Input/Output Organization

Review Notes

Yul Williams, D.Sc.

Purpose of Course Notes

These course notes are provided to increase your understanding of the course materials presented to you in the Course Content area of the WebTycho online classroom. They will generally follow the outline of the online modules and the course textbook.

The course notes offer insight into the various topics presented in the topics covered online, but, they are not to be considered a complete coverage of each topic area.

Goals

- At the conclusion of this session, the student should be able to:
- Distinguish between programmed and Interrupt-driven I/O
- Identify the 3 forms of interrupts
- Explain the operation of a Direct Memory Access Controller
- Describe the function of a Peripheral Processing Unit

Outline

- Introduction to Input/Output Organization
- Asynchronous Data Transfers
- Programmed I/O
- Interrupts and Interrupt Processing
- Direct Memory Access
- Input/Output Processors
- Serial Communications
- Summary

Asynchronous Data Transfers

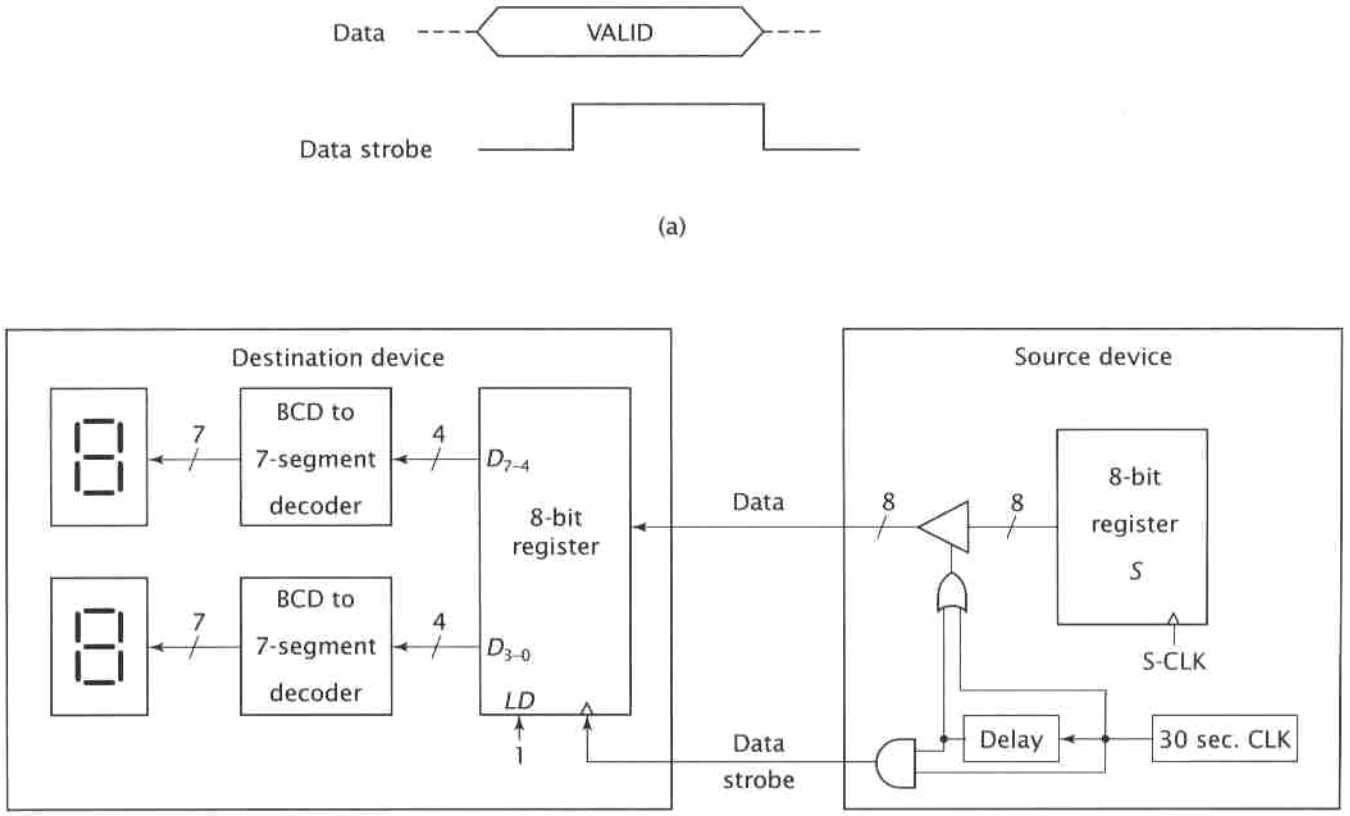
Asynchronous Data Transfers

- All of our previous examples for data exchange between components have been facilitated by a synchronous signal (namely the clock)
- Asynchronous data transfers, however, happen without the aid of a clock
- Some mechanism, however, must be implemented to assure data transfer
- There are 4 types of asynchronous data transfer

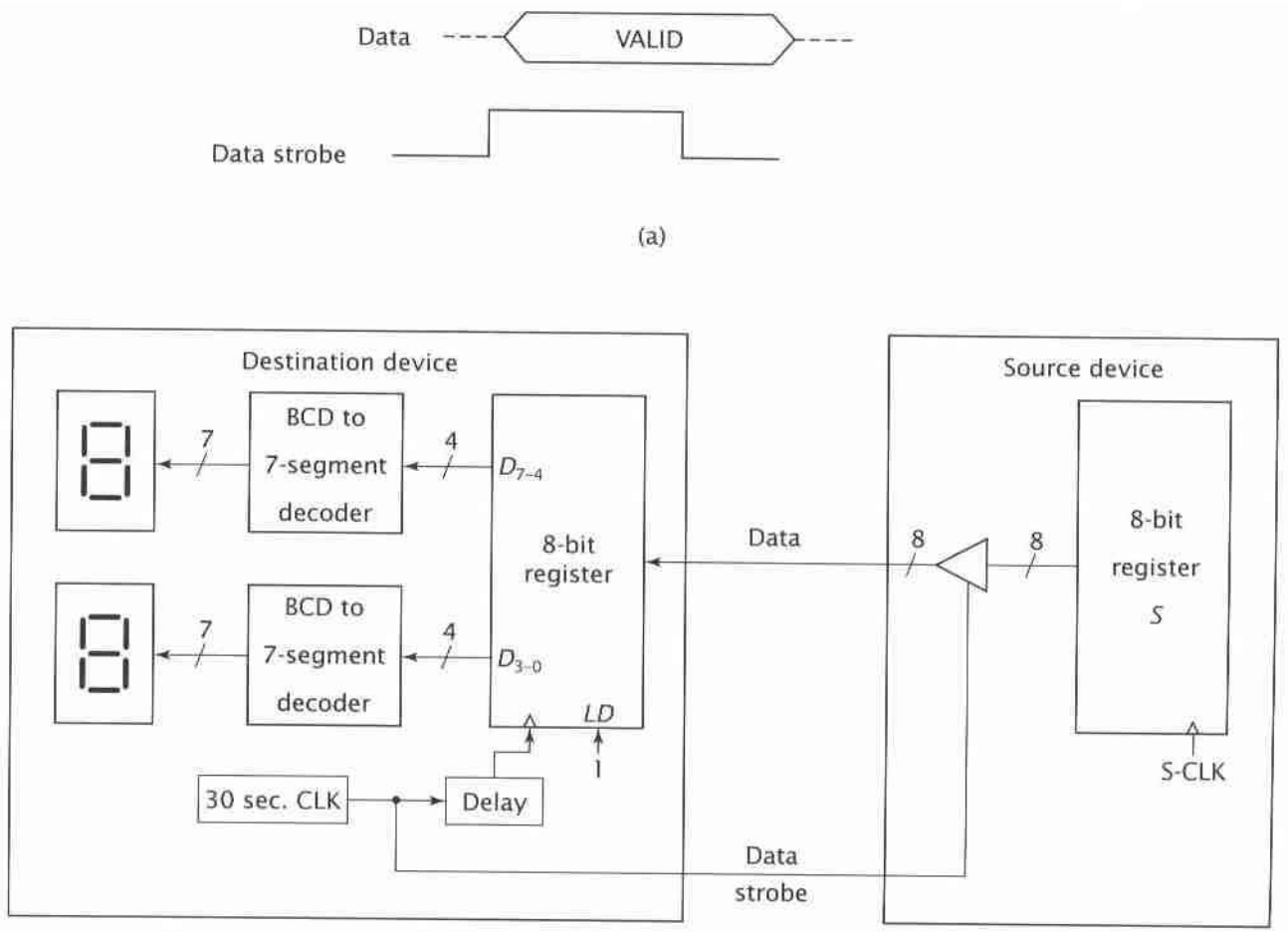
Four Types of Asynchronous Data Transfer

- Source-initiated data transfer without handshaking
- Destination-initiated data transfer without handshaking
- Source-initiated data transfer with handshaking
- Destination-initiated data transfer with handshaking

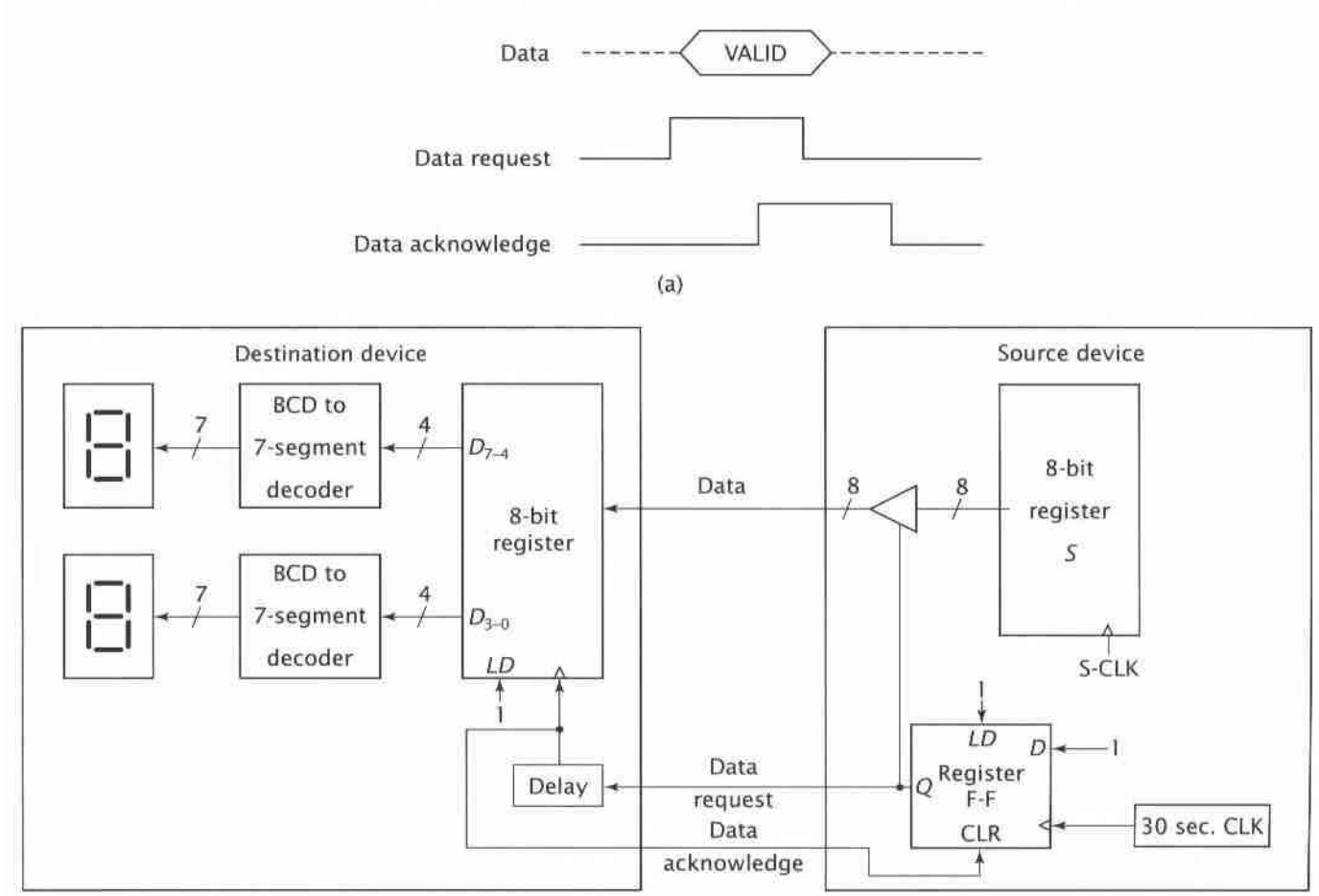
Source-initiated data transfer without handshaking



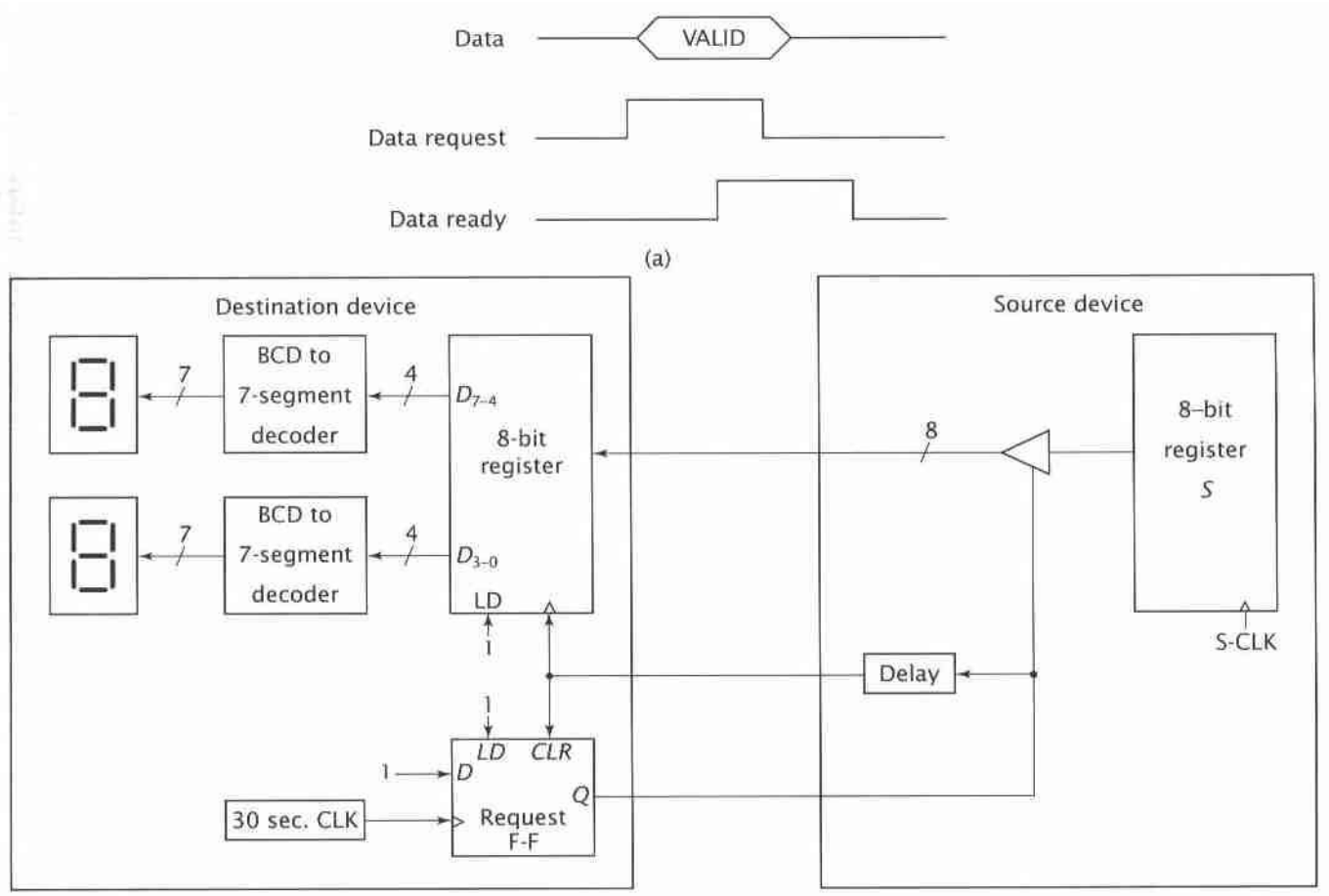
Destination-initiated data transfer without handshaking



Source-initiated data transfer with handshaking



Destination-initiated data transfer with handshaking



Programmed I/O

I/O Mapping

Memory Mapping

Programming I/O

- Two types of instructions can support I/O:
  - special-purpose I/O instructions;
  - memory-mapped load/store instructions.
- Intel x86 provides `in`, `out` instructions. Most other CPUs use memory-mapped I/O.
- I/O instructions do not preclude memory-mapped I/O.

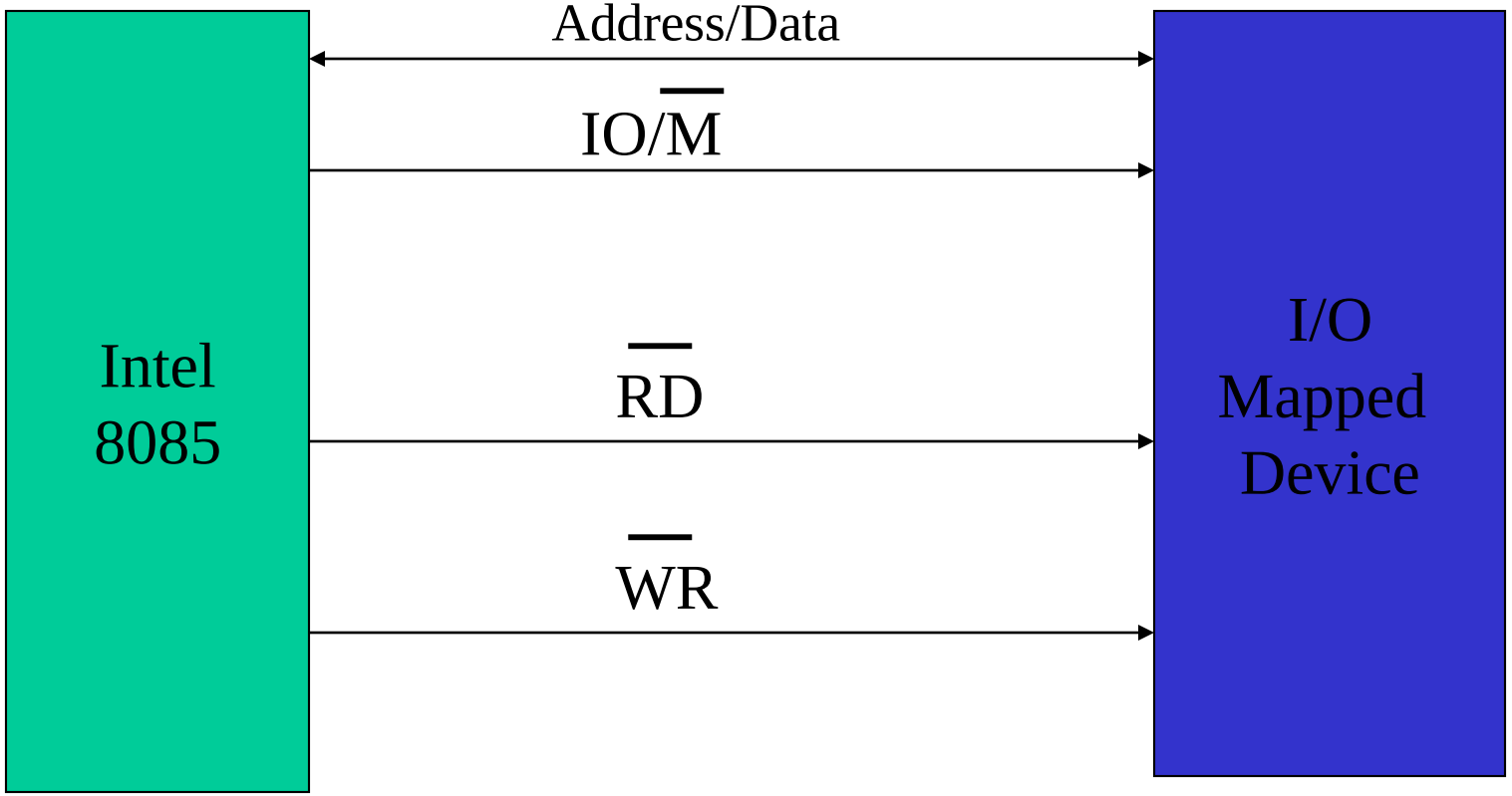
Example of Programmed I/O

```
;Intel 8085 Assembly Code Example
ORG 100H      ;Program starts at location 100H
MVI B, 05H    ; B = 5
MOV A, B      ; Accumulator = B
OUT 200H      ; Device_at_IO_location_0200H = 5
END
```

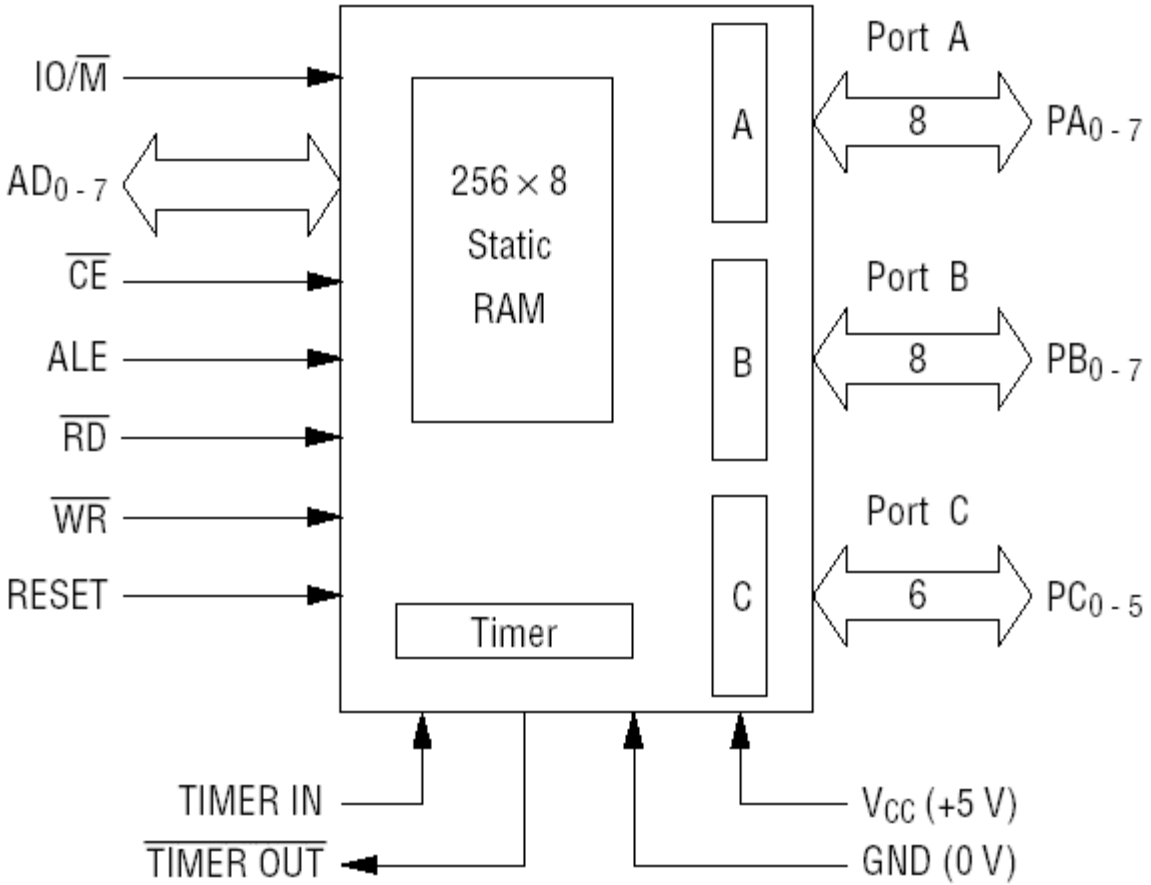
I/O Mapped Devices

- The use of explicit IN or OUT assembly instructions are used with the Intel microprocessors to directly control an I/O mapped device
- Special signals are also asserted with the IN and OUT instructions

IO Signals



Example: Parallel Port Interface



Intel P8155H-2

Memory-mapped I/O

- Define location for device:

DEV1 EQU 0x1000

- Read/write code:

```
MOV r1,#DEV1 ; set up device addr
LDR r0,[r1] ; read DEV1
LDR r0,#8 ; set up value to write
STR r0,[r1] ; write value to device
```

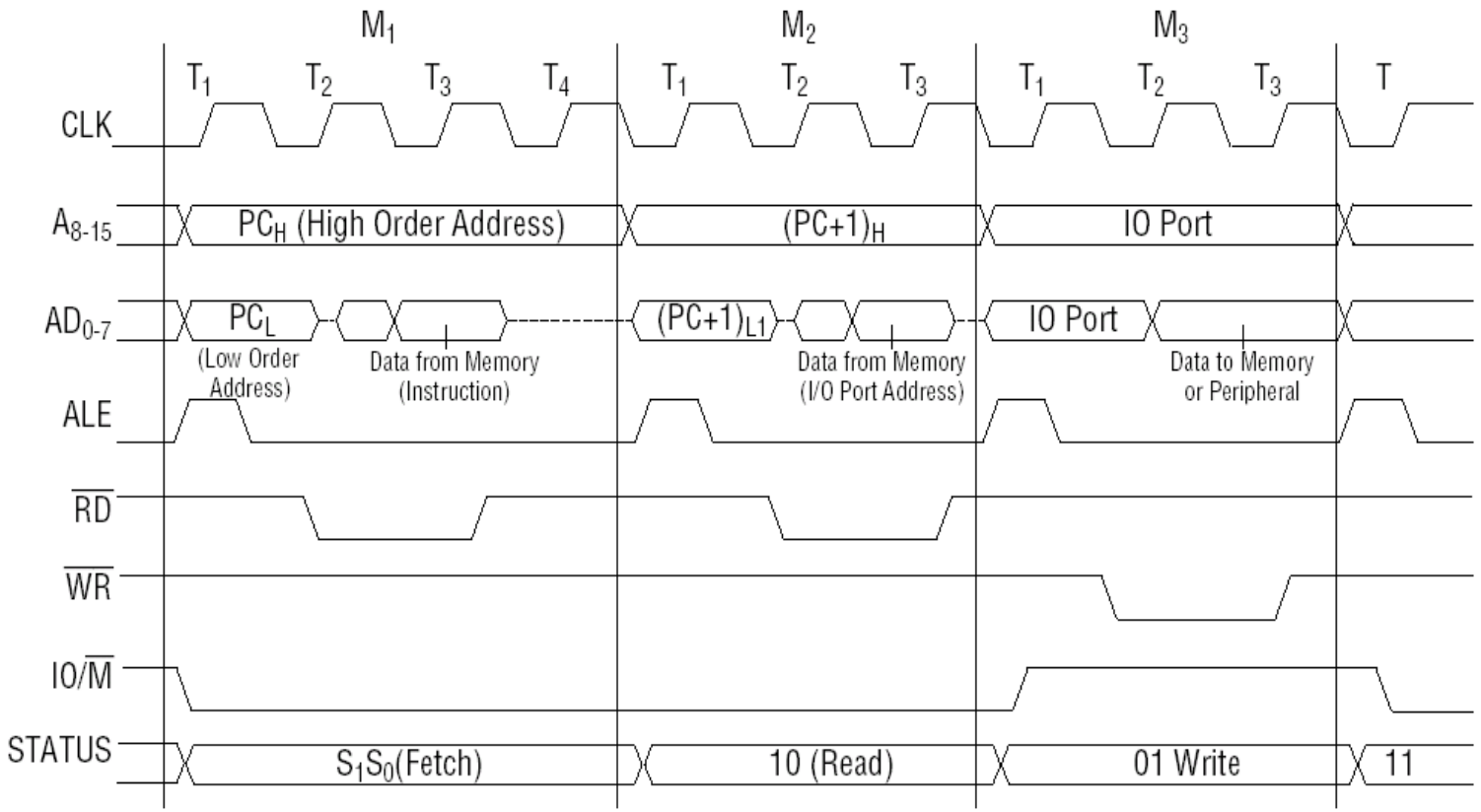
Simplified Device Control Model

- If all I/O devices are memory mapped, the methods used to access and control these devices is made uniform
- I/O devices are assigned valid addresses within the addressable memory space of the system
- All device references are performed using addresses, chip select and R/W signal assertions

Using I/O and Memory Mapped Devices Together

- Both types of devices may be exercised in the same system
- As we have seen, I/O mapped devices assert the IO/Mem signal so that the system can uniquely identify the proper device to control

System Timing Example (8085)



Interrupts and Interrupt Processing

Speeding up the I/O Process

- Since I/O devices are slower than memory, a lot of time may be wasted waiting for data transfers between the devices and the CPU
- We can solve this problem by allocating time slots to devices so that they receive equal attention from the CPU
- We can do this using a polling scheme

Polling

- The polling concept involves the CPU checking to see if each device needs to be serviced
- When a device is ready, the CPU collects its data (or provides data for output) and proceeds to the next I/O device
- As you can see, the CPU becomes dedicated to the I/O devices and is not free to do anything else

The Interrupt

- In order to free the CPU for tasks other than I/O tasks, the CPU could operate more efficiently if it service each I/O device when each actually required service
- The interrupt was created to address this need
- Using the interrupt scheme, each device would request service (asynchronously) from the CPU
- The CPU would them service the device and return to foreground processing activities

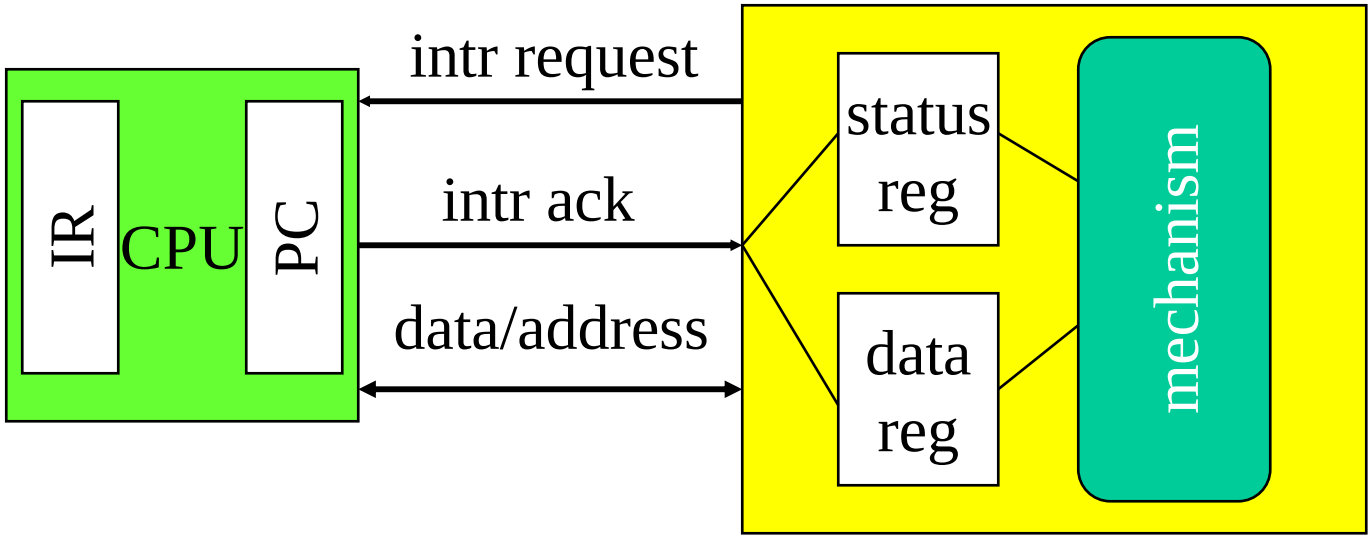
3 Types of Interrupts

- External Interrupts – used by the CPU to interact with external devices
- Internal Interrupts – occur entirely within the CPU (page faults, overflow, etc…)
- Software Interrupts – generated by interrupt instructions in the CPU's ISA

Interrupt I/O

- Polled I/O is very inefficient.
  - CPU can't do other work while servicing I/O devices
- Interrupts allow a device to change the flow of control in the CPU.
  - Causes subroutine call to handle device.

Interrupt interface



Interrupt behavior

- Based on subroutine call mechanism.
- Interrupt forces next instruction to be a subroutine call to a predetermined location
  - Return address is saved to resume executing foreground program.

Interrupt physical interface

- CPU and device are connected by CPU bus.
- CPU and device handshake:
  - device asserts interrupt request;
  - CPU asserts interrupt acknowledge when it can handle the interrupt.

Example: character I/O handlers

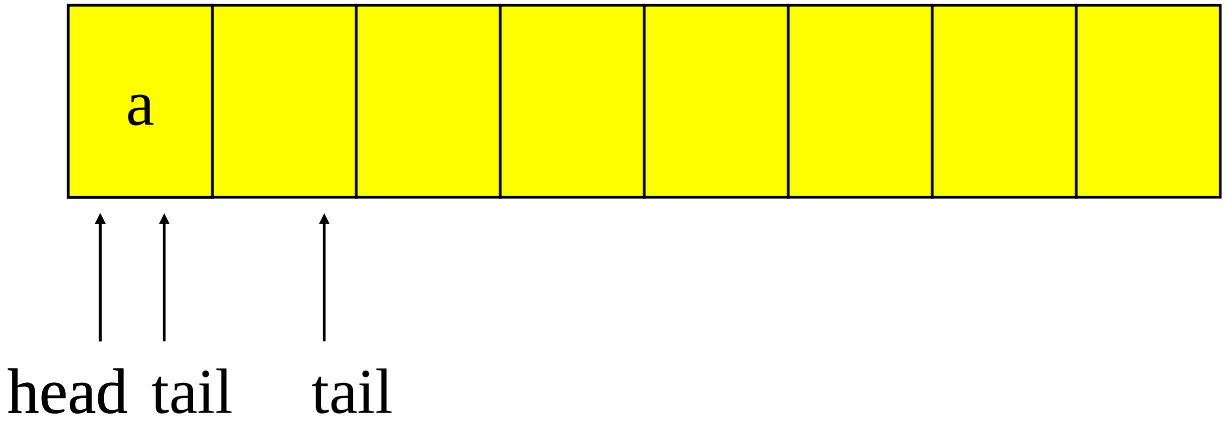
```
void input_handler() {
   achar = peek(IN_DATA);
   gotchar = TRUE;
   poke(IN_STATUS,0);
}
void output_handler() {
}
```

Example: interrupt-driven main program

```
main() {
    while (TRUE) {
        if (gotchar) {
                poke(OUT_DATA,achar);
                poke(OUT_STATUS,1);
                gotchar = FALSE;
                }
        }
    }
```

Example: interrupt I/O with buffers

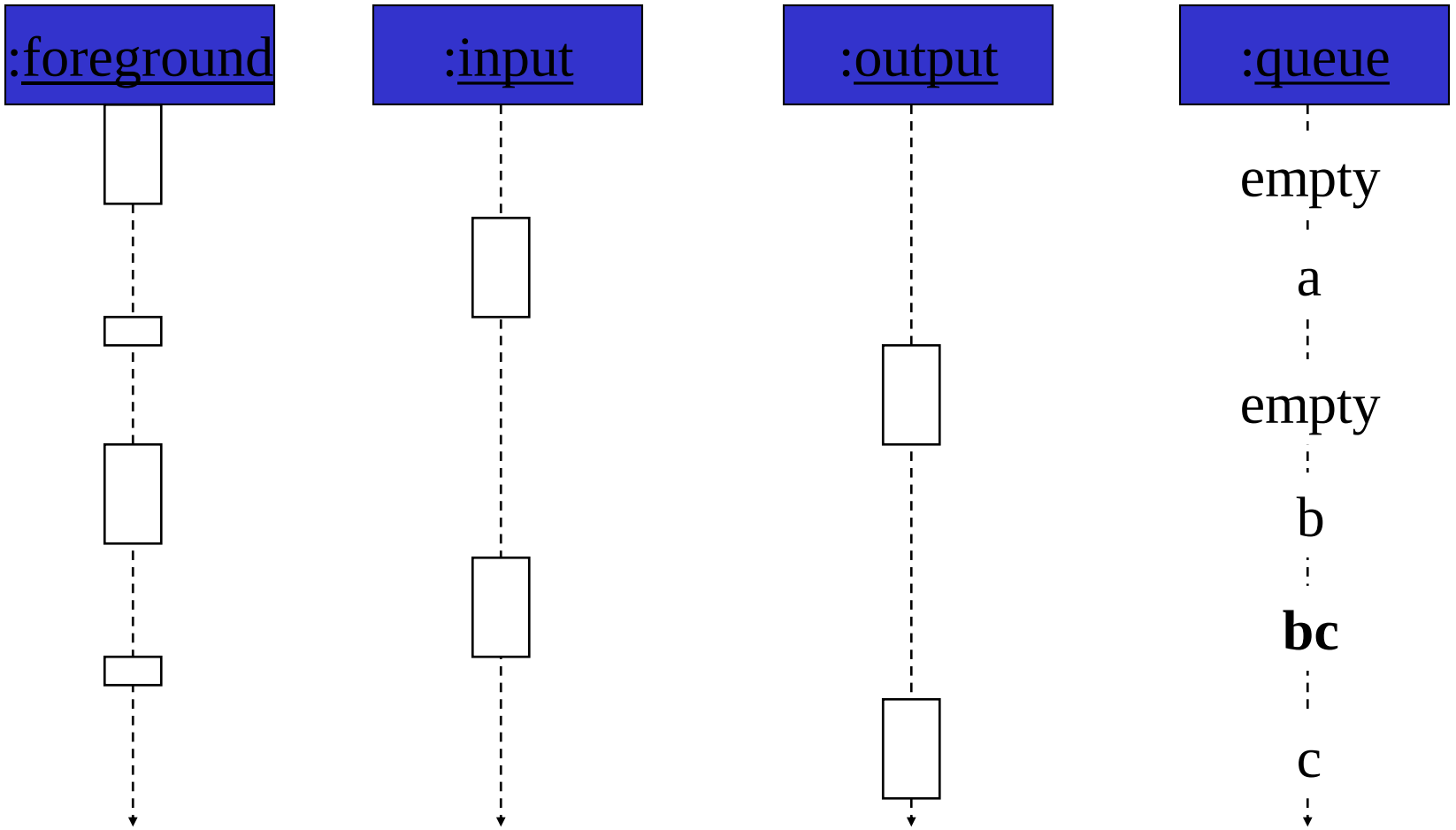
- Queue for characters:



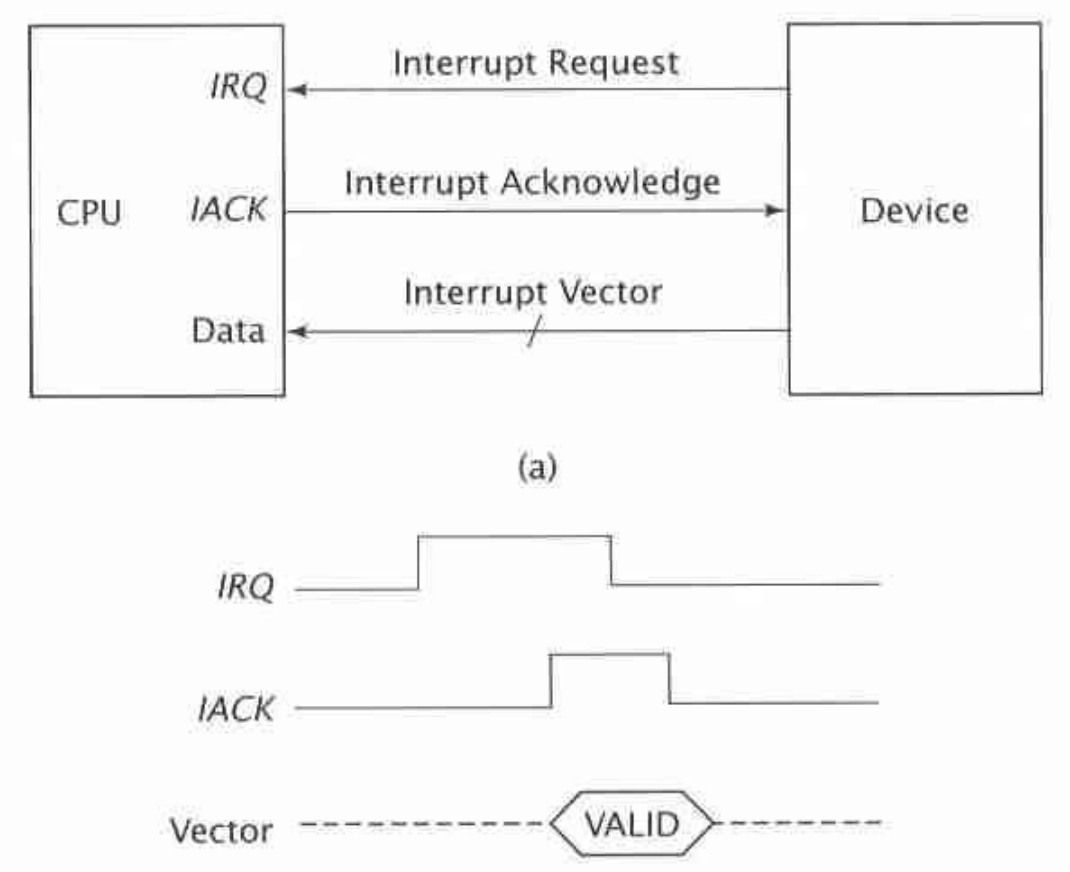
Buffer-based input handler

```
void input_handler() {
    char achar;
    if (full_buffer()) error = 1;
    else { achar = peek(IN_DATA); add_char(achar); }
    poke(IN_STATUS,0);
    if (nchars == 1)
        { poke(OUT_DATA,remove_char();
    poke(OUT_STATUS,1); }
    }
```

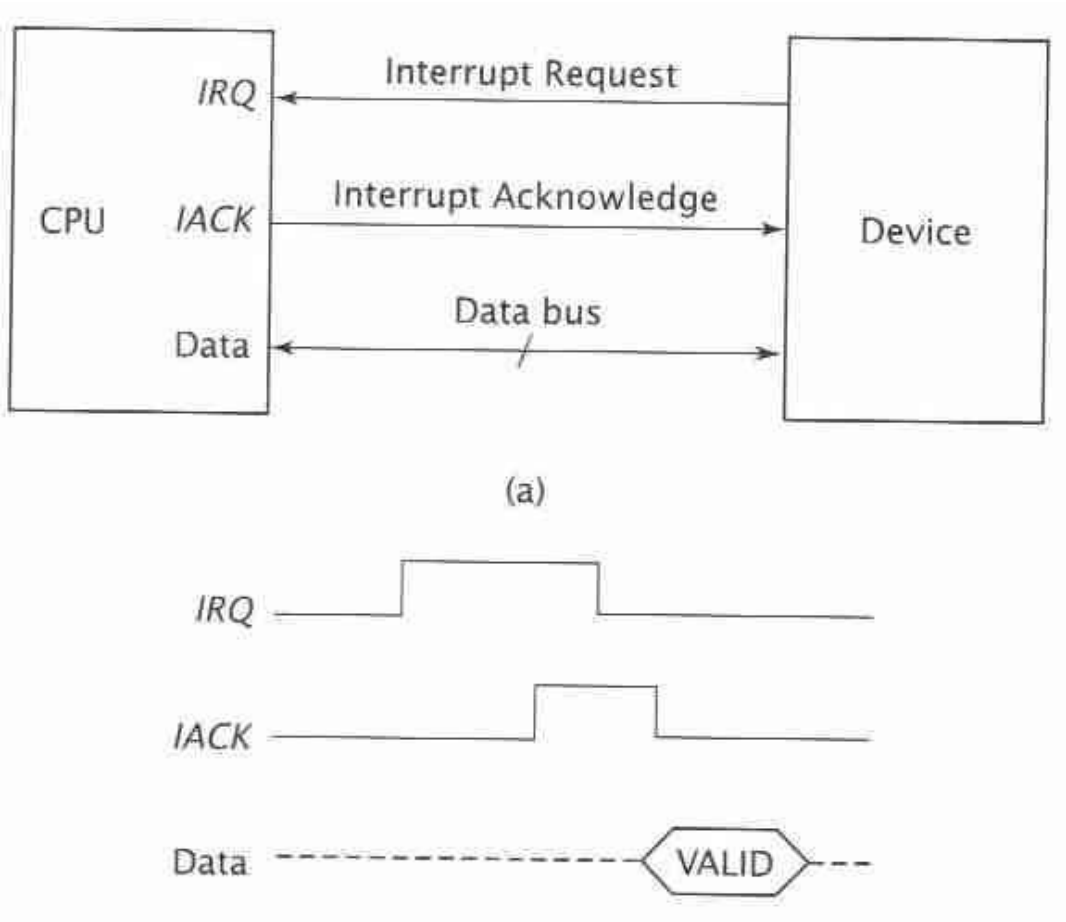
I/O sequence diagram



Vectored Interrupt Example



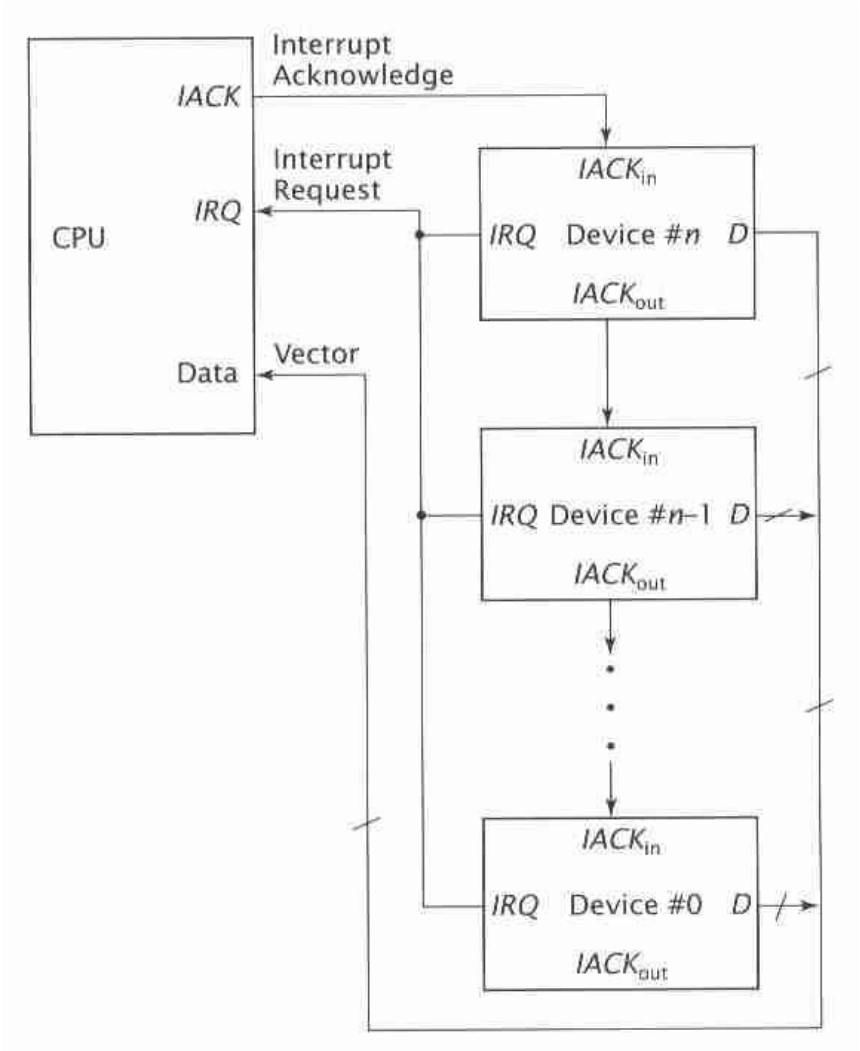
Non-vectored Interrupt Example



Connecting Multiple I/O Devices

- What happens if we need to connect more than a single I/O device to the CPU using the interrupt interface?
- We can use a technique called daisy chaining
- The following diagram illustrates the concept of daisy-chaining

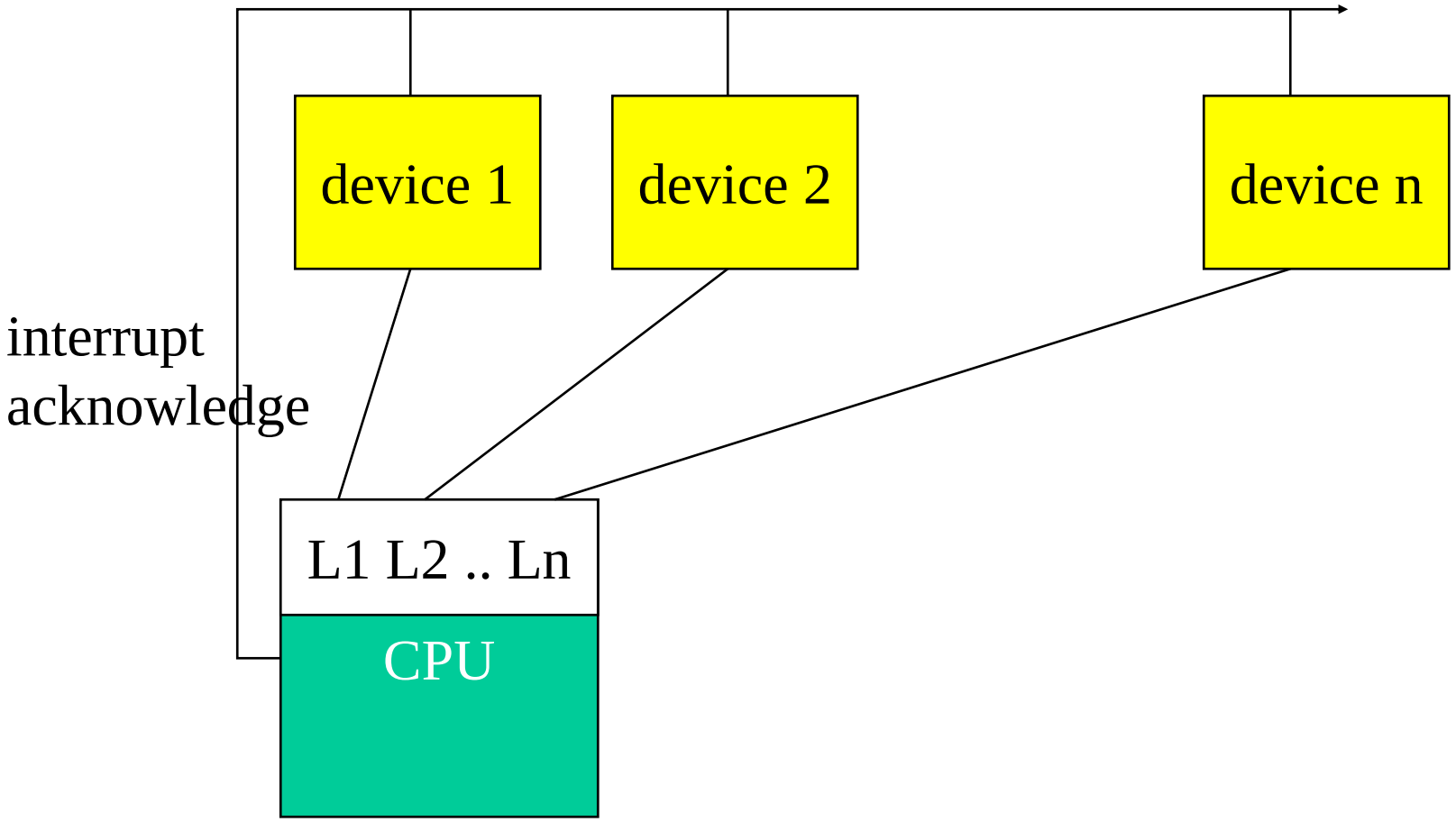
Daisy-Chaining I/O Devices



Priorities and vectors

- Two mechanisms allow us to make interrupts more specific:
  - Priorities determine what interrupt gets CPU first.
  - Vectors determine what code is called for each type of interrupt.
- Mechanisms are orthogonal: most CPUs provide both.

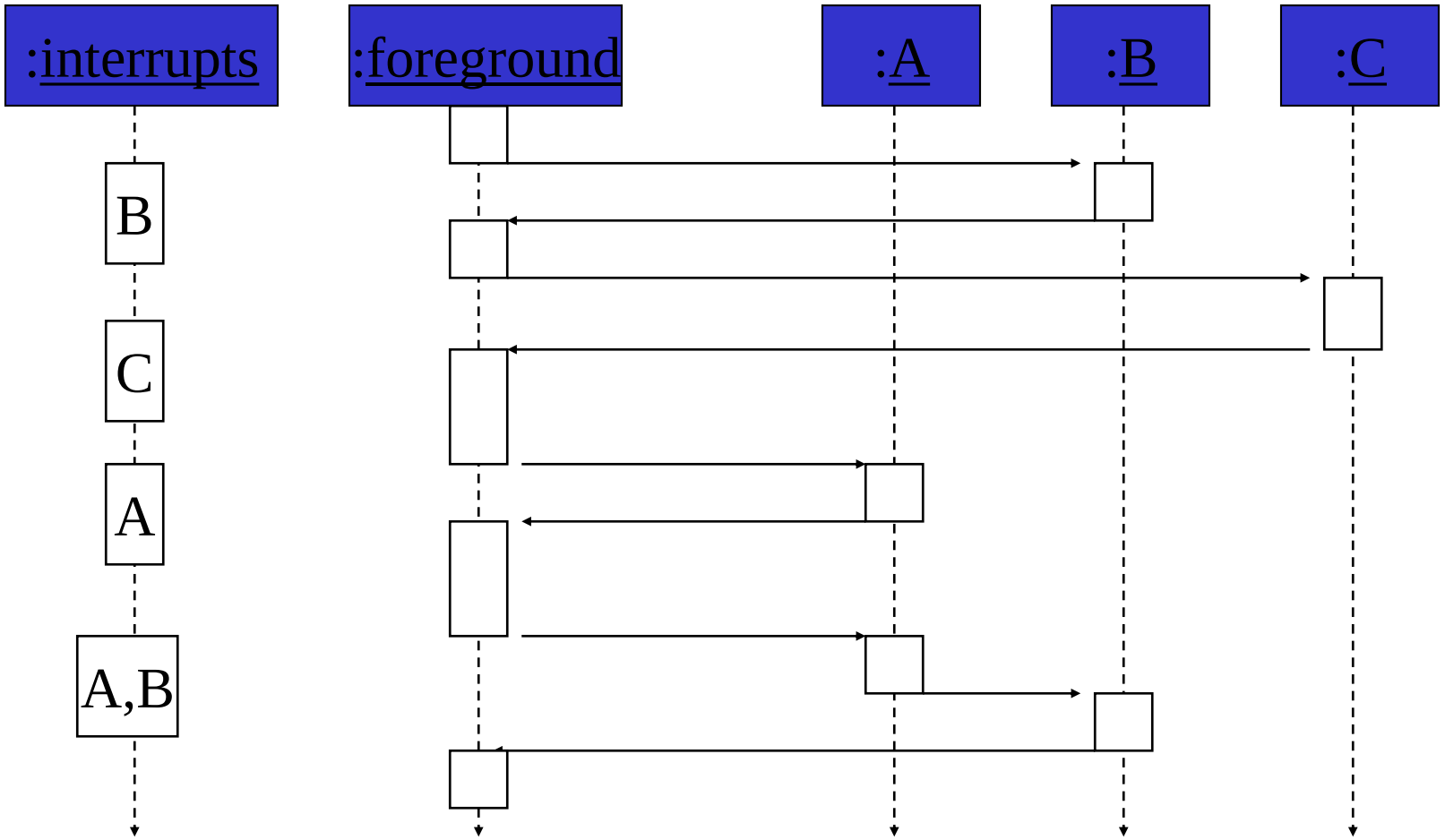
Prioritized interrupts



Interrupt prioritization

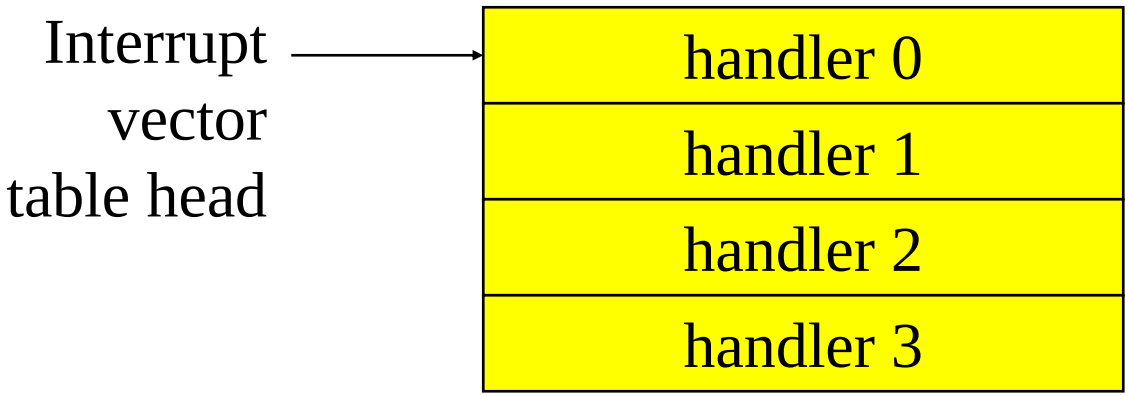
- Masking: interrupt with priority lower than current priority is not recognized until pending interrupt is complete.
- Non-maskable interrupt (NMI): highest-priority, never masked.
  - Often used for power-down.

Example: Prioritized I/O

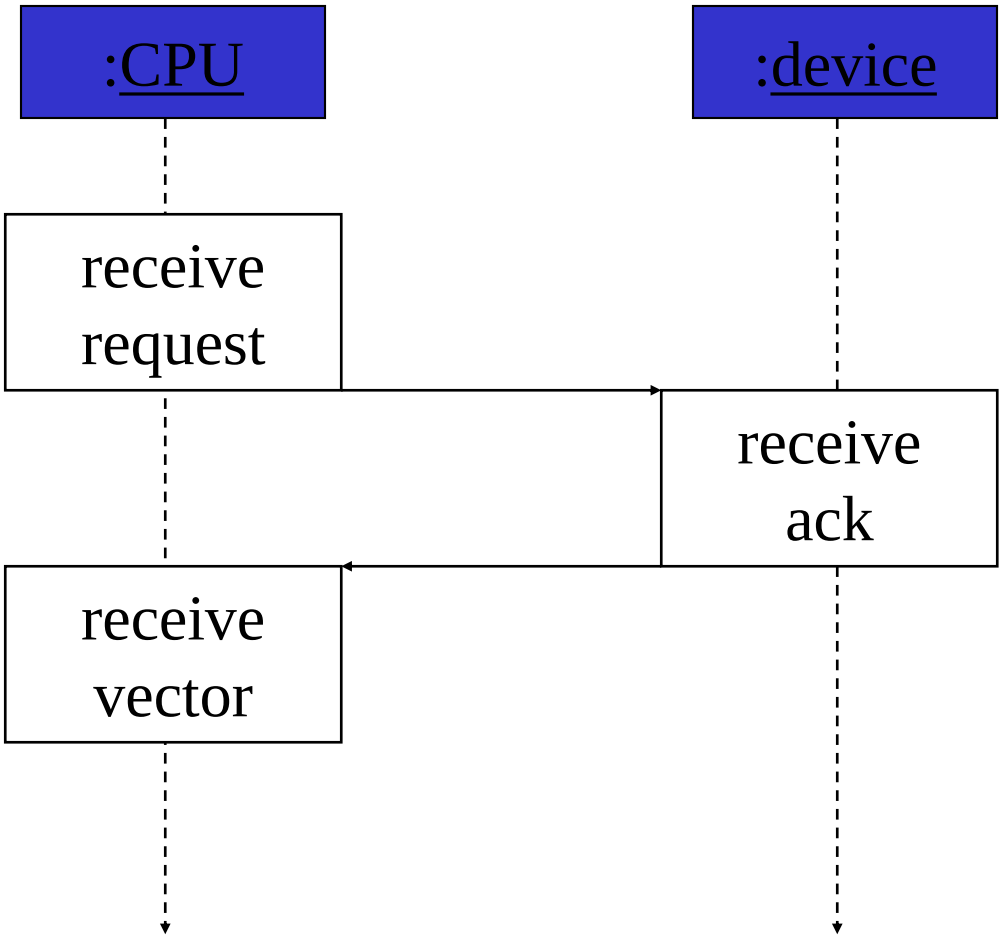


Interrupt vectors

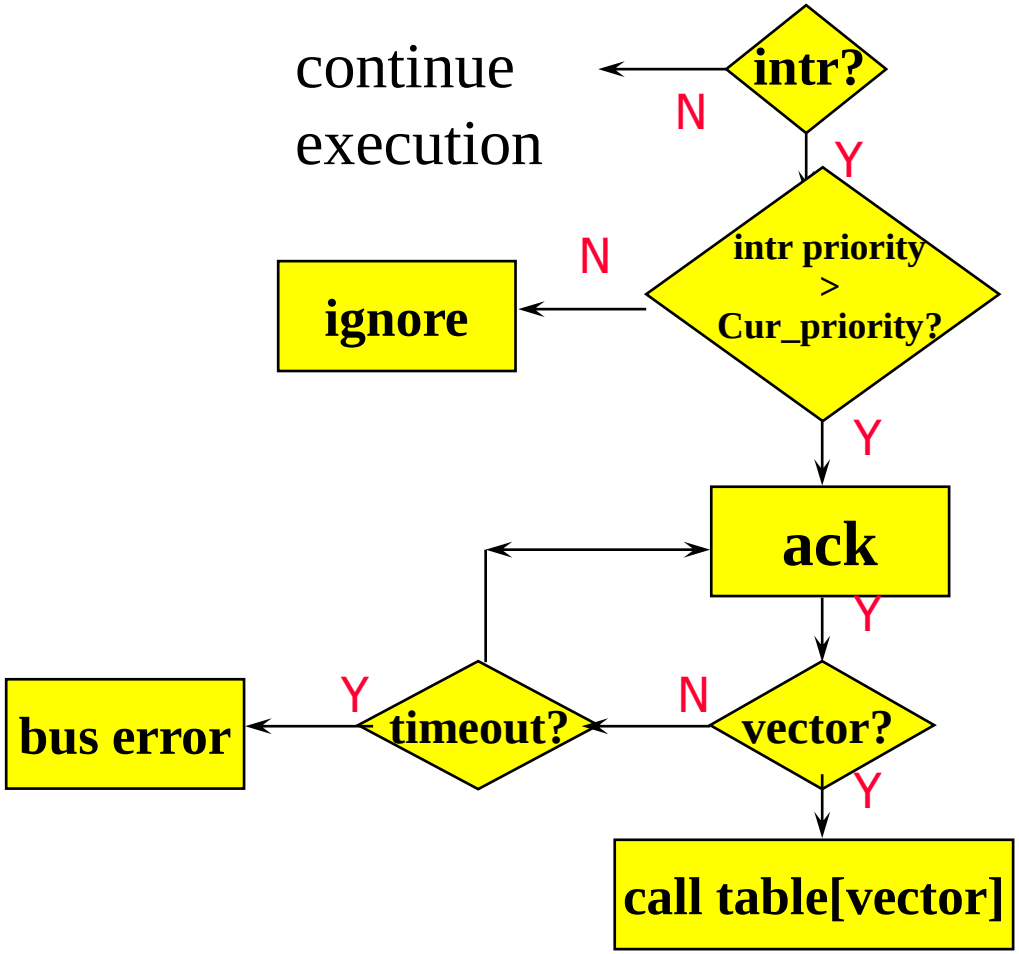
- Allow different devices to be handled by different code.
- Interrupt vector table:



Interrupt vector acquisition



Generic interrupt mechanism



Assume priority selection is handled before this point.

Interrupt sequence

- Device asserts interrupt request signal
- CPU acknowledges request.
- Device sends vector.
- CPU calls handler.
- Software processes request.
- CPU restores state to foreground program.

Sources of interrupt overhead

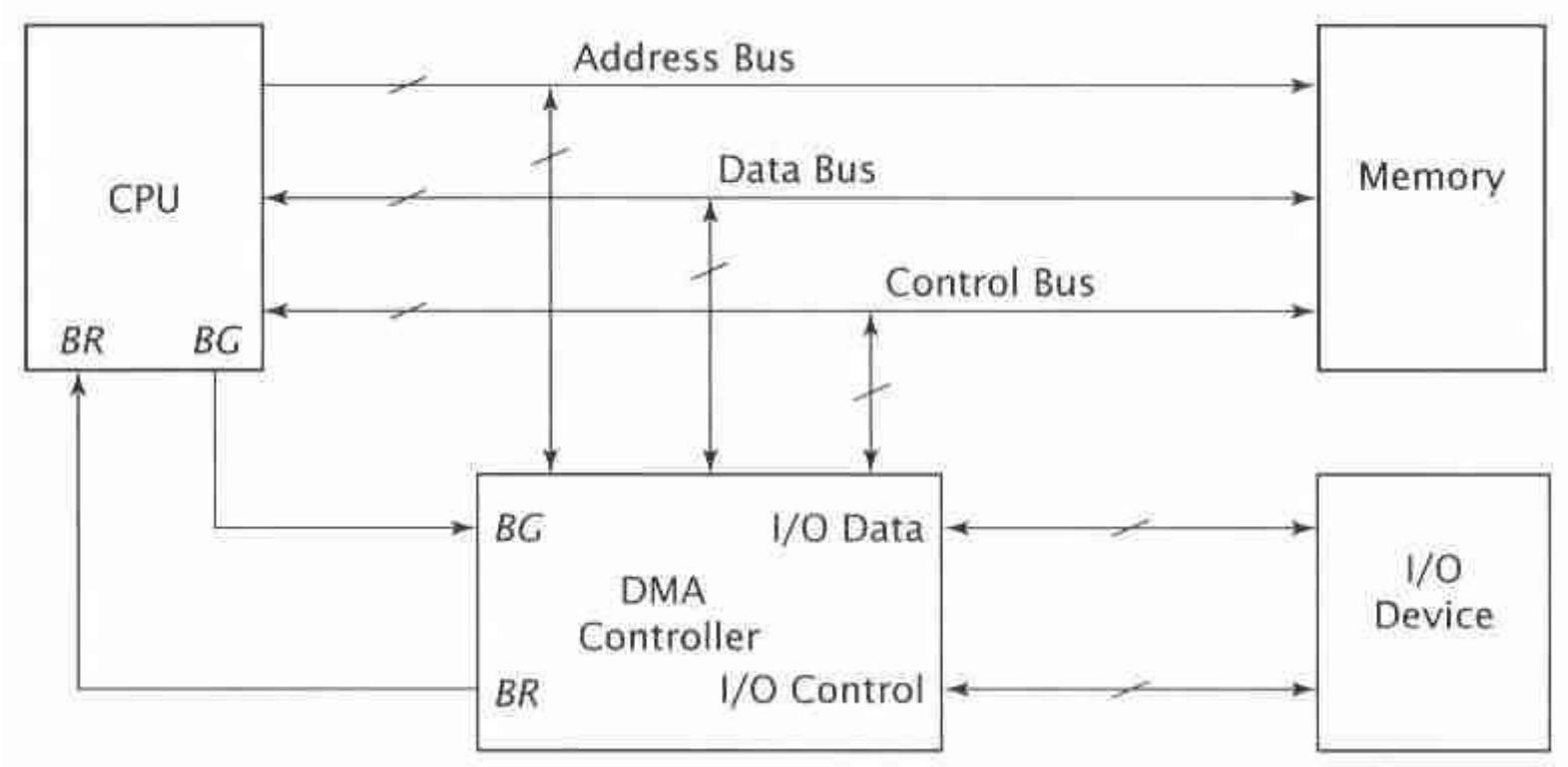
- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.

Exception

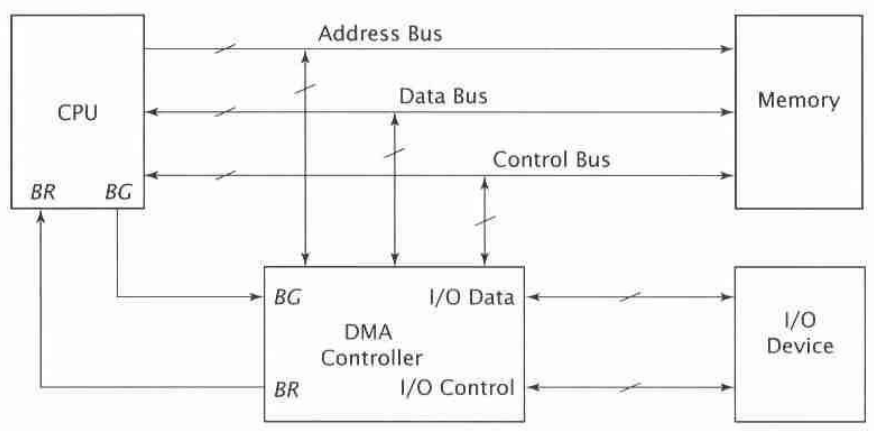
- Exception: internally detected error.
- Exceptions are synchronous with instructions but unpredictable.
- Examples of exceptions:
  - Divide by Zero
  - Overflow Condition
  - Underflow condition
  - Exceeded valid address range

Direct Memory Access

Direct Memory Access Example



DMA Process Explained



- DMA is programmed with memory transfer instructions
- DMA asserts Bus Request (BR) and CPU asserts the Bus Grant (BG) and relinquishes the BUS
- DMA transfers data and de-asserts BR
- CPU de-asserts BG and returns to normal function

DMA Transfer Modes

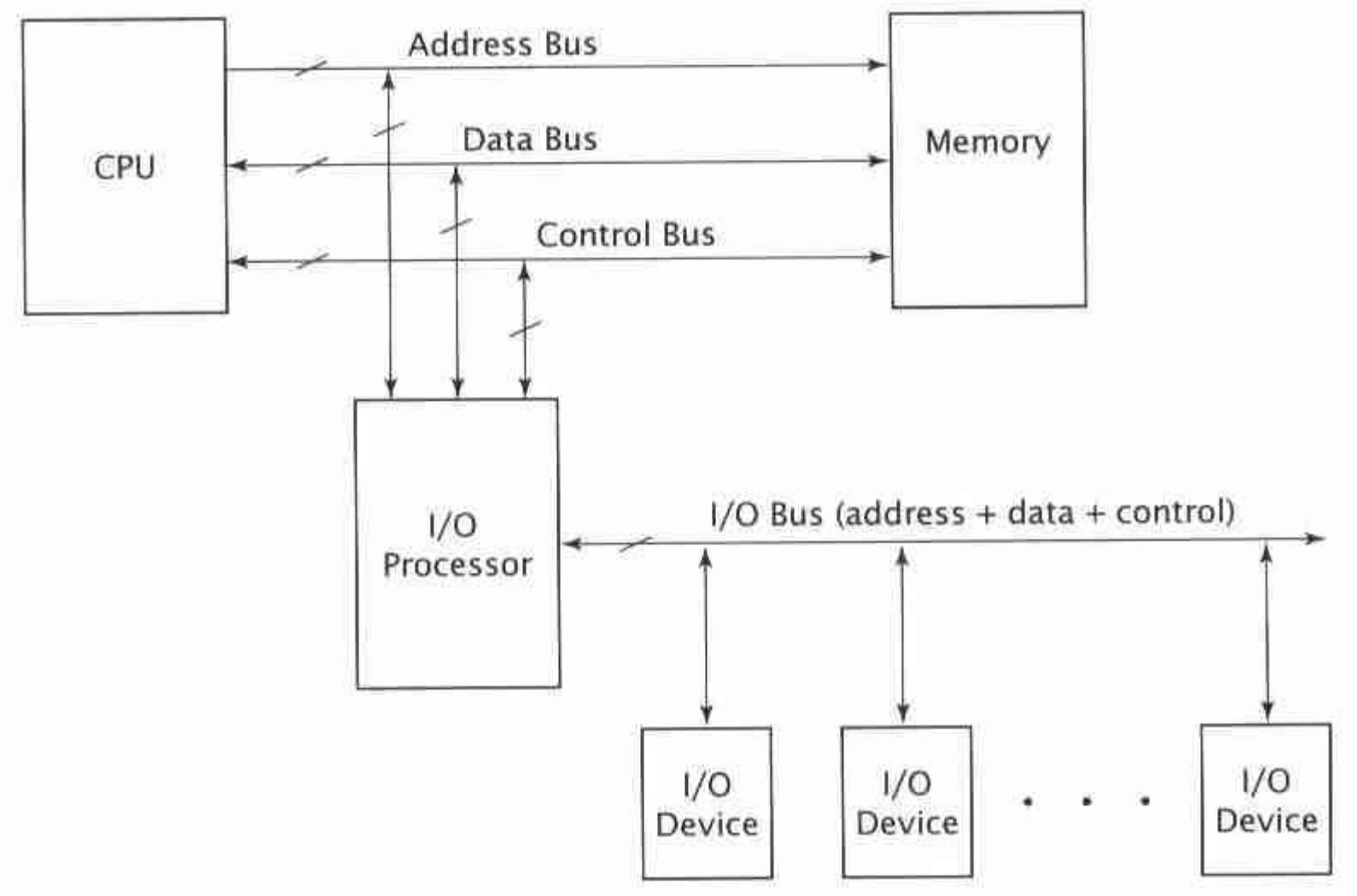
- Burst Mode – An entire block of data is transferred in one contiguous sequence
- Cycle Stealing Mode – DMA controller requests bus every other cycle and transfers a byte at a time until the block has been transferred
- Transparent Mode – DMA transfers occur when CPU is performing operations that do not require bus access

Input/Output Processors

Purpose of I/O Processors

- I/O Processors (IOPs) are used to offload the task of managing I/O devices from the CPU
- The IOP serves as an intermediary between the I/O devices and the CPU
- In fact, when an IOP is used, the only interaction between the CPU and the devices is through the IOP

I/O Processor Example



What's in a Name?

- I/O Processors are sometimes referred to as:
  - I/O Controllers
  - Channel Controllers
  - Peripheral Processing Units (PPUs)
- They perform the functions of DMA controllers and a lot more

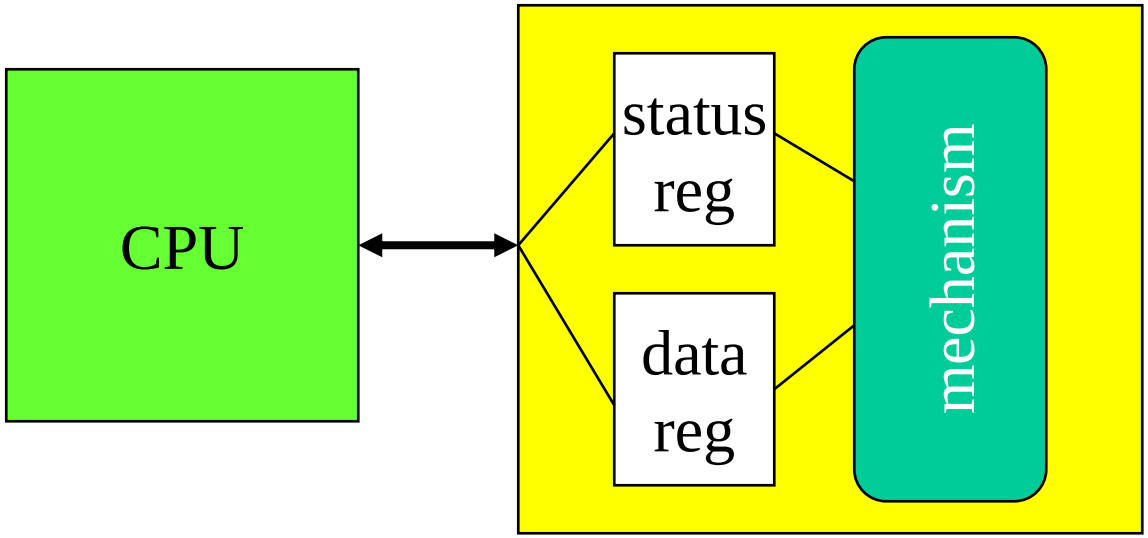
IOP Commands

- The IOP operates on 3 basic classes of commands
  - Block Transfer Commands
  - Arithmetic, Logic, and Branch Commands
  - Control Commands

Serial Communications

I/O devices

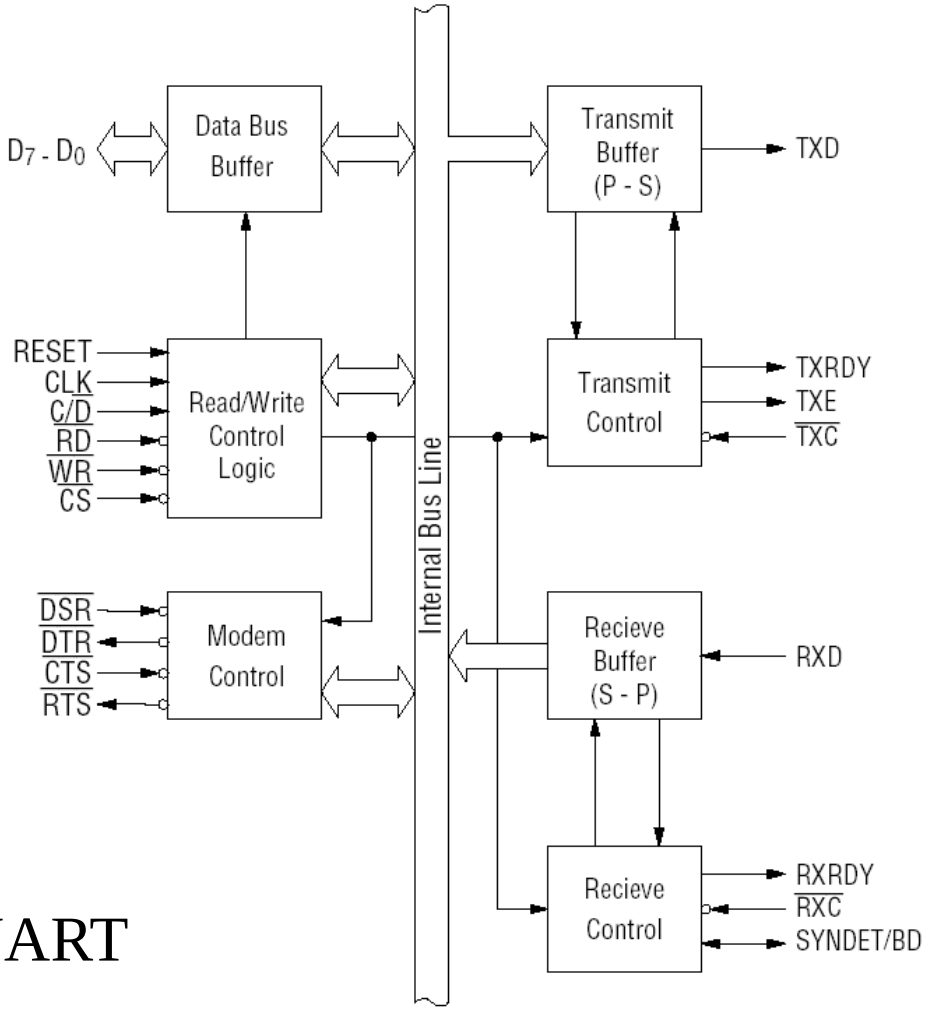
- Usually includes some non-digital component.
- Typical digital interface to CPU:



Application: UART

- Universal asynchronous receiver transmitter (UART) : provides serial communication.
- UART functions are integrated into standard PC interface chip.
- Allows many communication parameters to be programmed.

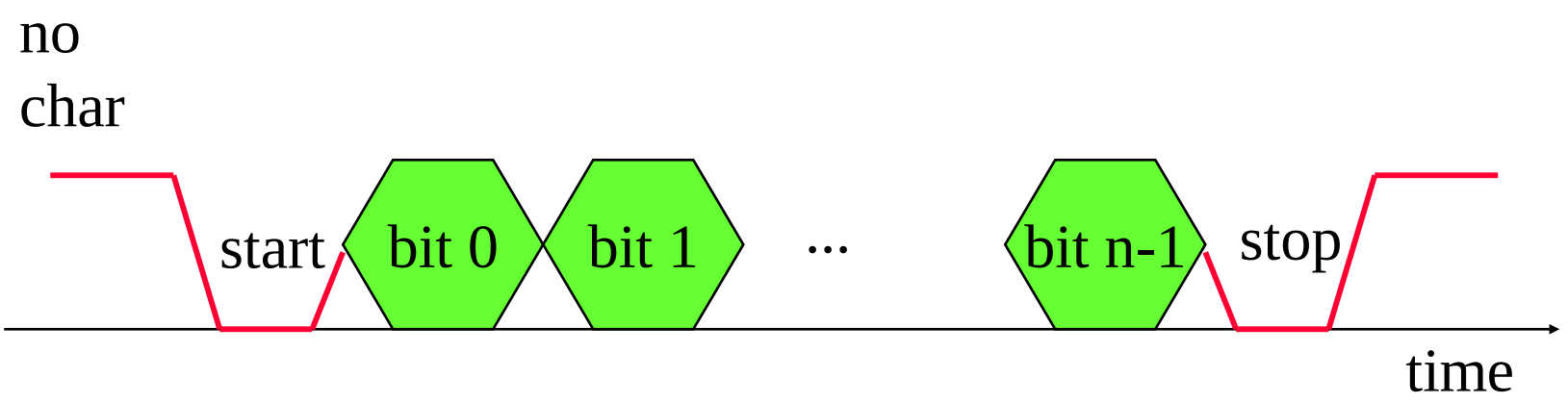
UART Functional View



Intel 8251 UART

Serial communication

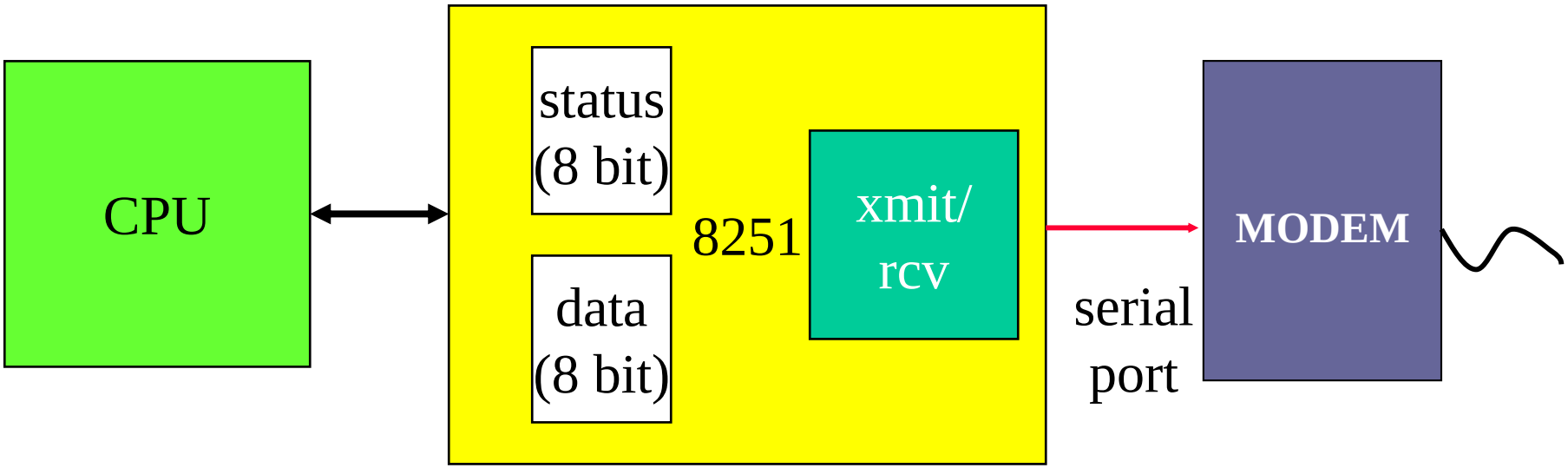
- Characters are transmitted separately:



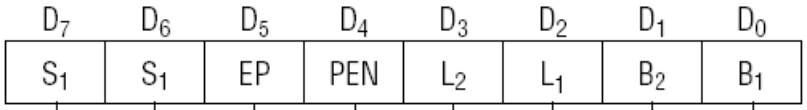
Serial communication parameters

- Baud (bit) rate.
- Number of bits per character.
- Parity/no parity.
- Even/odd parity.
- Length of stop bit (1, 1.5, 2 bits).

UART / CPU interface



Control Word Configuration



Baud Rate Factor

0	1	0	1
0	0	1	1
Refer to Fig. 3 SYNC	1 ×	16 ×	64 ×

Charactor Length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

Parity Check

0	1	0	1
0	0	1	1
Disable	Odd Parity	Disable	Even Parity

Stop bit Length

0	1	0	1
0	0	1	1
Inhabit	1 bit	1.5 bits	2 bits

Intel 8251UART Signals

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

Summary

- I/O devices are usually not clock synchronized with the CPU
- Data exchanges are coordinated either with no acknowledgement or full acknowledgement
- Acknowledgement is assured through the handshaking process

Summary Continued

- Polled I/O is not a desirable strategy for general purpose computers
- Interrupts offer a better solution to free the CPU to work on other system tasks
- Interrupts may be vectored or non-vectored
- Interrupts may be prioritized

Summary Continued

- Direct Memory Access improves the efficiency of the computer by transferring needed data without main CPU intervention
- The DMA function is incorporated into the I/O Processor (IOP)
- The IOP is used to manage all system I/O so that the main CPU can concentrate on more important tasks
- Serial communication is a technique used to connect external devices in an inexpensive fashion
- Several protocols are used to facilitate communications

Acknowledgements

- Some of the graphical material was excerpted from “Computers as Components” by Wayne Wolf
- Morgan Kaufmann Publishers