

Lecture 02: More on Memory and I/O Modules

William Stallings

Computer Organization

and Architecture

Chapter 7

Input/Output

Key Content

- The function of Input/Output module
- I/O addressing
- Different I/O techniques
 - Programmed
 - Interrupt
 - Direct Memory Access (DMA)

Input/Output Problems

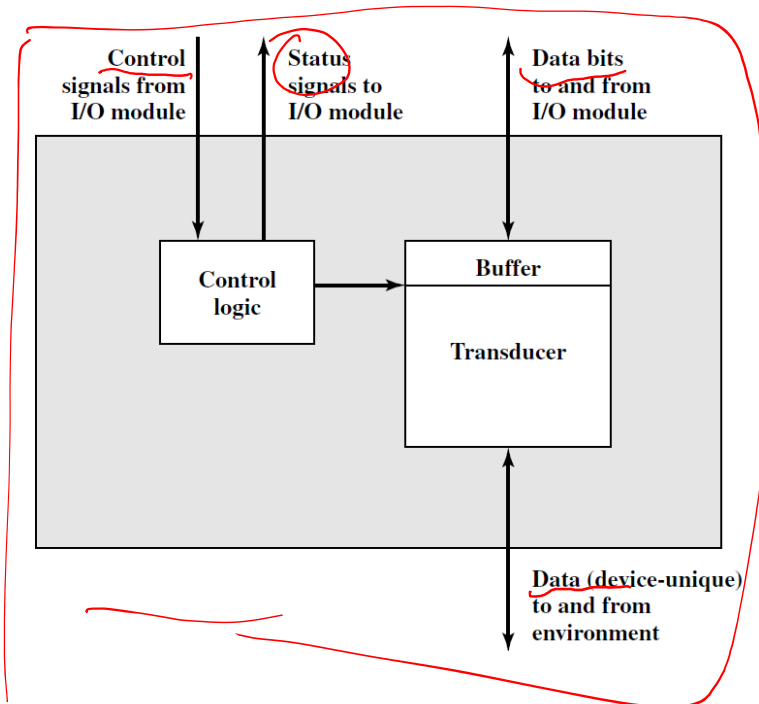
- Wide variety of peripherals
 - Different operation logic
 - > impractical for CPU to control all kinds of devices
 - Speak different "languages"
 - Delivering different amounts of data, e.g., serial/parallel
 - At different speeds
 - In different formats, e.g., analog/digital
 - > impractical for CPU to understand
 - Slower than CPU and RAM
 - > impractical to directly connect devices with high-speed system bus
- We need I/O modules (ports)!

Input/Output Module

- Interface to the CPU and Memory
- Interface to one or more peripherals
- It's like a bridge, an interpreter, a buffer, and ...

External Devices

- Human readable
 - Screen, printer, keyboard
- Machine readable
 - Monitoring and control
- Communication
 - Modem
 - Network Interface Card (NIC)



I/O Module Function

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection (⇒) known

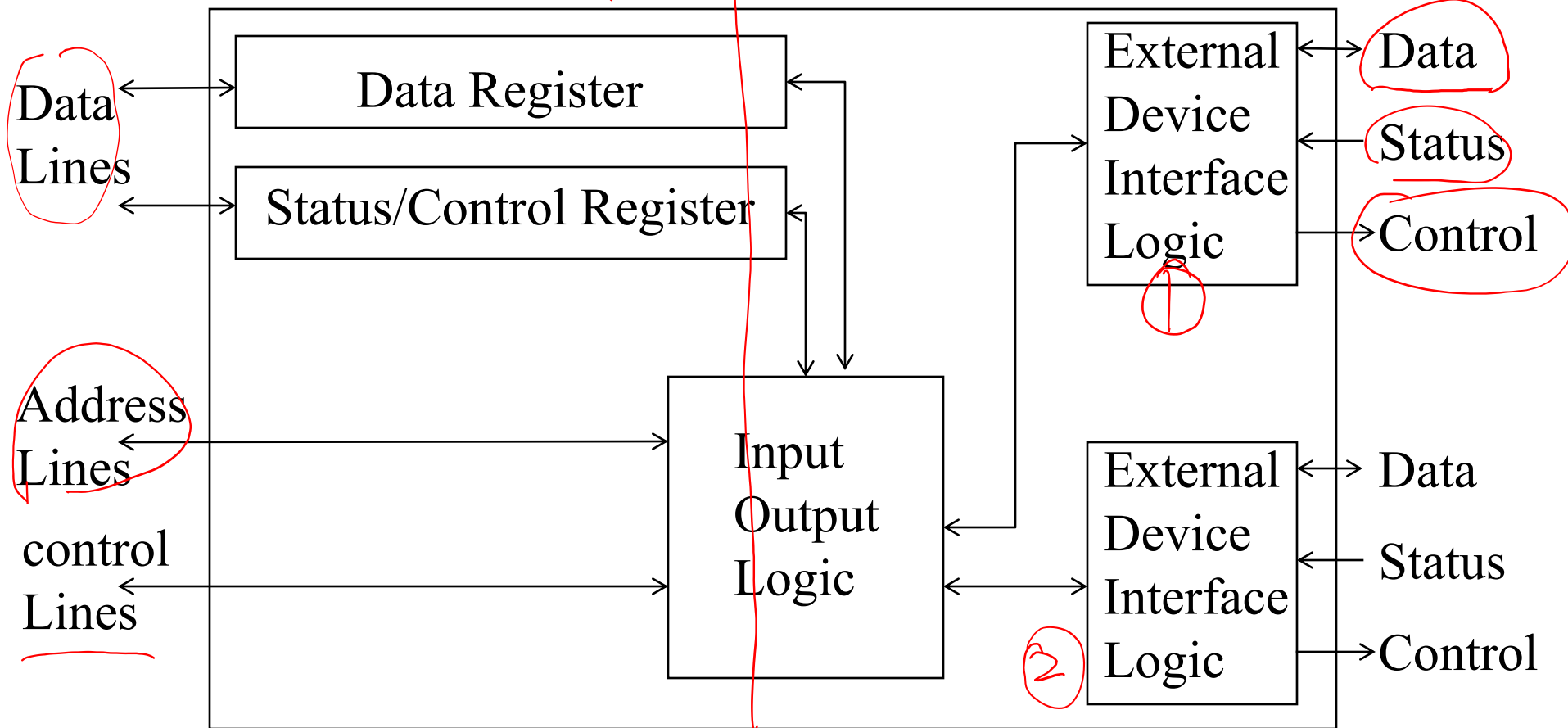
I/O Steps

- For example, the control of the transfer of data from an external device to the processor
 - CPU checks I/O module for device status
 - I/O module returns the device status
 - If the device is ready, CPU requests data transfer by means of a command to the I/O module
 - I/O module gets a unit of data from device
 - I/O module transfers the data to CPU
 - Variations for output, DMA, etc.

I/O Module Diagram

Interface to Systems Bus


Interface to External Device



I/O Module Design Decisions

- Hide or reveal device properties to CPU
- Support multiple or single device
- Control device functions or leave for CPU

Input Output Techniques

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

I/O at a High-level

■ Programmed I/O

Addition program in C

```
#include <stdio.h>

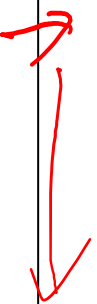
int main()
{
    int x, y, z;

    printf("Enter two numbers to add\n");
    scanf("%d%d", &x, &y);

    z = x + y;

    printf("Sum of the numbers = %d\n", z);

    return 0;
}
```



■ Question: what would happen when users do not enter the two numbers?

I/O at a High-level

■ Interrupt Driven I/O

- free the CPU from waiting for the I/O event

```
1  #include <signal.h>
2
3
4  void keyboard_handler(int signal)
5  {
6      // read the keyboard symbols
7  }
8
9  void main()
10 {
11
12     // first setup the keyboard interrupt handler
13     handler_init();
14     while (1) {
15         // do something else here
16     }
17 }
```

I/O at a High-level

■ Direct Memory Access

- free the CPU from waiting for the I/O event and waiting for the data

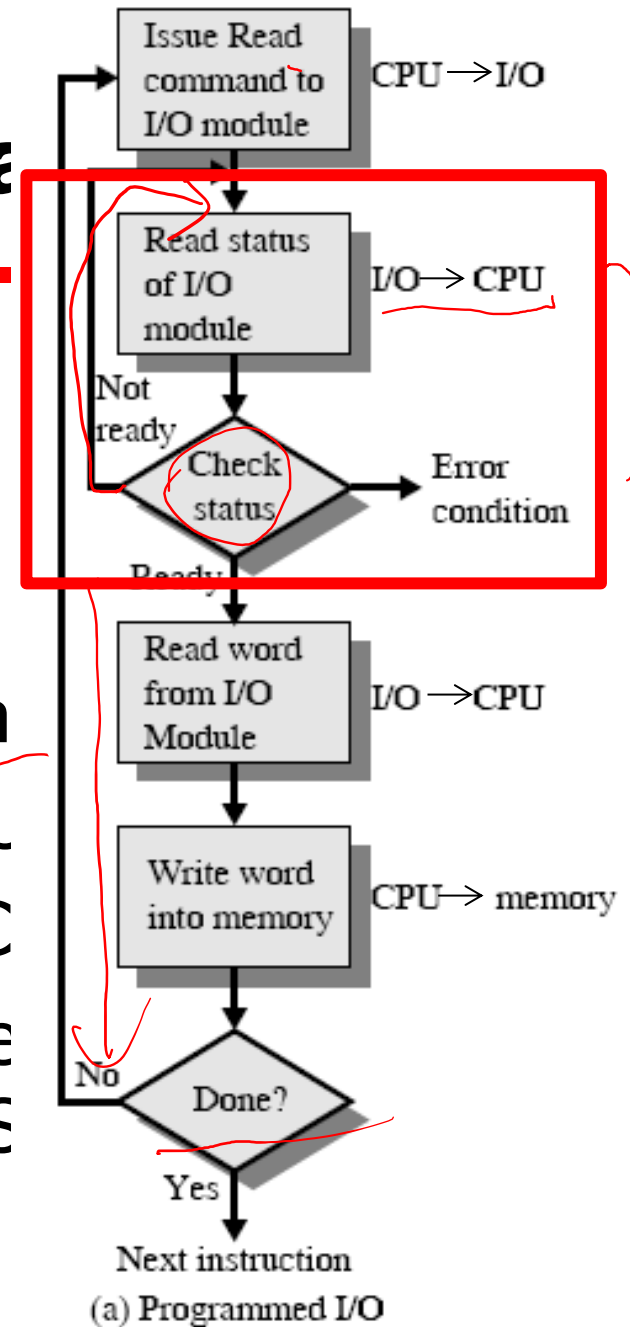
```
1  #include <dma.h>
2
3  void main()
4  ▼ {
5      int buffer[1024];
6
7      // first setup the dma
8      dma_init(buffer);
9
10     while (1) {
11         // do something else here
12     }
13 }
```

Programmed I/O

- CPU executes a program that gives it direct control of the I/O operation
 - Sensing device status
 - Read/write commands to the I/O module
 - Transferring data
- CPU waits for I/O module to complete operation

Programmed I/O - details

- 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
- I/O module does not interrupt CPU
- CPU may wait or come back later with the help of time-sharing OS



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?
 - 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 (from CPU viewpoint)
- Each device is given a unique identifier
- CPU commands contain the identifier (address) of the corresponding module (and device)

Addressing Schemes Revisited

■ 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

8086

Problem with Programmed I/O?

**Simple, but if CPU is faster,
it is a huge waste of CPU time**

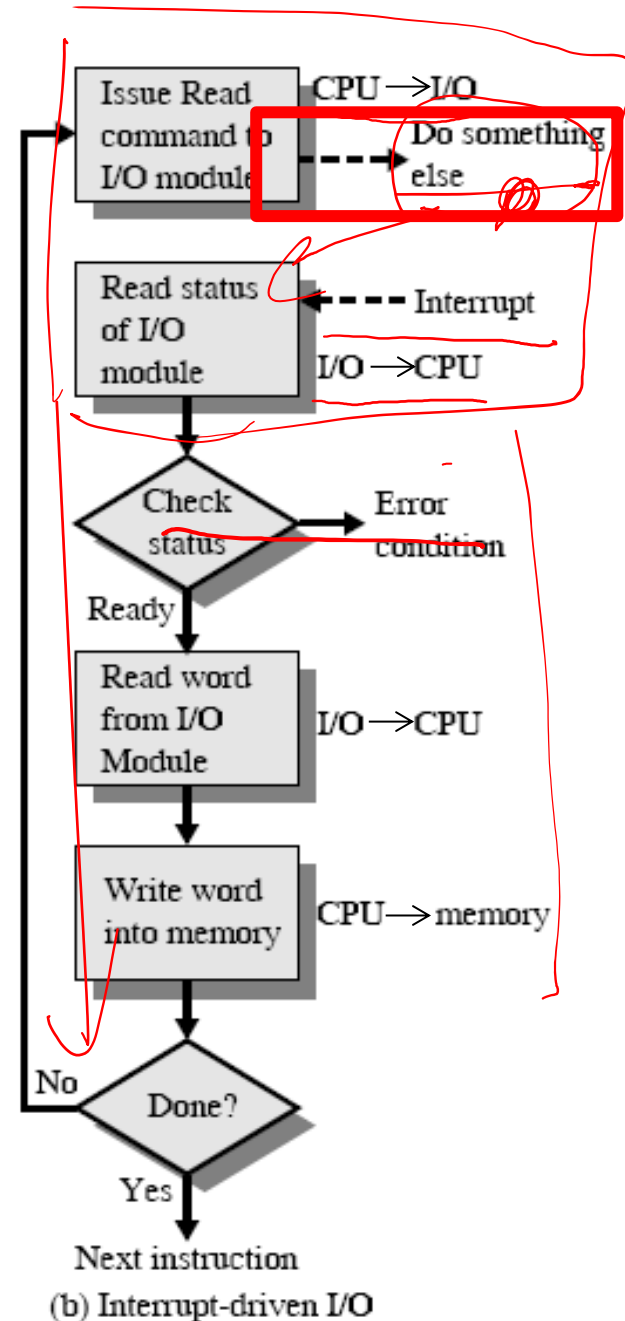
Interrupt Driven I/O

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

Interrupt Driven I/O

Basic Operation

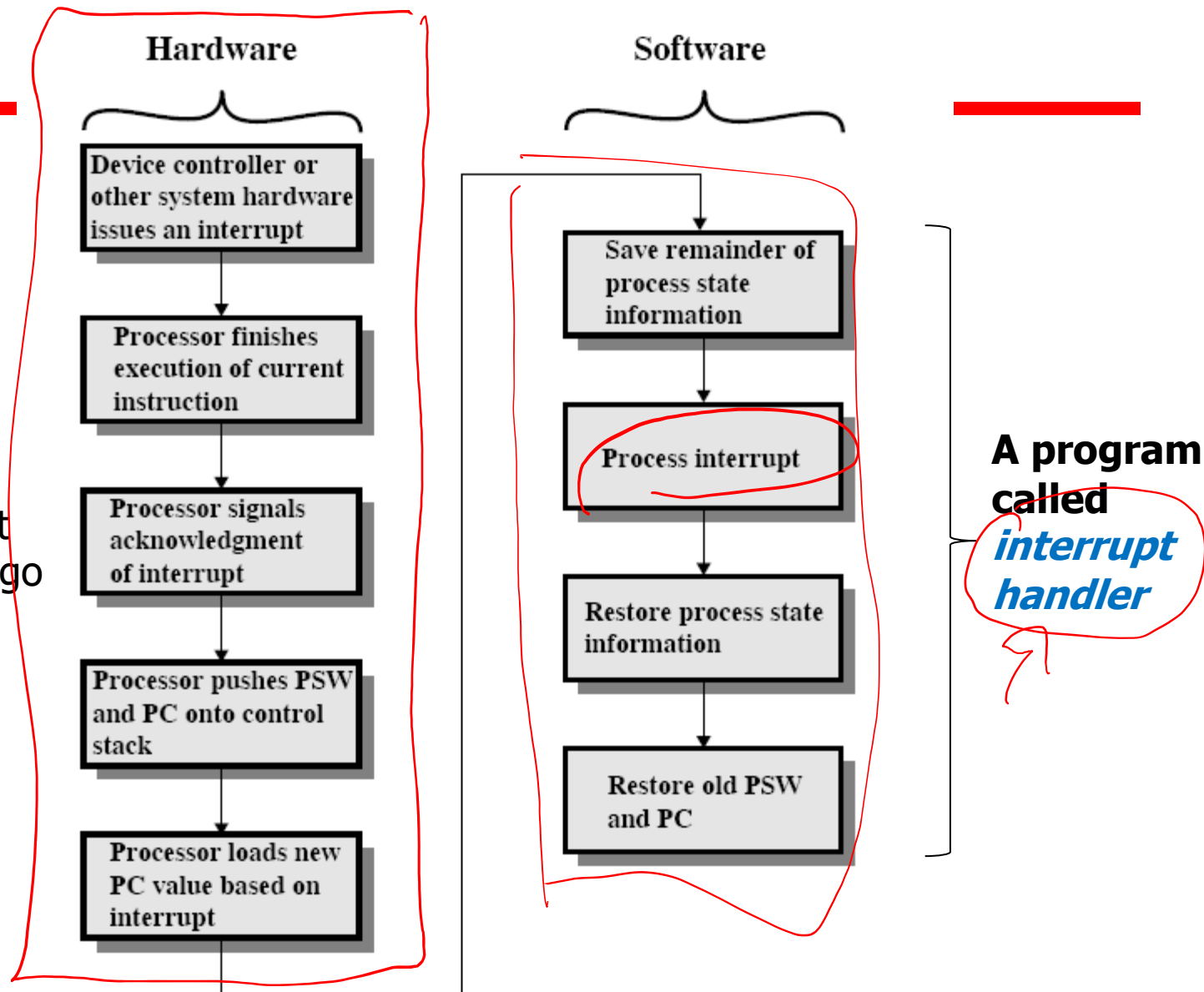
- CPU requests I/O operation
- I/O module performs operation other work
- I/O module informs CPU when set up by interrupting CPU
- CPU deals with this event



Handling an **Interrupt**: from a Protocol Perspective

Draw a protocol:
Which parties?
Interactions?

What's interrupt?
New event needs
CPU to handle first
but CPU needs to go
back to previous
work after that



CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at the end of each instruction cycle →
- If interrupted:-
 - Save context (registers)
 - Process interrupt
 - Fetch data & store
 - Recover from the saved context

Design Issues

- How can CPU know which module is issuing the interrupt?
 - when there are multiple devices connected to the system
- How to locate the corresponding handler program when interrupted?
- How do you deal with multiple interrupts?
 - Possible for more than one devices to issue an interrupt simultaneously or in a row

Identifying Interrupting Module (1)

- Connect a dedicated line for each module
 - Limits the number of devices
- Software poll
 - All devices share one common Interrupt Request line to interrupt CPU
 - Once get an interrupt, CPU asks each module in turn
 - CPU clears the interrupt request status of the module responsible

Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll
 - All devices share one common Interrupt Request line to interrupt CPU
 - Interrupt Acknowledge signal is sent down a chain
- Bus Master
 - Module must claim the bus before it can raise interrupt
 - e.g. PCI & SCSI
- Interrupt controller
 - 8259

Localizing Handler Programs

- Using a general handler program
 - CPU enters this handler every time it gets interrupted
 - looks for the module responsible and gets the address of the corresponding handler program
- Using **interrupt vectors**
 - instead of using fixed locations, a handler program can be stored anywhere in memory
 - a pointer is used to link to the handler program
 - the address of the pointer is fixed and known to CPU
 - such pointers are interrupt vectors
- Pros and cons?

Dealing with Multiple Interrupts

- Set **priorities** for interrupts
 - i.e., high-priority interrupts get served first
 - Given a interrupt identification scheme, how to set priorities?
 - Software and hardware polling, bus mastering
- Nesting of interrupts
 - i.e., high-priority interrupts can further interrupt low-priority interrupts

Problem with Programmed and Interrupt-driven I/O?

They both need the involvement of CPU.

Direct Memory Access

- Interrupt driven and programmed I/O require active CPU intervention
 - Transfer rate is limited
 - CPU is tied up
- DMA is the answer

DMA Function

- Additional Module (hardware) on bus
- DMA controller takes over from CPU for I/O

DMA Operation

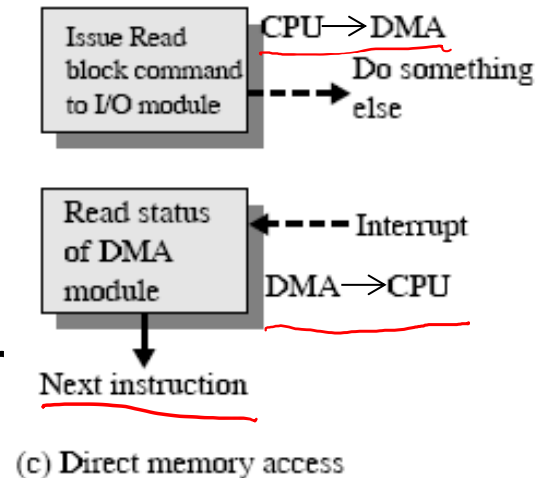
■ CPU tells DMA controller:-

- Read/Write
- Device address
- Starting address of memory block for
- Amount of data to be transferred

■ CPU carries on with other work

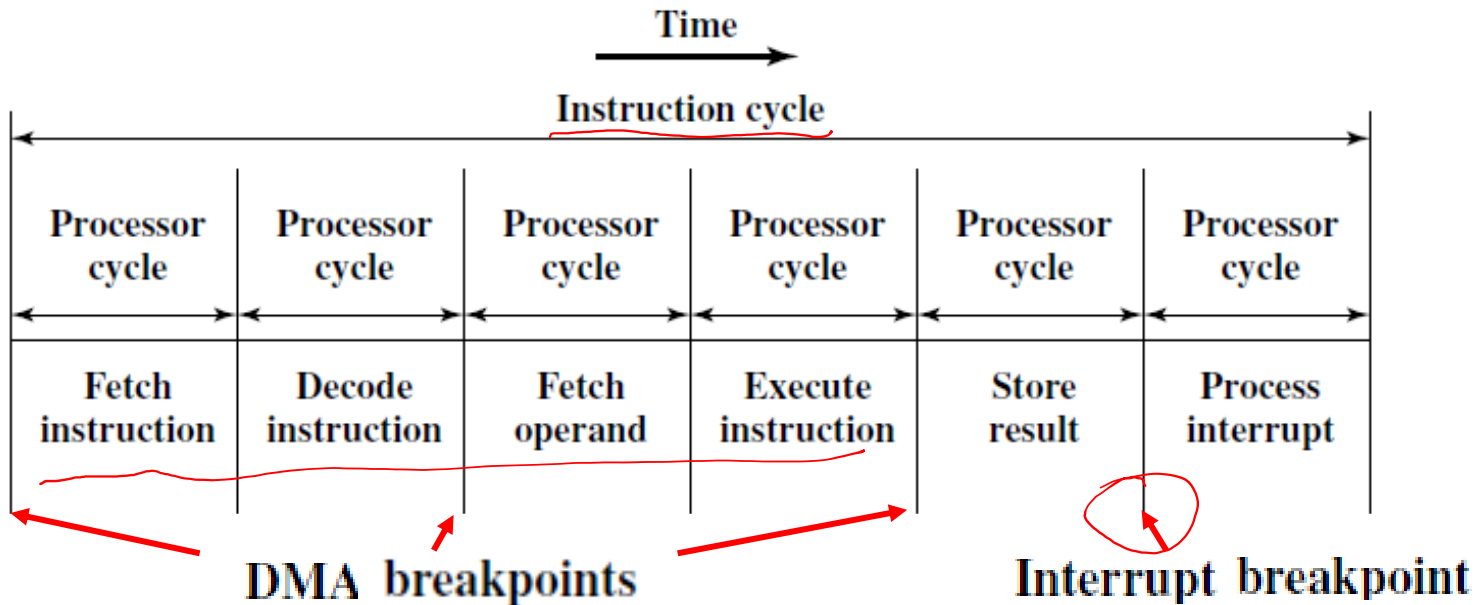
■ DMA controller deals with transfer

■ DMA controller sends interrupt when finished

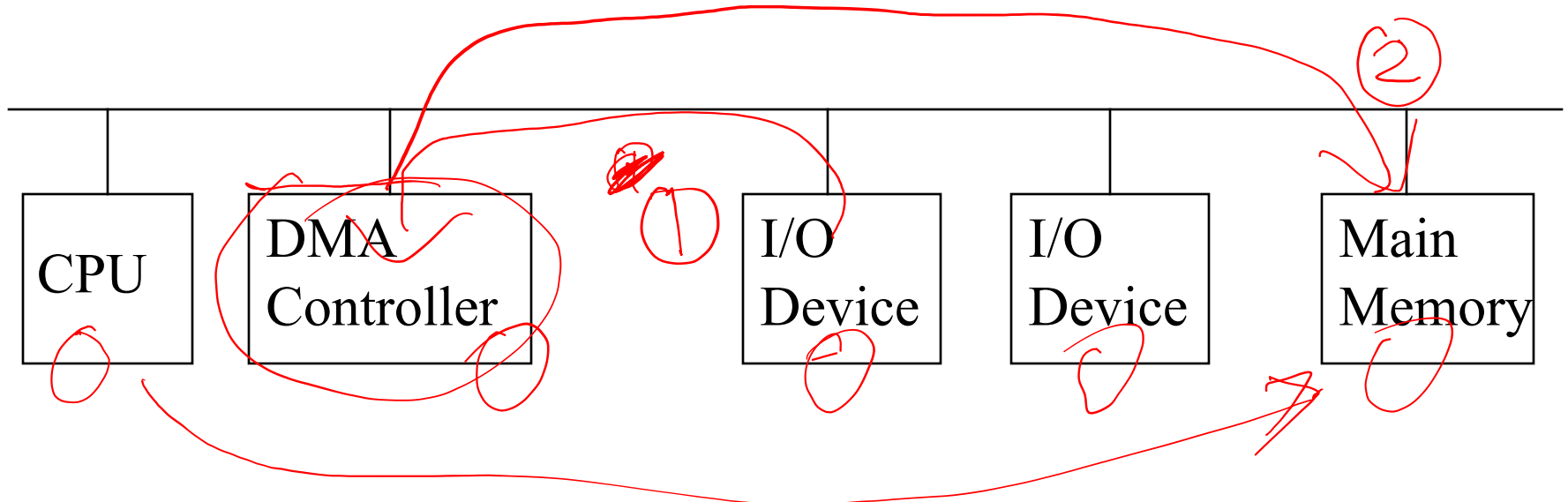


DMA Transfer Cycle Stealing

- In an instruction cycle, the processor may be suspended due to DMA operation
 - CPU suspended just before it accesses bus
 - DMA controller takes over bus for a cycle
 - Transfer of one word of data
 - Not an interrupt: CPU does not switch context
 - Slows down CPU but not as much as CPU doing transfer

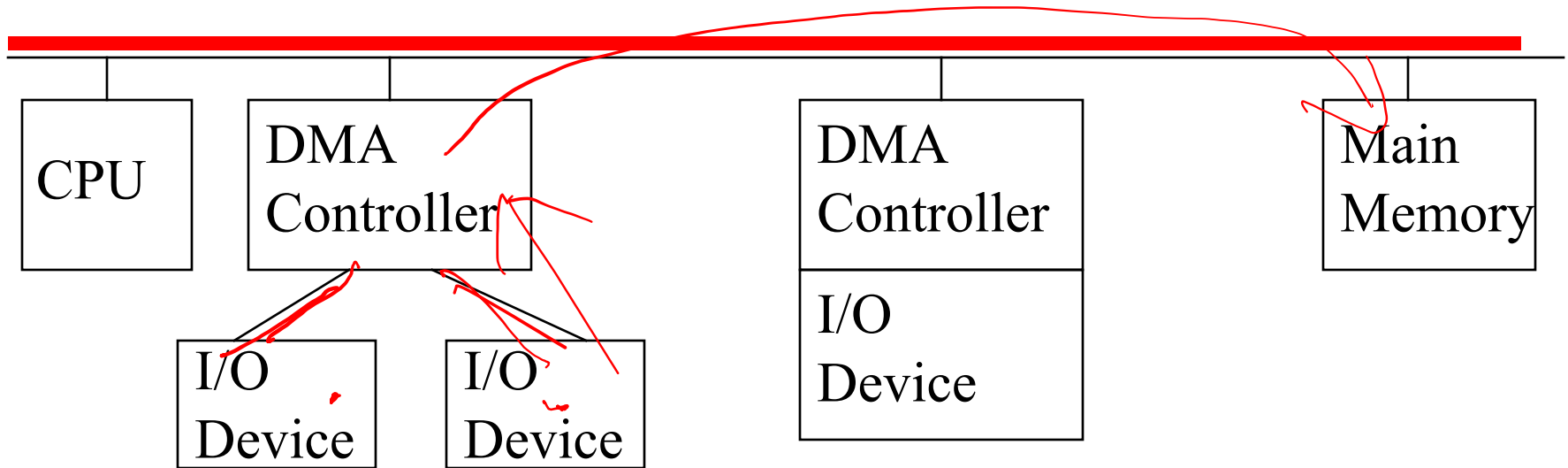


DMA Configurations (1)



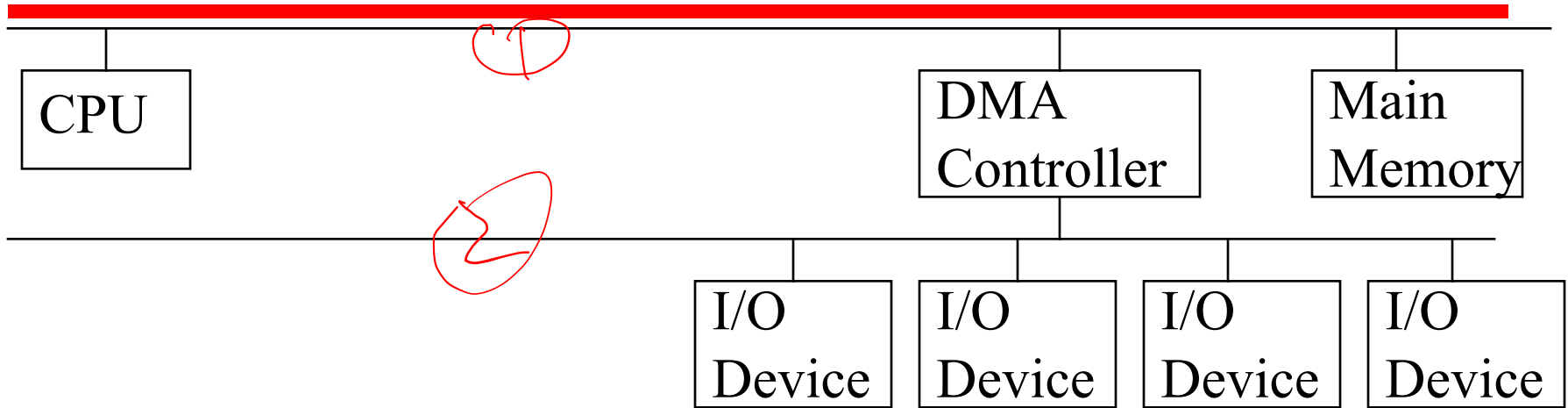
- Single Bus, Detached DMA controller
- Each transfer uses bus twice
 - e.g., I/O to DMA then DMA to memory
- For one transfer, CPU is suspended twice

DMA Configurations (2)



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

DMA Configurations (3)



- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
 - DMA to memory
- For one transfer, CPU is suspended once

Which way is the best?

- Simplicity ↓
- Performance ↓

trade-off

Understand Different I/O from the Hardware/Software

CPU \longleftrightarrow I/O device

Interaction with I/O Device	Programmed I/O	Interrupt-driven I/O	<u>Direct Memory Access</u>
<u>Waiting for the Device</u>	Software (<u>instructions running on the CPU</u>)	<u>Hardware</u>	<u>Hardware</u>
<u>Transfer the Device Data to the Memory</u>	<u>Software</u>	<u>Software</u>	<u>Hardware</u>

Assignment Three

- Computer Organization and Architecture 8e by William Stallings
 - Problem 7.8, 7.9, 7.10 (a, b)