



+ **Input/Output**



Generic Model of an I/O Module

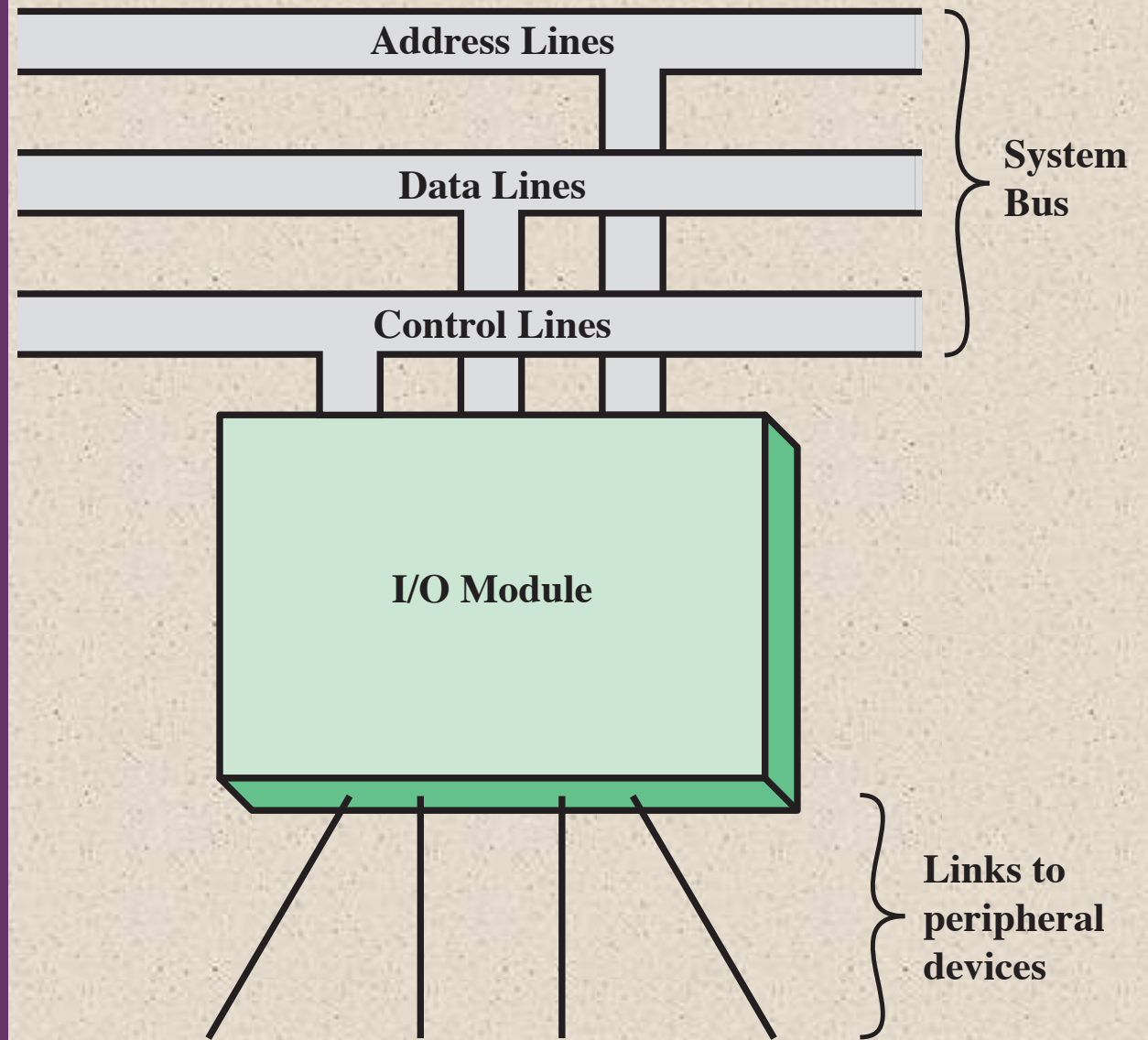


Figure 7.1 Generic Model of an I/O Module

+ External Devices



- Provide a means of exchanging data between the external environment and the computer
- Attach to the computer by a link to an I/O module
 - The link is used to exchange control, status, and data between the I/O module and the external device
- *peripheral device*
 - An external device connected to an I/O module

Three categories:

- Human readable
 - Suitable for communicating with the computer user
 - Video display terminals (VDTs), printers
- Machine readable
 - Suitable for communicating with equipment
 - Magnetic disk and tape systems, sensors and actuators
- Communication
 - Suitable for communicating with remote devices such as a terminal, a machine readable device, or another computer



External Device Block Diagram

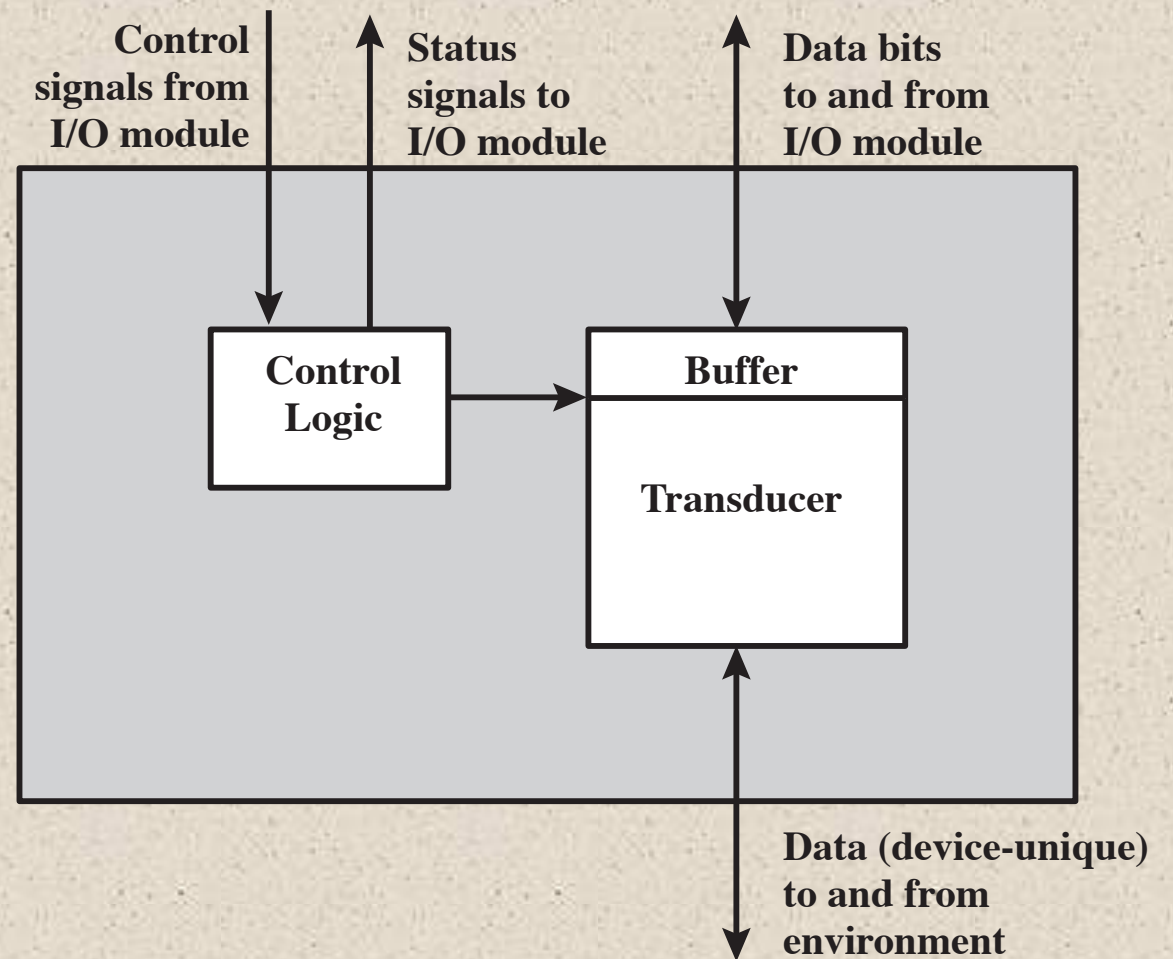


Figure 7.2 Block Diagram of an External Device



Keyboard/Monitor

International Reference Alphabet (IRA)

- Basic unit of exchange is the character
 - Associated with each character is a code
 - Each character in this code is represented by a unique 7-bit binary code
 - 128 different characters can be represented
- Characters are of two types:
 - Printable
 - Alphabetic, numeric, and special characters that can be printed on paper or displayed on a screen
 - Control
 - Have to do with controlling the printing or displaying of characters
 - Example is carriage return
 - Other control characters are concerned with communications procedures

Most common means of computer/user interaction

User provides input through the keyboard

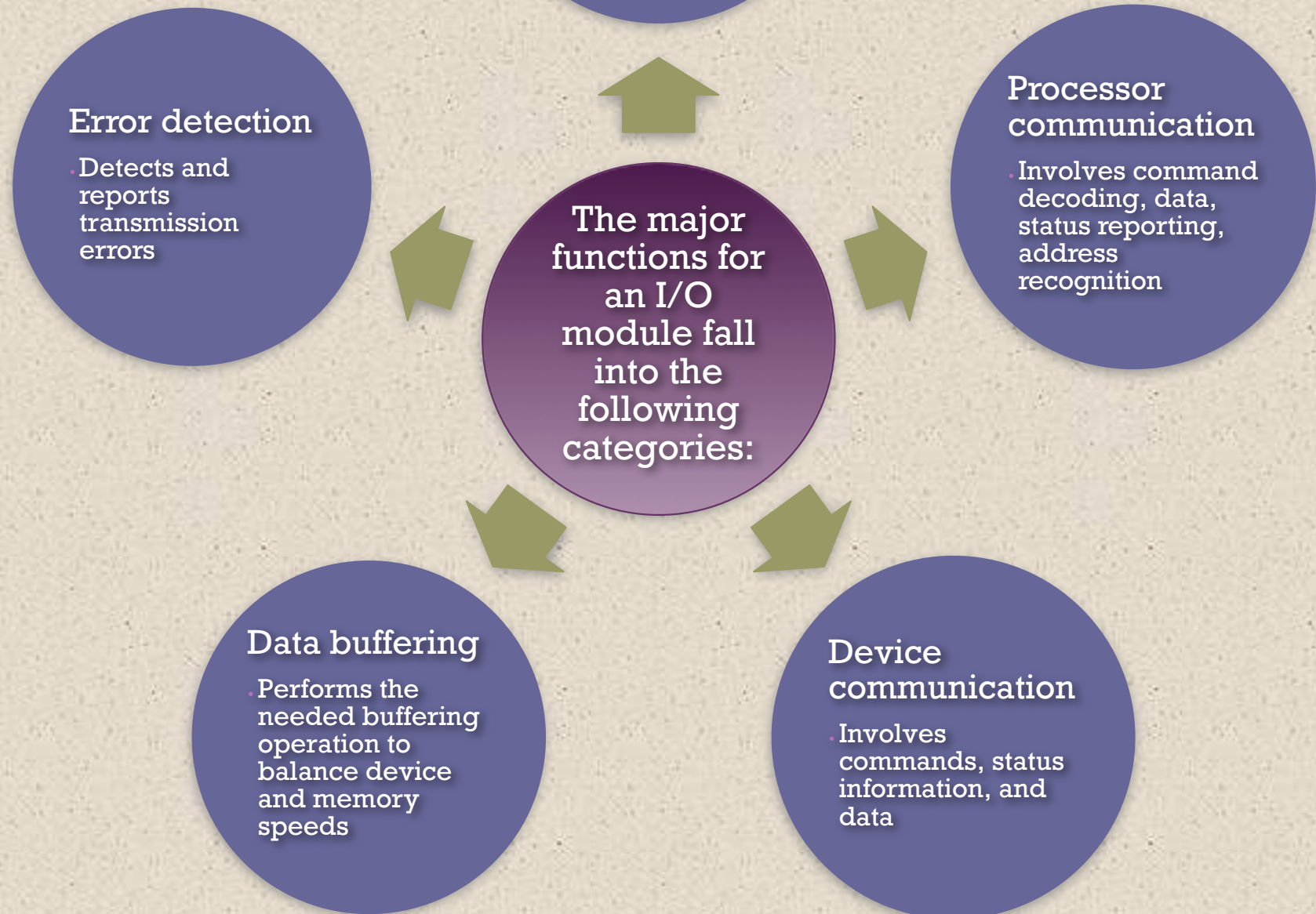
The monitor displays data provided by the computer

Keyboard Codes

- When the user depresses a key it generates an electronic signal that is interpreted by the transducer in the keyboard and translated into the bit pattern of the corresponding IRA code
- This bit pattern is transmitted to the I/O module in the computer
- On output, IRA code characters are transmitted to an external device from the I/O module
- The transducer interprets the code and sends the required electronic signals to the output device either to display the indicated character or perform the requested control function

I/O Modules

Module Function



I/O Module Structure

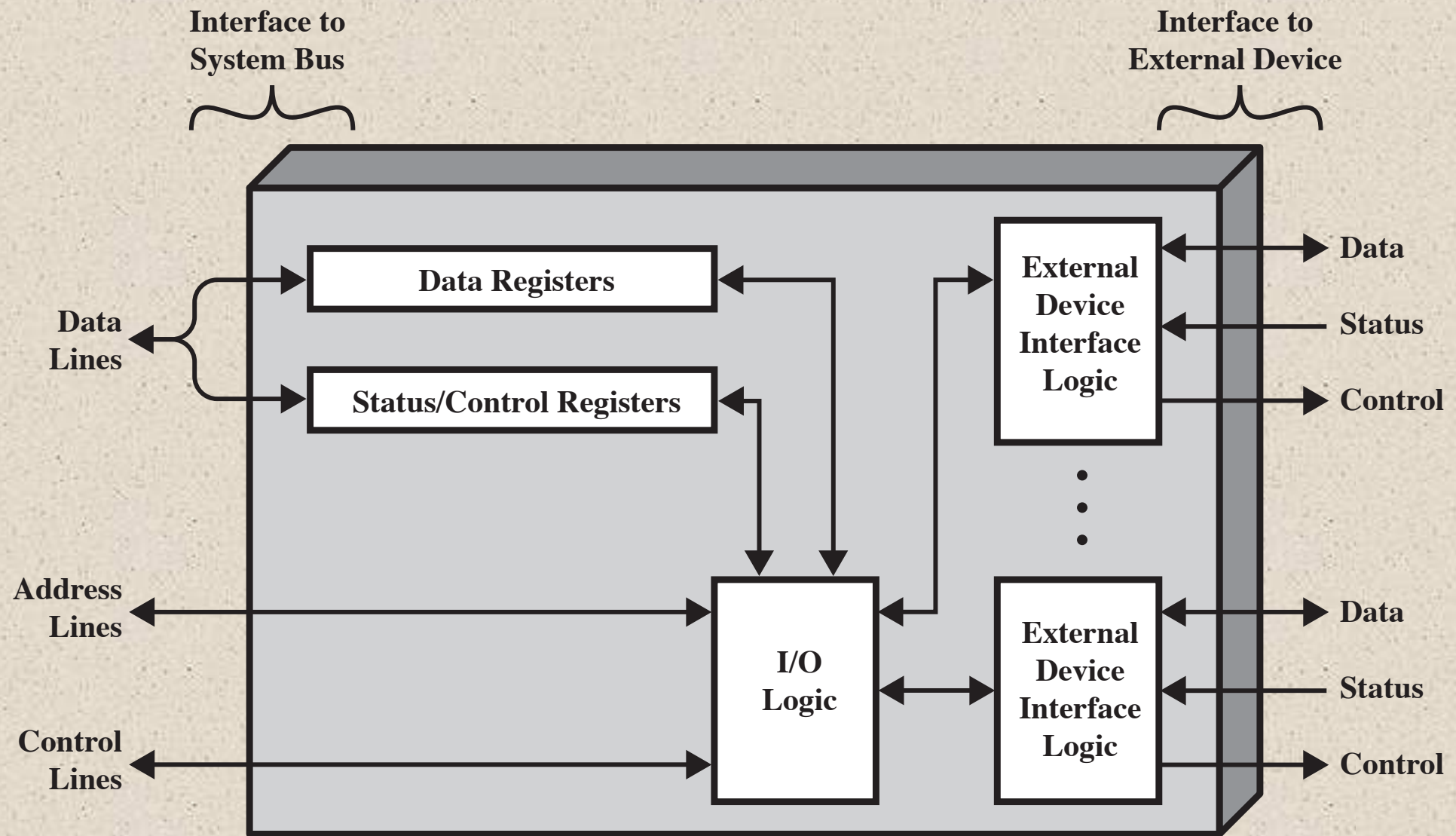
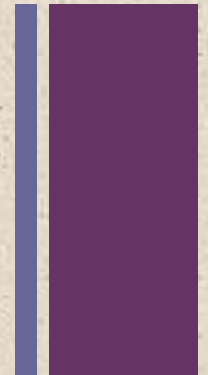


Figure 7.3 Block Diagram of an I/O Module



Programmed I/O



- Three techniques are possible for I/O operations:
- Programmed I/O
 - Data are exchanged between the processor and the I/O module
 - Processor executes a program that gives it direct control of the I/O operation
 - When the processor issues a command it must wait until the I/O operation is complete
 - If the processor is faster than the I/O module this is wasteful of processor time
- Interrupt-driven I/O
 - Processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- Direct memory access (DMA)
 - The I/O module and main memory exchange data directly without processor involvement

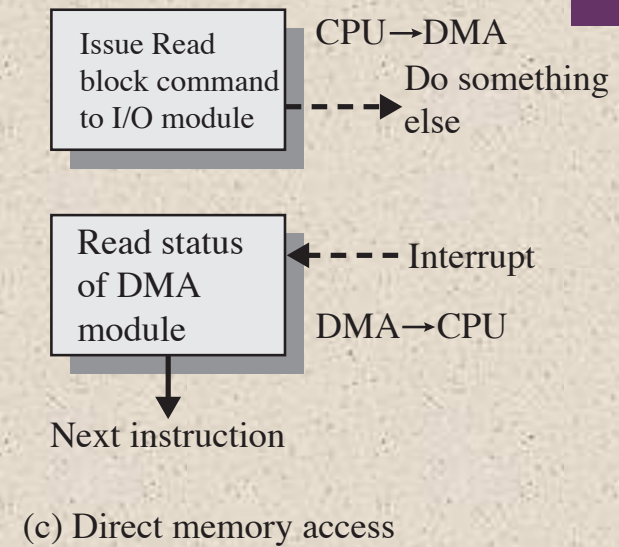
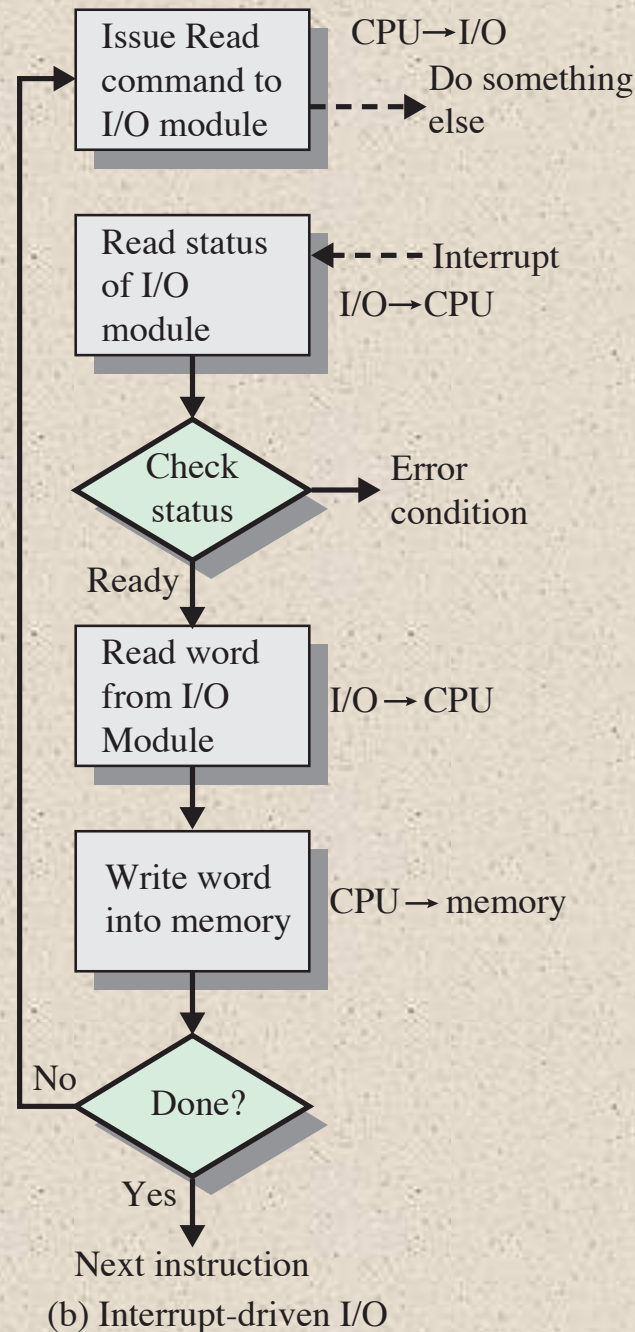
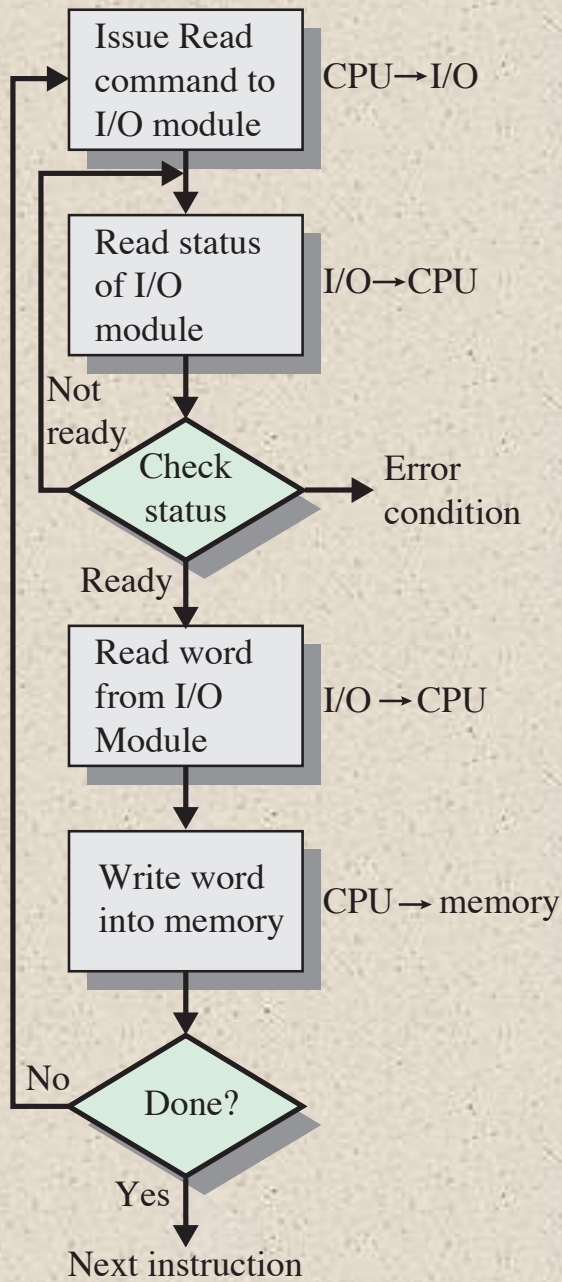
I/O Techniques

+

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

+ I/O Commands

- There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:
 - 1) Control
 - used to activate a peripheral and tell it what to do
 - 2) Test
 - used to test various status conditions associated with an I/O module and its peripherals
 - 3) Read
 - causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer
 - 4) Write
 - causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral



Three Techniques for Input of a Block of Data

Figure 7.4 Three Techniques for Input of a Block of Data

I/O Instructions

With programmed I/O there is a close correspondence between the I/O-related instructions that the processor fetches from memory and the I/O commands that the processor issues to an I/O module to execute the instructions

The form of the instruction depends on the way in which external devices are addressed

Each I/O device connected through I/O modules is given a unique identifier or address

When the processor issues an I/O command, the command contains the address of the desired device

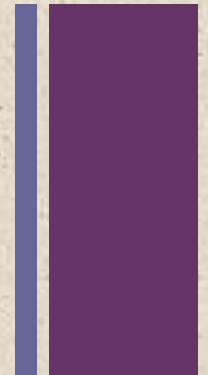
Thus each I/O module must interpret the address lines to determine if the command is for itself

Memory-mapped I/O

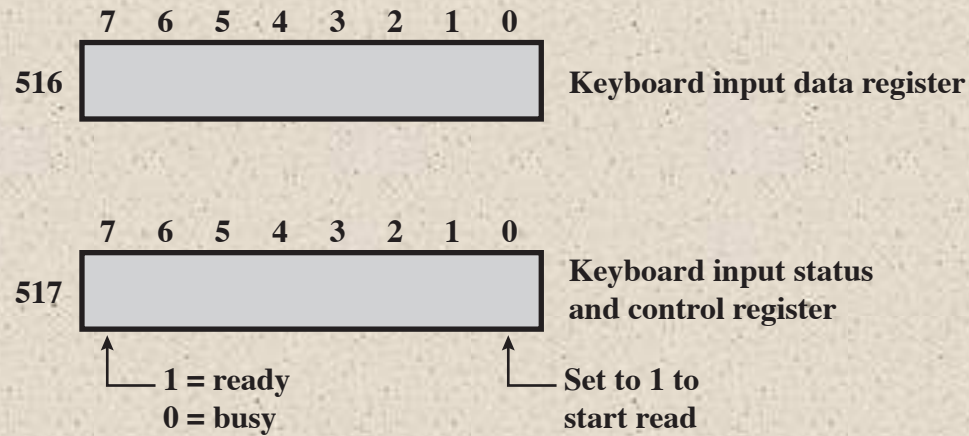
There is a single address space for memory locations and I/O devices

A single read line and a single write line are needed on the bus

+ I/O Mapping Summary



- 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



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

Memory
Mapped
I/O

Isolated
I/O

Figure 7.5 Memory-Mapped and Isolated I/O

Interrupt-Driven I/O



The problem with programmed I/O is that the processor has to wait a long time for the I/O module to be ready for either reception or transmission of data

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work

The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor

The processor executes the data transfer and resumes its former processing

Simple Interrupt Processing

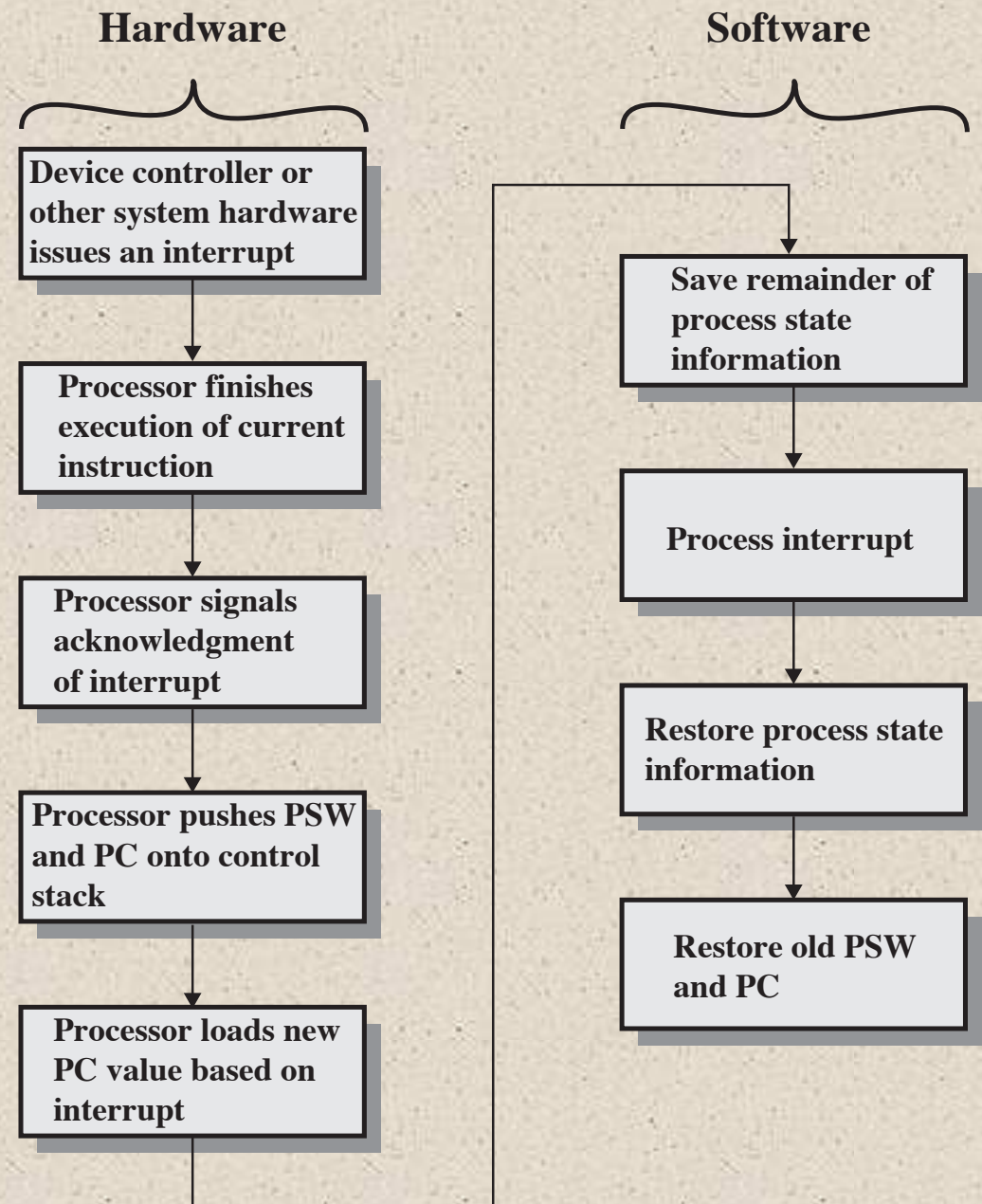


Figure 7.6 Simple Interrupt Processing



Changes in Memory and Registers for an Interrupt

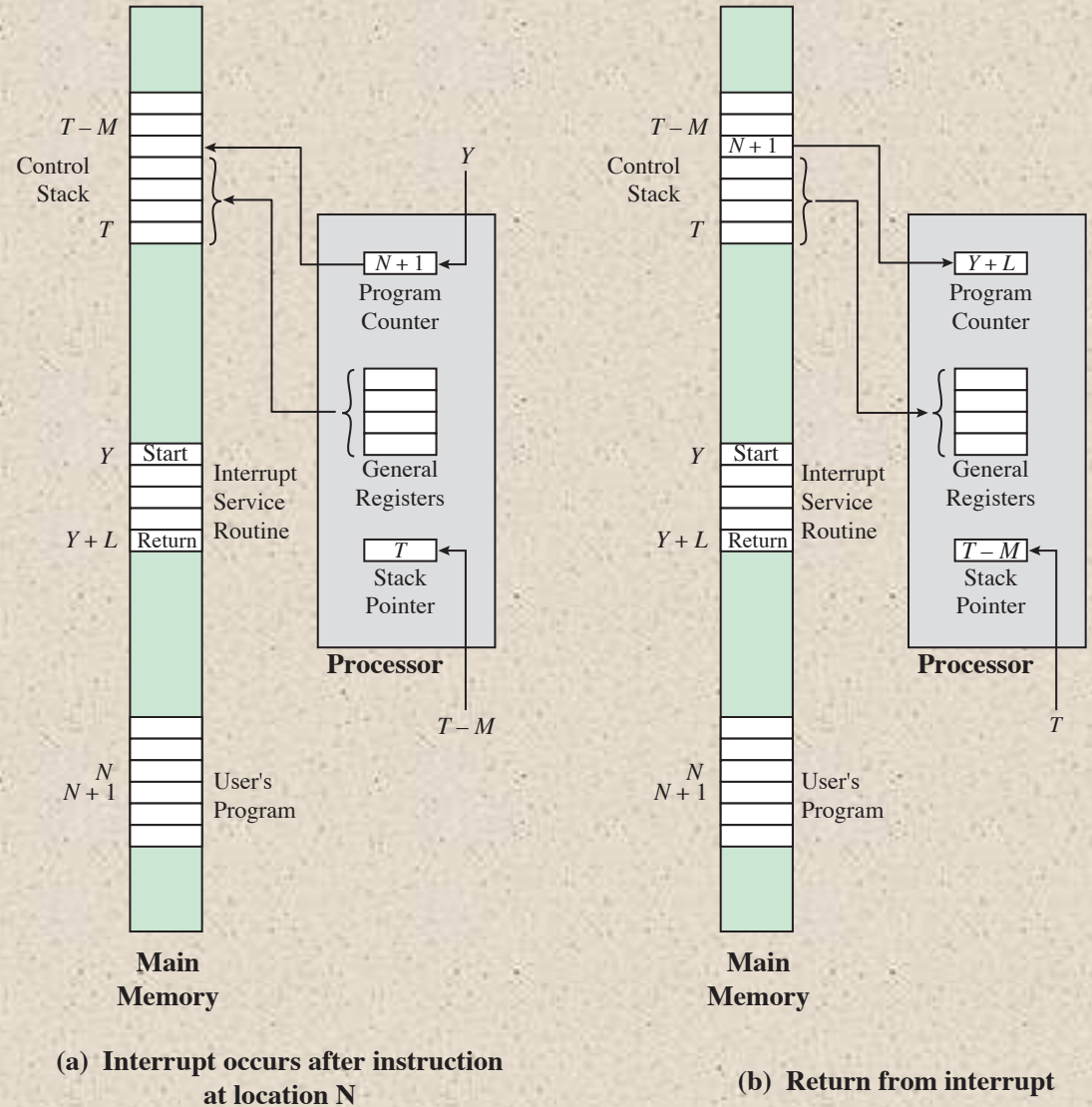


Figure 7.7 Changes in Memory and Registers for an Interrupt

Design Issues



Two design issues arise in implementing interrupt I/O:

- Because there will be multiple I/O modules how does the processor determine which device issued the interrupt?
- If multiple interrupts have occurred how does the processor decide which one to process?

+ Device Identification

Four general categories of techniques are in common use:

- **Multiple interrupt lines**
 - Between the processor and the I/O modules
 - Most straightforward approach to the problem
 - Consequently even if multiple lines are used, it is likely that each line will have multiple I/O modules attached to it
- **Software poll**
 - When processor detects an interrupt it branches to an interrupt-service routine whose job is to poll each I/O module to determine which module caused the interrupt
 - Time consuming
- **Daisy chain (hardware poll, vectored)**
 - The interrupt acknowledge line is daisy chained through the modules
 - Vector – address of the I/O module or some other unique identifier
 - Vectored interrupt – processor uses the vector as a pointer to the appropriate device-service routine, avoiding the need to execute a general interrupt-service routine first
- **Bus arbitration (vectored)**
 - An I/O module must first gain control of the bus before it can raise the interrupt request line
 - When the processor detects the interrupt it responds on the interrupt acknowledge line
 - Then the requesting module places its vector on the data lines

Intel 82C59A Interrupt Controller

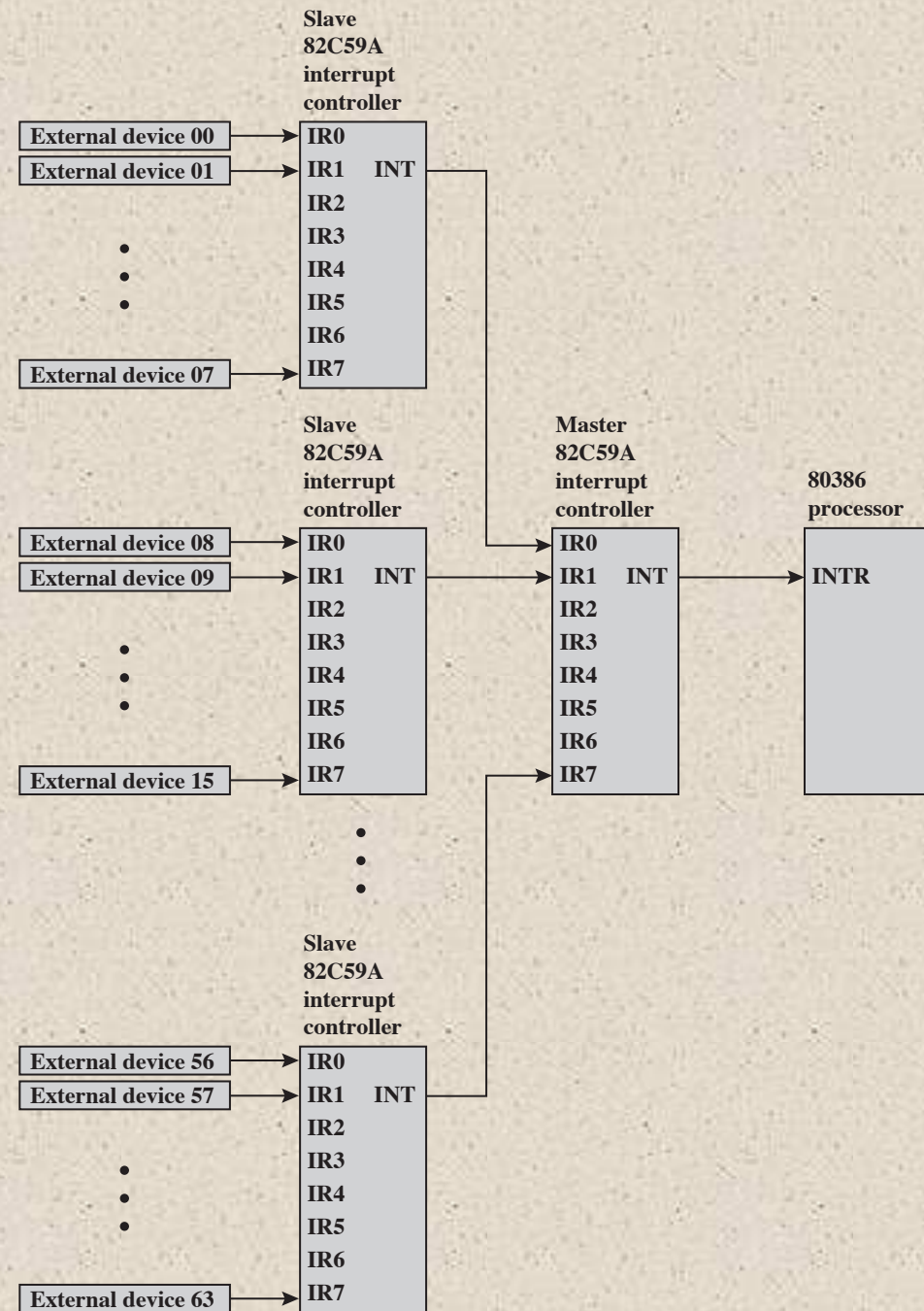


Figure 7.8 Use of the 82C59A Interrupt Controller

Drawbacks of Programmed and Interrupt-Driven I/O

- Both forms of I/O suffer from two inherent drawbacks:
 - 1) The I/O transfer rate is limited by the speed with which the processor can test and service a device
 - 2) The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer
- When large volumes of data are to be moved a more efficient technique is *direct memory access* (DMA)



Typical DMA Module Diagram

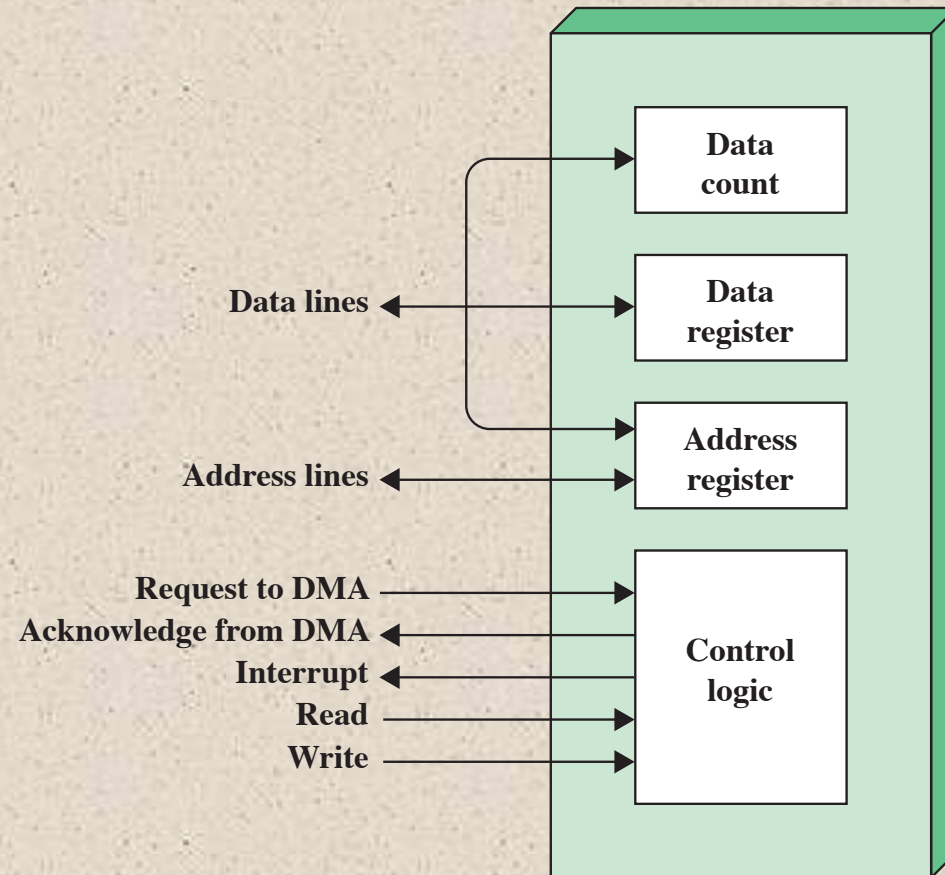


Figure 7.11 Typical DMA Block Diagram

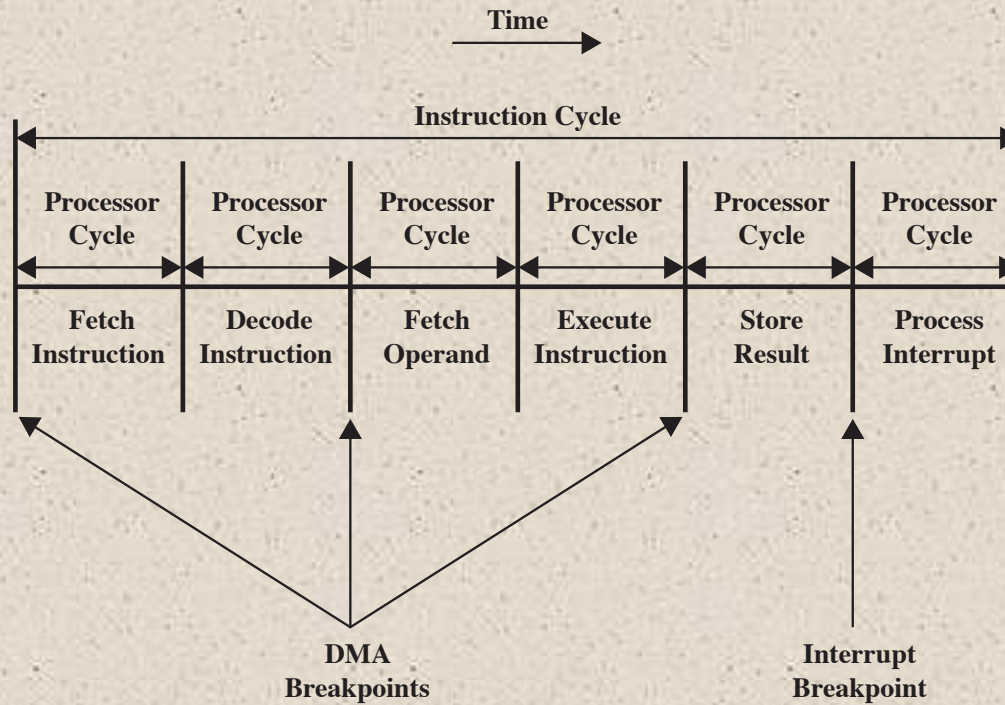


Figure 7.12 DMA and Interrupt Breakpoints During an Instruction Cycle

DMA

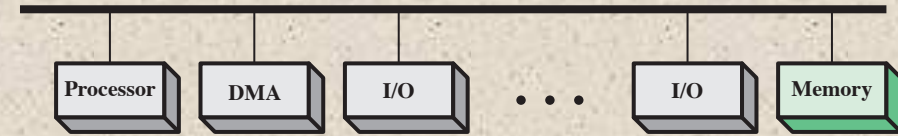
DMA

+

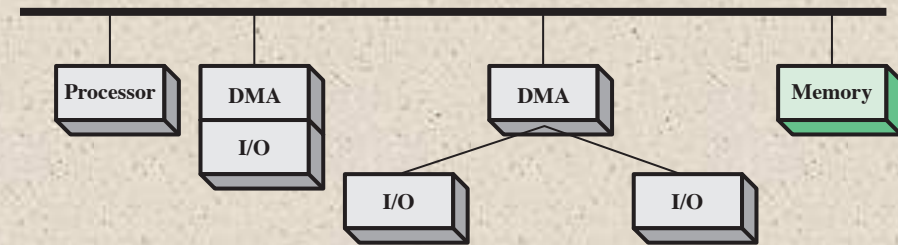
DMA Operation



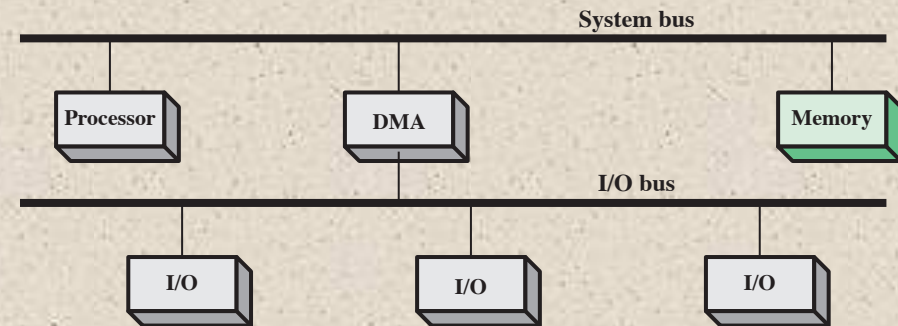
Alternative DMA Configurations



(a) Single-bus, detached DMA



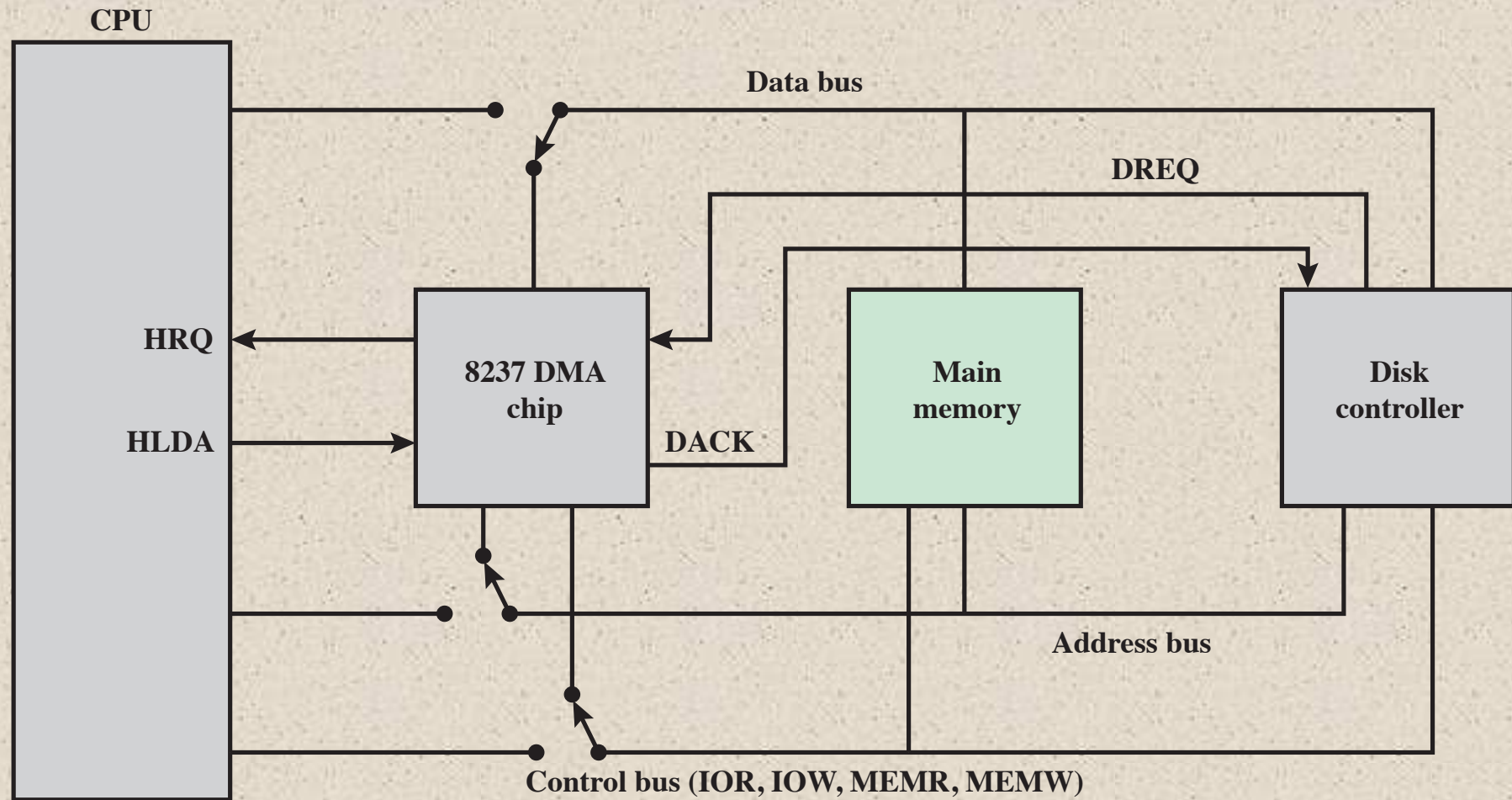
(b) Single-bus, Integrated DMA-I/O



(c) I/O bus

Figure 7.13 Alternative DMA Configurations

8237 DMA Usage of System Bus

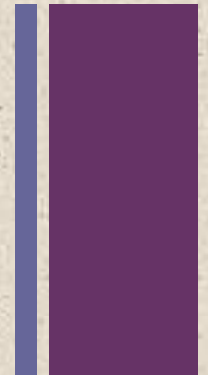


DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

Figure 7.14 8237 DMA Usage of System Bus



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

+ Evolution of the I/O Function

1. The CPU directly controls a peripheral device.
2. A controller or I/O module is added. The CPU uses programmed I/O without interrupts.
3. Same configuration as in step 2 is used, but now interrupts are employed. The CPU need not spend time waiting for an I/O operation to be performed, thus increasing efficiency.
4. The I/O module is given direct access to memory via DMA. It can now move a block of data to or from memory without involving the CPU, except at the beginning and end of the transfer.
5. The I/O module is enhanced to become a processor in its own right, with a specialized instruction set tailored for I/O
6. The I/O module has a local memory of its own and is, in fact, a computer in its own right. With this architecture a large set of I/O devices can be controlled with minimal CPU involvement.