William Stallings
Computer Organization and Architecture
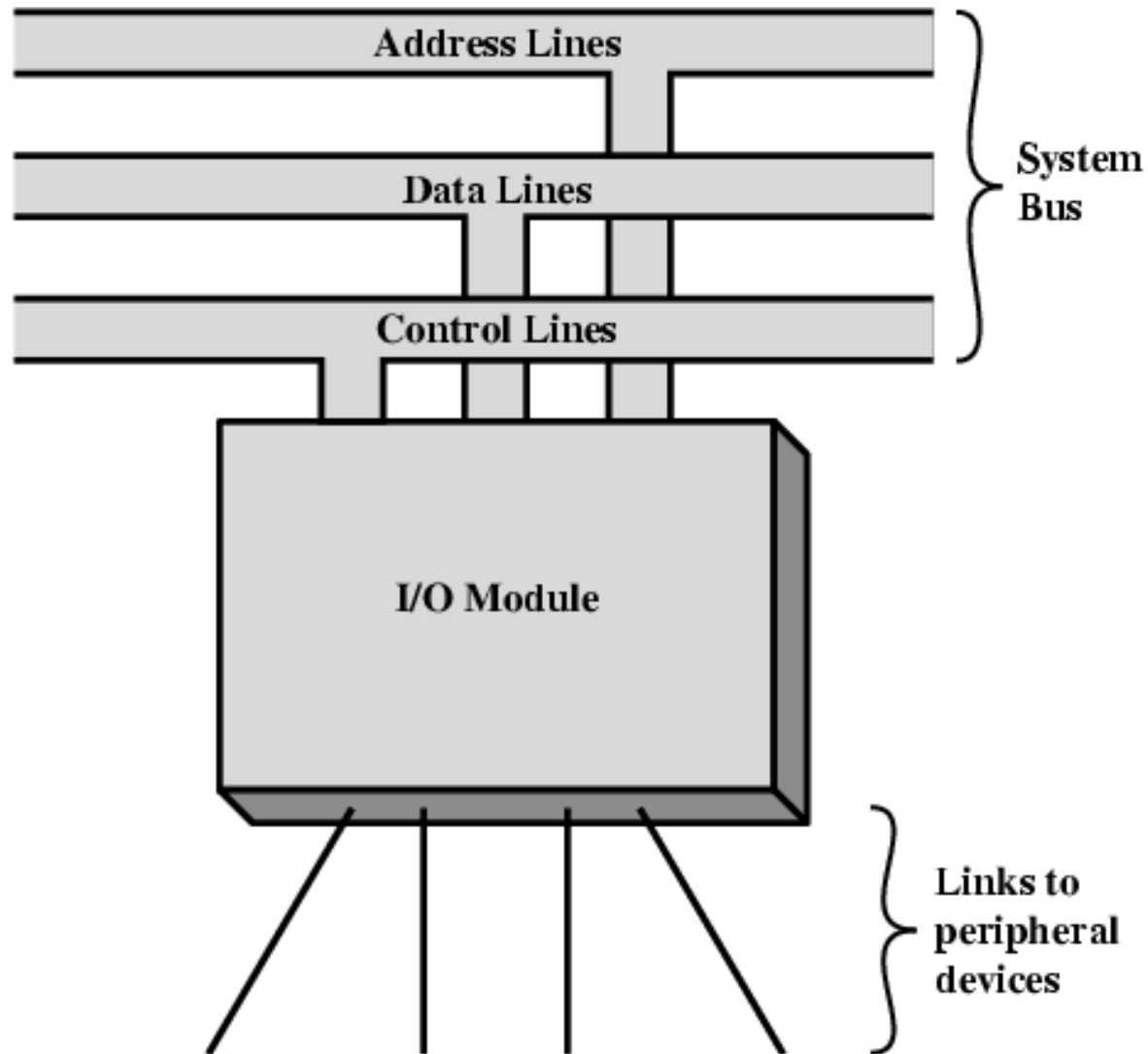# 7th Edition

Chapter 7
Input/Output

Input/Output Problems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- Almost all slower than CPU and RAM
- Need I/O modules

Input/Output Module

- Interface to CPU and Memory
- Interface to one or more peripherals

Generic Model of I/O Module

External Devices

- Human readable
  - —Screen, printer, keyboard
- Machine readable
  - —Monitoring and control
- Communication
  - —Modem
  - —Network Interface Card (NIC)
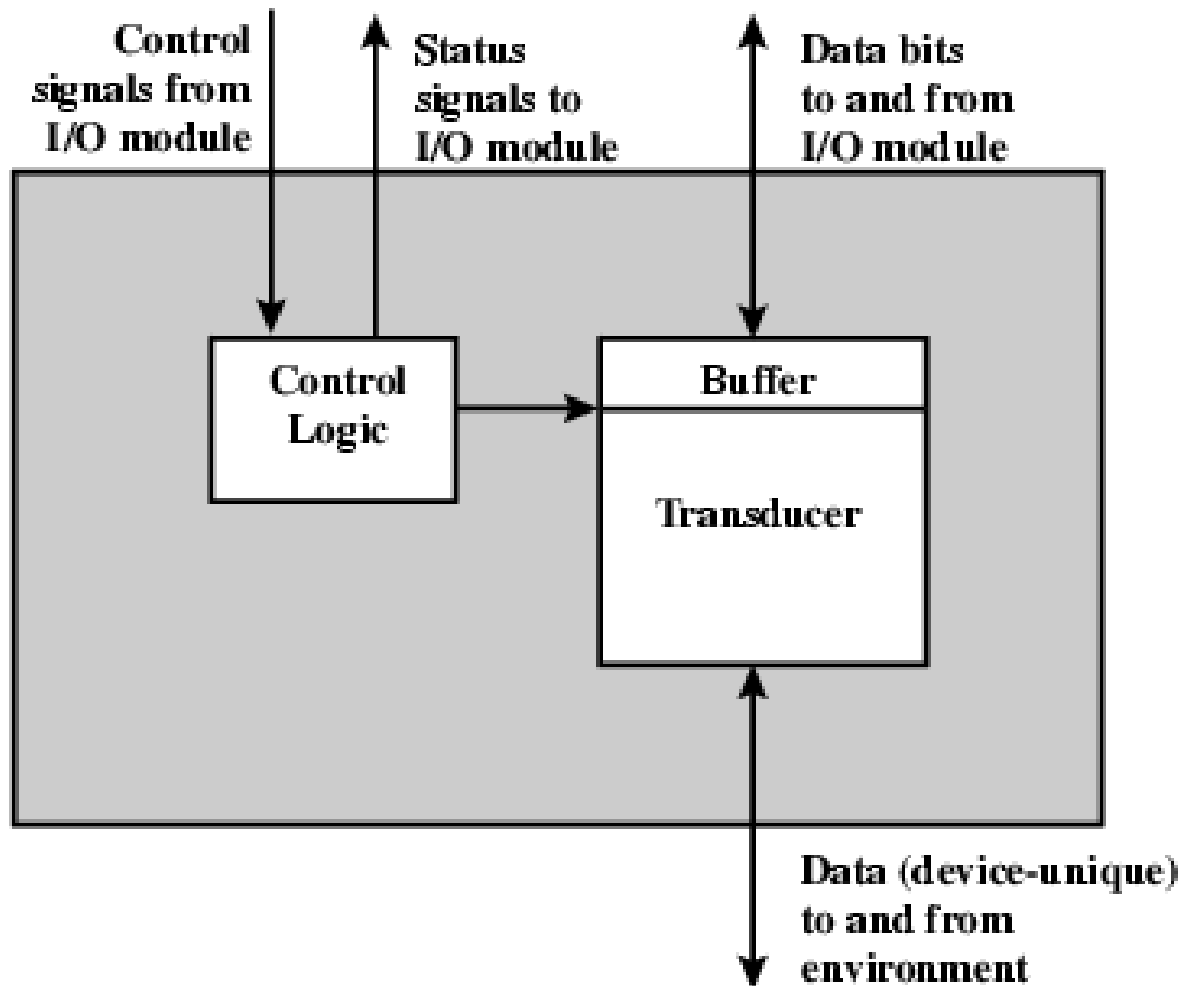
External Device Block Diagram

Table 7.1 The International Reference Alphabet (IRA)

bit position

b_7			0	0	0	0	1	1	1	1
	b_6		0	0	1	1	0	0	1	1
		b_5	0	1	0	1	0	1	0	1

| b_4	b_3	b_2	b_1								
0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l		
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

I/O Module Function

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection

I/O Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
- Variations for output, DMA, etc.

I/O Module Diagram
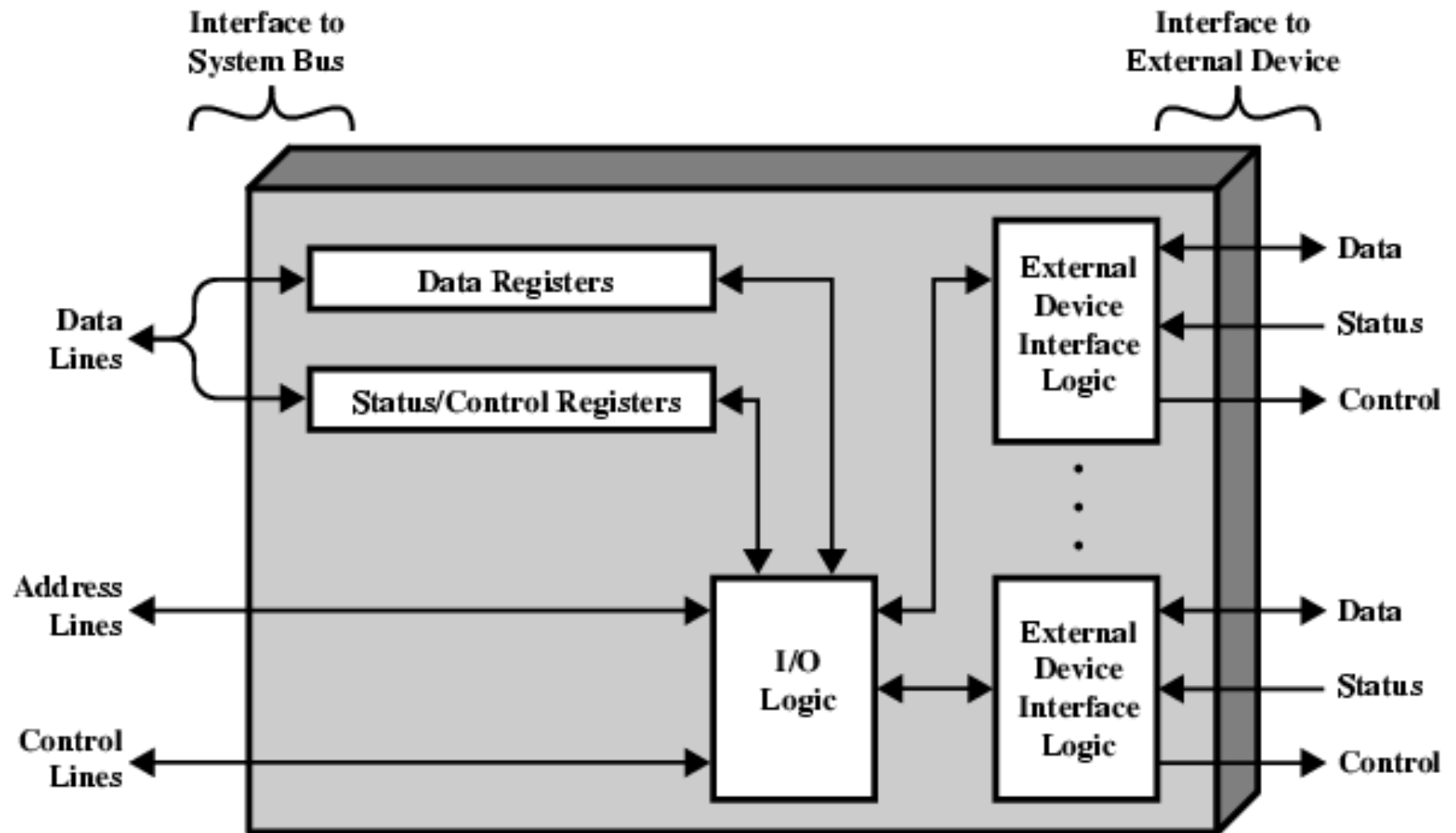
I/O Module Decisions

- Hide or reveal device properties to CPU
- Support multiple or single device
- Control device functions or leave for CPU
- Also O/S decisions
  - e.g. Unix treats everything it can as a file

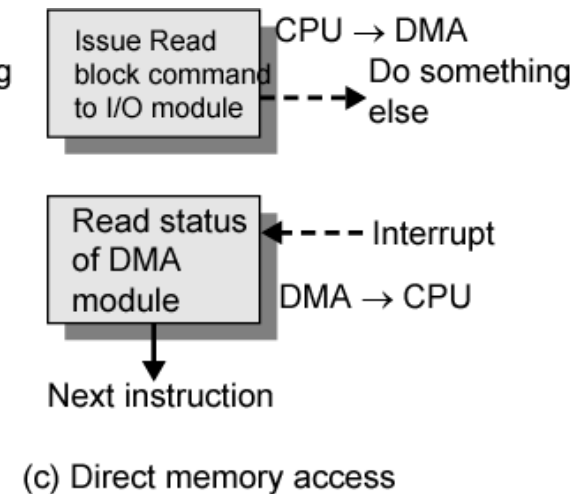I/O Channel or I/O Processor (High level interface, detailed processing burden)
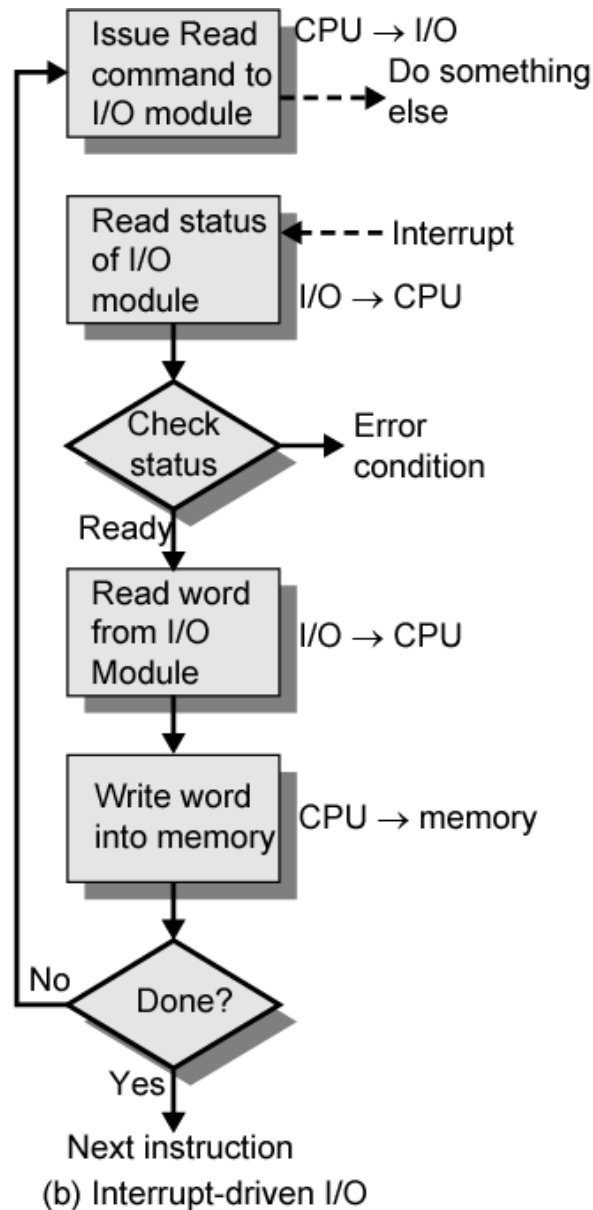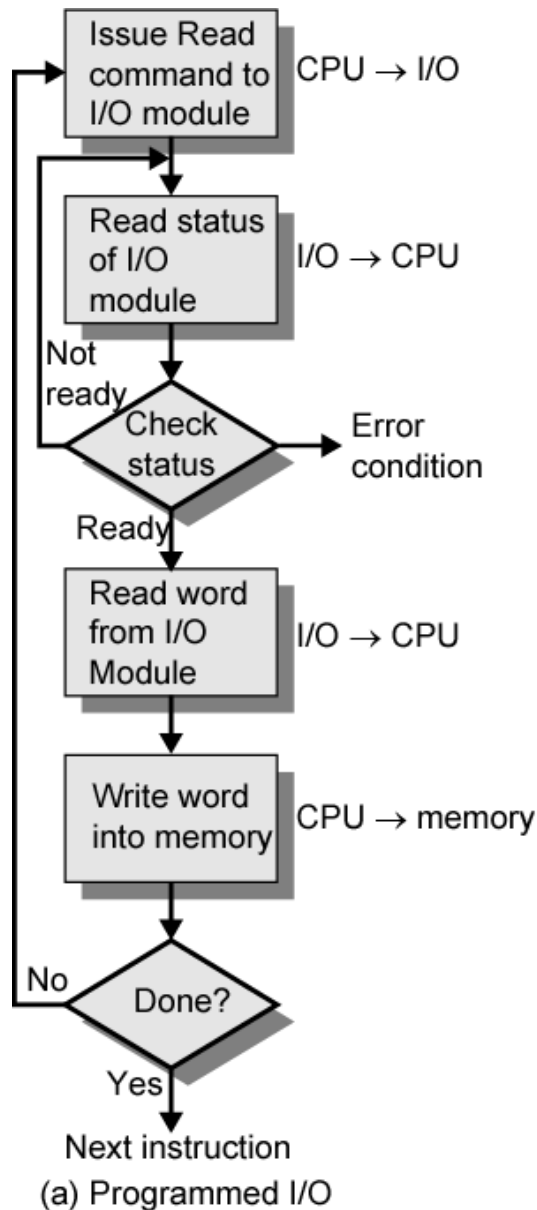
I/O Controller of Device Controller (primitive and requires detailed control)

I/O Module: A common name

Input Output Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

Three Techniques for Input of a Block of Data

(a) Programmed I/O

- Issue Read command to I/O module — CPU → I/O
- Read status of I/O module — I/O → CPU
- Check status — Not ready (loop back) / Error condition
- Ready → Read word from I/O Module — I/O → CPU
- Write word into memory — CPU → memory
- Done? — No (loop back) / Yes → Next instruction

(b) Interrupt-driven I/O

- Issue Read command to I/O module — CPU → I/O → Do something else
- Read status of I/O module ← Interrupt — I/O → CPU
- Check status — Error condition
- Ready → Read word from I/O Module — I/O → CPU
- Write word into memory — CPU → memory
- Done? — No (loop back) / Yes → Next instruction

(c) Direct memory access

- Issue Read block command to I/O module — CPU → DMA → Do something else
- Read status of DMA module ← Interrupt — DMA → CPU
- Next instruction

Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time

Programmed I/O - detail

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

I/O Commands

- CPU issues address
  —Identifies module (& device if >1 per module)
- CPU issues command
  —Control - telling module what to do
    – e.g. spin up disk
  —Test - check status
    – e.g. power? Error? Completion of job?
  —Read/Write
    – Module transfers data via buffer from/to device

Addressing I/O Devices

- Under programmed I/O data transfer is very like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)

I/O Mapping

- Memory mapped I/O
  - —Devices and memory share an address space
  - —I/O looks just like memory read/write
  - —No special commands for I/O
    - – Large selection of memory access commands available
- Isolated I/O
  - —Separate address spaces
  - —Need I/O or memory select lines
  - —Special commands for I/O
    - – Limited set

Memory Mapped and Isolated I/O

```
  7  6  5  4  3  2  1  0
516 [                    ]   Keyboard input data register
```

```
  7  6  5  4  3  2  1  0
517 [                    ]   Keyboard input status
                             and control register
     ↑                  ↑
     └ 1 = ready        └ Set to 1 to
       0 = busy           start read
```
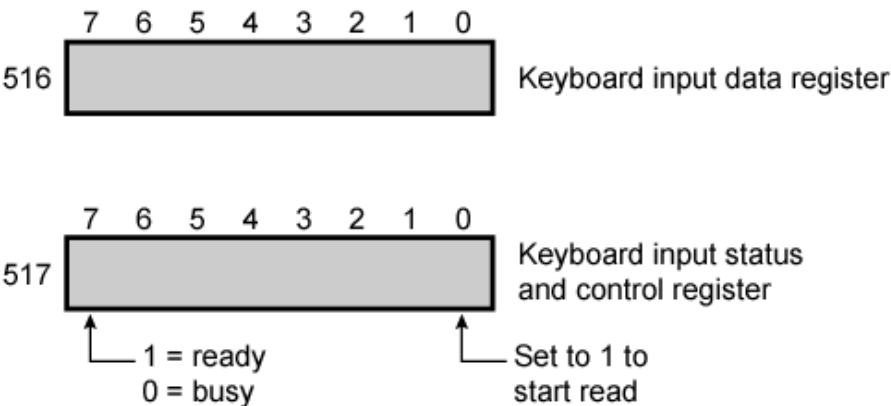
ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O

Interrupt Driven I/O

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

Interrupt Driven I/O
Basic Operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Simple Interrupt Processing

Hardware

Software

Device controller or other system hardware issues an interrupt

Processor finishes execution of current instruction

Processor signals acknowledgment of interrupt

Processor pushes PSW and PC onto control stack

Processor loads new PC value based on interrupt

Save remainder of process state information

Process interrupt

Restore process state information

Restore old PSW and PC

CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
  - Save context (registers)
  - Process interrupt
    - Fetch data & store
- See Operating Systems notes

Changes in Memory and Registers for an Interrupt
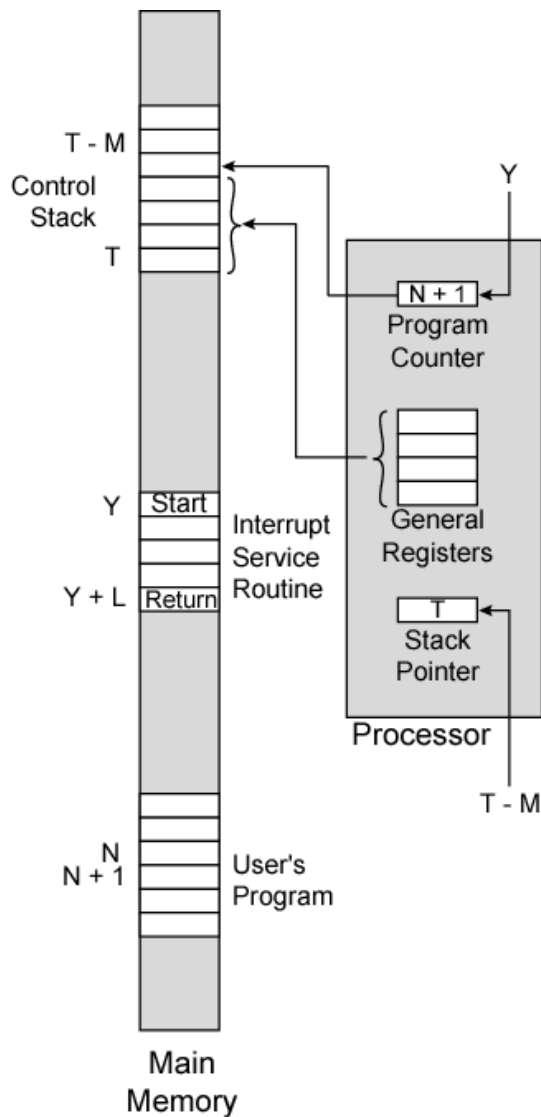
(a) Interrupt occurs after instruction at location N
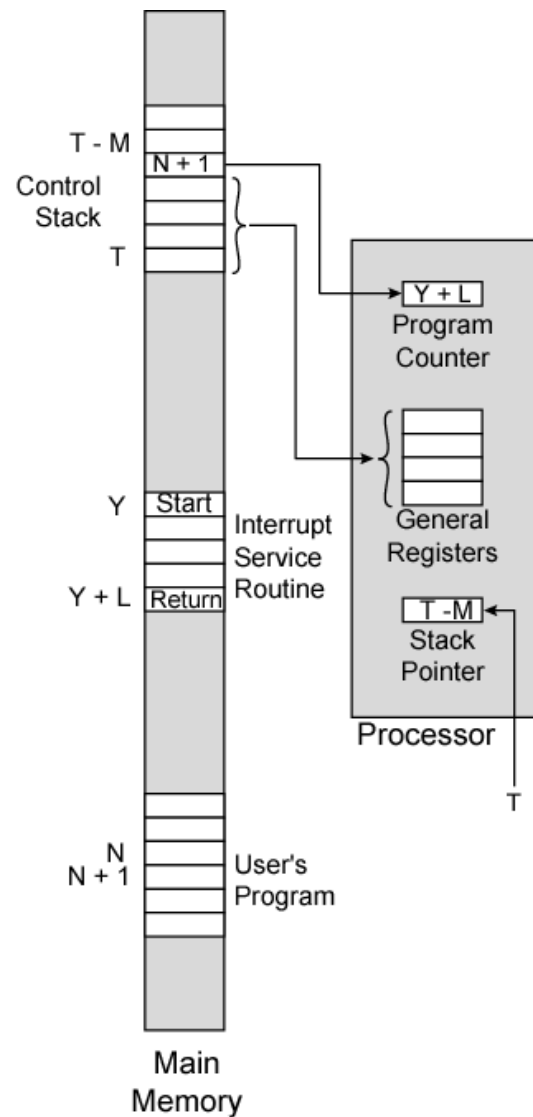
(b) Return from interrupt

Design Issues

- How do you identify the module issuing the interrupt?

- How do you deal with multiple interrupts?
  - —i.e. an interrupt handler being interrupted

Identifying Interrupting Module (1)

- Different line for each module
  - —impractical to dedicate many pins or bus lines
  - —Limits number of devices
- Software poll
  - —CPU asks each module in turn
  - —Or reads the status register of each module in turn
  - —Once the interrupting module is identified, branches to a service routine specific to that device.
  - —Slow

Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll
  - —All I/O modules share a common interrupt request line. The interrupt acknowledge line is daisy chained through the modules.
  - —Interrupt Acknowledge sent down a chain
  - —Module responsible places a word (vector) on bus
  - —CPU uses vector to identify handler routine (Vectored Interrupt)
- Bus Master (Bus Arbitration)
  - —Module must claim the bus before it can raise interrupt
  - —Thus, only one device may raise interrupt at a time
  - —Processor responds on the interrupt acknowledge line
  - —The requesting module then, puts its vector on data lines
  - —e.g. PCI & SCSI

Distributed Arbitration
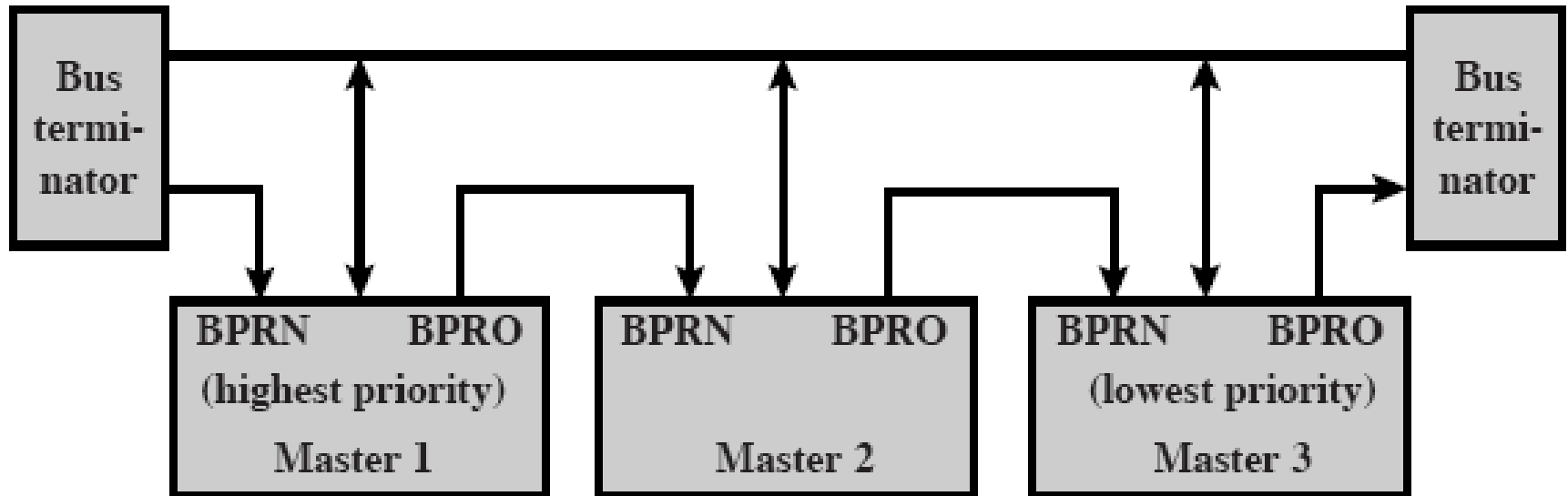
Multiple Interrupts

- Each interrupt line has a priority
- Order in which modules are polled, order of modules on a daisy chain or a priority scheme for bus arbitration determines the priority.
- Higher priority lines can interrupt lower priority lines
- In the case of bus mastering, only current master can interrupt and bus arbitration can employ a priority scheme.

Example

- 80386 has one interrupt line and one INTA (Interrupt Acknowledge) line
- 80386 based systems use 82C59A programmable interrupt controller
- 82C59A has 8 interrupt lines. A cascade arrangement handles up to 64 lines.

Sequence of Events

- 82C59A accepts interrupts
- 82C59A determines priority
- 82C59A signals 80386 (raises INTR line)
- CPU Acknowledges
- 82C59A puts correct vector on data bus
- CPU processes interrupt and communicate directly with the I/O module to read or write data.

82C59A Interrupt Controller
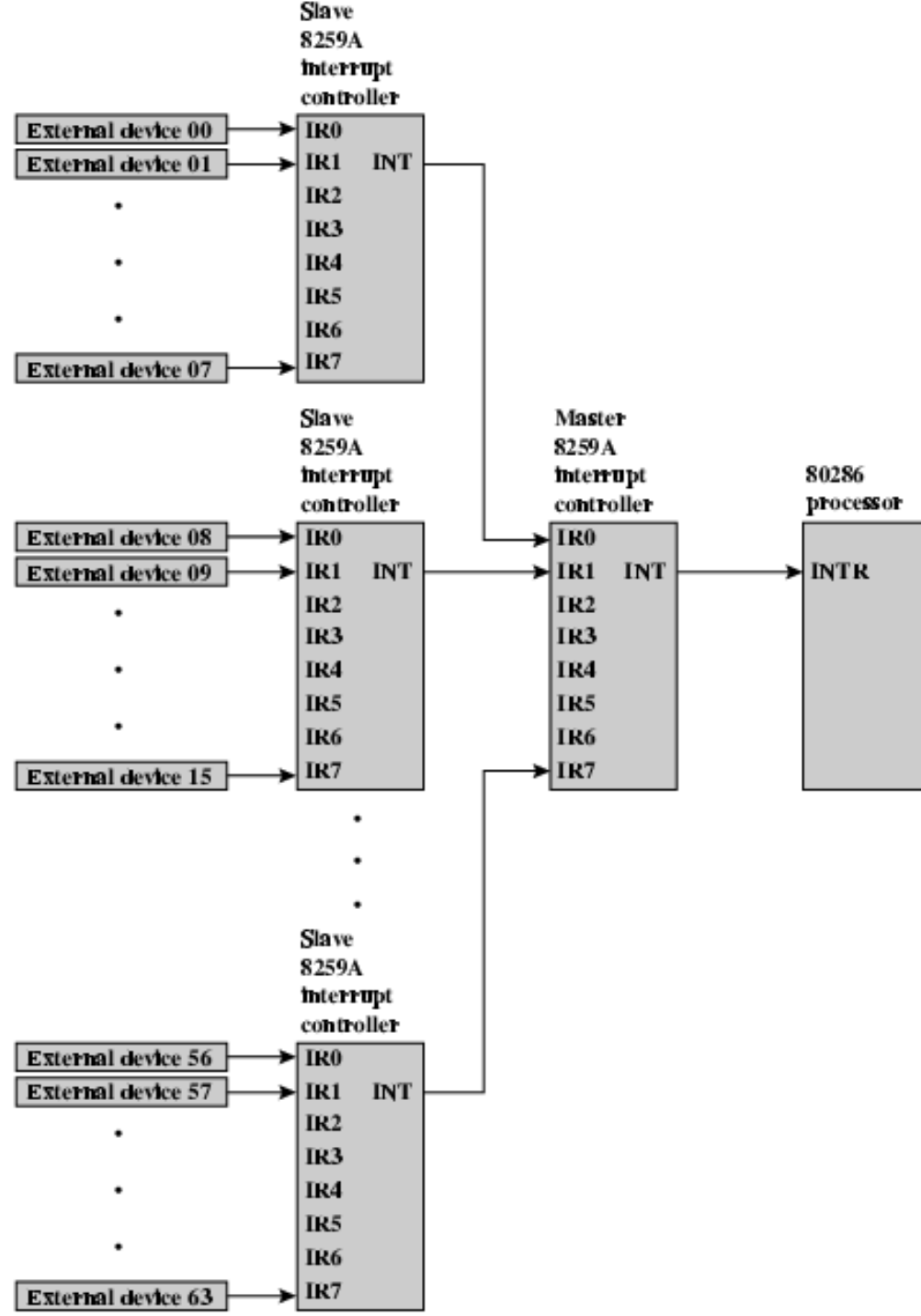
Intel 82C55A
Programmable Peripheral Interface
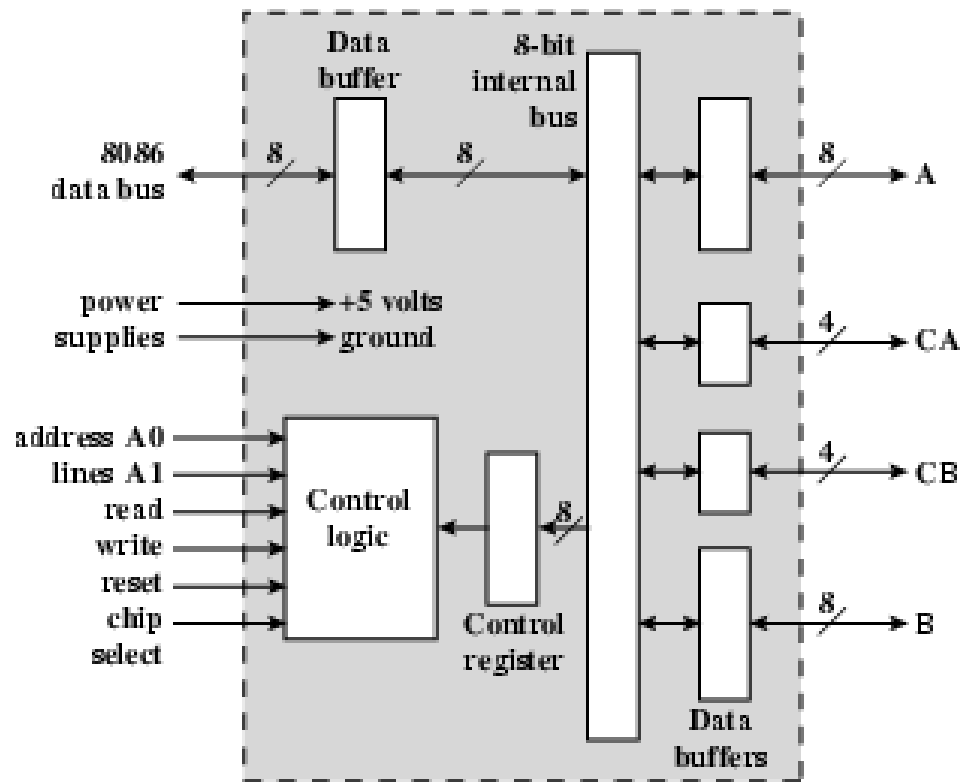
(a) Block diagram
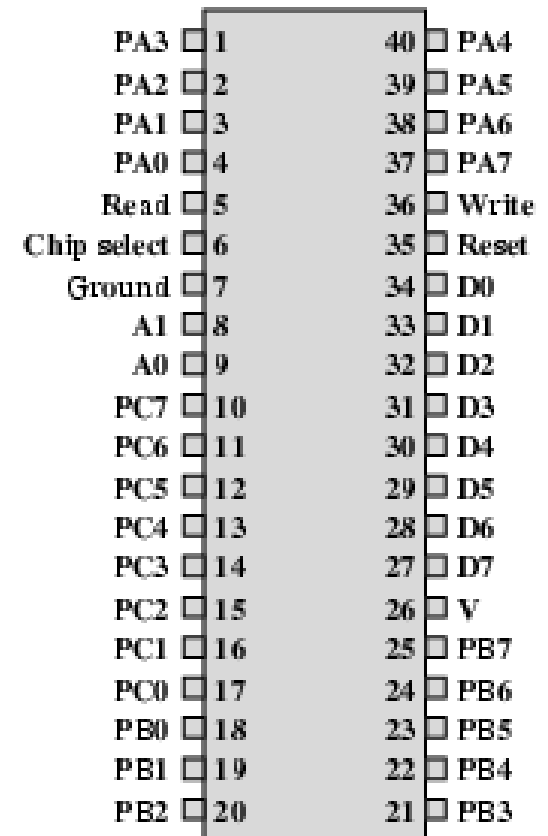
(b) Pin layout

Keyboard/Display Interfaces to 82C55A
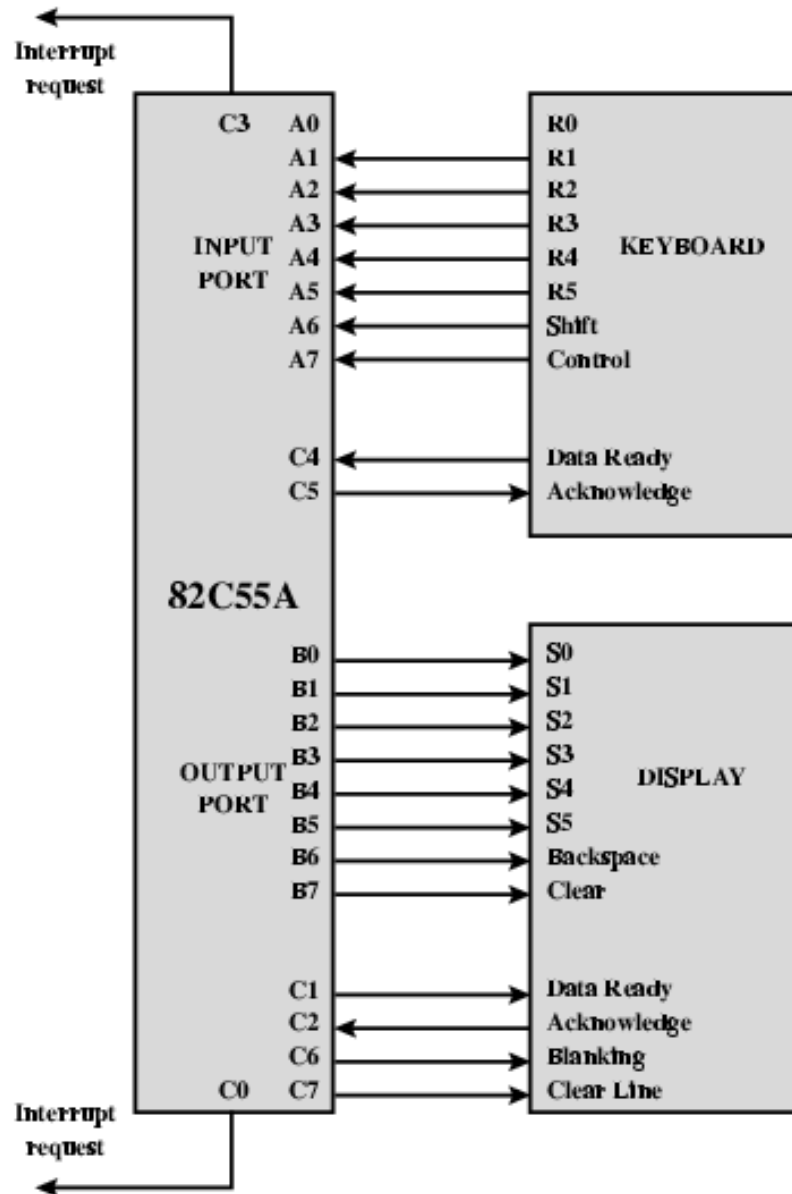
Direct Memory Access

- Interrupt driven and programmed I/O require active CPU intervention
  - Transfer rate is limited by the speed with which the processor can test and service a device (programmed I/O is faster but at a cost of doing nothing else)
  - CPU is tied up (in the case of interrupt I/O freed to some extent at the expense of the I/O transfer rate)
- For large volumes of data, DMA is a more efficient technique

DMA Function

- Additional Module (hardware) on bus
- DMA controller takes over the bus from CPU for I/O
  - Either when the processor does not need it
  - Or by forcing the processor to suspend operation temporarily (cycle steeling)

Typical DMA Module Diagram
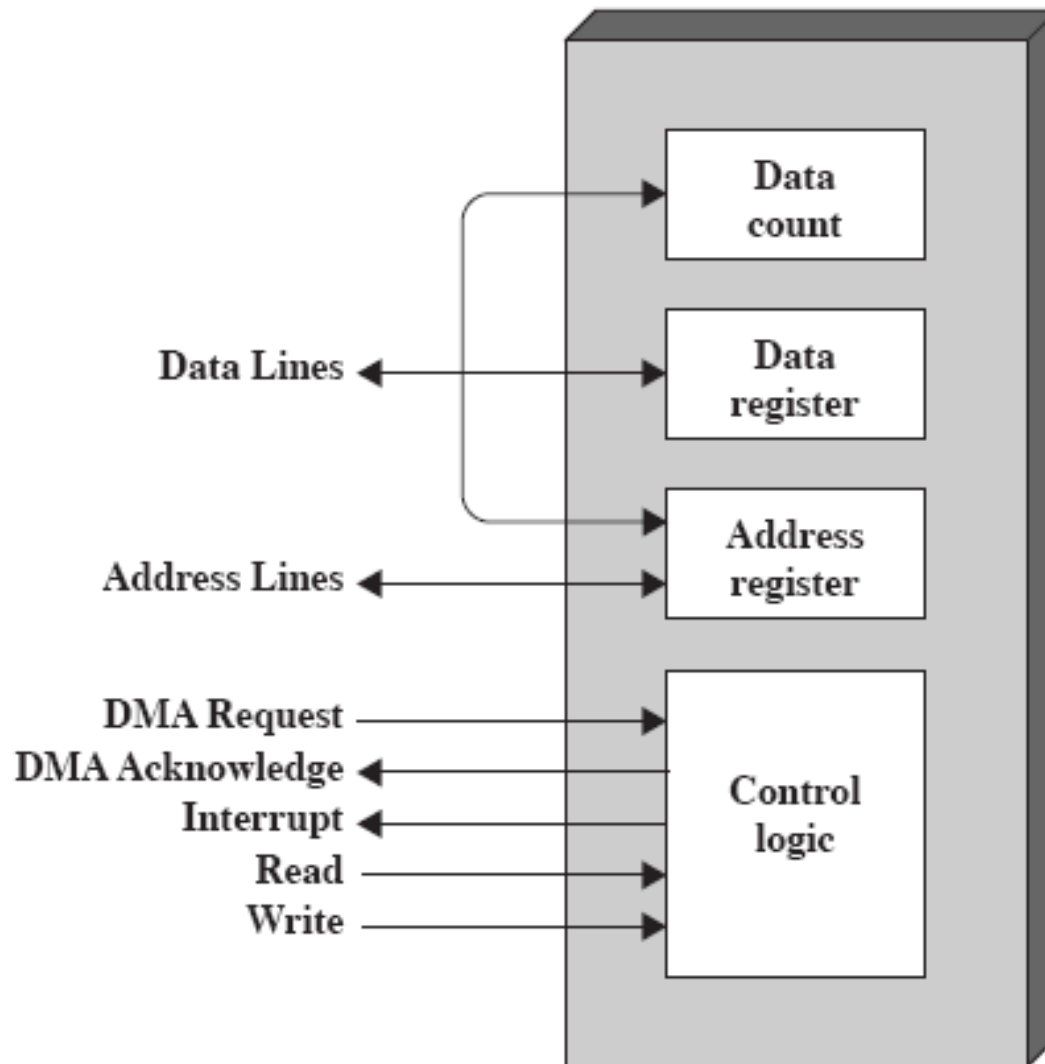
DMA Operation

- CPU tells DMA controller:-
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

DMA Transfer
Cycle Stealing

- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - —CPU does not switch context
- CPU suspended just before it accesses bus
  - —i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer

DMA and Interrupt Breakpoints During an Instruction Cycle
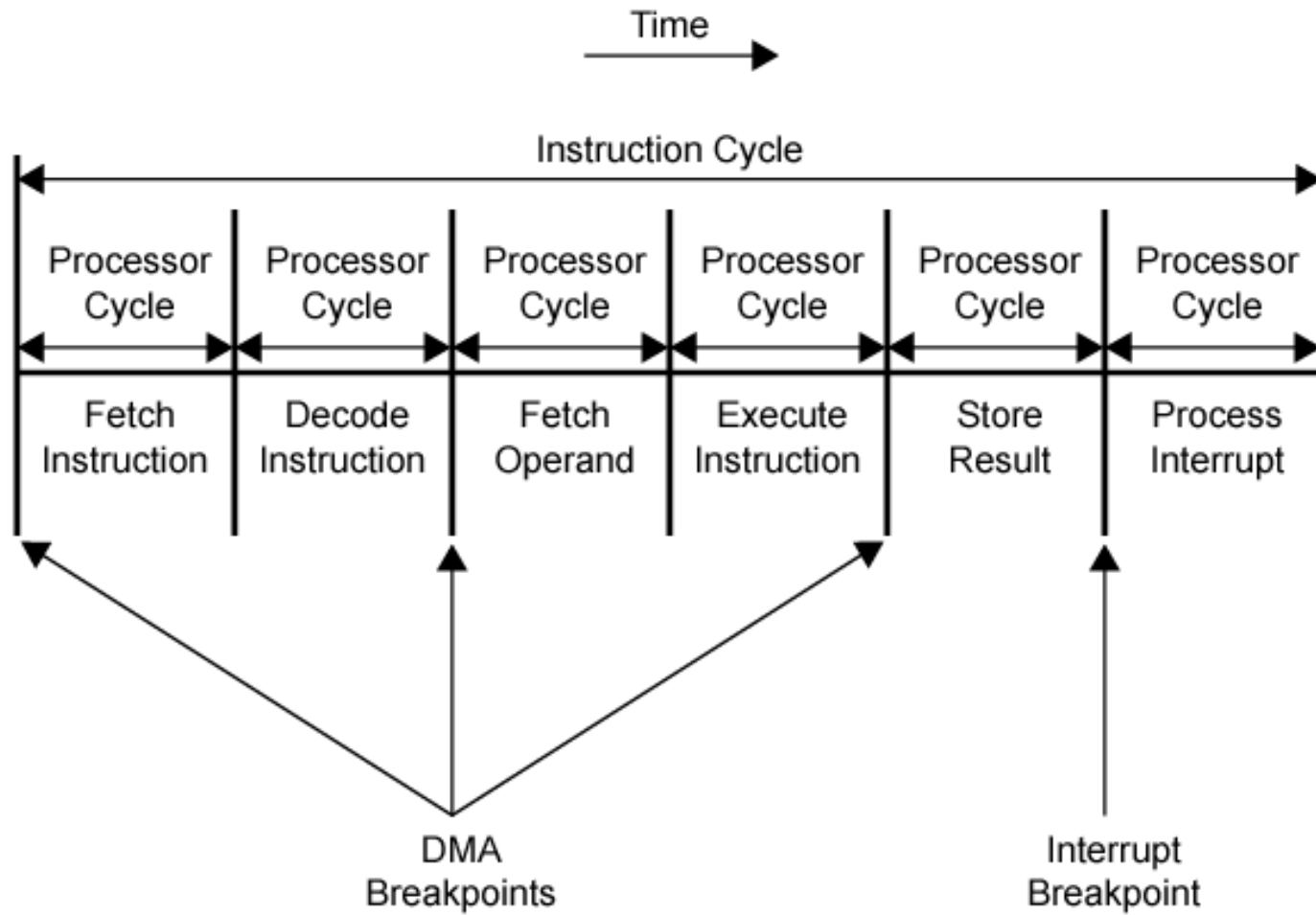
Aside

- What effect does caching memory have on DMA?

- What about on board cache?

- Hint: how much are the system buses available?

DMA Configurations (1)
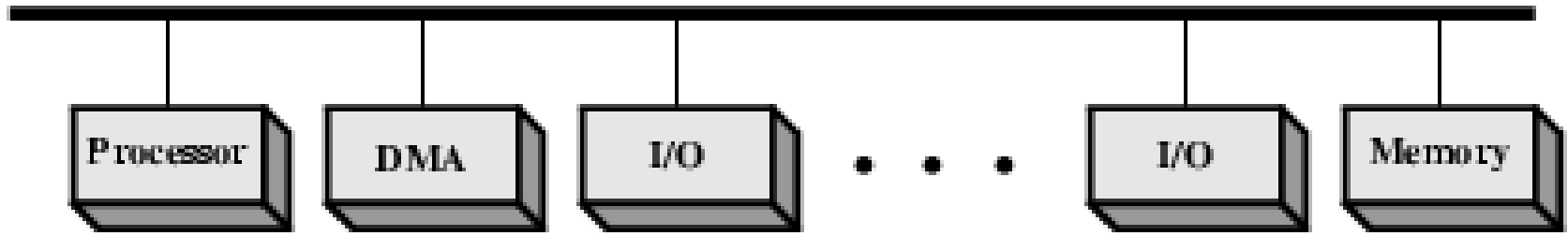
- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - —I/O to DMA then DMA to memory
- CPU is suspended twice

DMA Configurations (2)
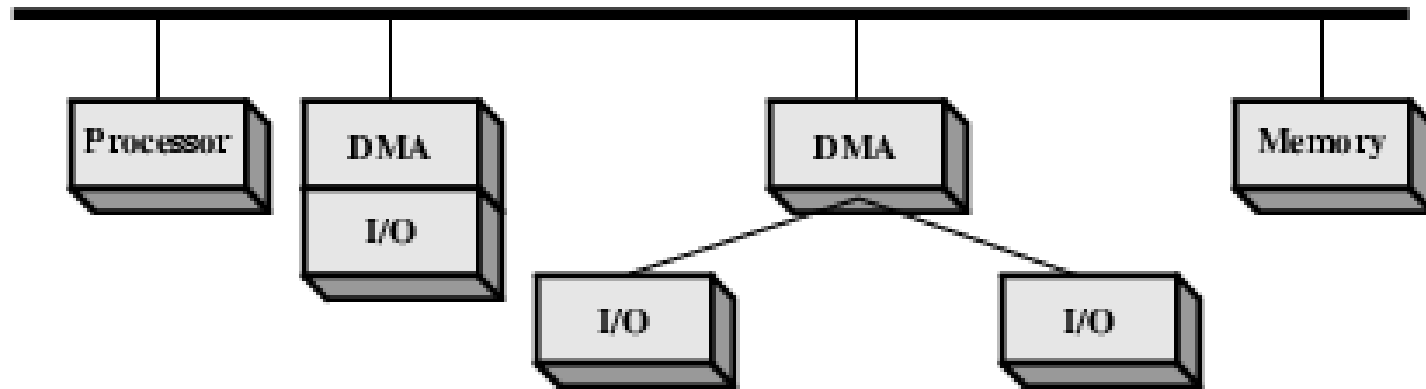
(b) Single-bus, Integrated DMA-I/O

- Single Bus, Integrated DMA controller
- Controller may support >1 device
- Each transfer uses bus once
  - —DMA to memory
- CPU is suspended once

DMA Configurations (3)

System bus

| Processor | DMA | Memory |

I/O bus

| I/O | I/O | I/O |
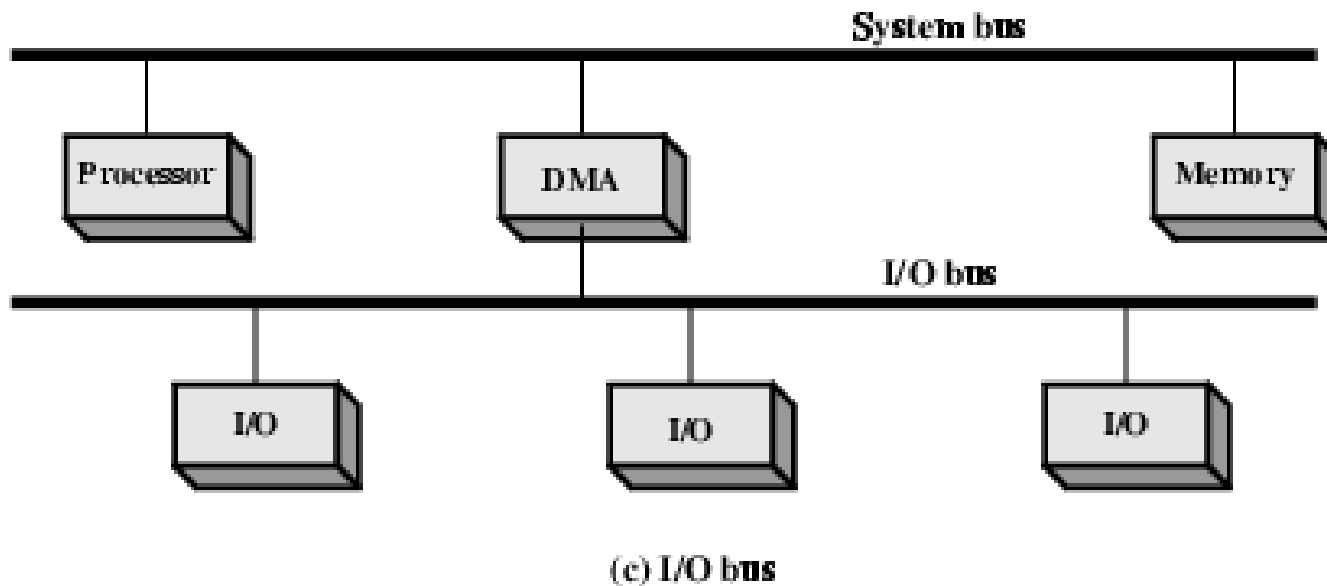
(c) I/O bus

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses the system bus once
  —DMA to memory
- CPU is suspended once

Intel 8237A DMA Controller

- Interfaces to 80x86 family and DRAM
- When DMA module needs buses it sends HOLD signal to processor
- CPU responds HLDA (hold acknowledge)
  - DMA module can use buses
- E.g. transfer data from memory to disk
  1. Device requests service of DMA by pulling DREQ (DMA request) high
  2. DMA puts high on HRQ (hold request),
  3. CPU finishes present bus cycle (not necessarily present instruction) and puts high on HDLA (hold acknowledge). HOLD remains active for duration of DMA
  4. DMA activates DACK (DMA acknowledge), telling device to start transfer
  5. DMA starts transfer by putting address of first byte on address bus and activating MEMR; it then activates IOW to write to peripheral. DMA decrements counter and increments address pointer. Repeat until count reaches zero
  6. DMA deactivates HRQ, giving bus back to CPU

8237 DMA Usage of Systems Bus
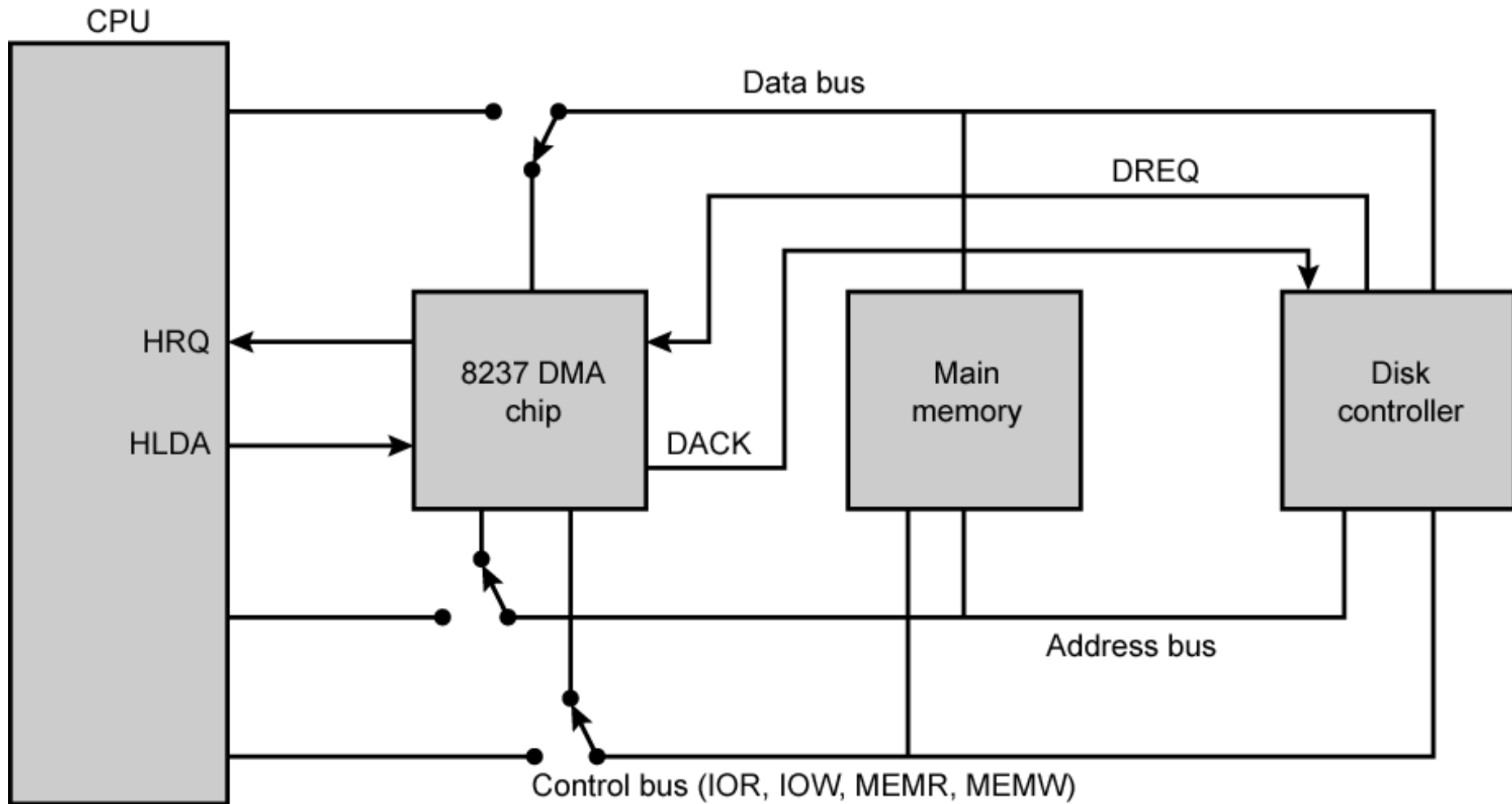
DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

Fly-By

- While DMA using buses processor idle
- Processor using bus, DMA idle
  - Known as fly-by DMA controller
- Data does not pass through and is not stored in DMA chip
  - DMA only between I/O port and memory
  - Not between two I/O ports or two memory locations
- Can do memory to memory via register
- 8237 contains four DMA channels
  - Programmed independently
  - Any one active
  - Numbered 0, 1, 2, and 3

The Evolution of the I/O function

1. CPU directly control a peripheral device

2. A controller or I/O module is added. CPU uses programmed I/O.

3. Interrupts are employed at the same configuration.

4. The I/O module is given direct access to memory via DMA.

5. The I/O module is enhanced to become a processor in its own right, with a specialized instruction set tailored for I/O. The CPU directs the I/O processor to execute an I/O program in memory. (I/O Channel)

6. The I/O processor has a local memory of its own and is, in fact, a computer in its own right. (I/O Processor)

I/O Channels

- I/O devices getting more sophisticated
- e.g. 3D graphics cards
- CPU instructs I/O controller to do transfer
- I/O controller does entire transfer
- Improves speed
  - Takes load off CPU
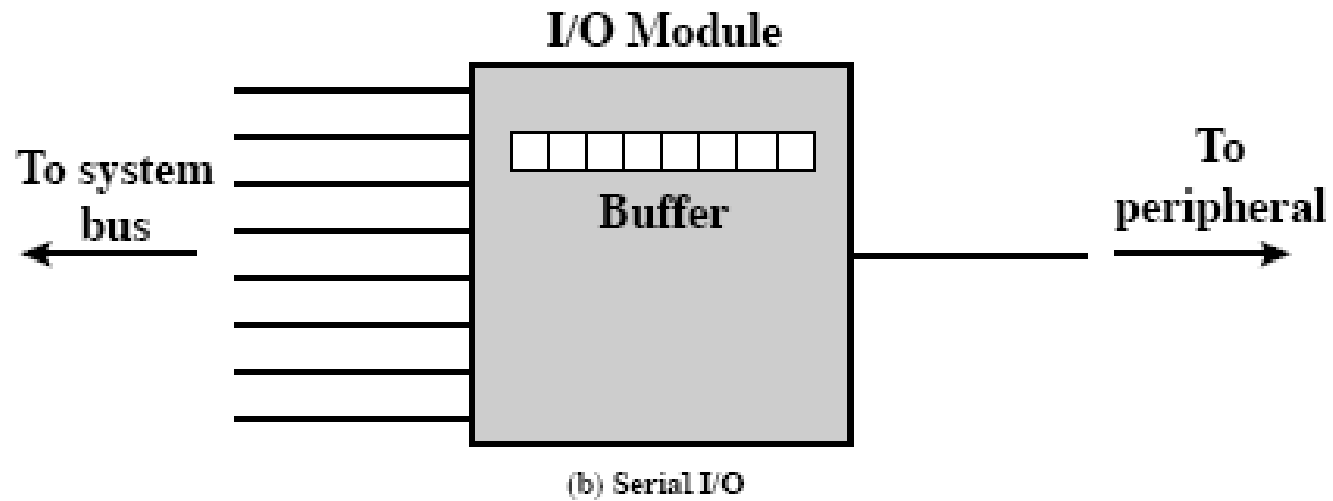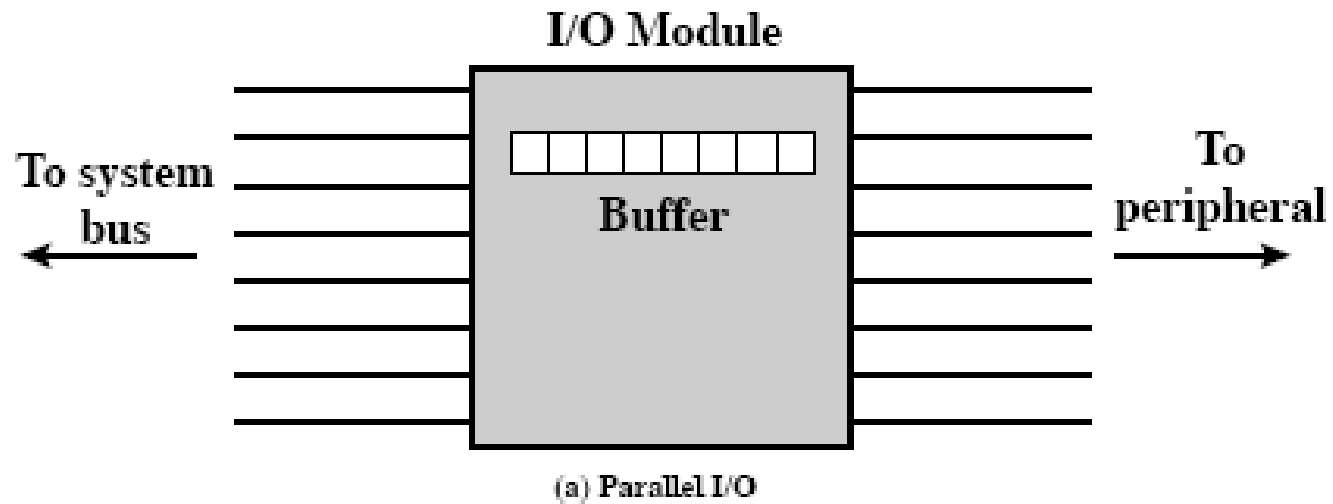 - Dedicated processor is faster

I/O Channel Architecture
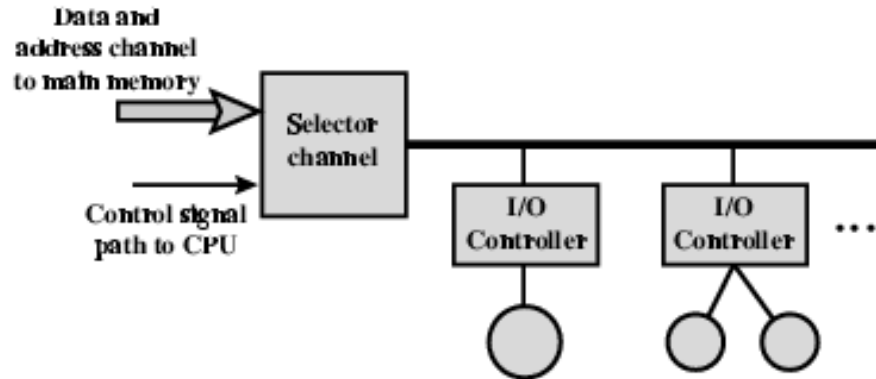
(a) Selector
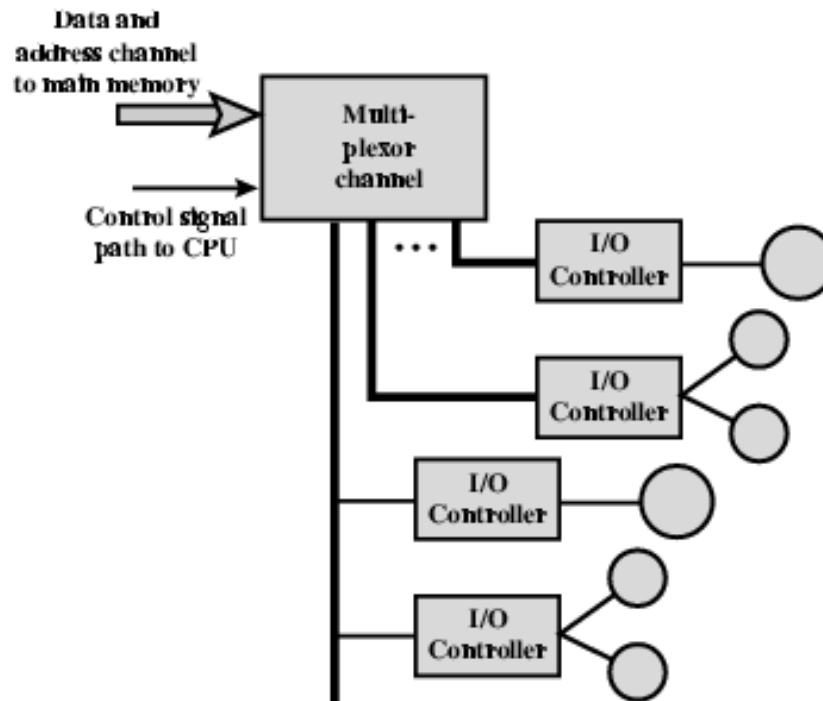
(b) Multiplexor

Interfacing

- Connecting devices together
- Bit of wire?
- Dedicated processor/memory/buses?
- E.g. FireWire, InfiniBand

I/O Module

To system bus

Buffer

To peripheral

(a) Parallel I/O

I/O Module

To system bus

Buffer

To peripheral

(b) Serial I/O

IEEE 1394 FireWire

- High performance serial bus
- Fast
- Low cost
- Easy to implement
- Also being used in digital cameras, VCRs and TV
- Serial transmission
- Simpler & cheaper connectors & cabling
- No need for shielding & synchronizing between wires

FireWire Configuration

- Daisy chain
- Up to 63 devices on single port
  - Really 64 of which one is the interface itself
- Up to 1022 FireWire buses can be connected with bridges
- Hot plugging (plug-in/out without power down)
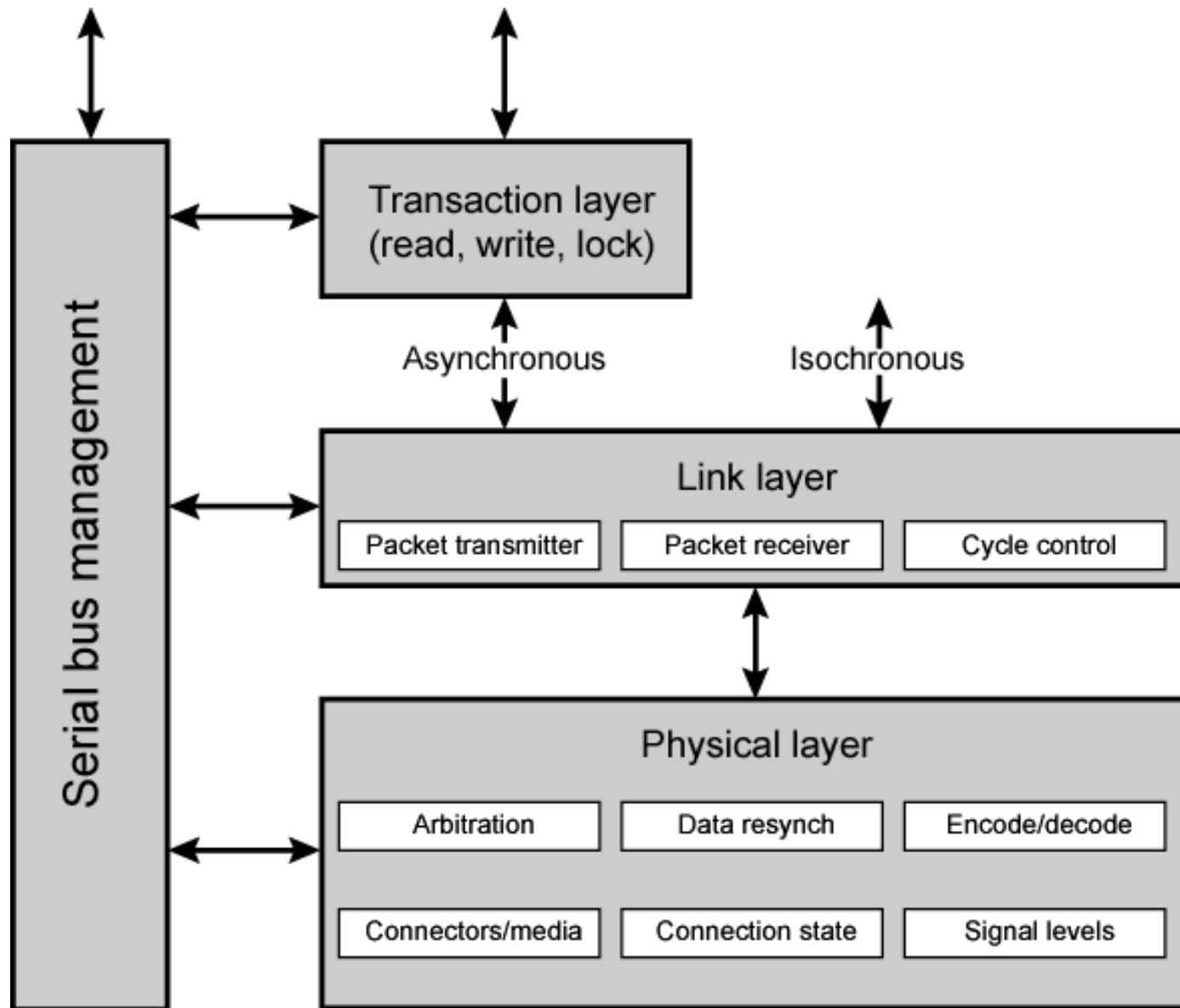- Automatic configuration
- No bus terminators
- May be tree structure

Simple FireWire Configuration
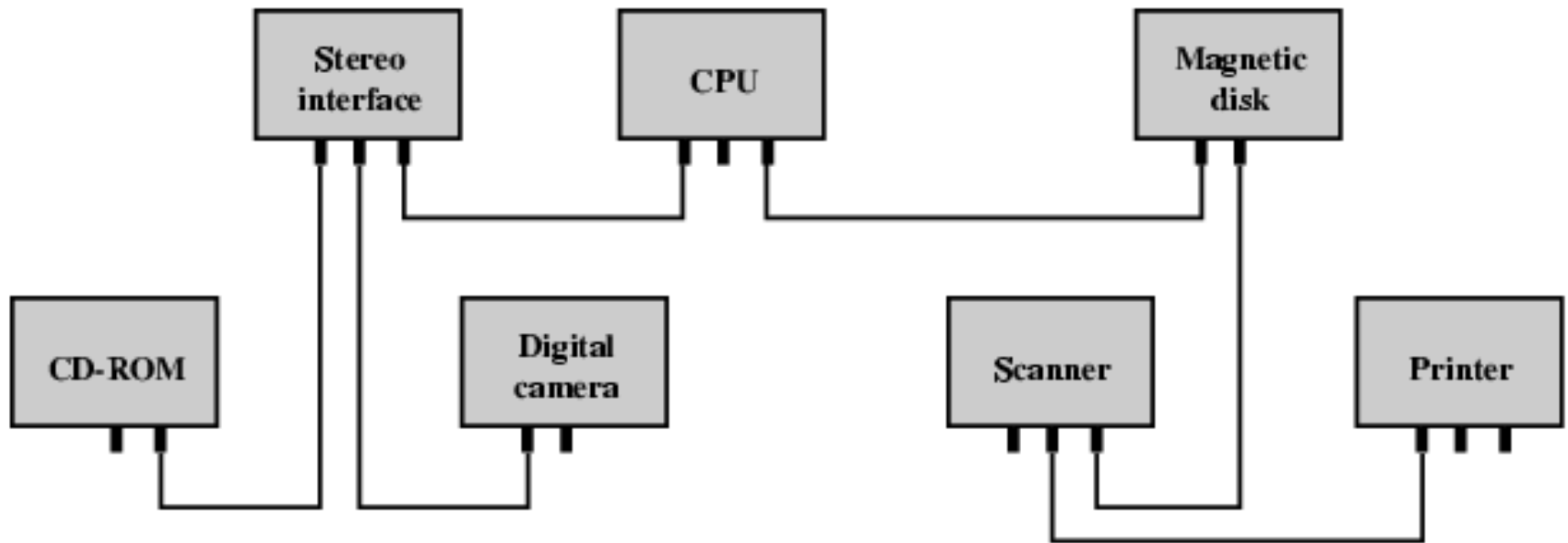
FireWire 3 Layer Stack (3 layers of Protocols)

- Physical
  - Transmission medium, electrical and signaling characteristics
- Link
  - Transmission of data in packets
- Transaction
  - Request-response protocol

# FireWire Protocol Stack

FireWire - Physical Layer

- Data rates from 25 to 400Mbps
- Two forms of arbitration
  - Based on tree structure
  - Root acts as arbiter
  - First come first served
  - Natural priority controls simultaneous requests
    - i.e. who is nearest to root
  - Fair arbitration (may compete only once in a fairness interval)
  - Urgent arbitration (for each nonurgent packet, 3 urgent packets in an fairness interval)

FireWire - Link Layer

- Two transmission types
  - —Asynchronous
    - Variable amount of data and several bytes of transaction data transferred as a packet
    - To explicit address
    - Acknowledgement returned
  - —Isochronous
    - Variable amount of data in sequence of fixed size packets at regular intervals
    - Simplified addressing
    - No acknowledgement
    - Guarantees that data can be delivered within a specified latency with a guaranteed data rate (ex. digital sound or video)

For a mixed traffic, a node (cycle master) periodically issues a cycle start packet. This signals all other nodes that an isochronous cycle has started. During that cycle only isochronous packets may be sent.
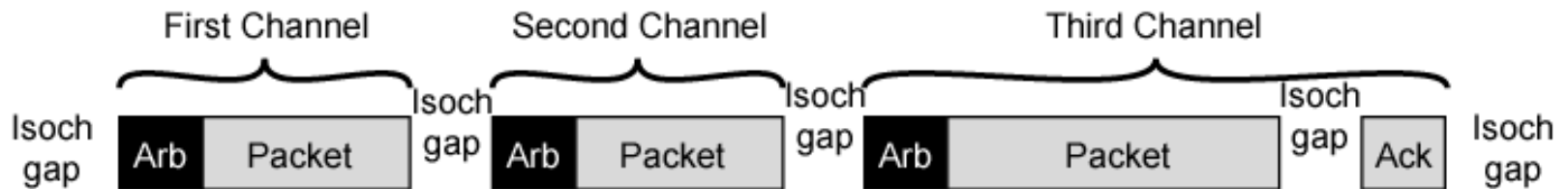
FireWire Subactions
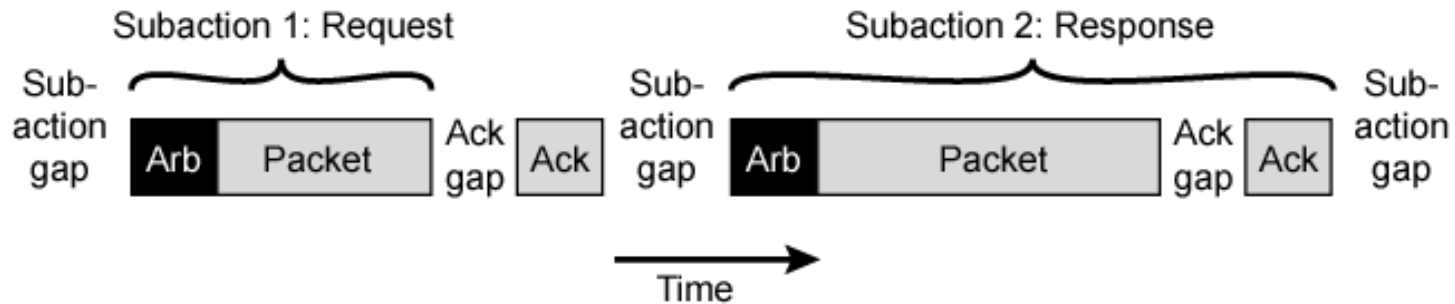
(a) Example asynchronous subaction
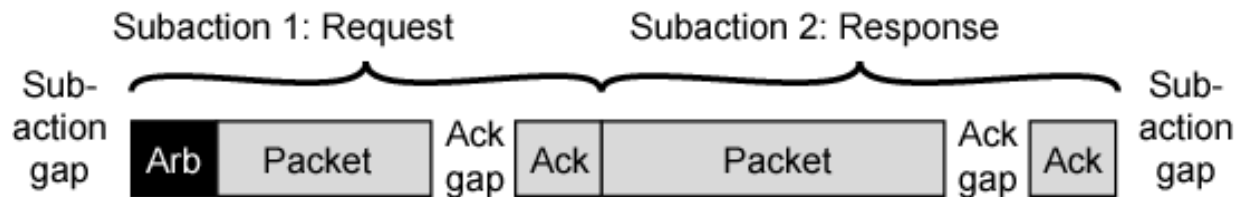
(b) Concatenated asynchronous subactions

(c) Example isochronous subactions

InfiniBand

- I/O specification aimed at high end servers
  - —Merger of Future I/O (Cisco, HP, Compaq, IBM) and Next Generation I/O (Intel)
- Version 1 released early 2001
- Architecture and spec. for data flow between processor and intelligent I/O devices
- Intended to replace PCI in servers
- Increased capacity, expandability, flexibility

InfiniBand Architecture

- Remote storage, networking and connection between servers
- Attach servers, remote storage, network devices to central fabric of switches and links
- Greater server density
- Scalable data centre
- Independent nodes added as required
- Up to 64,000 servers, storage systems and networking devices
- I/O distance from server up to
  - 17m using copper
  - 300m multimode fibre optic
  - 10km single mode fibre
- Up to 30Gbps

InfiniBand Switch Fabric
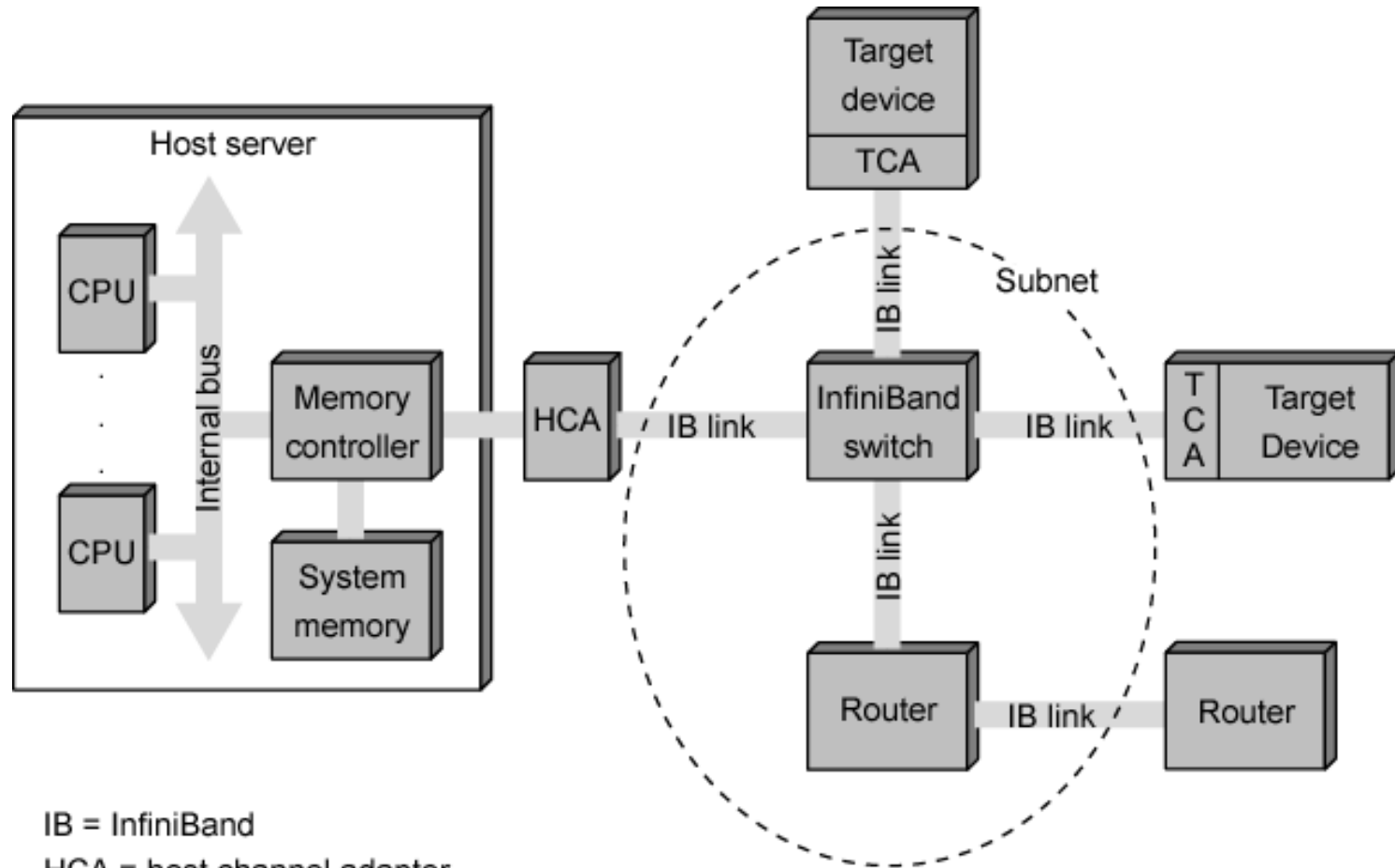
InfiniBand Operation

- 16 logical channels (virtual lanes) per physical link

- One lane for management, rest for data

- Data in stream of packets

- Virtual lane dedicated temporarily to end to end transfer

- Switch maps traffic from incoming to outgoing lane

InfiniBand Protocol Stack

IB = InfiniBand
WQE = work queue element
CQE = completion queue entry
QP = queue pair
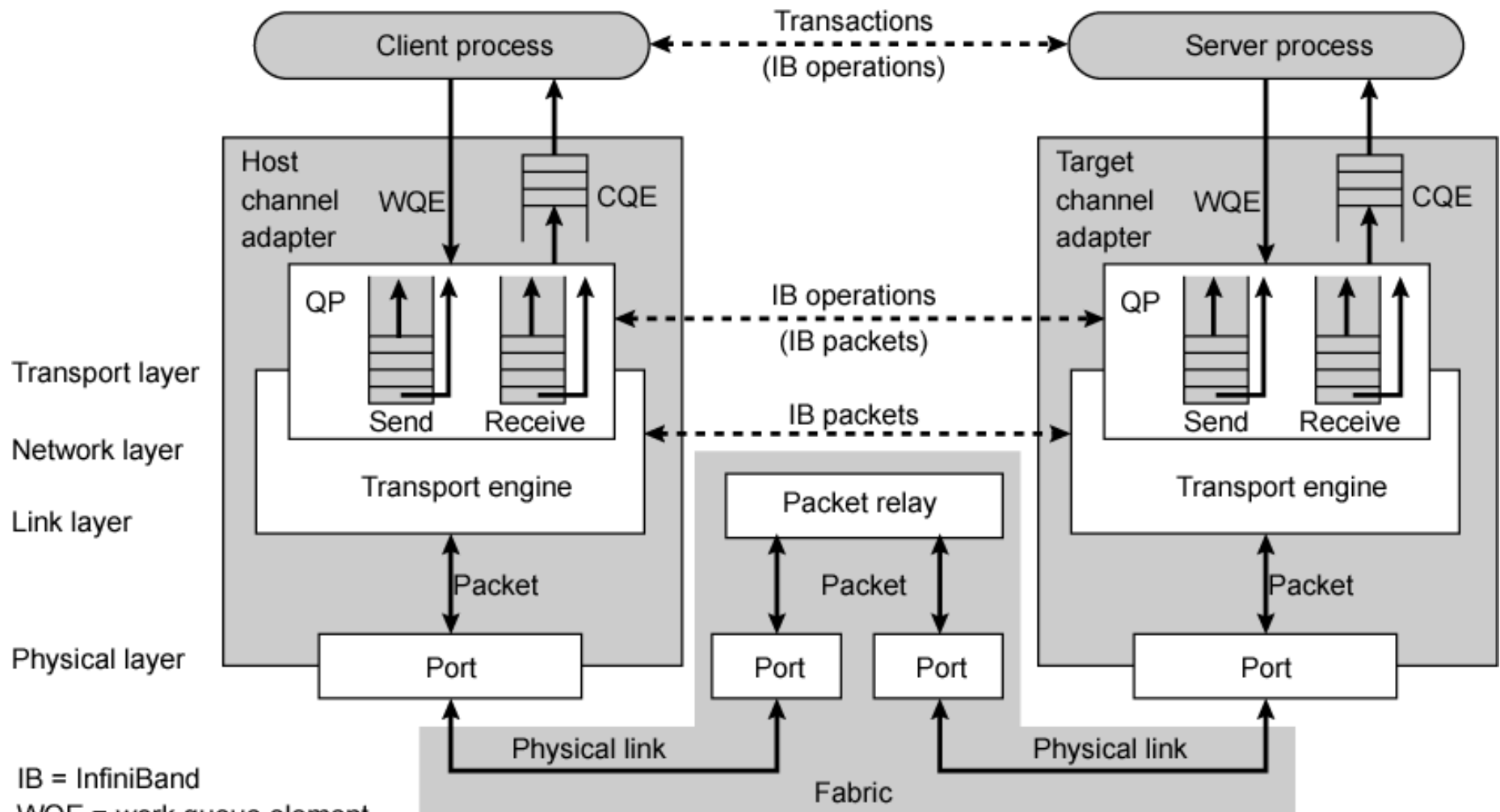
Foreground Reading

- Check out Universal Serial Bus (USB)
- Compare with other communication standards e.g. Ethernet