I/O Controllers and Interfacing

November 27

CSC201 Section 002

Fall, 2000

Definitions

- Interface = a set of registers that allow CPU and device to exchange information
- Controller manages a device and handles bus access for the device
- Device driver is part of the operating system which handles details of the device operation
- The interrupt system is the way the CPU requests attention / service

Device Controllers

I/O Controller	I/O Addresses	Interrupt vector
Clock	040 – 043	8
Keyboard	060 - 063	9
Secondary RS232	2F8 - 2FF	11
Hard disk	320 - 32F	13
Printer	378 - 37F	15
Monochrome display	380 - 3BF	–
Color display	3D0 - 3DF	–
Floppy disk	3F0 - 3F7	14
Primary RS232	3F8 - 3FF	12

Fig. 5-2. Some examples of controllers, their I/O addresses, and their interrupt vectors on the IBM PC.

Interfaces

- Sections of memory holding commands, data, and status information
- Accessible by both the CPU and the device
  - Device and CPU both read/write status
  - CPU writes commands, device reads them
  - Device and CPU read/write data

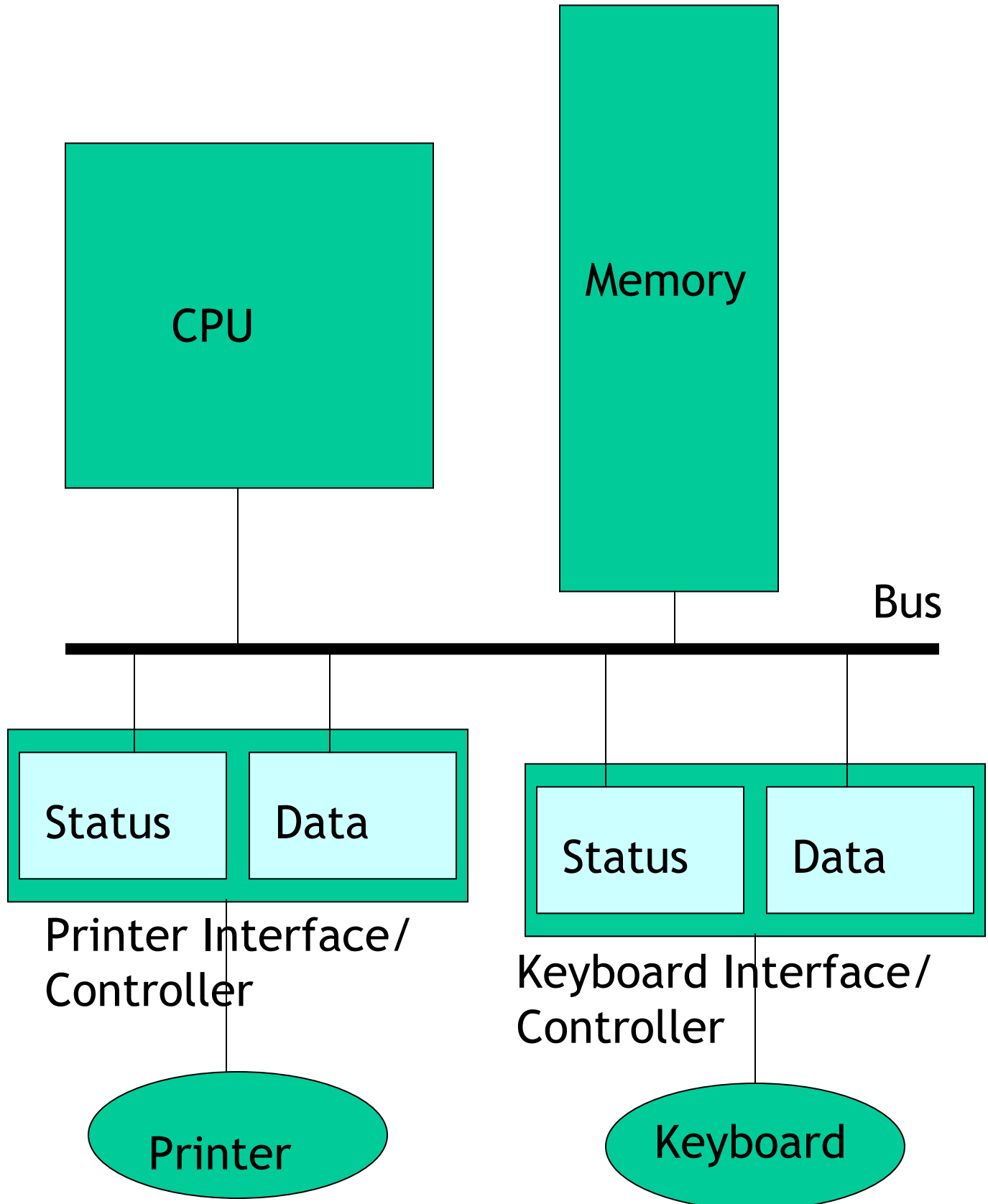
Dedicated I/O Instructions?

- I/O interface accessed differently than main memory
- Dedicated I/O instructions for input and output operations
- Not the method used by most processors

Memory-Mapped I/O

- I/O interface is accessed just like main memory
- I/O operations = reading from or writing to memory!
- Every interface has data, status, and control registers

Interfaces



Example: Reading From Keyboard

- Keyboard_Status MSB:
  - '1' means there's a character waiting to be read by CPU (set by keyboard)
  - '0' means there isn't (cleared by CPU)
- Reading 1 character from keyboard:

```
.data
chars       db    100 dup (?)
...
.code
        mov   EAX, Keyboard_Data
        mov    chars, AL
        and    Keyboard_Status, 7FFFFFFFh
```

Reading 3 Characters

```
mov   EAX, Keyboard_Data
mov    chars, AL
and    Keyboard_Status, 7FFFFFFFh
mov   EAX, Keyboard_Data
mov    chars+1, AL
and    Keyboard_Status, 7FFFFFFFh
mov   EAX, Keyboard_Data
mov    chars+2, AL
and    Keyboard_Status, 7FFFFFFFh
```

- Problems?

Example: Outputting to Printer

- Printer_Status MSB:
  - '1' means there's a character waiting to be printed by printer (set by CPU)
  - '0' means there isn't (cleared by printer)
- Printing 1 character:

```
mov   AL, 'a'
mov    Printer_Data, EAX
or     Printer_Status, 80000000h
```

Printing 3 Characters

```
mov   AL, 'a'
mov    Printer_Data, EAX
or     Printer_Status, 80000000h
mov   AL, 'b'
mov    Printer_Data, EAX
or     Printer_Status, 80000000h
mov   AL, 'c'
mov    Printer_Data, EAX
or     Printer_Status, 80000000h
```

- Problems?

Checking Status First: Input

- Code to read a character:

```
keyboard_wait:
        test  Keyboard_Status, 80000000h
        jz    keyboard_wait
        mov   EAX, Keyboard_Data
        mov    chars, AL
        and    Keyboard_Status, 7FFFFFFFh
```

- This is called a "spin loop", or "busy waiting"

Reading 3 Characters, Again

```
keyboard_wait1:
        test  Keyboard_Status, 80000000h
        jz    keyboard_wait1
        mov   EAX, Keyboard_Data
        mov    chars, AL
        and    Keyboard_Status, 7FFFFFFFh
keyboard_wait2:
        test  Keyboard_Status, 80000000h
        jz    keyboard_wait2
        mov   EAX, Keyboard_Data
        mov    chars, AL
        and    Keyboard_Status, 7FFFFFFFh
keyboard_wait3:
        test  Keyboard_Status, 80000000h
        jz    keyboard_wait3
        mov   EAX, Keyboard_Data
        mov    chars, AL
        and    Keyboard_Status, 7FFFFFFFh
```

Checking Status First: Output

- Code to print a character:

```
printer_wait:
        test  Printer_Status, 80000000h
        jnz    printer_wait
        mov   AL, ‘a’
        mov    Printer_Data, EAX
        or       Printer_Status, 80000000h
```

Printing 3 Characters

```
printer_wait1:
        test  Printer_Status, 80000000h
        jnz    printer_wait1
        mov   AL, ‘a’
        mov    Printer_Data, EAX
        or       Printer_Status, 80000000h
printer_wait2:
        test  Printer_Status, 80000000h
        jnz    printer_wait2
        mov   AL, ‘b’
        mov    Printer_Data, EAX
        or       Printer_Status, 80000000h
printer_wait3:
        test  Printer_Status, 80000000h
        jnz    printer_wait3
        mov   AL, ‘c’
        mov    Printer_Data, EAX
        or       Printer_Status, 80000000h
```

Is Busy Waiting Good?

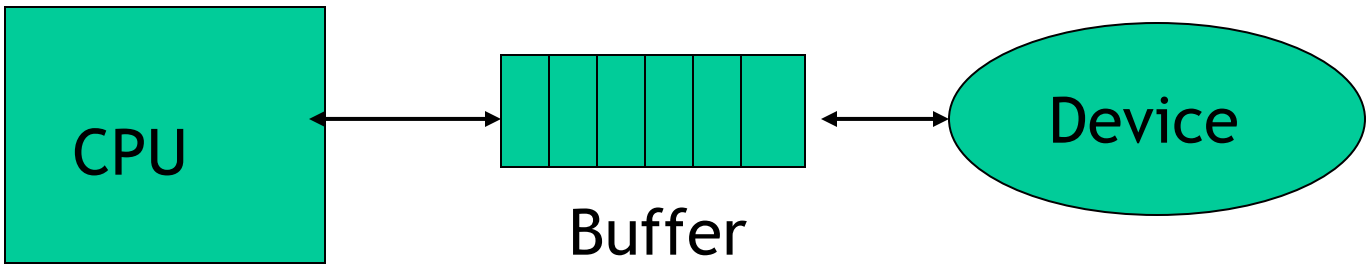
- Achieves maximum I/O performance
  - CPU responds as quickly as possible to "ready" indication
- Is CPU used efficiently? Assume...
  - Actual transfer of 1 character by CPU takes 10ns (a few instructions)
  - Printer has a maximum speed of 100KB/s = 10,000ns / character
  - CPU doing "useful" work only 10ns / 10,000ns = .01% of the time!

Improvements?

- Let CPU do other work while waiting for I/O devices: multitasking
- Periodically resume the necessary I/O processing
- Interrupt "other work" by…
  - a signal from the device (e.g., keyboard pressed by user)
  - a signal from a timer (e.g., for output to the printer)

Buffering

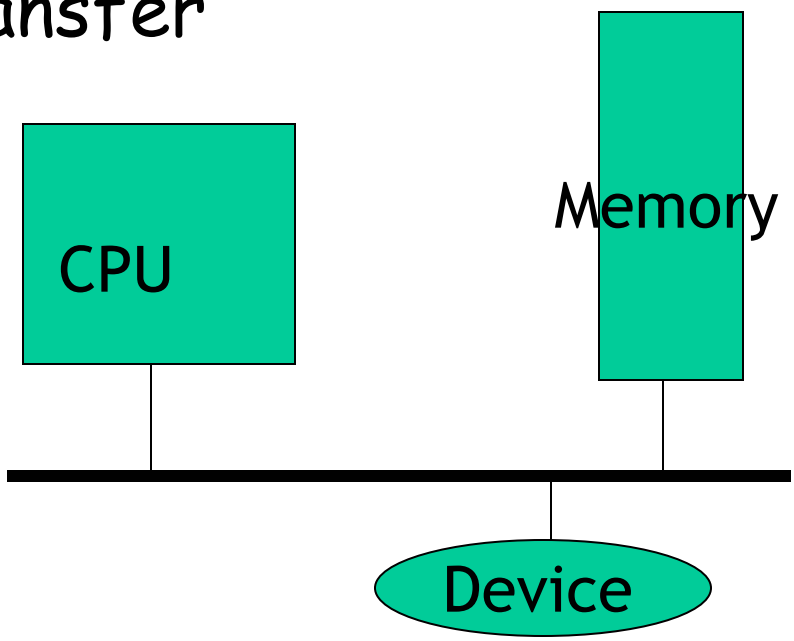
- Buffer = an area of memory for storing data transferred between a program and a device
  - ex.: characters typed on keyboard but not yet input to the program
  - ex.: characters output by the program but not yet printed



- Allows program to "get ahead of" the output device, or "be behind" the input device
  - greater flexibility, less overhead, more efficient

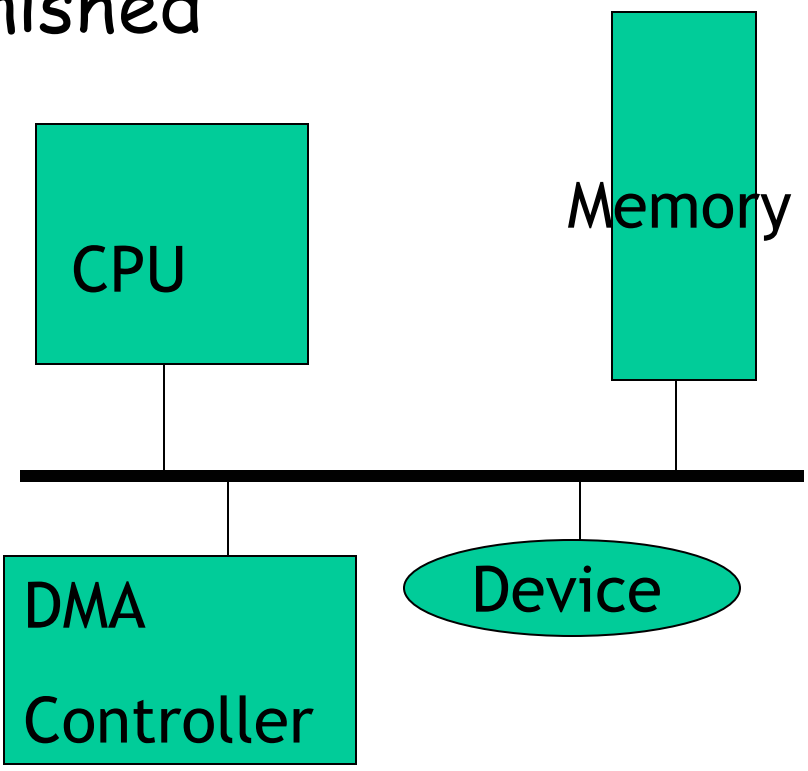
Direct Memory Access (DMA)

- Some devices are "block-oriented", like disk drives
  - Transfer byte 1 to chars
  - Transfer byte 2 to chars+1
  - Transfer byte 3 to chars+2
  - …
  - Transfer byte 512 to chars+511
- Inefficient for CPU to supervise the transfer



Direct Memory Access (DMA)

- Instead, CPU tells a co-processor
  - what device to transfer from
  - how many bytes to transfer
  - where in memory the bytes are to be stored
- Co-processor does the work, and notifies CPU (via interrupt) when finished



- A detail: how handle conflicts for the bus?