5.b Principles of I/O Hardware

CPU-I/O communication

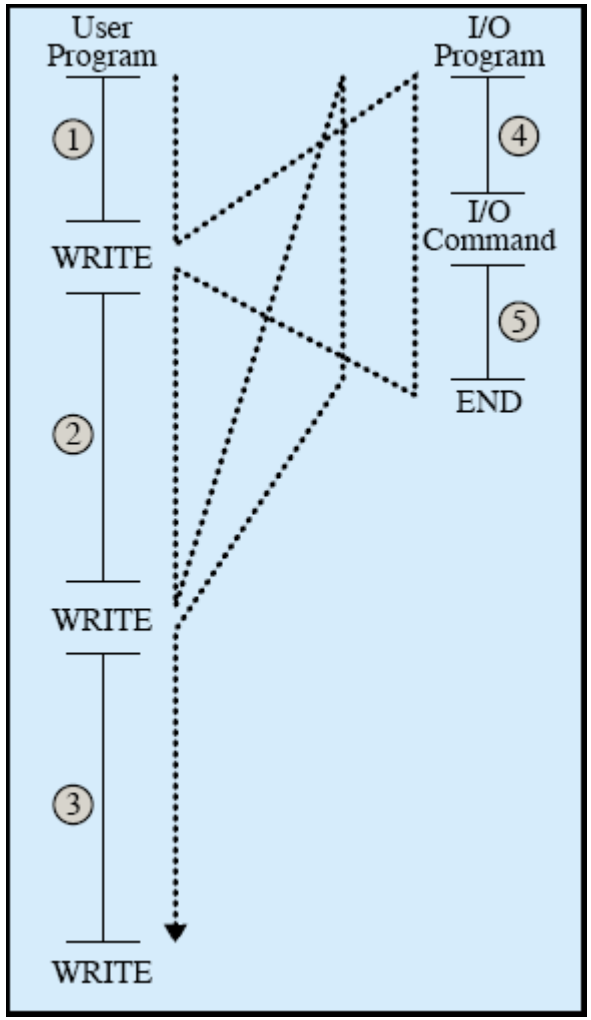
- Three communication protocols between CPU and I/O
  1. Programmed I/O (or “polling” or “busy waiting”)
     - the CPU must repeatedly poll the device to check if the I/O request completed
  2. Interrupt-driven I/O
     - the CPU can switch to other tasks and is (frequently) interrupted by the I/O device
  3. Direct Memory Access (DMA)
     - the CPU is involved only at the start and the end of the whole transfer; it delegates control to the I/O controller that accesses memory directly without bothering the CPU

5.b Principles of I/O Hardware

CPU-I/O communication

1. Programmed I/O

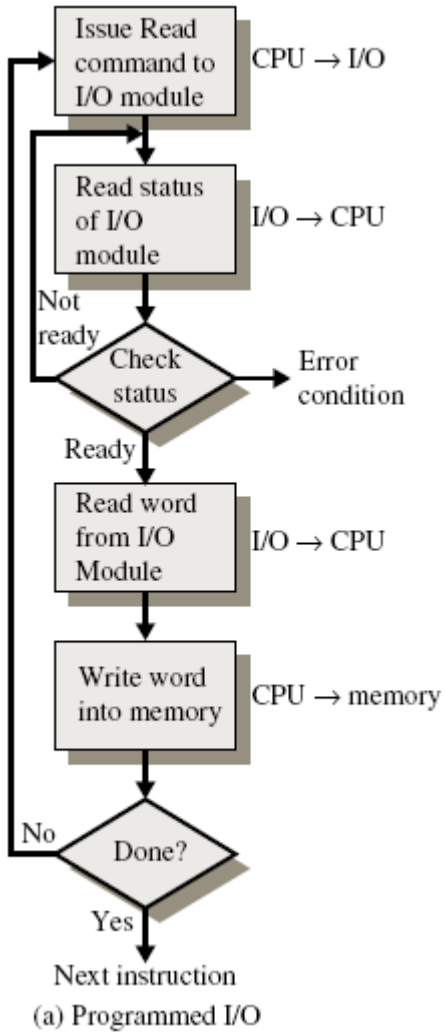
- ✓ the CPU issues an I/O command (on behalf of a process) to an I/O module
- ✓ the CPU (the process) then busy waits for completion before proceeding
- ✓ also called “busy waiting” or “polling”



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication



(a) Programmed I/O

Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

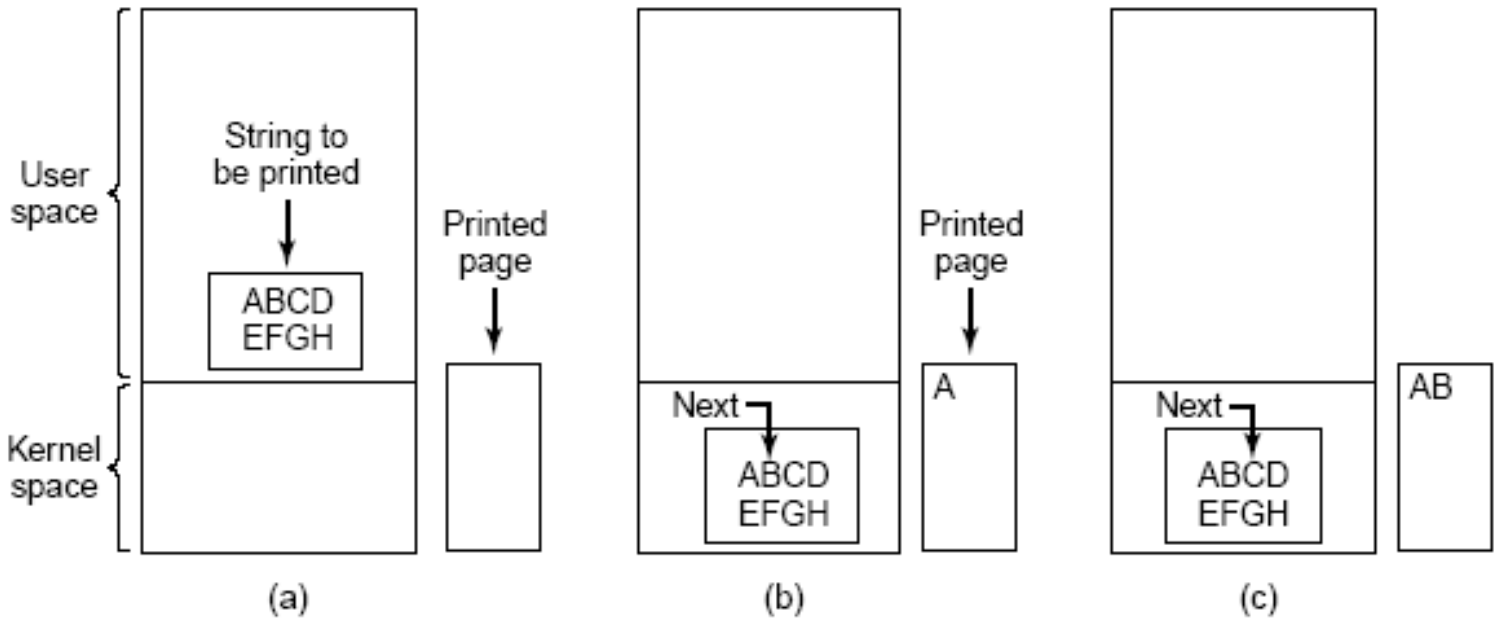
1. Programmed I/O

- ✓ basic handshake protocol between CPU and I/O module
  a. host repeatedly polls *occupied* bit in status register of I/O module until cleared (this is the busy waiting part)
  b. host sets *write* bit in *command* register of I/O module and writes byte into *data-out* register of I/O module
  c. host sets *command-ready* bit of I/O module
  d. I/O module notices *command-ready* bit and sets *occupied* bit
  e. I/O module reads *command* and *data-out* registers and orders I/O device to perform I/O
  f. when suceeded, I/O module clears *command-ready* bit, *error* bit and *occupied* bit

5.b Principles of I/O Hardware

CPU-I/O communication

- Example: writing a string to the printer



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Steps in printing a string

5.b Principles of I/O Hardware

CPU-I/O communication

- Example of “driver” logic: writing a string to the printer

1. . . . using programmed I/O:

```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
for (i = 0; i < count; i++) {              /* loop on every character */
      while (*printer_status_reg != READY) ;/* loop until ready */
      *printer_data_register = p[i];       /* output one character */
}
return_to_user( );
```

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Programmed I/O code

5.b Principles of I/O Hardware

CPU-I/O communication

1. Programmed I/O problems

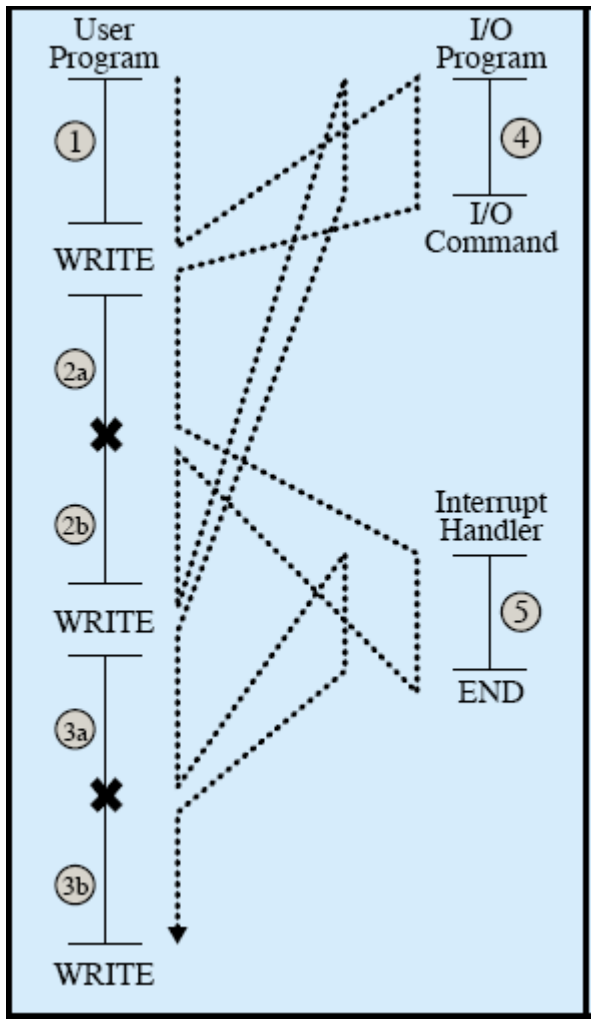
- ✓ the I/O device (module) is passive and needy: it does not alert the CPU that it is ready and does not transfer data to/from memory by itself
- ✓ the CPU needs to continually check the I/O status and data registers
  - to minimize the CPU waiting time
  - but also to avoid overflow in the small buffer of the controller: needs to be regularly cleared
- ✓ naturally this is a waste of CPU time if the I/O transfer is slower than the CPU. . . which it always is!
- ✓ no longer an option today

5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O

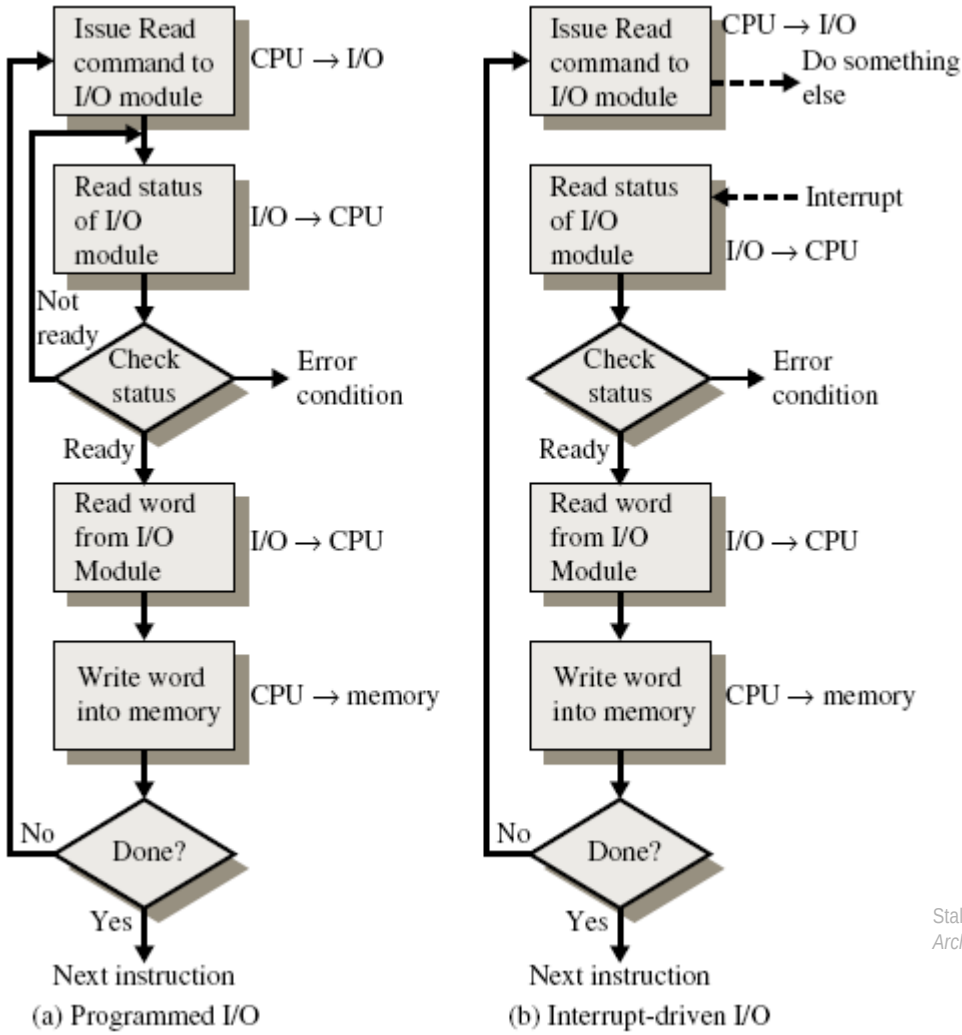
- ✓ the CPU issues an I/O command (on behalf of a process) to an I/O module
- ✓ . . . but does not wait for completion; instead, it continues executing subsequent instructions
- ✓ then, later, it is interrupted by the I/O module when work is complete
- ✓ note: the subsequent instructions may be in the same process or not, depending on whether I/O was requested asynchronously or not: *process wait ≠ CPU wait!*



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication



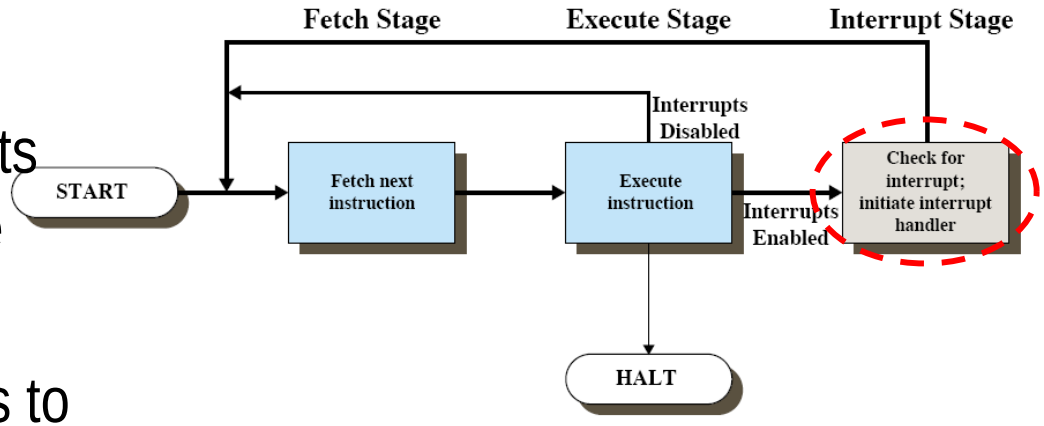
Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

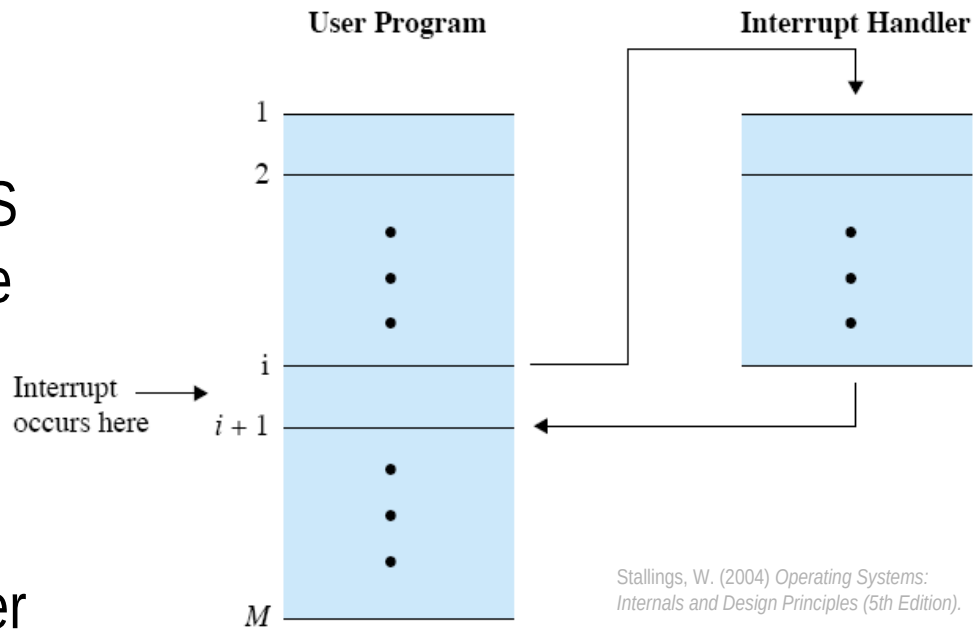
5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O

- ✓ CPU senses interrupts in a third stage of the fetch/execute cycle
- ✓ control (PC) transfers to an interrupt handler in kernel space,
- ✓ which branches to O/S routines specific to the type of interrupt;
- ✓ the CPU is eventually returned to this user program . . . or another





Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

- Example of "driver" and "interrupt handler" logic: writing a string to the printer

2. . . . using interrupts:

driver logic

```
                copy_from_user(buffer, p, count);
                enable_interrupts( );
only            while (*printer_status_reg != READY) ;
1st char        *printer_data_register = p[0];
                scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

interrupt handler logic

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Interrupt-driven I/O code: (a) system call and (b) interrupt service procedure

5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O

- ✓ relies on an efficient hardware mechanism that saves a small amount of CPU state, then calls a privileged kernel routine
- ✓ note that this hardware mechanism is put to good use by the O/S for other events:
  - in virtual memory paging, a page fault is an exception that raises an interrupt
  - system calls execute a special instruction (TRAP), which is a software interrupt

5.b Principles of I/O Hardware

CPU-I/O communication

2. Interrupt-driven I/O problems

- ✓ the I/O device (module) is more active but still very needy
- ✓ wasteful to use an expensive general-purpose CPU to feed a controller 1 byte at a time

5.b Principles of I/O Hardware

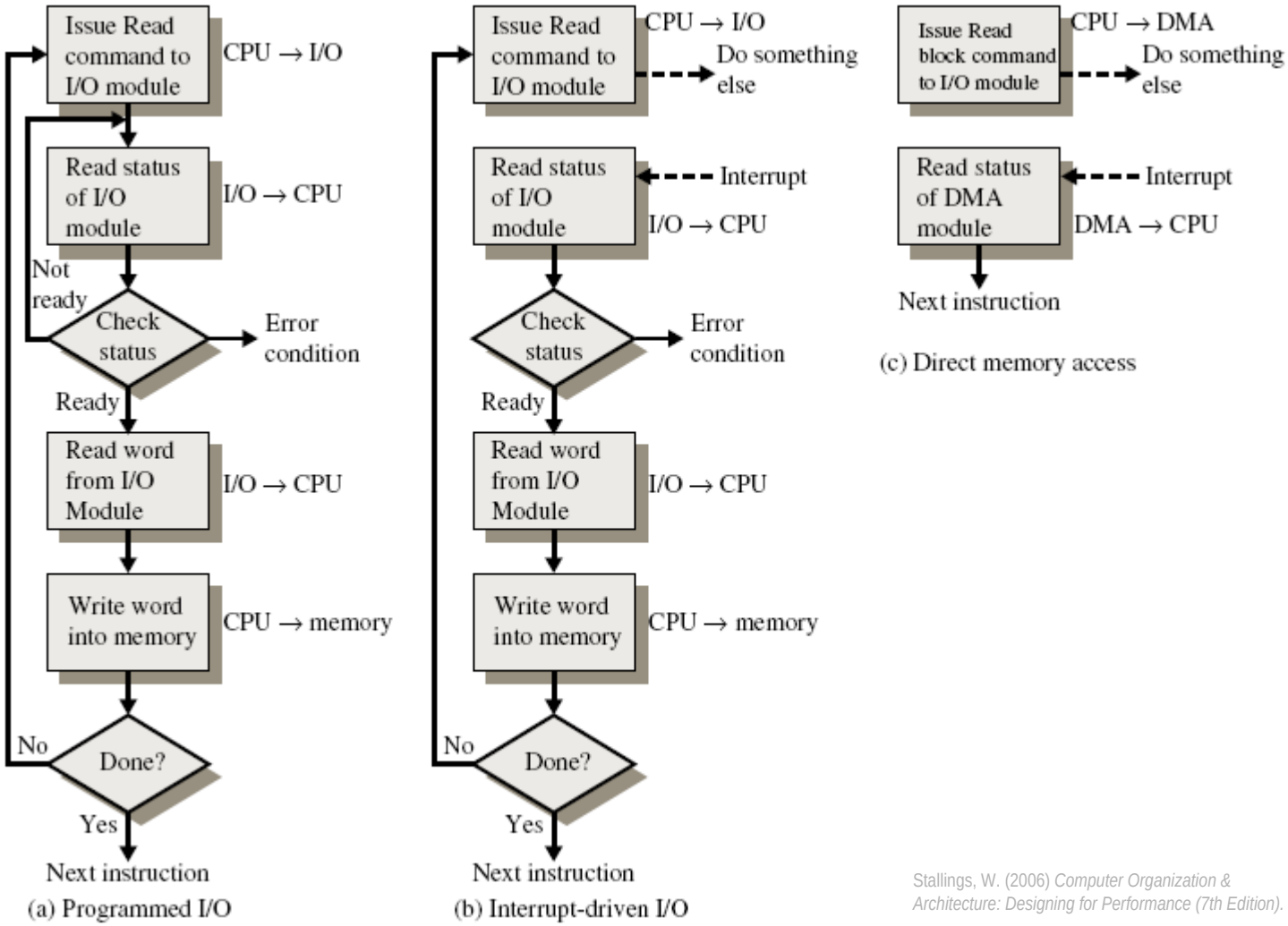
CPU-I/O communication

3. Direct Memory Access (DMA)

- ✓ avoids programmed/interrupted I/O for large data movement
- ✓ requires a special-purpose processor called DMA controllerbypasses CPU to transfer data directly between I/O device and memory
- ✓ the handshaking is performed between the DMA controller and the I/O module: in essence, the DMA controller is going to do the programmed I/O instead of the CPU
- ✓ only when the entire transfer is finished does the DMA controller interrupt the CPU

5.b Principles of I/O Hardware

CPU-I/O communication

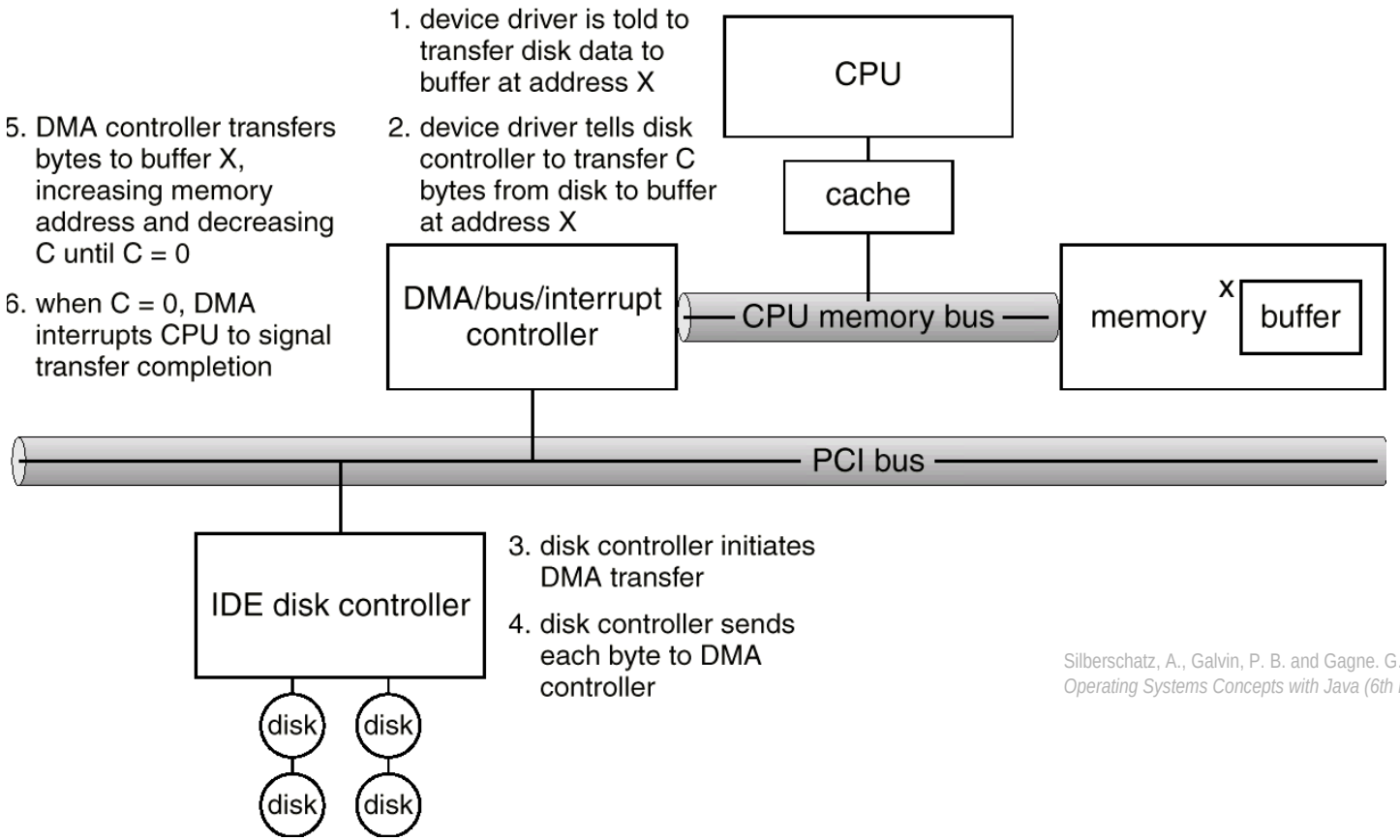


Stallings, W. (2006) *Computer Organization & Architecture: Designing for Performance (7th Edition).*

5.b Principles of I/O Hardware

CPU-I/O communication

3. Direct Memory Access (DMA)



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Steps in a DMA transfer

5.b Principles of I/O Hardware

CPU-I/O communication

- Example of "driver" logic: writing a string to the printer

3. . . . using DMA:

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

DMA-supported I/O code: (a) system call and (b) interrupt service procedure

5.b Principles of I/O Hardware

CPU-I/O communication

- Summary

	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)

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Principles of Operating Systems

CS 446/646

5. Input/Output

a. Overview of the O/S Role in I/O

b. Principles of I/O Hardware

- ✓ The diversity of I/O devices
- ✓ I/O bus architecture
- ✓ I/O devices & modules
- ✓ CPU-I/O communication

c. I/O Software Layers

d. Disk Management