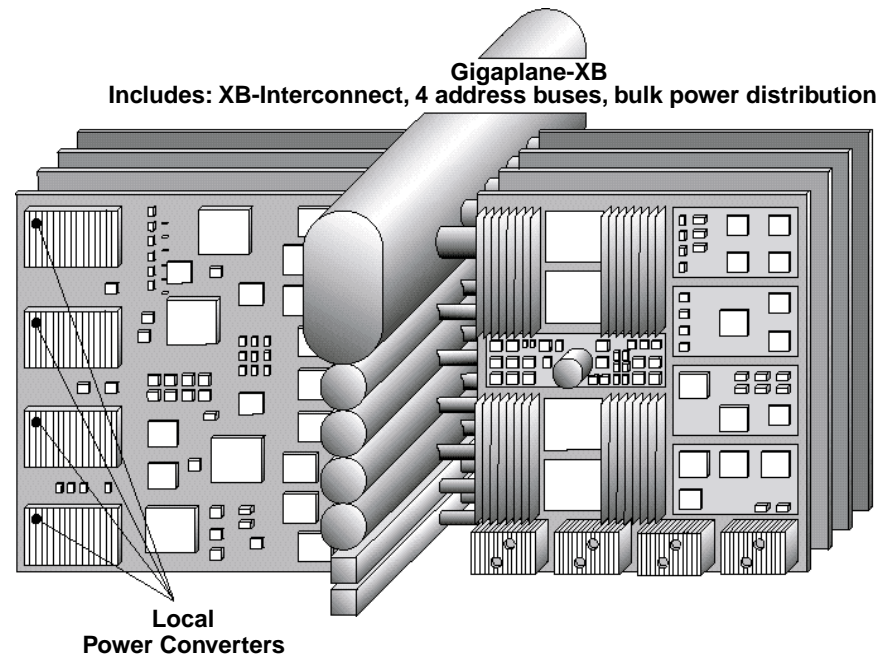
MACINTOSH 72xx I/O SYSTEM



PENTIUM II I/O SYSTEM



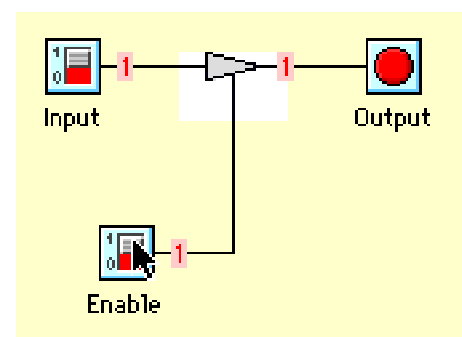
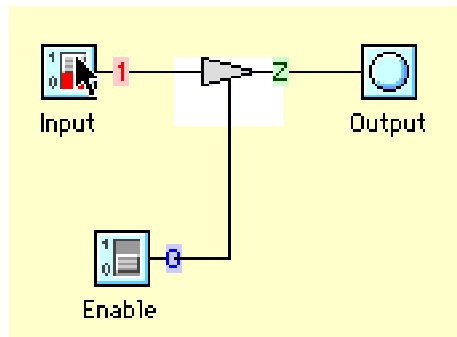
Enterprise 10000 hardware architecture



- Data is packet-switched using a crossbar
- Addresses are broadcast

3-STATE BUFFER

- A **3-state buffer** has 2 inputs and 1 output
 - ▷ Enable asserted: Output = input (state is either 0 or 1)
 - ▷ Enable deasserted: High-impedance state (denoted \times or Z)
 - Output can be driven by another device
 - ▷ Equivalent to a mechanical switch

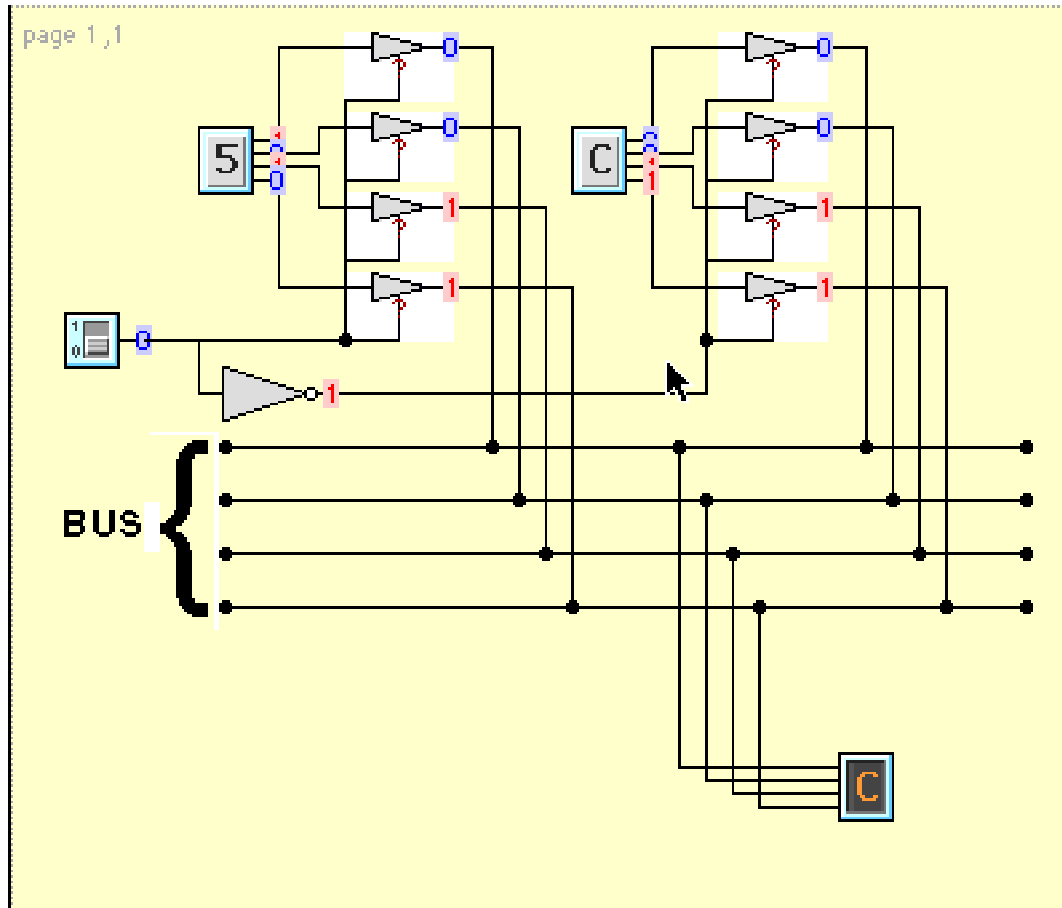


EXCITATION TABLE FOR 3-STATE BUFFER

- A tristate buffer has 3 possible output values:
 - ▷ Asserted
 - ▷ Deasserted
 - ▷ High impedance (floating)

enable	in	out
0	0	Z
0	1	Z
1	0	0
1	1	1

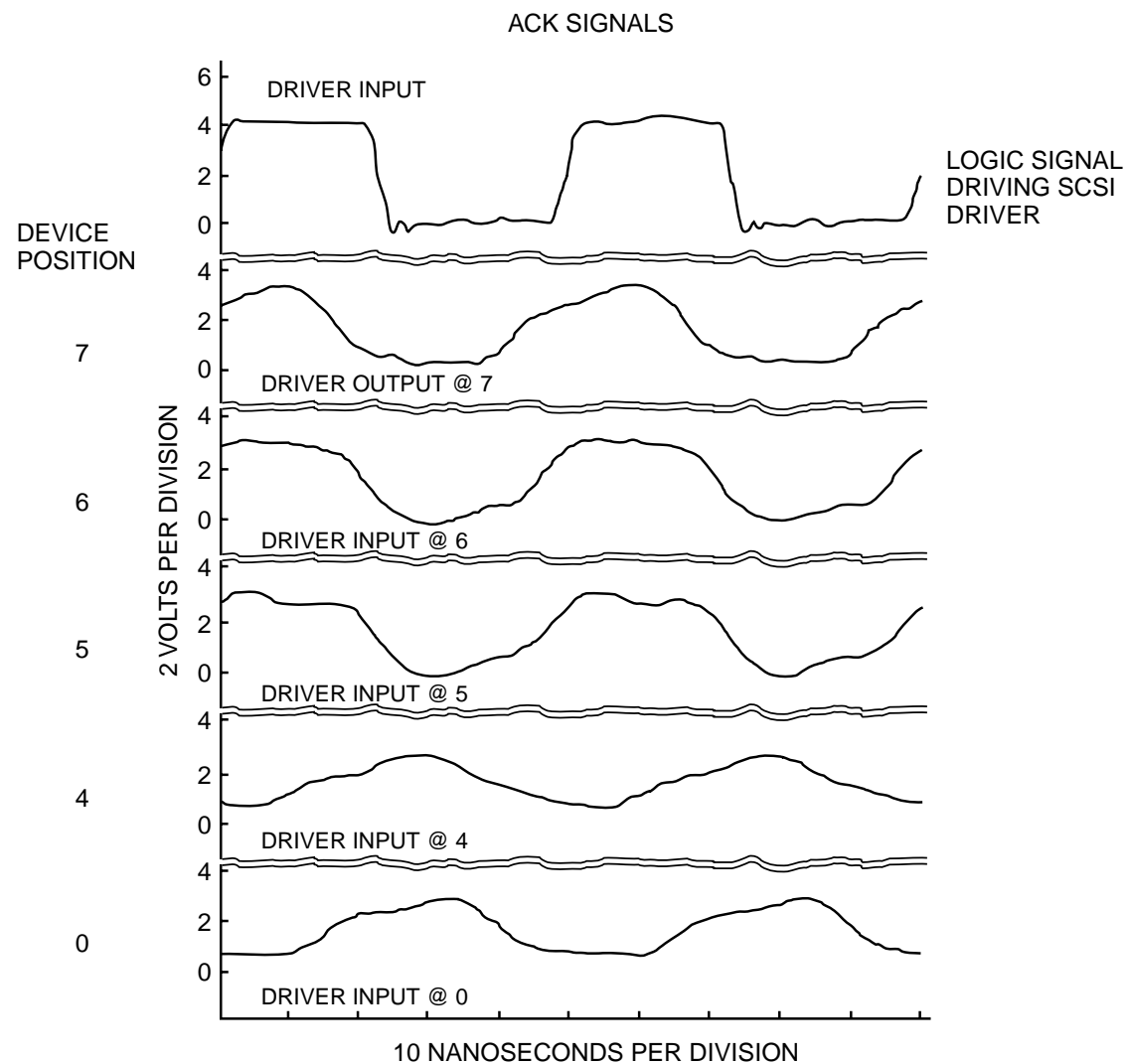
USE OF TRISTATES TO ENABLE/DISABLE BUS ACCESS



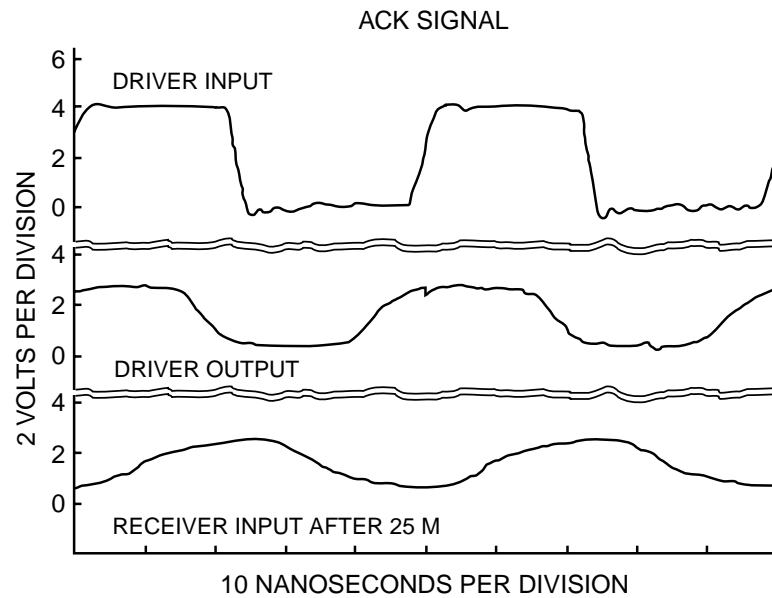
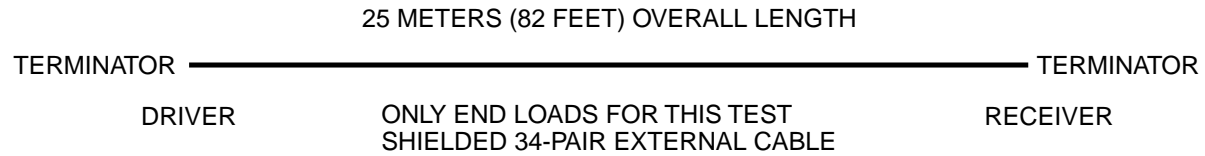
BUS DESIGN CONSTRAINTS

- Laws of physics limit bus speeds
 - ▷ Transmission speed \leq speed of light
 - ▷ Crosstalk
 - Occurs because:
 - ◇ A time-varying voltage on a conductor induces a charge $q_2 = C_{12} v_1$ on another, parallel conductor
 - ◇ A time-varying current in a conductor induces a voltage $v_2 = L_{12} di_1/dt$ in another, parallel conductor
 - Limits bus clock frequency
 - Can be reduced by:
 - ◇ Grounding alternate conductors
 - ◇ Abandoning the bus concept and using twisted-pair, point-to-point connections (Seymour Cray)
 - EMI & reflections limit number of devices connected to bus
- Real estate on die or PC board limits number of lines

ACK SIGNALS ON COMPLEX ULTRA-SCSI CHAIN



ACK SIGNALS ON POINT-TO-POINT ULTRA-SCSI BUS



ASYNCHRONOUS vs. SYNCHRONOUS BUSES

- Bus communication protocol: Specification of sequence of events and timing requirements for transferring information on a bus
- Asynchronous bus transfers:
 - ▷ Certain conductors on the bus are control lines
 - ▷ Signals on the control lines control the sequence of events
- Synchronous bus transfers:
 - ▷ Events are sequenced relative to a master clock signal
 - ▷ Once a certain kind of transfer has been initiated, no further command signaling is necessary to control the transfer

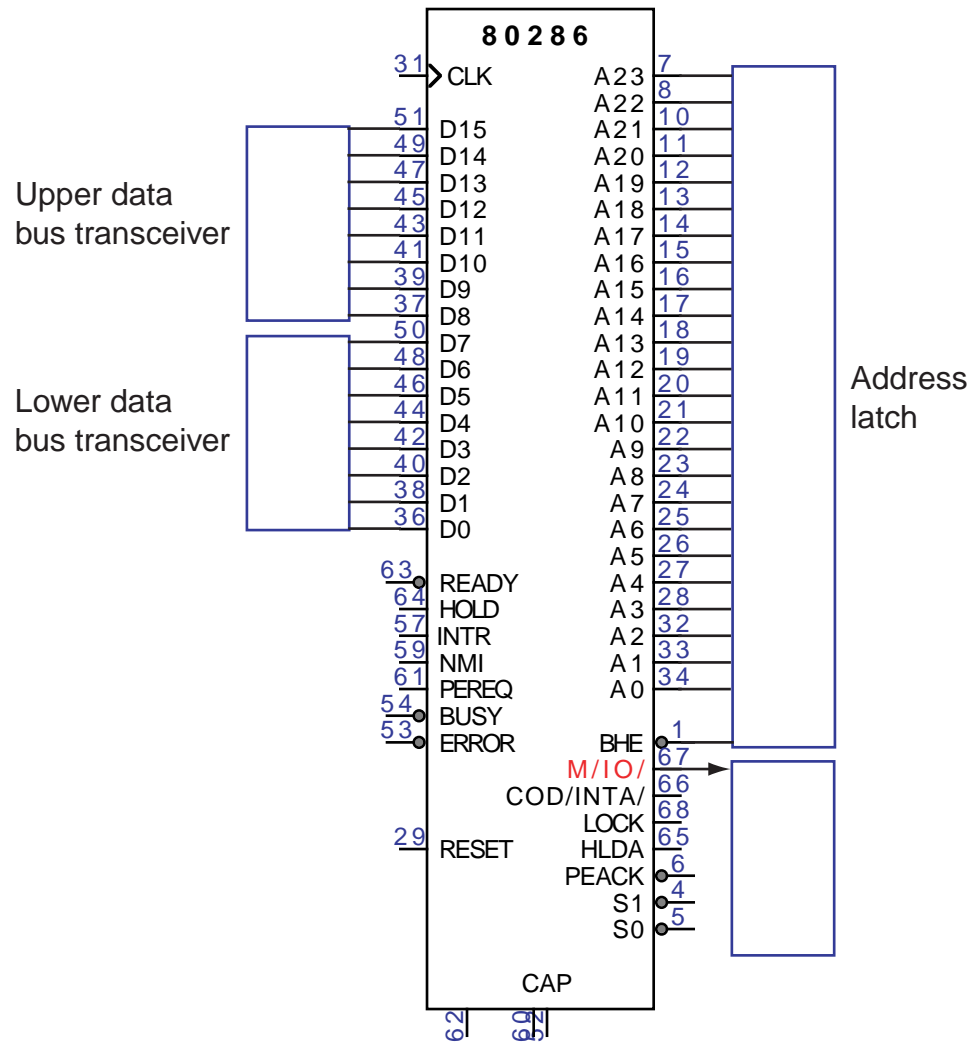
SYNCHRONOUS BUSES

- Bus clock is phase-locked to processor clock
 - ▷ Bus clock frequency = $\frac{1}{n} \times$ processor clock frequency ($n = 1$ to 6)
 - ▷ Clock signal is carried on a control line
 - ▷ Communications protocol defined with reference to bus clock signal
 - ▷ Local bus (*e.g.*, VESA Local Bus):
 - Extends the processor's bus control signals
 - May connect processor to L2 cache
 - May connect processor and memory to high-speed I/O devices
- Advantages:
 - ▷ Fast & wide
 - ▷ Simple logic (finite state machine)
- Disadvantages:
 - ▷ Must be short (bus skew; attenuation; crosstalk)
 - ▷ All devices must run at same frequency

80286 – PENTIUM I/O

- Separate I/O and memory address spaces
 - ▷ Since the 8086, I/O or memory access is signaled by M/I0# (memory access if high, I/O if low)
 - For MOVE (memory–CPU copy), M/I0# is high
 - For IN or OUT (I/O), M/I0# is low
 - M/I0# is a processor signal that does not appear on the ISA bus
 - Instead, M/I0# is an input to the bus controller
 - ▷ I/O address space is 0x0000 to 0xffff

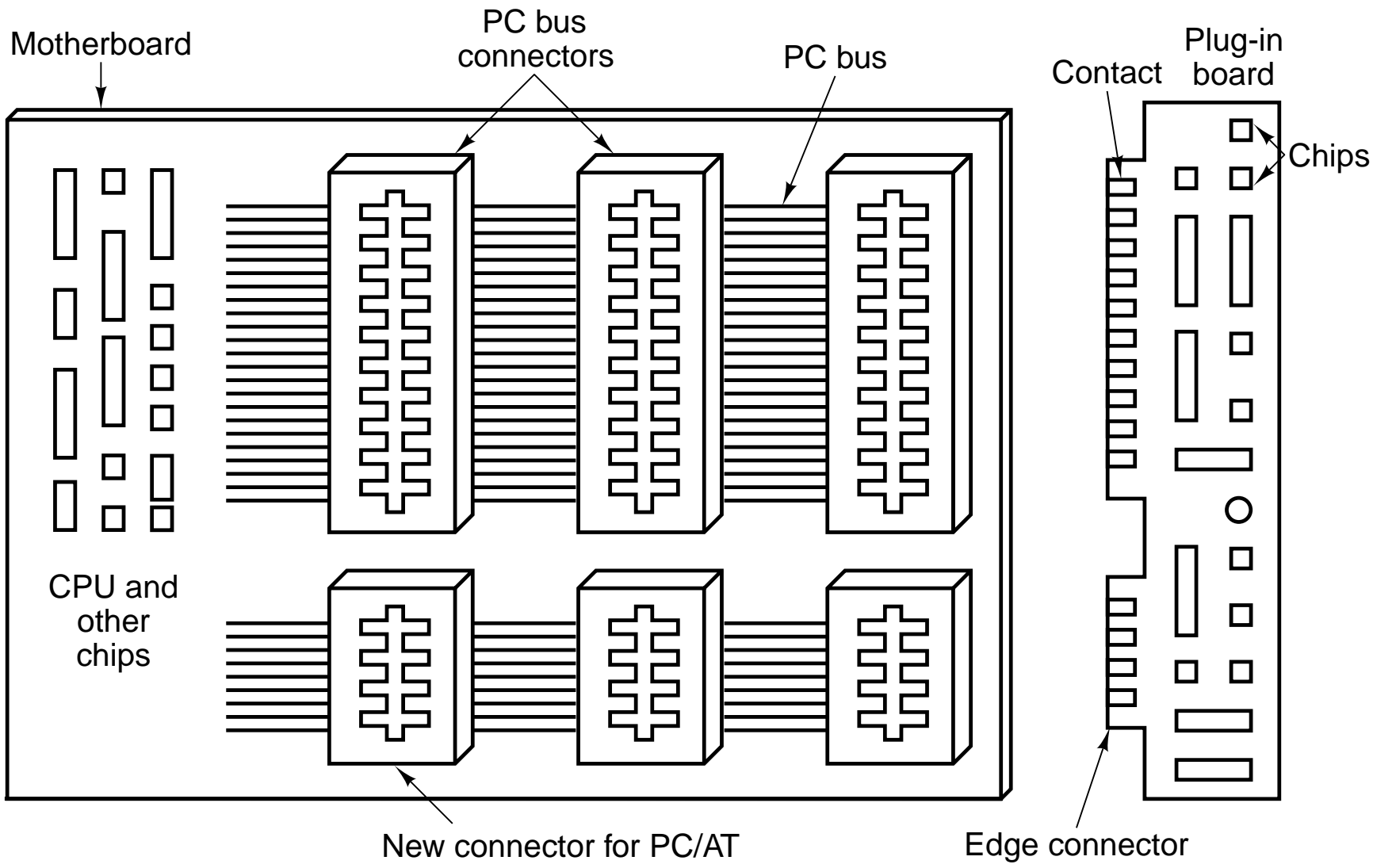
80286 SIGNALS



ISA BUS

- ISA \equiv Industry Standard Architecture
 - ▷ Synchronous
 - ▷ Industry response to IBM's MicroChannel architecture
 - ▷ Uses both the PC/AT and the IBM PC bus standards
 - Interface cards have 2 sets of connectors
 - PC bus: 8 data lines, 20 address lines
 - ISA bus: 16 data lines, 24 address lines; bus frequency 8.33 MHz
Maximum possible throughput: $2 \text{ bytes} \times 8.33 \text{ MHz} = 16.67 \text{ MB/s}$
 - ▷ Separate I/O and memory address spaces
 - Since the 8085, I/O or memory access is signaled by IO/M# (I/O if high, memory access if low)
 - ◇ For MOVE (memory–CPU copy), IO/M# is high
 - ◇ For IN or OUT (I/O), IO/M# is low
 - I/O address space is 0x0000 to 0xffff

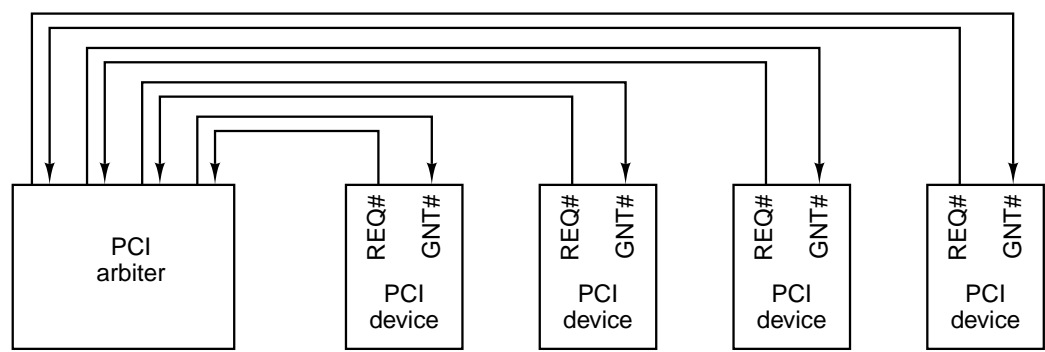
ISA BUS CONNECTORS



PCI BUS

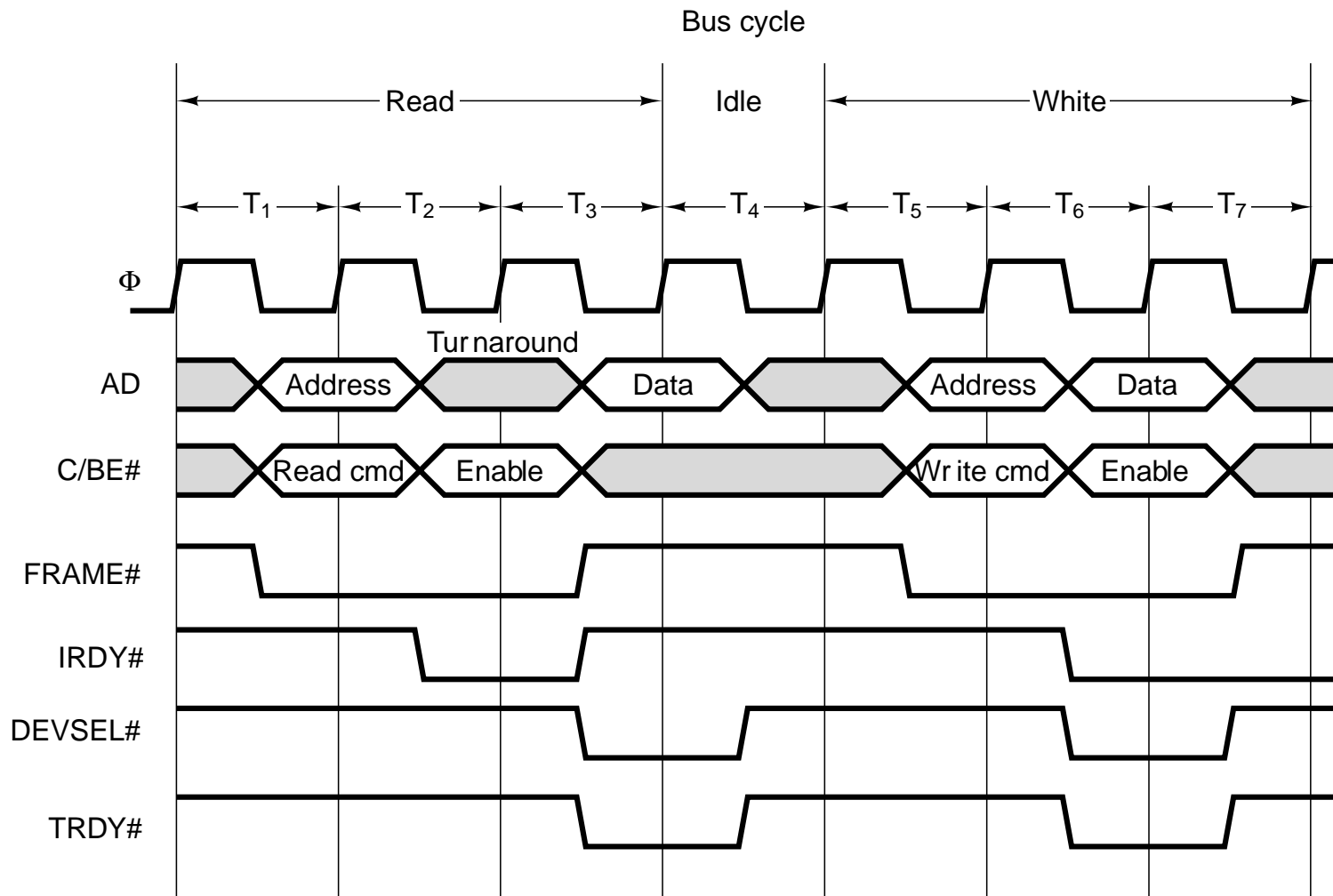
- PCI \equiv Peripheral Component Interconnect
 - ▷ Synchronous
 - ▷ PCI 1.0: Clock frequency 33 MHz, 32-bit-wide data path
 - ▷ PCI 2.1: Clock frequency 66 MHz, 64-bit-wide data path
 - Maximum theoretical bandwidth:
 $8 \text{ bytes} \times 66 \text{ MHz} = 528 \text{ MB/s}$
 - ▷ Transactions are negative-edge-triggered
 - ▷ Address and data lines are multiplexed
 - ▷ Bus arbiter usually built into the chipset
 - ▷ Every PCI device has a 256-byte configuration address space that is readable by other devices \Rightarrow Plug 'n Play
- PCI cards
 - ▷ Options include voltage (5 V vs. 3.3 V), width (32 bits/120 pins vs. 64 bits/184 pins) and frequency (33 vs. 66 MHz)

PCI BUS ARBITER



Tanenbaum, *Structured Computer Organization*

PCI BUS TIMING FOR READ AND WRITE CYCLES



PCI BUS SIGNALS

MANDATORY PCI BUS SIGNALS

Signal	Lines	Master	Slave	Description
CLK	1			Clock (33 MHz or 66 MHz)
AD	32	×	×	Multiplexed address and data lines
PAR	1	×		Address or data parity bit
C/BE	4	×		Bus command/bit map for bytes enabled
FRAME#	1	×		Indicates that AD and C/BE are asserted
IRDY#	1	×		Read: master will accept; write: data present
IDSEL	1	×		Select configuration space instead of memory
DEVSEL#	1		×	Slave has decoded its address and is listening
TRDY#	1		×	Read: data present; write: slave will accept
STOP#	1		×	Slave wants to stop transaction immediately
PERR#	1			Data parity error detected by receiver
SERR#	1			Address parity error or system error detected
REQ#	1			Bus arbitration: request for bus ownership
GNT#	1			Bus arbitration: grant of bus ownership
RST#	1			Reset the system and all devices

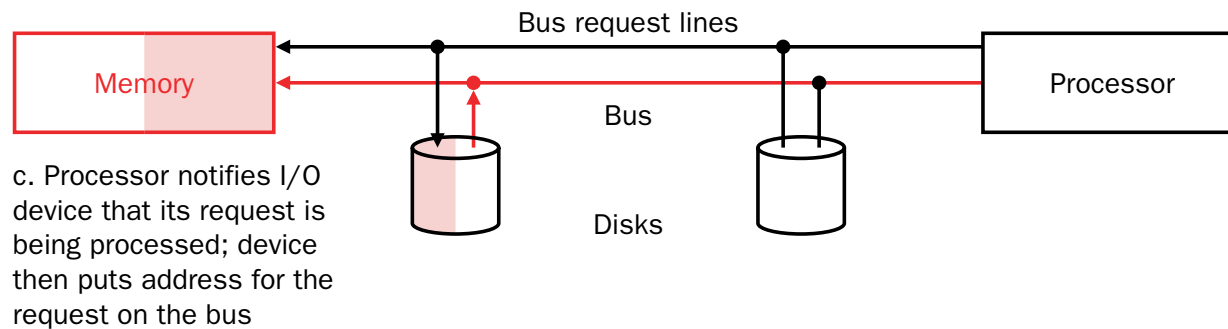
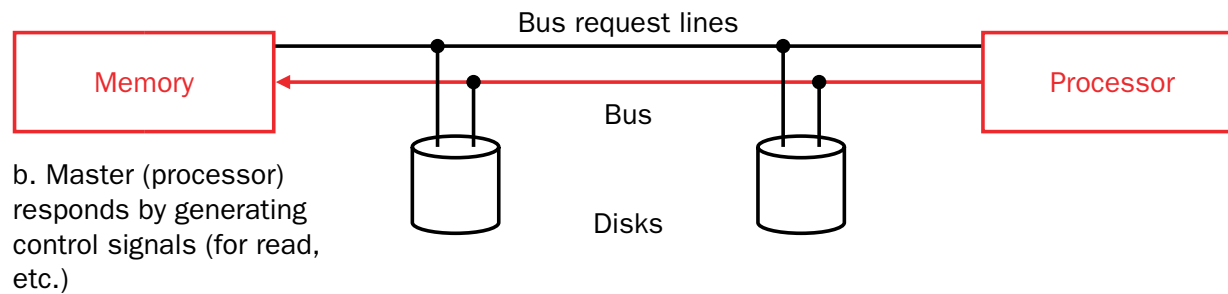
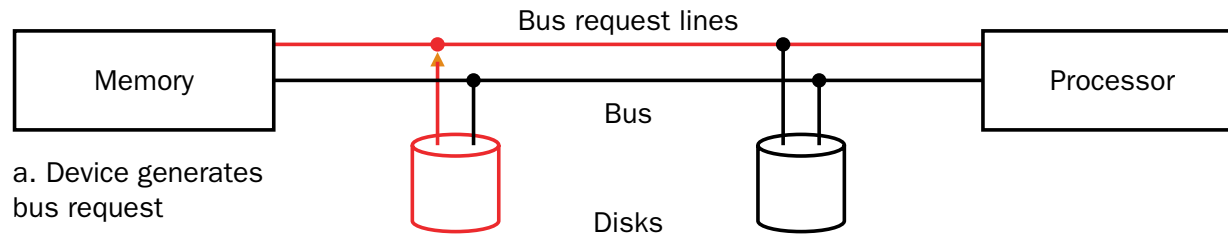
OPTIONAL PCI BUS SIGNALS

Sign	Lines	Master	Slave	Description
REQ64#	1	×		Request to run a 64-bit transaction
ACK64#	1		×	Permission is granted for a 64-bit transaction
AD	32	×		Additional 32 bits of address or data
PAR64	1	×		Parity for the extra 32 address/data bits
C/BE#	4	×		Additional 4 bits for byte enables
LOCK	1	×		Lock the bus to allow multiple transactions
SBO#	1			Hit on a remote cache (for a multiprocessor)
SDONE	1			Snooping done (for a multiprocessor)
INTx	4			Request an interrupt
JTAG	5			IEEE 1149.1 JTAG test signals
M66EN	1			Wired to power or ground (66 MHz or 33 MHz)

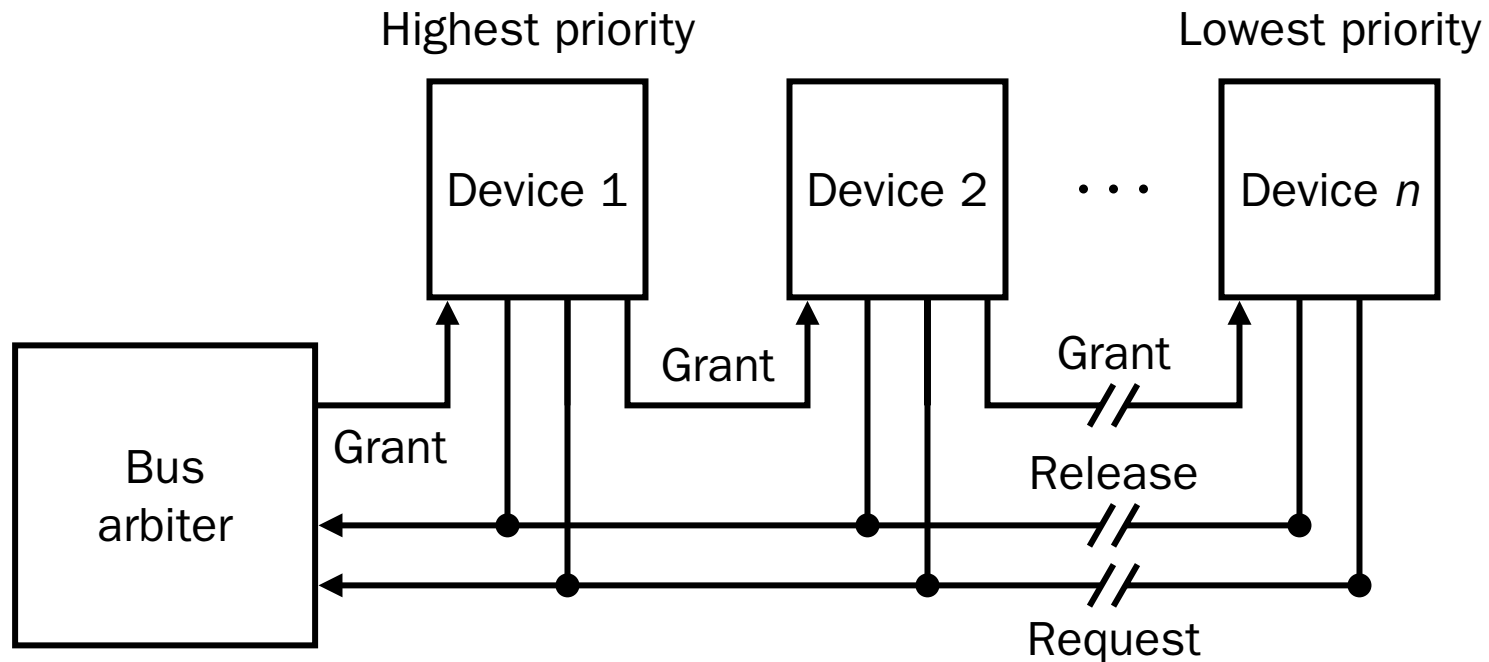
OBTAINING BUS ACCESS

- Goal: Give every device fair access
- Method: Use **bus masters**
 - ▷ A master enables bus access for one or more devices (by enabling/disabling tristate buffers)
 - ▷ Single bus master can be a bottleneck
 - ▷ Multiple masters require **arbitration**
 - Every device has a priority (IRQ number, SCSI ID, ...)
 - Extra control lines needed for bus request/access
 - ▷ Arbitration methods:
 - Centralized & parallel (SCSI)
 - Daisy chain (VMEbus)
 - Distributed arbitration using self-selection (NuBus)
 - Distributed arbitration using collision detection (Ethernet)

A BUS TRANSACTION WITH A SINGLE MASTER



DAISY CHAIN



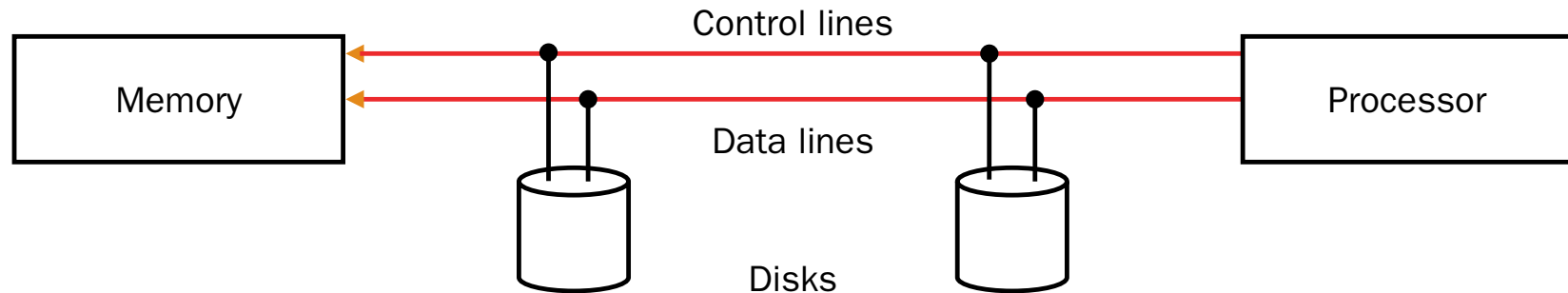
A daisy chain bus uses a bus grant line that chains through each device from highest to lowest priority. The protocol is:

1. Signal on the request line
2. Wait for a low-to-high transition on the grant line (indicates reassignment)
3. Intercept the grant signal and stop asserting the request line
4. Use the bus
5. Signal that the bus is no longer required by asserting the release line

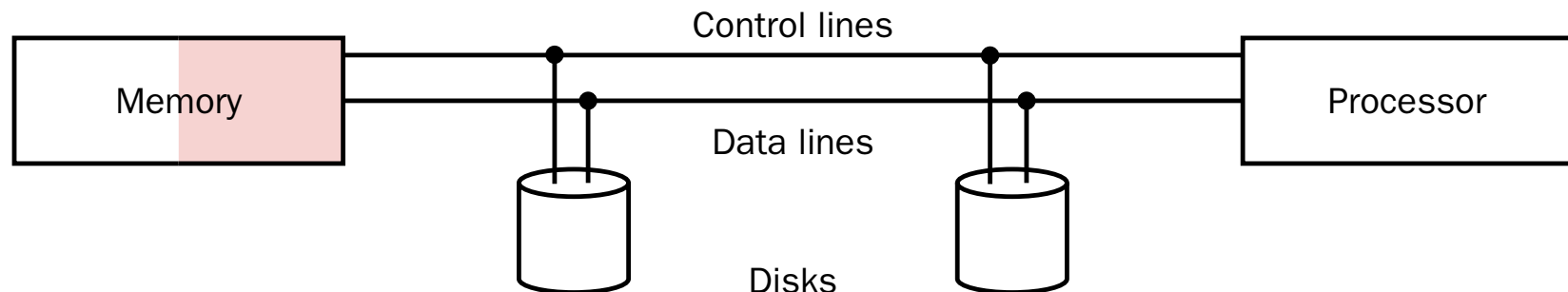
ASYNCHRONOUS BUSES

- Not clocked
 - ▷ Can accommodate many kinds of devices (disk, tape, scanner, ...)
- Data transfer controlled with handshaking protocol on dedicated control lines; represent with a finite state machine for each device
- Example (SCSI-1 bus):
 - ▷ Bus controller asserts **Sel** (select device) and transmits device ID
 - ▷ Selected device responds with **Ack**
 - ▷ Controller asserts **Cmd** (command), **Msg** (message), and **Req** (request a data transfer) signals, then transmits command bytes
 - ▷ Device responds to each byte with **Ack**
 - ▷ Controller deasserts **Cmd**, asserts **I/O**, then transmits data bytes
 - ▷ Device responds to each byte with **Ack**

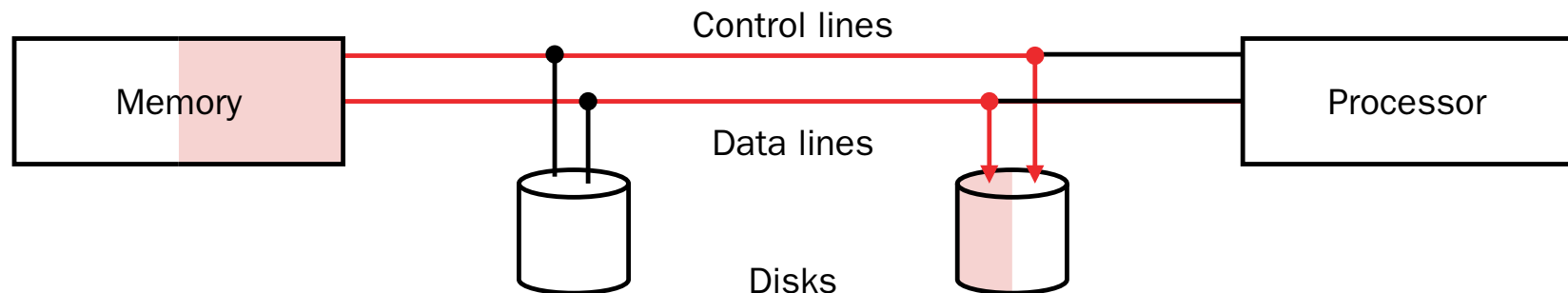
STEPS OF AN ASYNCHRONOUS OUTPUT OPERATION



a. Initiation of a read operation from memory. Control lines: *Read* command; Data lines: Address

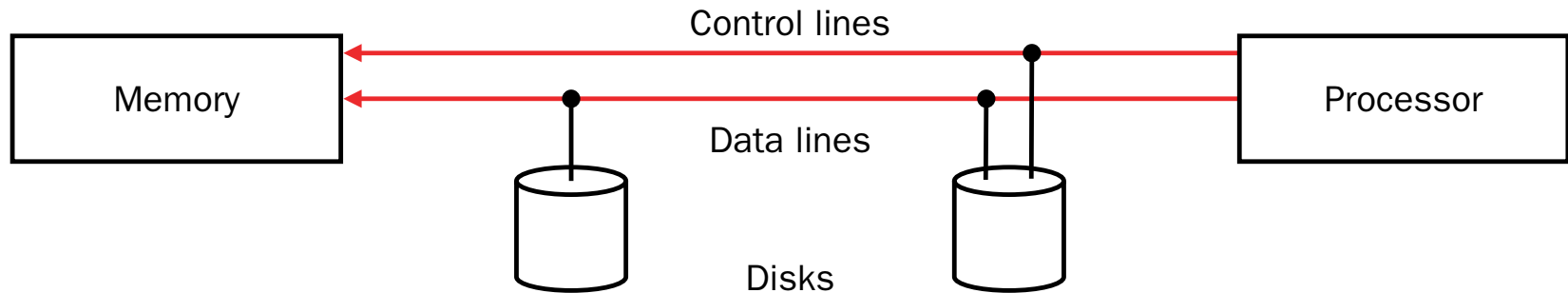


b. Memory access

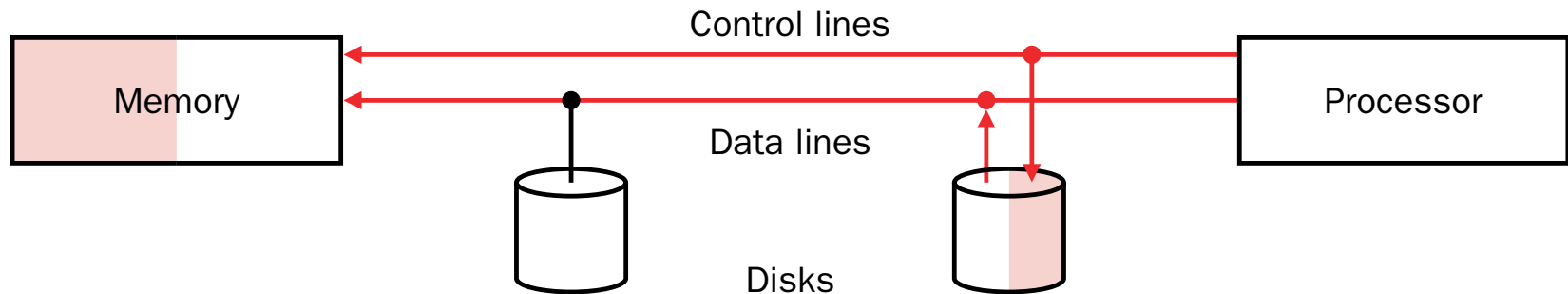


c. Memory puts the data on the data lines of the bus and uses the control lines to signal the I/O device that the data is available

STEPS OF AN ASYNCHRONOUS INPUT OPERATION

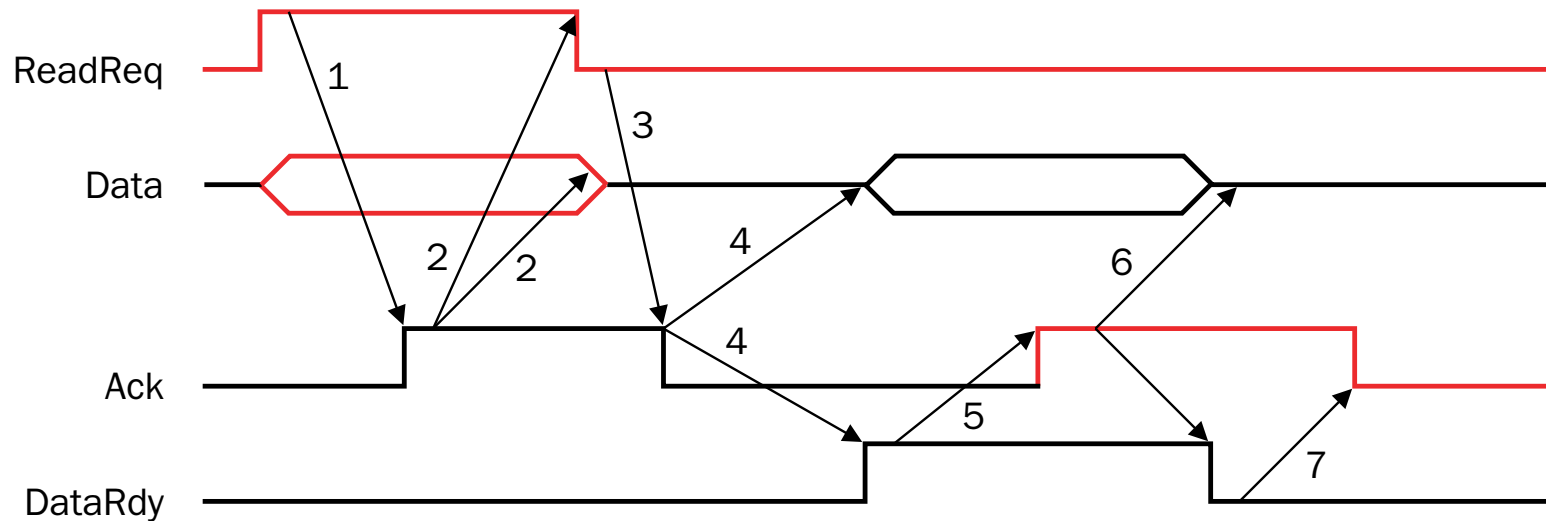


a. Control lines: *Write* request to memory; Data lines: Address



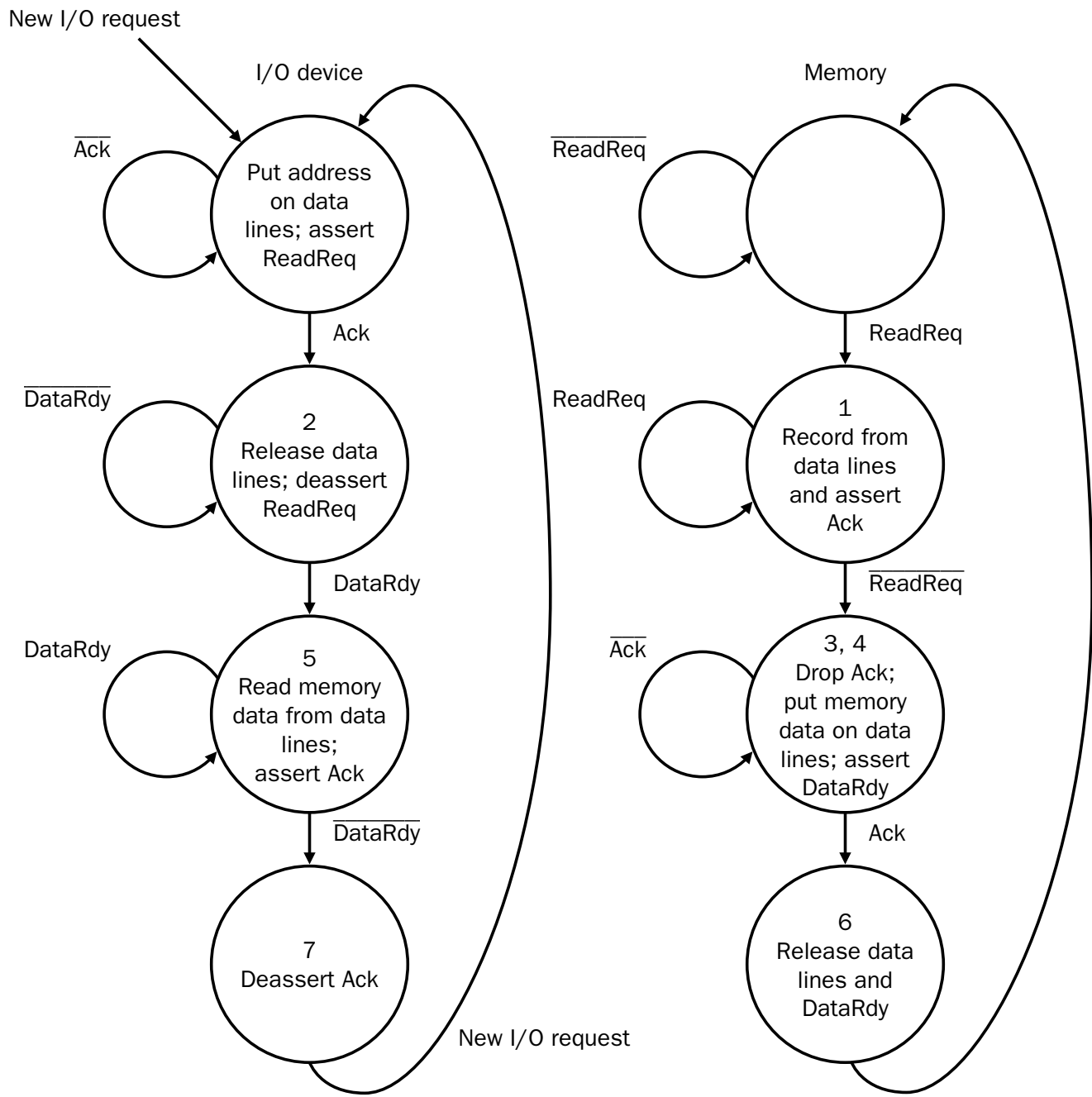
b. Memory signals the device that it is ready; Data is transferred

ASYNCHRONOUS BUS HANDSHAKING PROTOCOL



1. When memory sees ReadReq asserted, it reads the address from the data bus and asserts Ack
2. I/O device sees Ack asserted, releases ReadReq and data lines
3. Memory sees ReadReq deasserted, drops Ack to acknowledge ReadReq
4. Memory puts requested data on the data lines, asserts DataRdy
5. I/O device sees DataRdy, reads data, signals that it has seen the data by asserting Ack
6. Memory sees Ack, drops DataRdy, releases data lines
7. I/O device sees DataRdy deasserted, drops Ack to signal end of transmission

— I/O device
— Memory



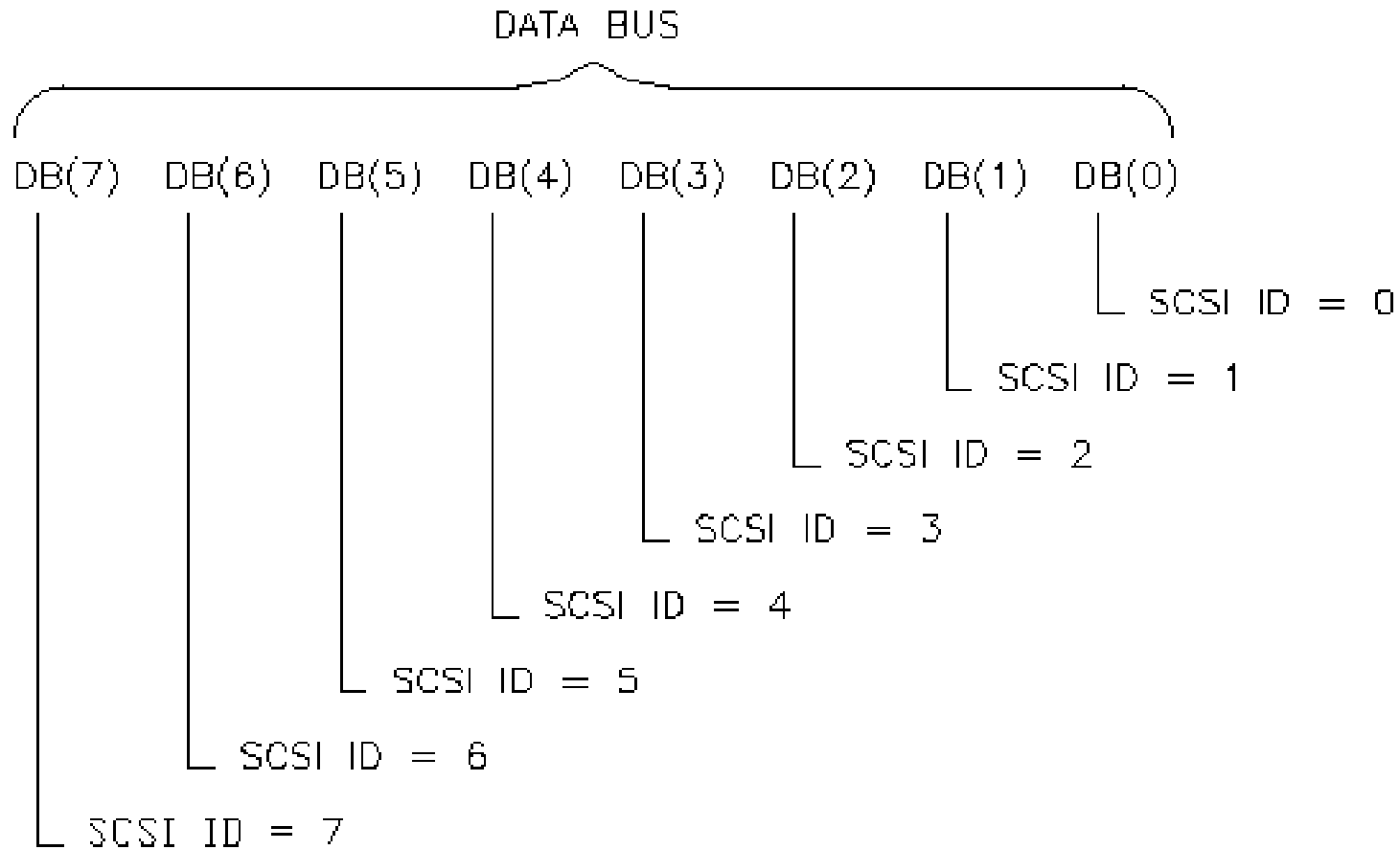
SCSI-1: AN ASYNCHRONOUS BUS (1)

- SCSI := Small Computer System Interface
 - ▷ Many “standard” implementations
 - ▷ Can connect many different kinds of devices:
 - Logic board
 - Hard drive
 - CD-ROM drive
 - Tape drive
 - Scanner
 - ▷ Controller chip on logic board or plug-in
 - ▷ Controller is connected by cable to internal or peripheral devices
 - ▷ Devices are daisy-chained
 - ▷ Device ID is set by hardware switches

SCSI-1: AN ASYNCHRONOUS BUS (2)

- SCSI-1 bus configuration
 - ▷ Peripheral SCSI-1 devices are connected by cable
 - ▷ Each bit of a data byte is transferred on a separate wire (line) of the cable
 - ▷ Each device must have a unique ID number between 0 and 7
 - The ID is signaled by asserting one of the lines DB(0) – DB(7)
 - In case of contention, the device with the highest ID wins
 - The logic board has ID 7, so it always wins

SCSI ID BITS

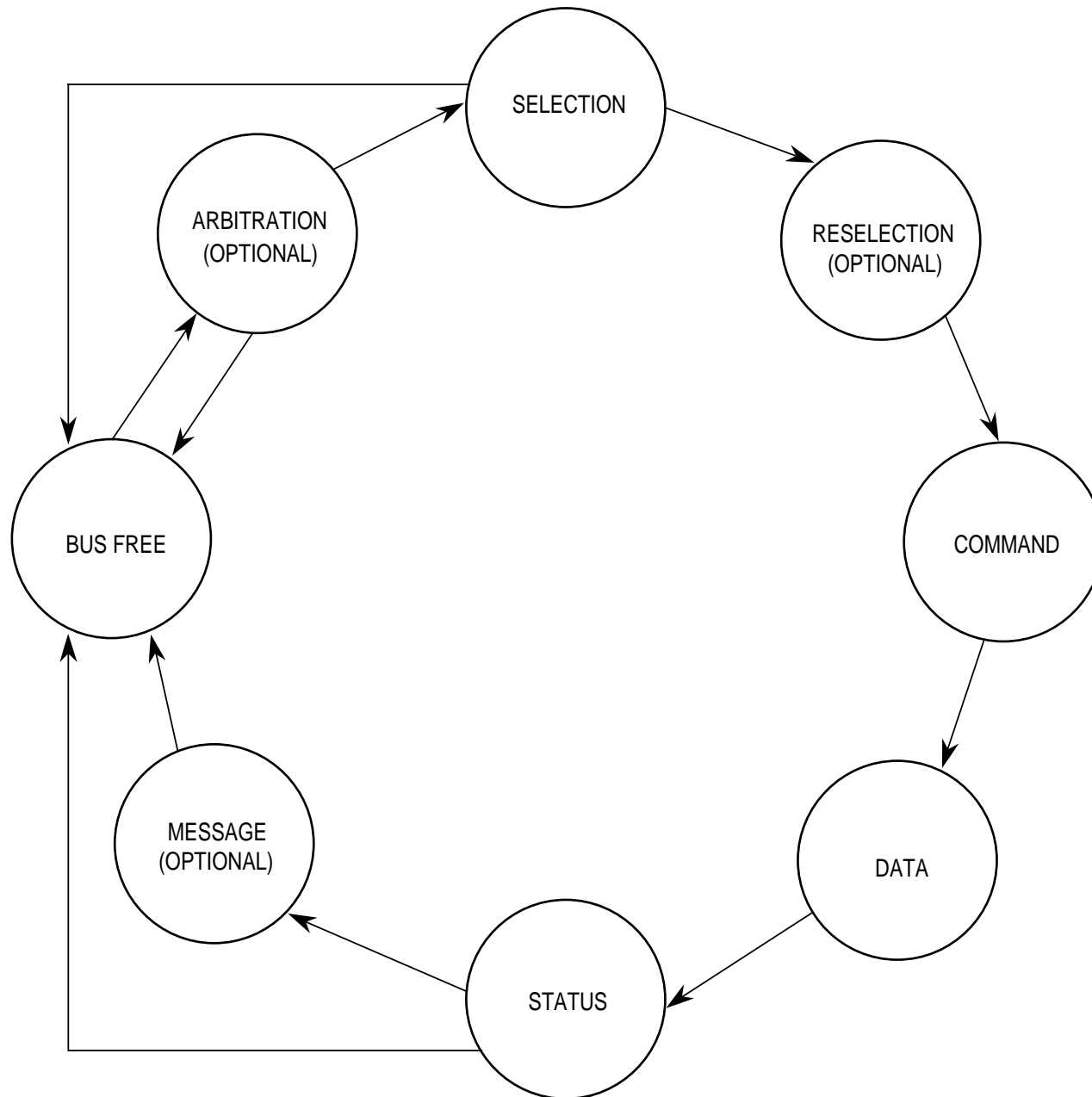


SCSI-1: AN ASYNCHRONOUS BUS (3)

- SCSI signaling sequence for data transfer
 - ▷ Controller broadcasts SEL (select) signal on pin 44 and the ID number on one of the data lines
 - ▷ Device selected responds with ACK (acknowledge) signal on pin 48 (handshake)
 - ▷ Controller sends REQ (request) signal on pin 48 to order device to perform a task (such as transferring a data byte)
 - ▷ Command bytes are transferred on the data bus
 - ▷ A handshake must take place for each data byte transferred

SCSI Bus Signals

Signal	Driven By	Signal Explanation
DB0–DB7	Initiator/Target	8-Bit Bidirectional Data Bus.
DBP	Initiator/Target	Data-Bus Parity Line. Optional.
ATN	Initiator	Attention. Used to send a message to the target when it controls the bus.
BSY	Initiator/Target	Busy. Indicates that the bus is unavailable for use.
ACK	Initiator	Acknowledge. Used by the initiator for handshaking.
RST	Any Device	Reset. Used to initiate a bus-free phase.
MSG	Target	Driven by the target to indicate that the current transfer is a message.
SEL	Initiator	Select. Used by the initiator to select a target before command execution. Also used by the target to reconnect when the reselection phase is implemented.
C/D	Target	Control/Data. Used during the information transfer phases to transfer commands, status, data or messages over the bus.
REQ	Target	Request. Used by the target during information transfer phases.
I/O	Target	Input/Output. Determines the direction of the transfer.

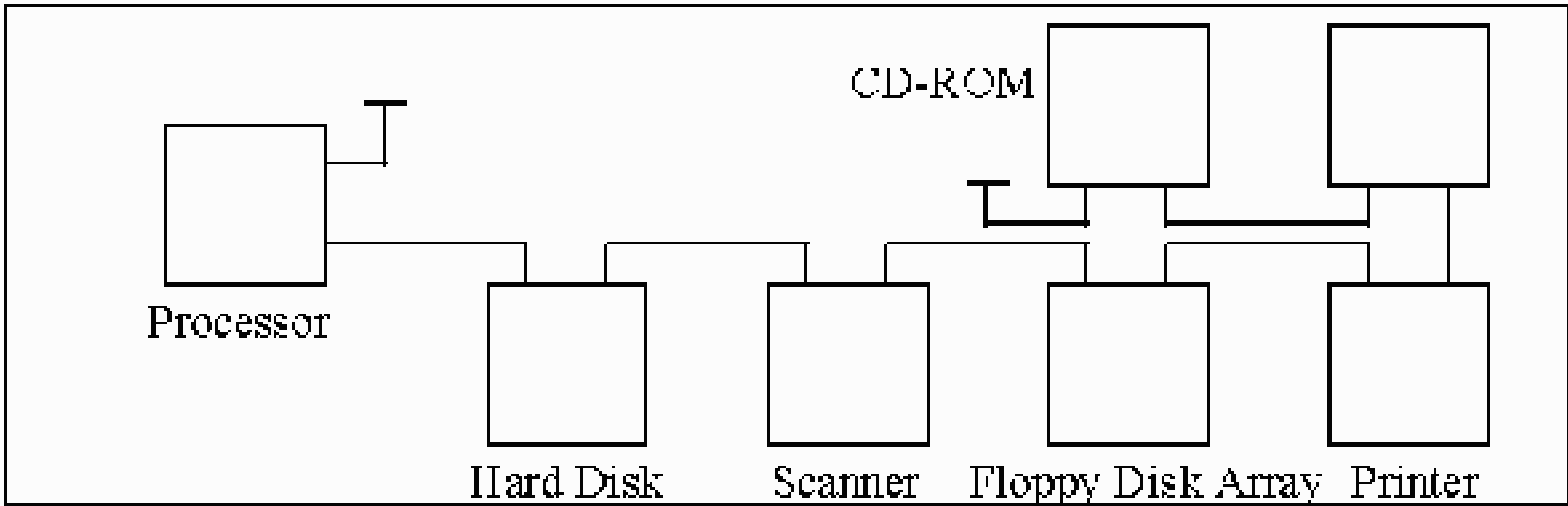


Phase Sequences of the SCSI Bus

SCSI Information Transfer Phases

Signal					Direction	Phase
SEL	BSY	MSG	C/D	I/O		
0	1	0	0	0	To Target	Data Out
0	1	0	0	1	From Target	Data In
0	1	0	1	0	To Target	Command
0	1	0	1	1	From Target	Status
0	1	1	0	0	—	Reserved
0	1	1	0	1	—	Reserved
0	1	1	1	0	To Target	Message Out
0	1	1	1	1	From Target	Message In

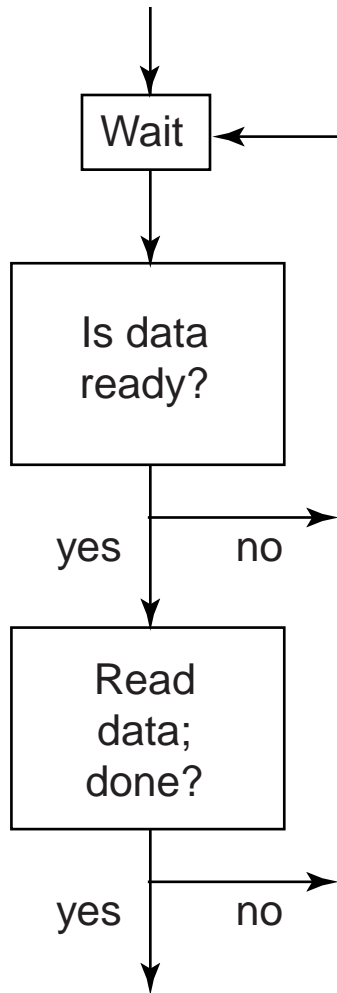
SCSI BUS TOPOLOGY



I/O AS SYNCHRONIZATION OF DATA TRANSFERS

- Fundamental problems of communication between devices, or between the CPU and peripheral devices:
 - ▷ Detection that a data transfer is necessary
 - Dedicated polling
 - Interrupts
 - Periodic polling
 - ▷ Synchronization of two devices, or a device and a CPU, with different speeds
 - Wait state insertion
 - DMA
 - Dual-ported memory
 - FIFO buffers
 - Caches

POLLED I/O



A **polling loop** is not an efficient way to use a CPU unless the device is very fast. If the device is fast, then “data ready” checks can be interspersed among useful instructions.

In most cases it is more efficient for the I/O device to tell the CPU when data is ready, or when a transfer is complete, than for the CPU to check the device frequently. An I/O device can use **interrupts** to tell the CPU that a data transfer should be started, or is finished.

DEDICATED vs. PERIODIC POLLING

- **Periodic polling** means that the CPU periodically interrogates the I/O device (*e.g.*, via an oscillator–counter–decoder combination) to see whether data is ready
- **Dedicated polling (spin waiting)** means that the I/O device controller sets or clears bits in a status register that is read in a tight loop by the CPU
 - ▷ When a system call for keyboard input is issued, and dedicated polling is in use, the CPU executes code somewhat like this:

```
get_loop: lw $a0, Device_Status
          bgez $a0, get_loop
          lb $2, Device_Data
          rfe
```
 - ▷ This operation transfers only a single byte; data may be missed
 - ▷ A different approach is necessary for block transfers

INTERRUPT-DRIVEN I/O (1)

- An interrupt is an event that occurs outside the execution cycle and that causes processing of the current thread to stop
 - ▷ Interrupts can be used to give I/O devices a means to signal the CPU that an event has occurred that requires action by the CPU (data is ready, etc.)
 - ▷ An interrupt causes an exception, which results in a jump to the appropriate exception handling code (MIPS: address 0x80000080)
 - ▷ There are (at least) two principal methods for detecting interrupts in hardware:
 - Connect the interrupt request output of an I/O device to one of the inputs of an **interrupt controller**
 - ◇ Interrupts may be level-triggered or edge-triggered
 - Connect one interrupt line to an OR of inputs from several devices that are periodically strobed for data ready
 - ◇ Device that caused the interrupt can be detected by reading a status word formed from inputs from the devices

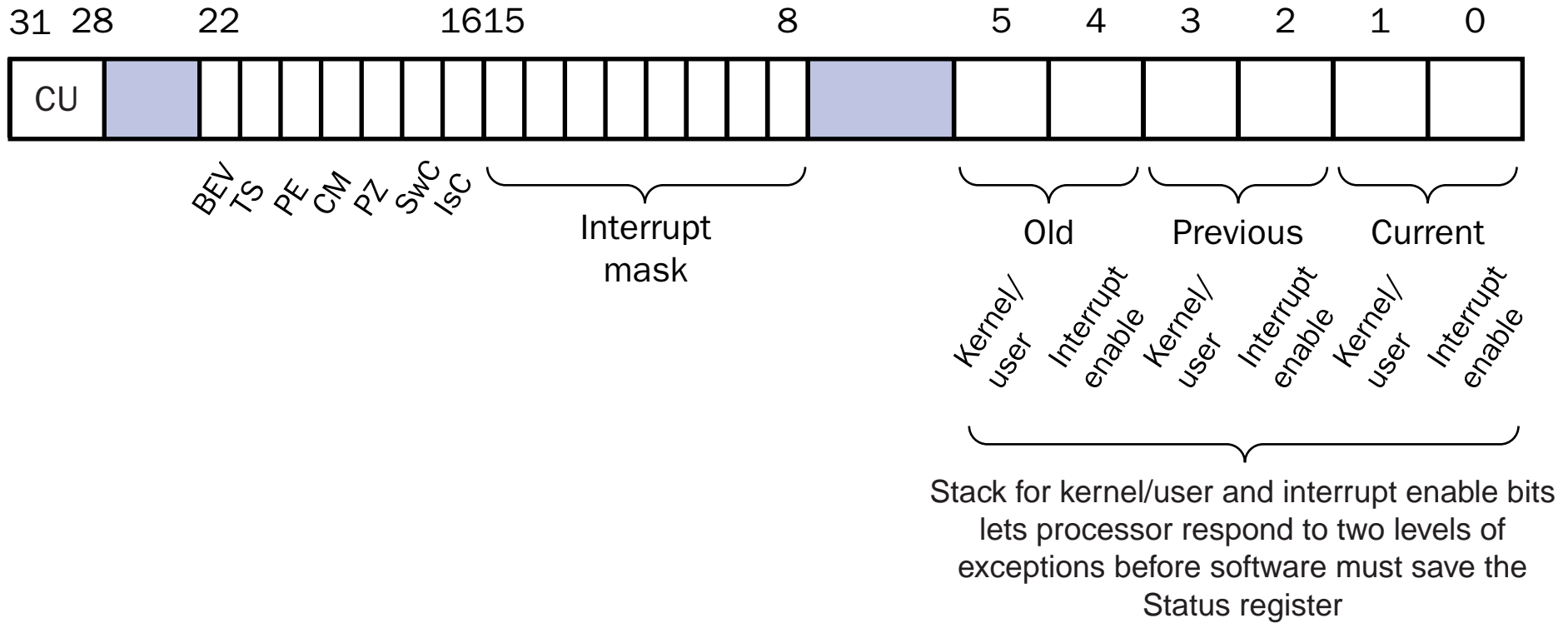
INTERRUPT-DRIVEN I/O (2)

- On a RISC machine, an interrupt causes a jump to the general exception handling code (with a few special cases such as Reset and UTLB Miss)
 - ▷ Method of P&H Chapter 5: Execution is suspended immediately
 - This method is required for some exceptions (TLB miss, page fault) unless execution can be undone
 - Restarting is hard in ISAs where memory is accessed at multiple times during execution of an instruction
 - ▷ Method of choice: The instruction that caused the exception is allowed to finish; subsequent instructions are suspended
 - ▷ Pending interrupts must be handled before next instruction is fetched
 - ▷ The exception handler determines the code to execute, based on the Cause register contents
 - ▷ The operating system determines what state needs to be saved (if any) besides the EPC and Cause registers

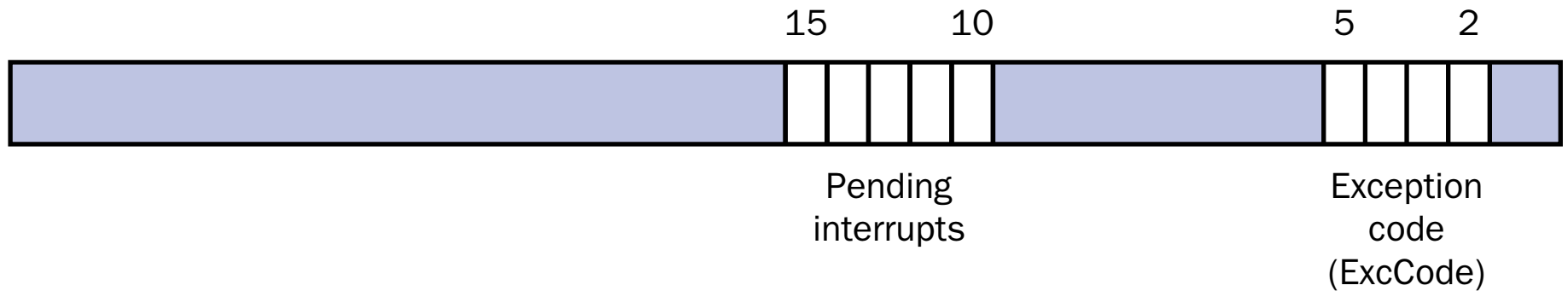
INTERRUPT-DRIVEN I/O (3)

- MIPS R2000 interrupt handler:
 - ▷ Saves `$a0` and `$v0` in special locations
 - `save0` is at address `0x90000250`; `save1` is at address `0x90000254`
 - `$a0` and `$v0` can't be pushed onto the stack, because the cause of the exception may be a bad stack pointer!
 - ▷ Copies coprocessor 0 Cause and EPC registers into `$k0` and `$k1`
 - ▷ Pushes current Kernel/User mode and Interrupt Enable Mode bits onto the stack in the Status register (see next slide)
 - ▷ The kernel's exception handler uses a jump table (or a sequence of `beq`'s) to determine the right code to execute (see SPIM kernel text)
 - ▷ The operating system clears the interrupts, if any
 - ▷ After executing an `rfe` instruction, the processor may restart execution at the address in the EPC

MIPS R2000 STATUS REGISTER



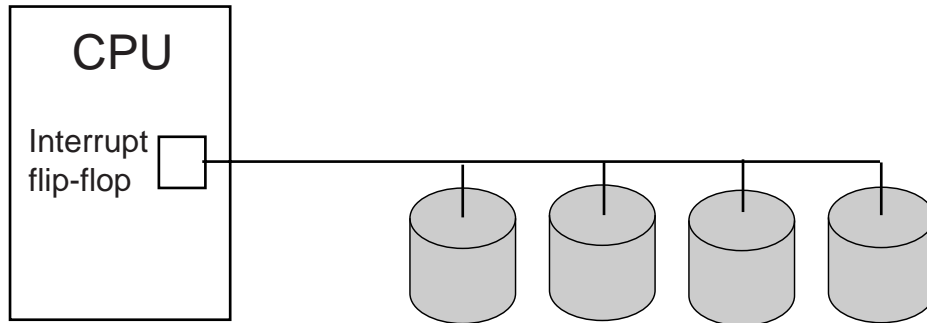
MIPS R2000 CAUSE REGISTER



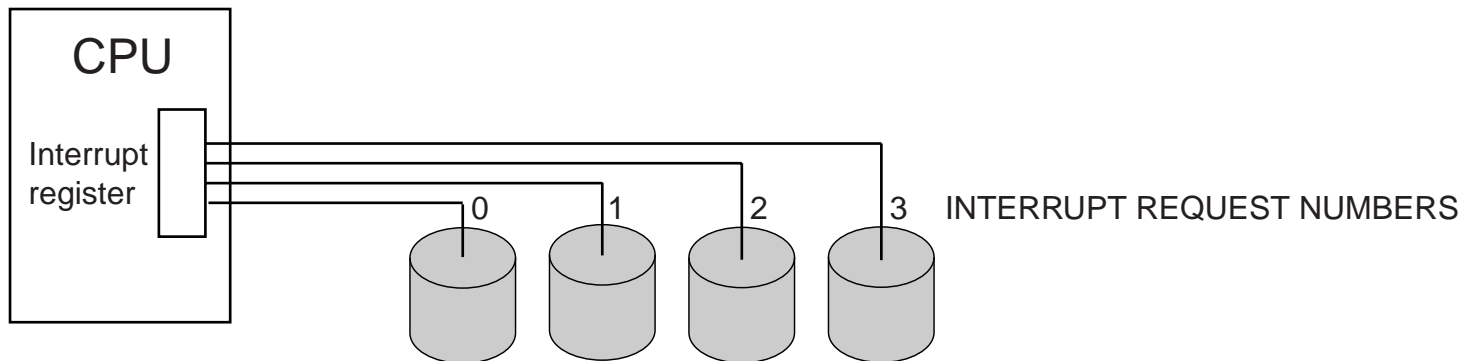
EXCEPTION CODES IN THE MIPS R2000 ISA

ExcCode	Name	Description
0	Int	External interrupt
1	MOD	TLB modification exception
2	TLBL	TLB miss exception (Load or instruction fetch)
3	TLBS	TLB miss exception (Store)
4	AdEL	Address error exception (Load or instruction fetch)
5	AdES	Address error exception (Store)
6	IBE	Instruction fetch bus error exception
7	DBE	Data load or store bus error exception
8	Sys	System call exception
9	Bp	Breakpoint exception
10	RI	Reserved or undefined instruction exception
11	CpU	Coprocessor unusable exception
12	Ovf	Arithmetic overflow exception

SINGLE- AND MULTIPLE-LINE INTERRUPT SYSTEMS

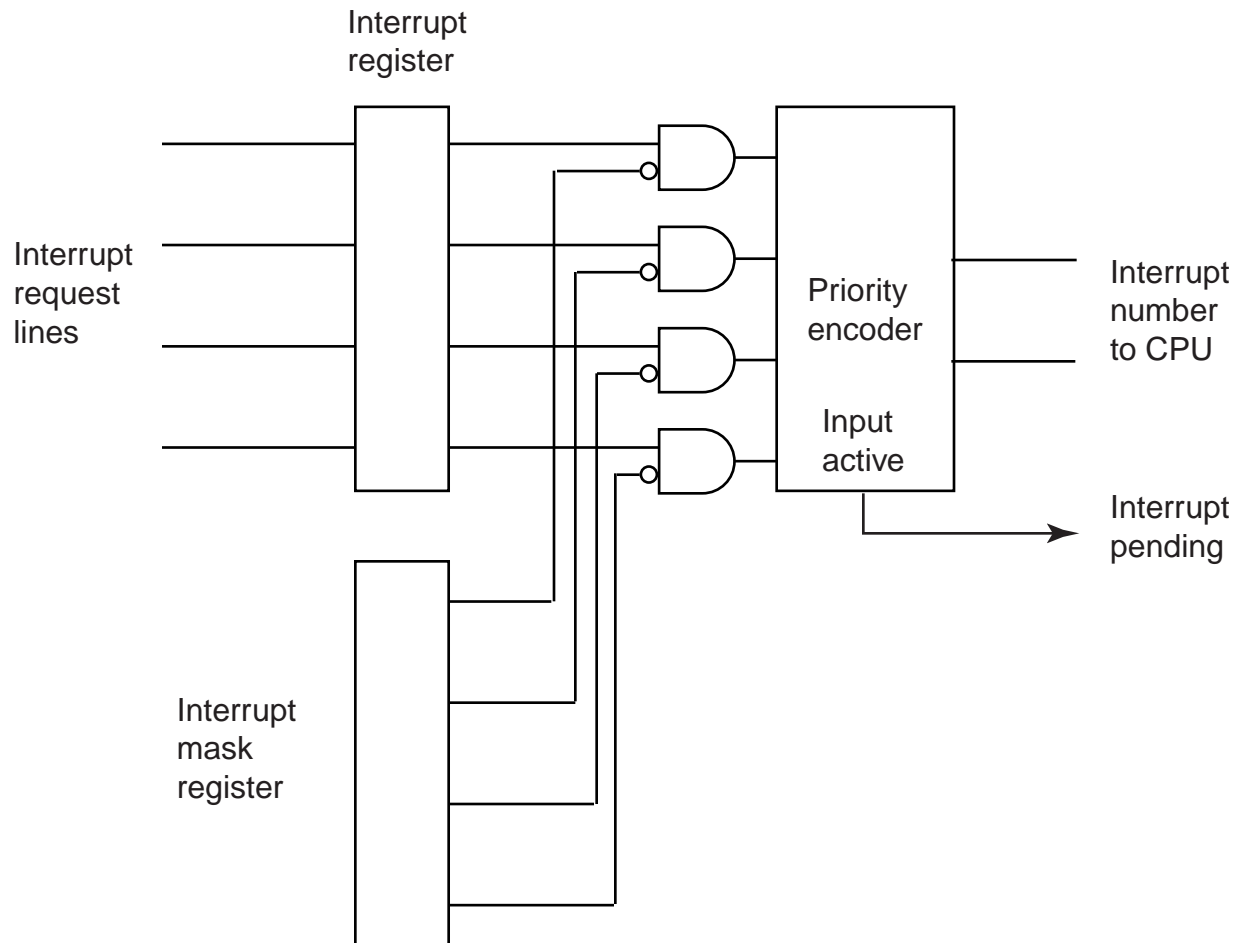


SINGLE-LINE INTERRUPT SYSTEM



MULTIPLE-LINE INTERRUPT SYSTEM

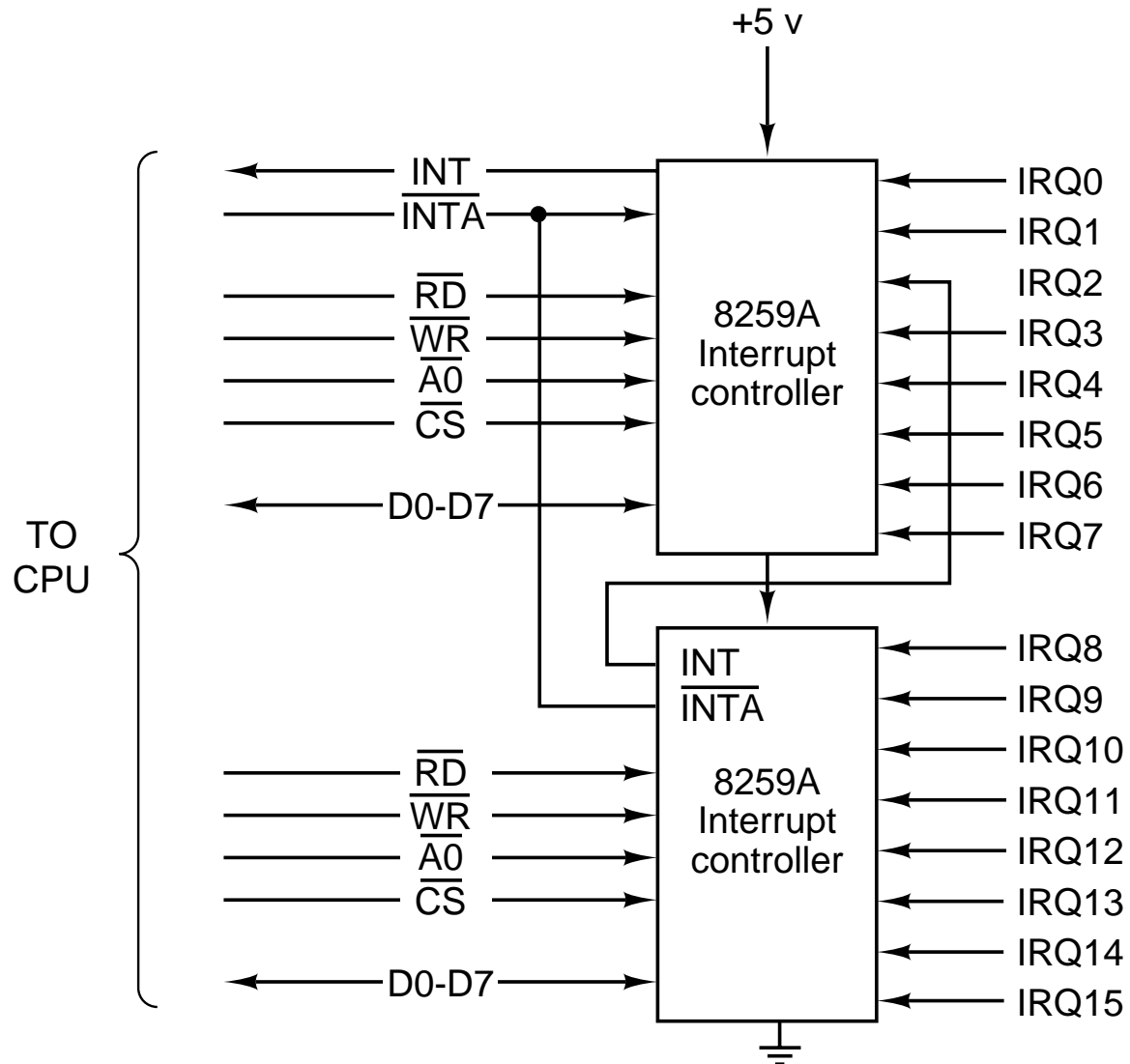
VECTORED INTERRUPT SYSTEM



INTERRUPT-DRIVEN I/O (4)

- In the Motorola 68000 series, the CPU checks for pending interrupts after execution of each instruction
 - ▷ CPU saves status register (SR) and enters supervisor mode
 - ▷ After determining the interrupt number N , the CPU saves state information and executes $M[4N] \rightarrow PC$, causing a branch to the text at the location pointed to by $M[4N]$

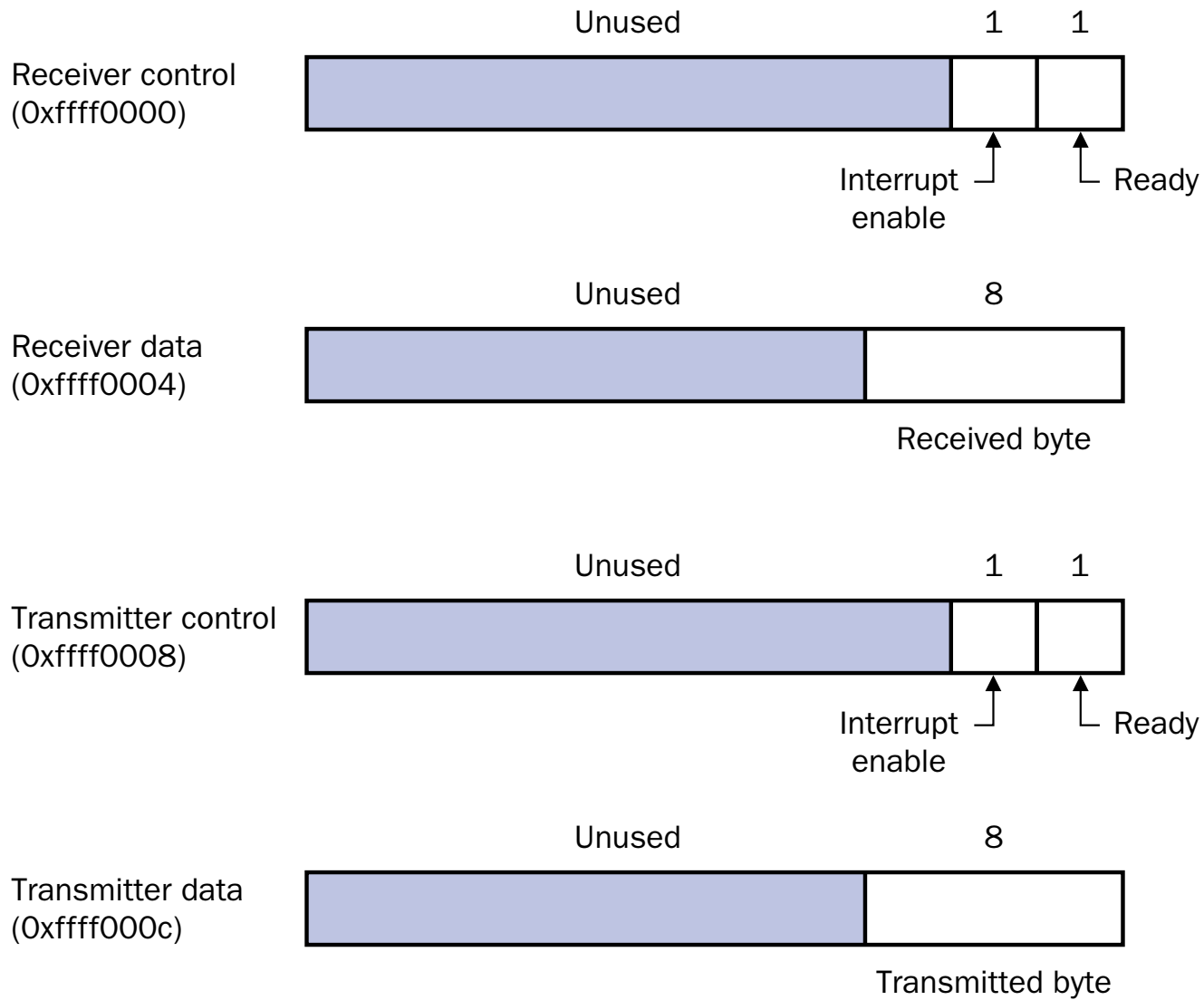
VECTORED INTERRUPTS IN THE IBM PC



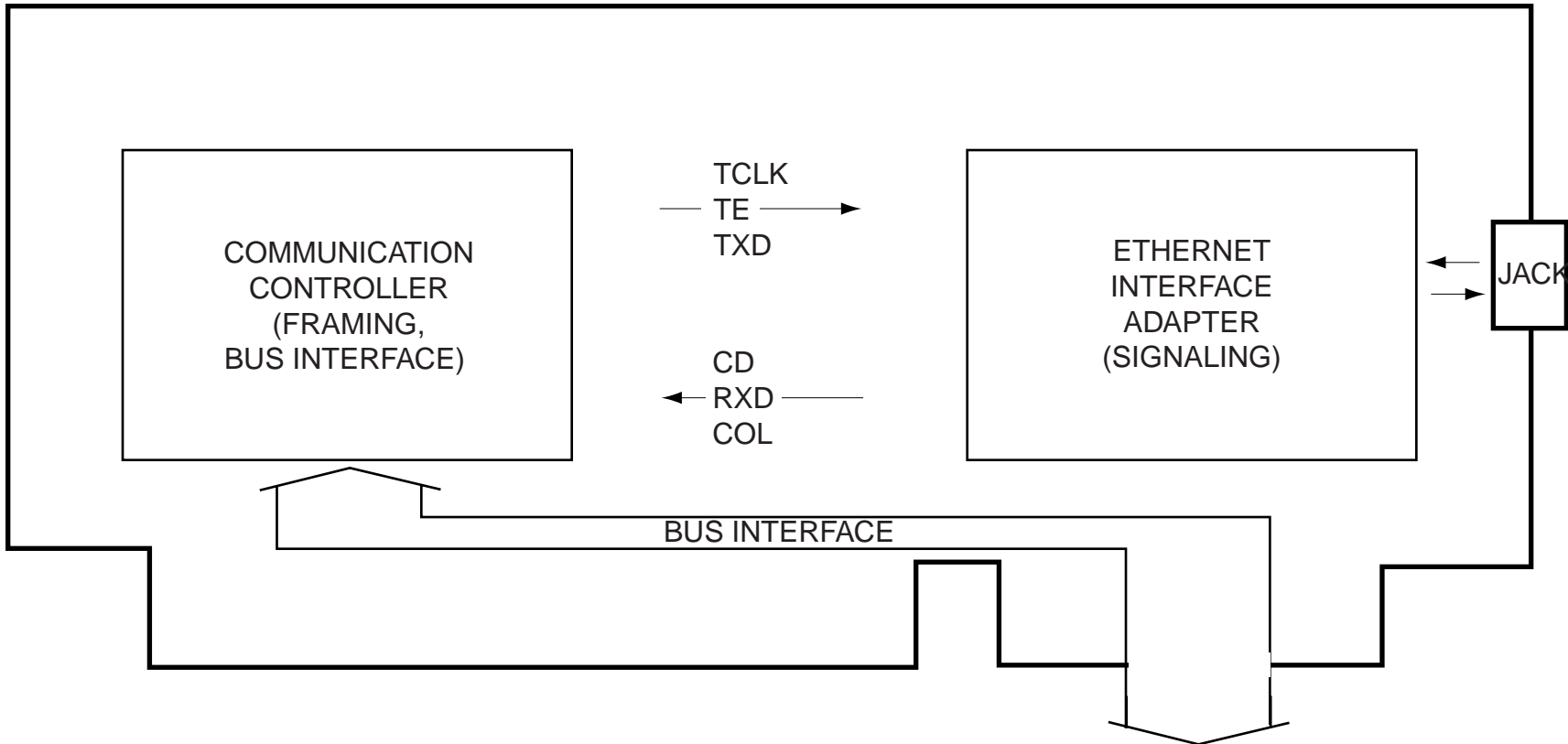
MEMORY-MAPPED I/O

- Instead of having multiple address spaces for memory, I/O, etc., have a single address space
 - ▷ Loading from a memory location that is mapped to an I/O device reads a data byte or word from the device
 - ▷ Storing to a memory location that is mapped to an I/O device writes a data byte or word to the device
 - ▷ Used in Motorola 68000 series
- In order to synchronize I/O properly, additional memory locations may be mapped to status words for the I/O devices

SPIM's MEMORY-MAPPED I/O REGISTERS



NETWORK INTERFACE CARD



I/O PROCESSORS

- An **I/O processor** (IOP) is a processor with (usually) a more restricted instruction set than the CPU
 - ▷ Purpose: Offload I/O processing from the CPU
 - Used in CDC 6600, IBM S/360–370, ...
 - ▷ I/O instructions executed by an IOP are called channel command words in the IBM world
 - ▷ A CPU and its IOPs are really a shared-memory multiprocessor

RELATION OF I/O TO PROCESSOR ARCHITECTURE

- I/O instructions and buses have disappeared
- Interrupt vectors have been replaced by jump tables
- Interrupt stack replaced by shadow registers
 - ▷ Handler saves registers and re-enables higher-priority interrupts
- Interrupt types reduced in number
 - ▷ Handler must query interrupt controller
- Caches cause problems for I/O
 - ▷ Flushing degrades performance heavily
 - ▷ Solution: “snooping” (borrowed from shared-memory multiprocessors)
- Virtual memory frustrates DMA
- Load-store architecture inconsistent with atomic I/O operations
- Stateful processors hard to context switch