



HARDWARE SUPPORT FOR OS

Instructor: William Zheng

Email:

william.zheng@digipen.edu

PHONE EXT: 1745

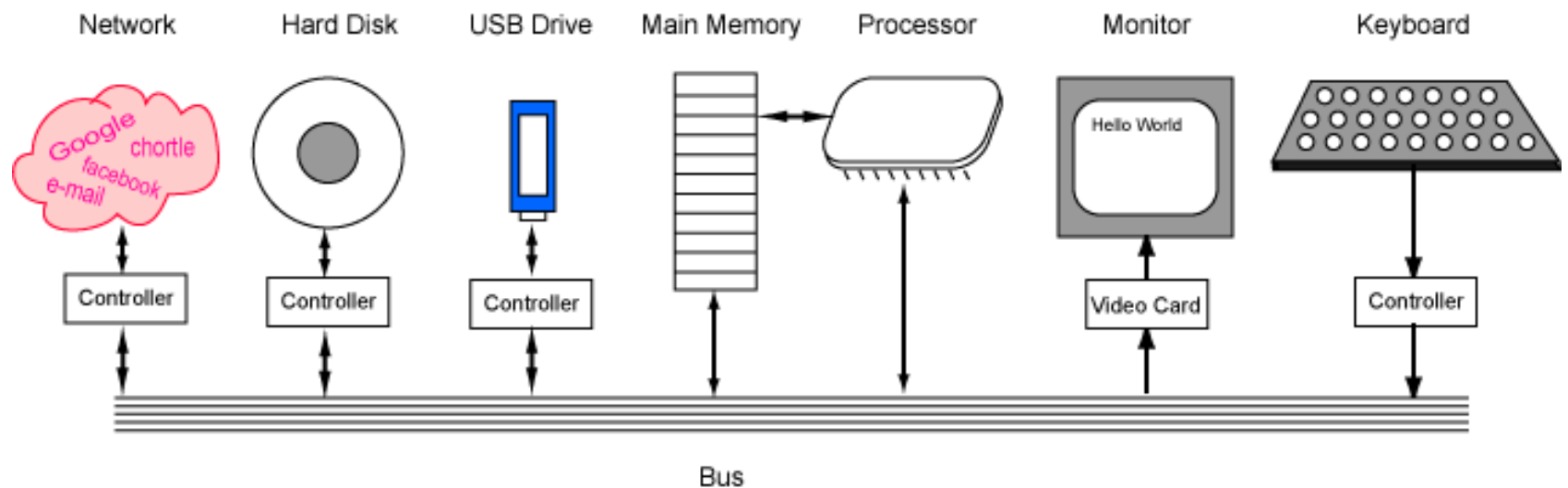
Goals for this lecture

- HW and SW
- CPU - I/O Communication
 - Interrupt handling
 - DMA
 - More details in chapter 13.1-13.2 of textbook
- CPU modes
 - Backward-compatibility modes
 - Privilege modes

HW & SW

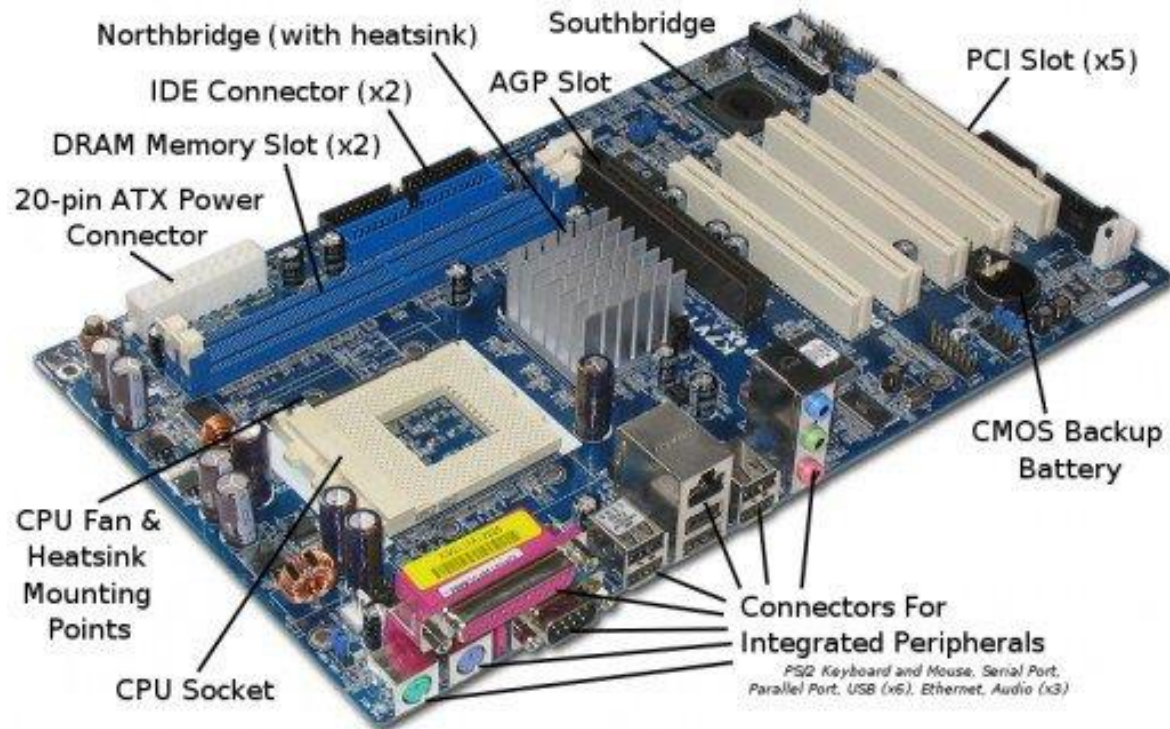
- Difference?
- Important to know the distinction
 - OS (SW)
 - But what OS can do
 - Limited by the hardware it's running on

Computer System Components

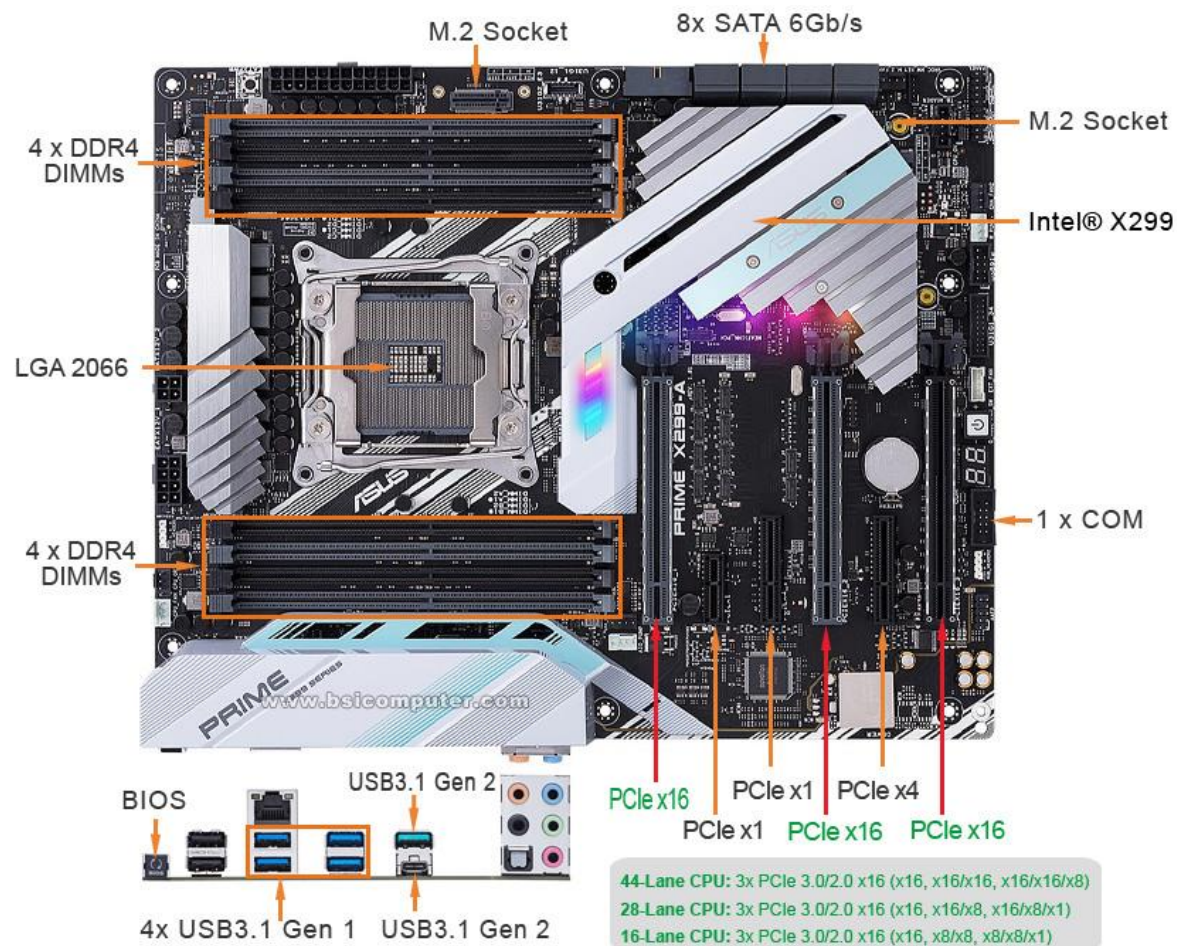


Main Components of a Computer System

Motherboard - 1

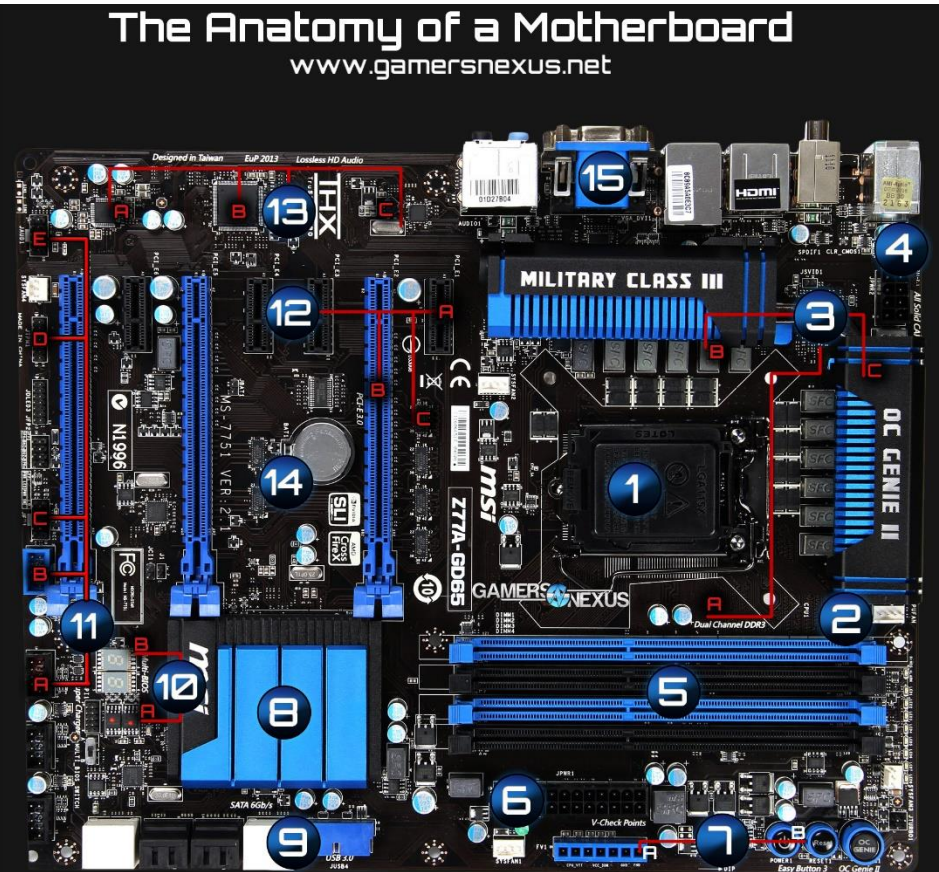


Motherboard - 2

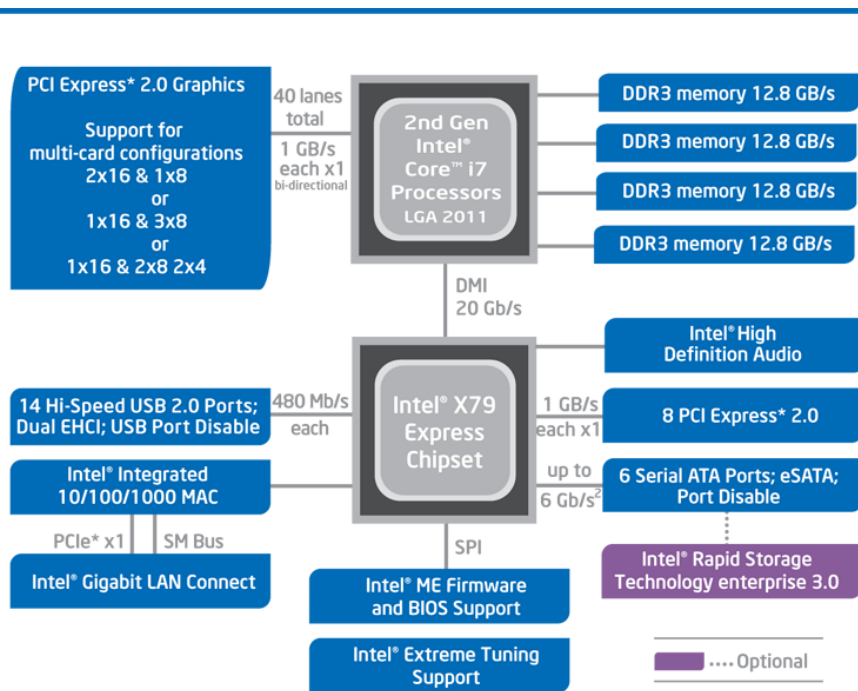


Motherboard - 3

1. CPU Socket
2. CPU Fan
3. VRM
 - a. Capacitors
 - b. Chokes (SFC)
 - c. MOSFET/sink
4. 12v CPU Power
5. RAM Slots
6. 24pin Power
7. Troubleshooting
8. Chipset
9. USB3 + SATA
10. Displays
 - a. BIOS chips
 - b. Segment Display (for POST)
11. Front I/O
 - a. USB2.0
 - b. 1394/FireWire
 - c. Front Panel
 - d. TPM header
 - e. HD Audio
12. PCI-e
 - a. PCI-e x1
 - b. PCI-e x16/x8
 - c. PLX / PEX Chip
13. Southwest Chips
 - a. Network Chip
 - b. Audio Chipset
 - c. Oscillating Clock Crystals
14. CMOS battery
15. Rear I/O panel



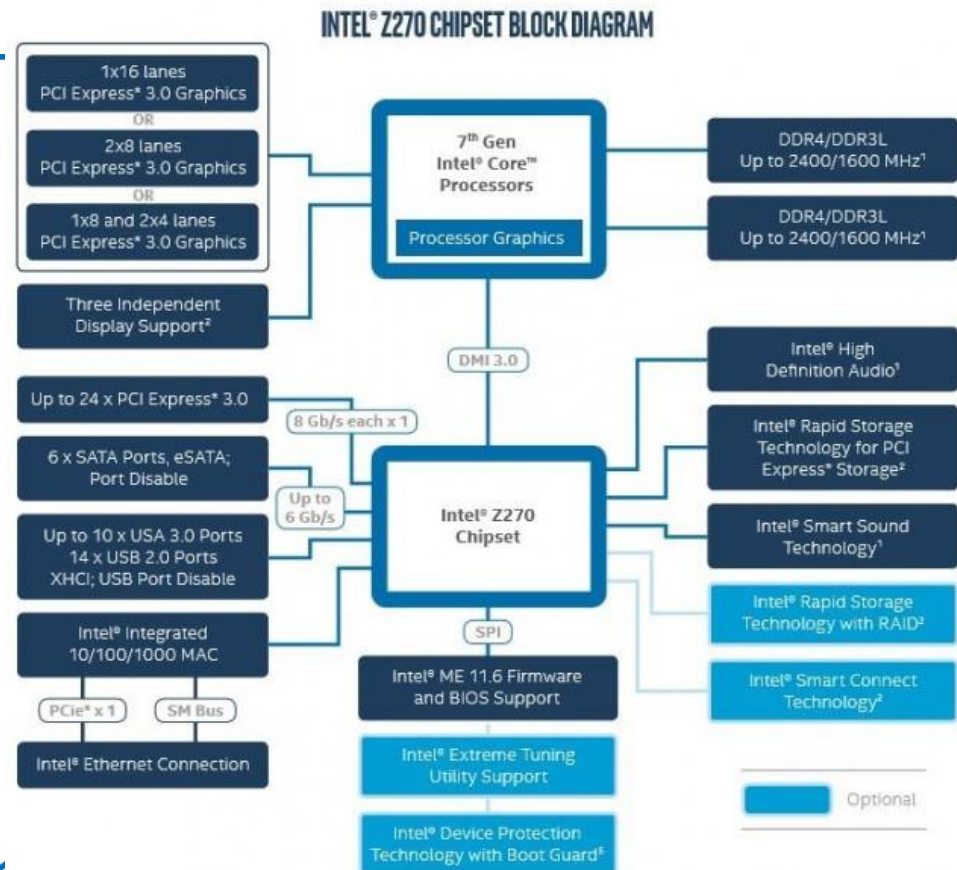
Chipset Block Diagram - 1



¹Theoretical maximum bandwidth

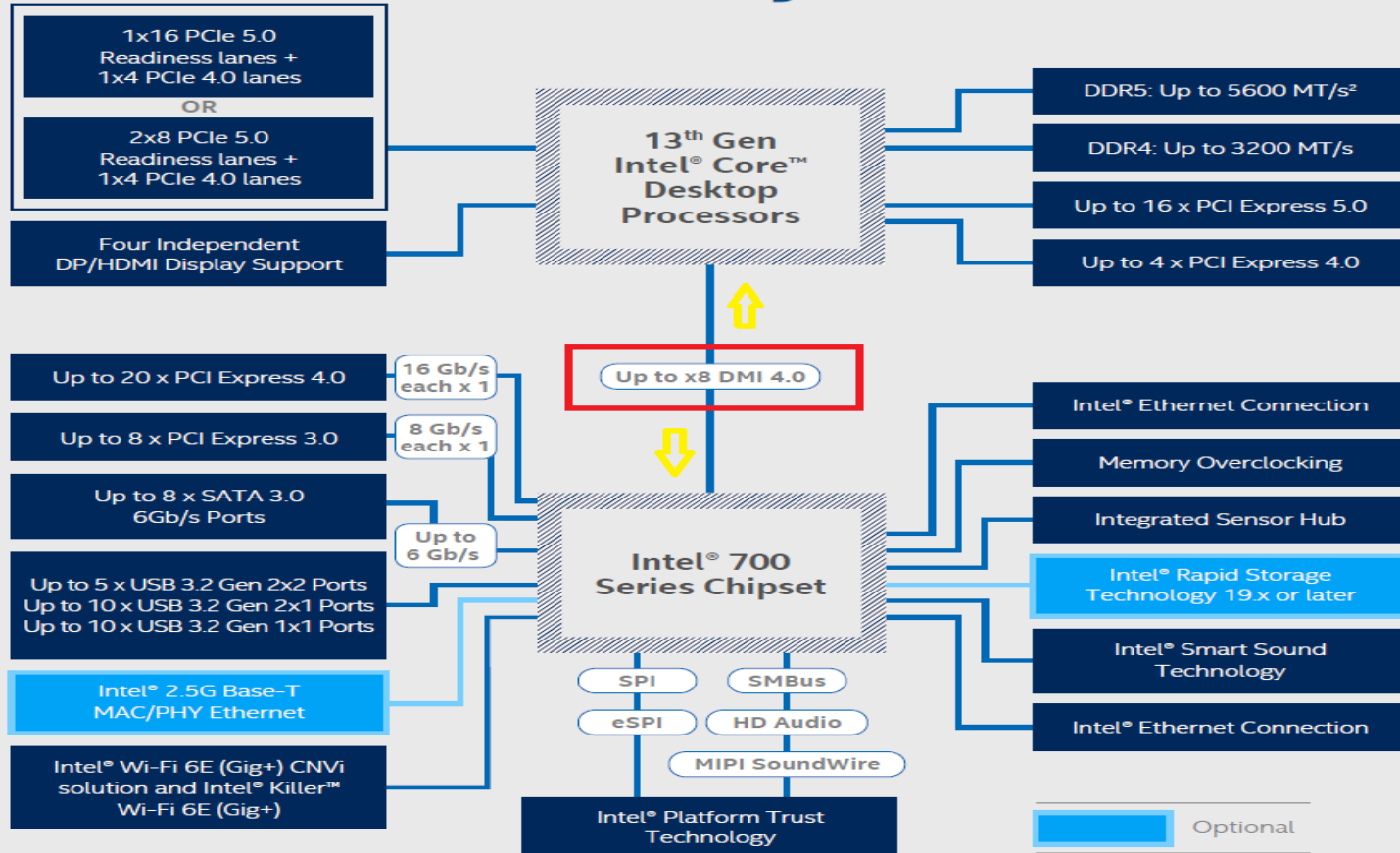
²All SATA ports capable of 3 Gb/s. 2 ports capable of 6 Gb/s.

Intel® X79 Express Chipset Block Diagram



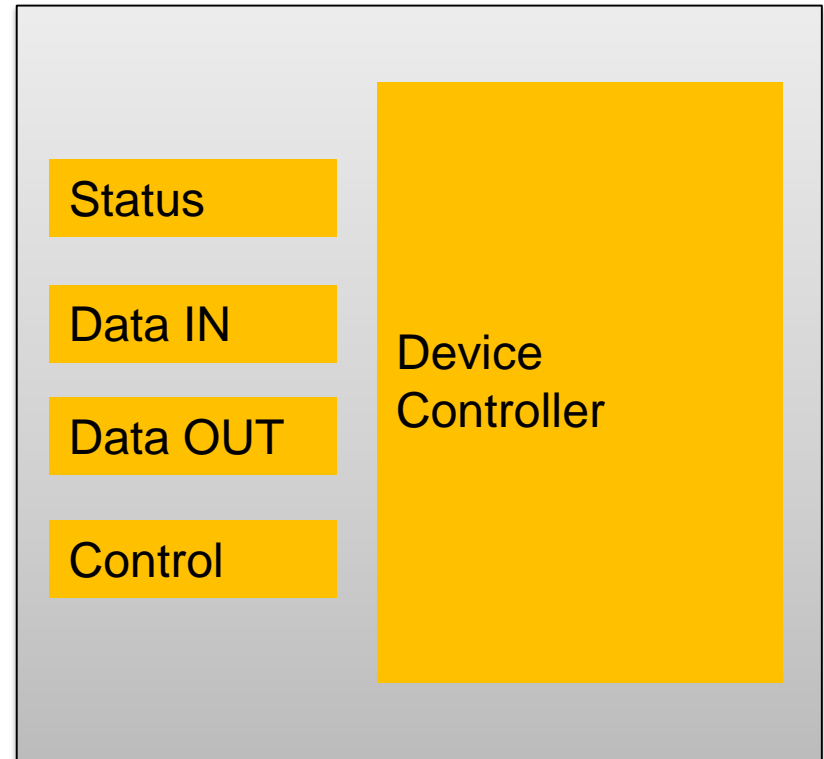
Chipset Block Diagram - 2

Intel® 700 Series Chipset
Block Diagram



Device Control Registers

- Status register
 - Done bit
 - Error bit
- Data register
 - Data to be printed
- Control registers
 - Commands



CPU – I/O Communication

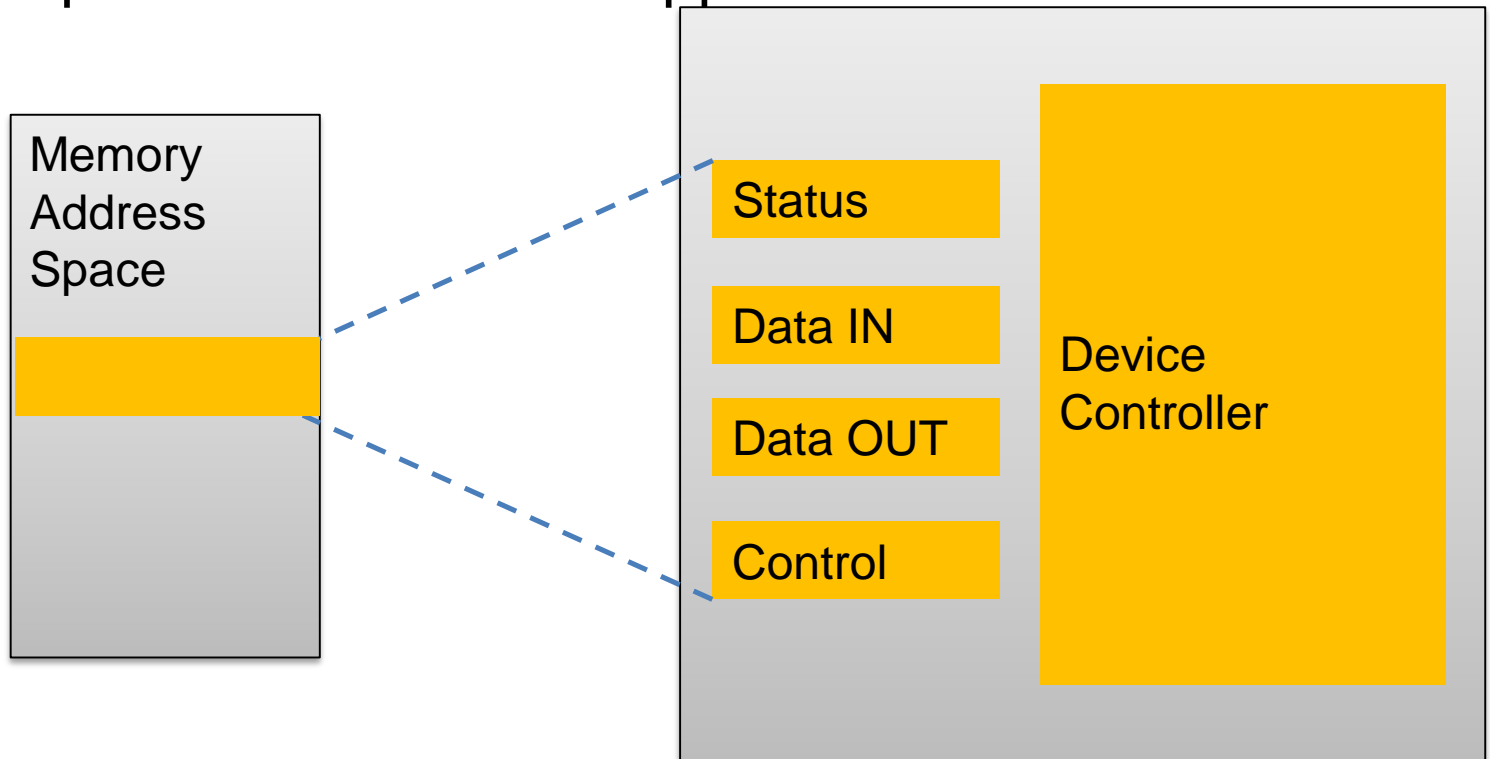
- From CPU to I/O
 - Special instructions
 - Memory-mapped I/O
 - Device data and command registers mapped to processor address space
 - Especially for large address spaces (graphics)
- From I/O to CPU
 - Polling
 - Interrupt

Special I/O Instructions

- Instruction specially for I/O
 - I/O ports
 - Different address spaces for memory and I/O devices
 - E.g. Intel IN, OUT instructions
 - Register I/O instructions: IN/OUT move data between I/O parts and the EAX (32-bit I/O), AX (16-bit I/O) or AL (8-bit I/O) general registers.
 - Block I/O instructions: INS/OUTS move blocks of data between I/O ports and memory space

Memory-mapped I/O - I

- Use memory read/write
 - E.g. intel MOV instructions
 - Specific addresses mapped to I/O devices

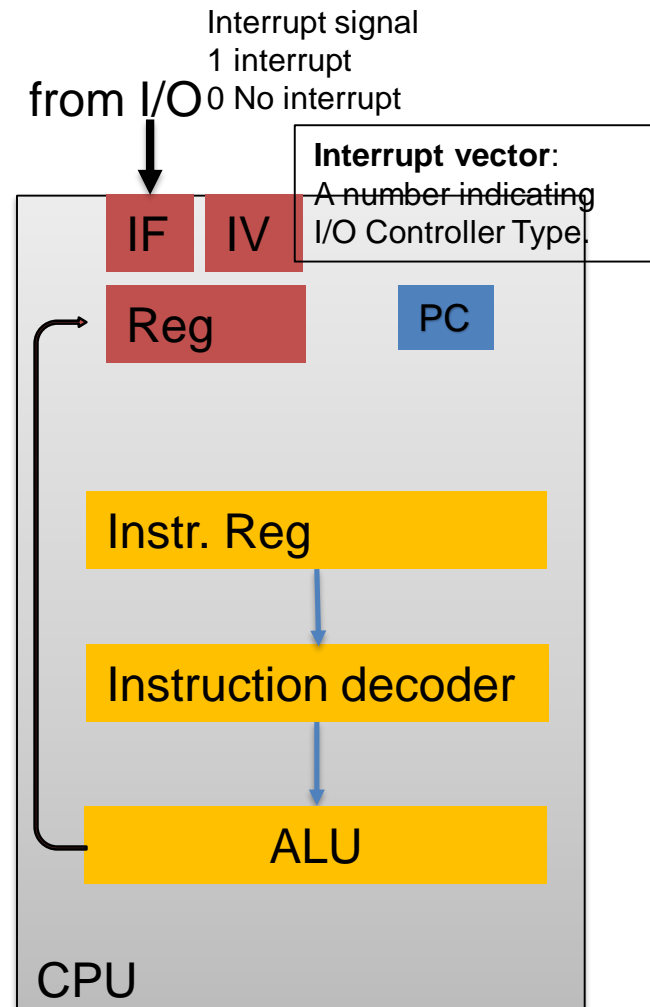


Memory-mapped I/O - II

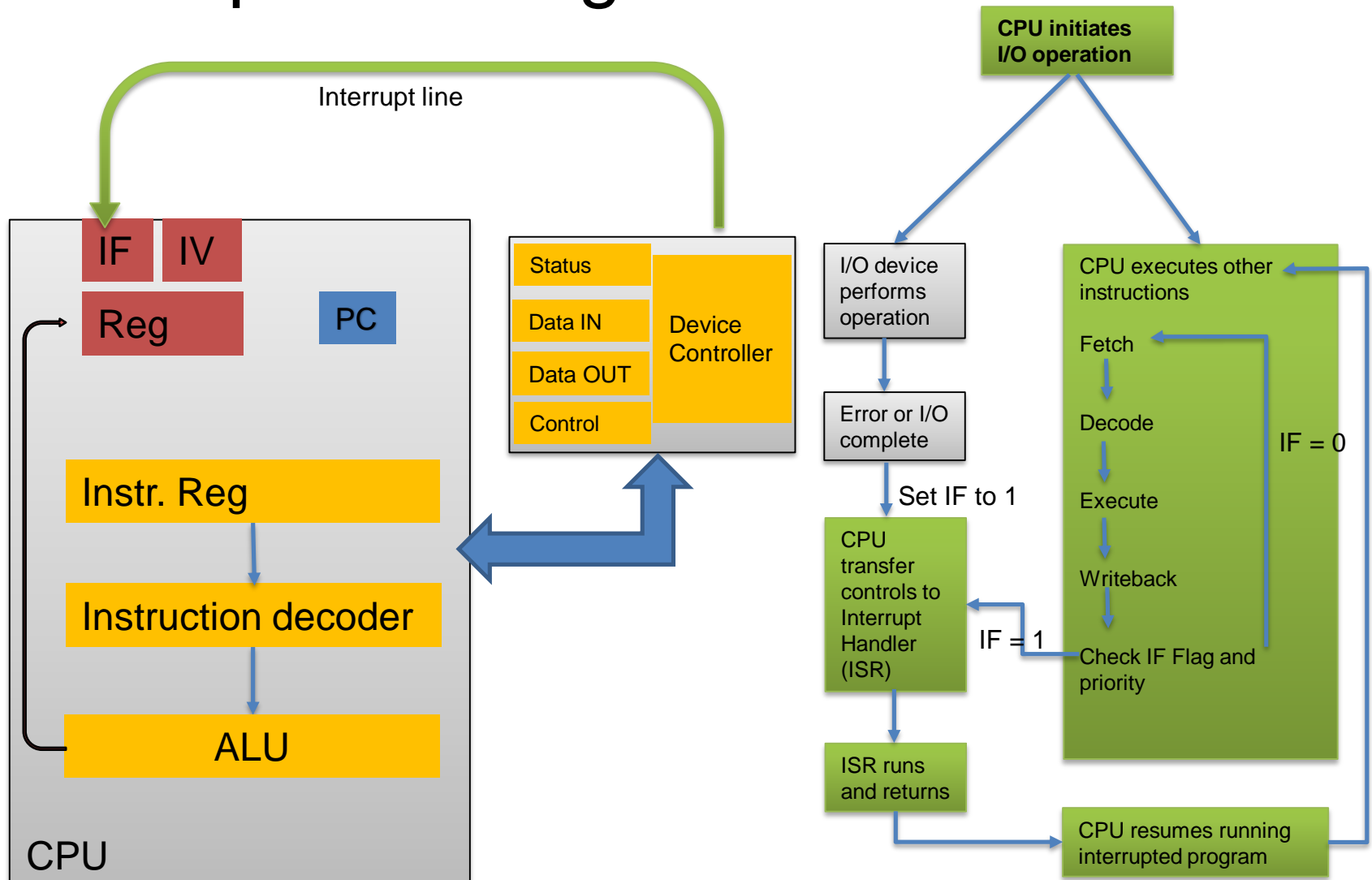
- Using software to wait for I/O *//wait for I/O to be done*
//done bit is on status register
- Can check status register at fixed times ***while (done_bit==0) ;***
//process further data for I/O
- Pros
 - Fast response time (for program waiting)
- Cons
 - Low CPU utilization

HW Setup for Interrupt Handling

- While waiting for I/O, do other jobs first
- Wait for I/O device to assert interrupt signal
- IF and IV register
 - Interrupt flag
 - Interrupt vector
 - A number indicating I/O controller type



Interrupt Handling

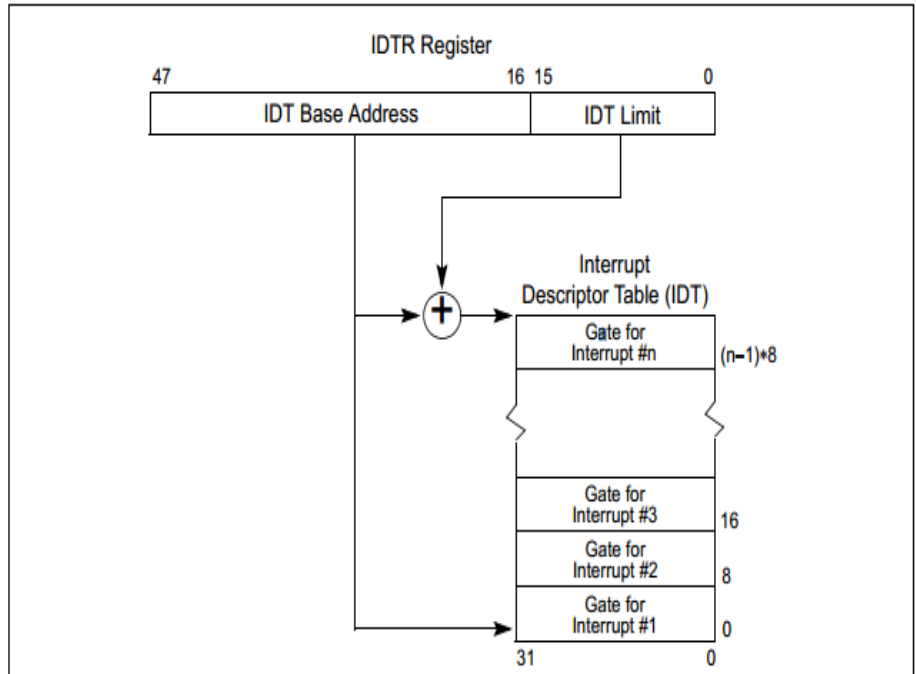


Interrupts (Cont.)

- 3 kinds of interrupt
 - Exceptions/Faults: Errors: e.g., divide by zero
 - Hardware Interrupt: I/O!
 - Software-generated interrupt: System calls
- Maskable and non-maskable
 - Maskable means turned-off
 - Usually exceptions/faults are not maskable.
- Interrupt priority
 - Interrupt handling can be interrupted by higher priority interrupts.

Interrupt Vector Table

- Interrupt Vector
 - Unique for each interrupt
- ISR
 - SW handling of interrupts
- Interrupt Vector Table
 - Intel – IDT
 - Real addressing mode: IVT
 - Protected mode: IDT
 - Array of function pointers



Intel Pentium Processor Event-Vector Table

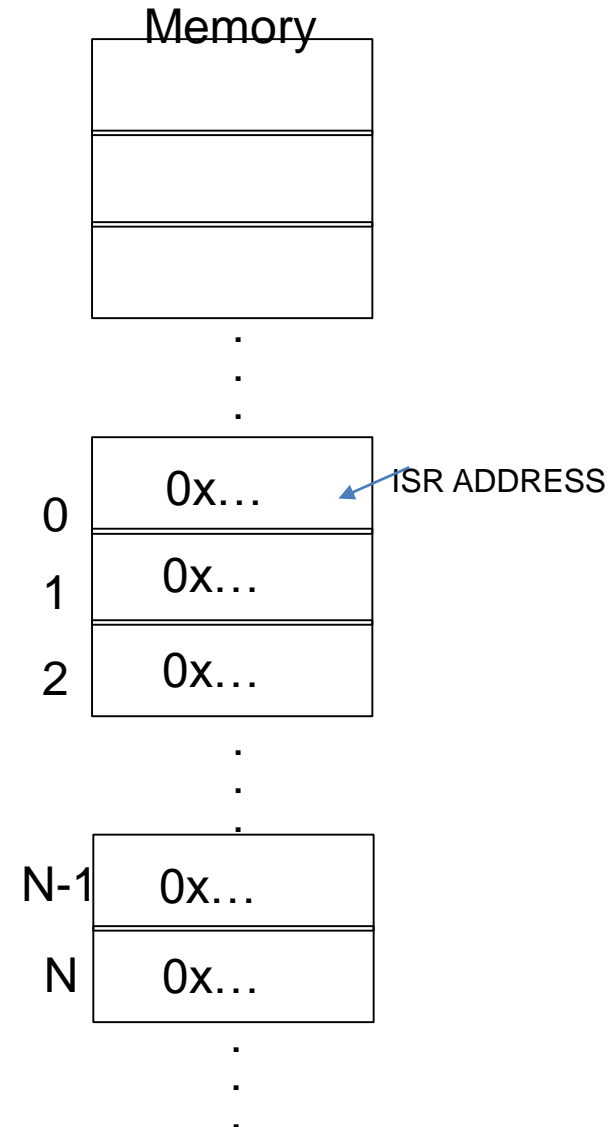
vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Interrupts (Cont.)

- Interrupt mechanism also used for **exceptions**
 - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
 - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

How does CPU transfer control to ISR?

- Must know the address of a function to call a function.
- Where is the interrupt vector table located?
 - Memory
- We want to know address containing the starting address of ISR for vector N
 - $\text{Base Address} + 8 * N$ for 64 bit
 - $\text{Base Address} + 4 * N$ for 32 bit
- Question: how does the HW know where is the base?
- Hardcoded (0x0000 to 0x03ff) during real mode
- IDTR (Interrupt Descriptor Table Register) (Protected mode)
 - Who sets up the IDTR?
 - Who sets up the IDT?

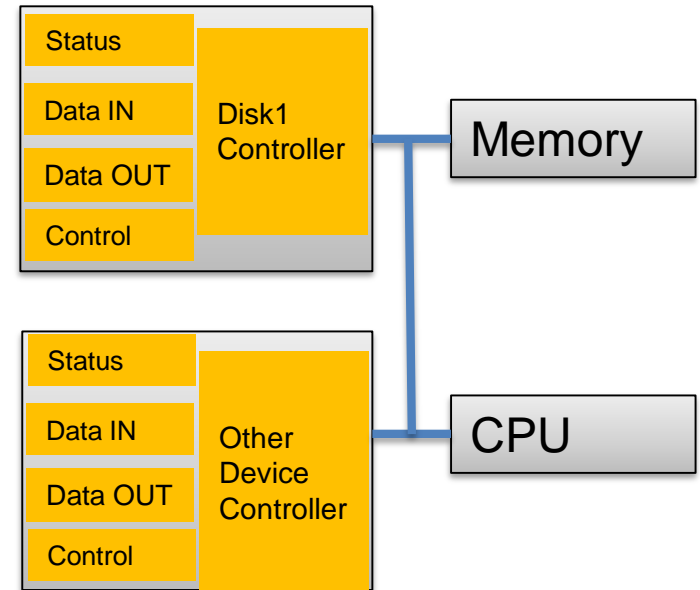


DMA: Motivation Scenario

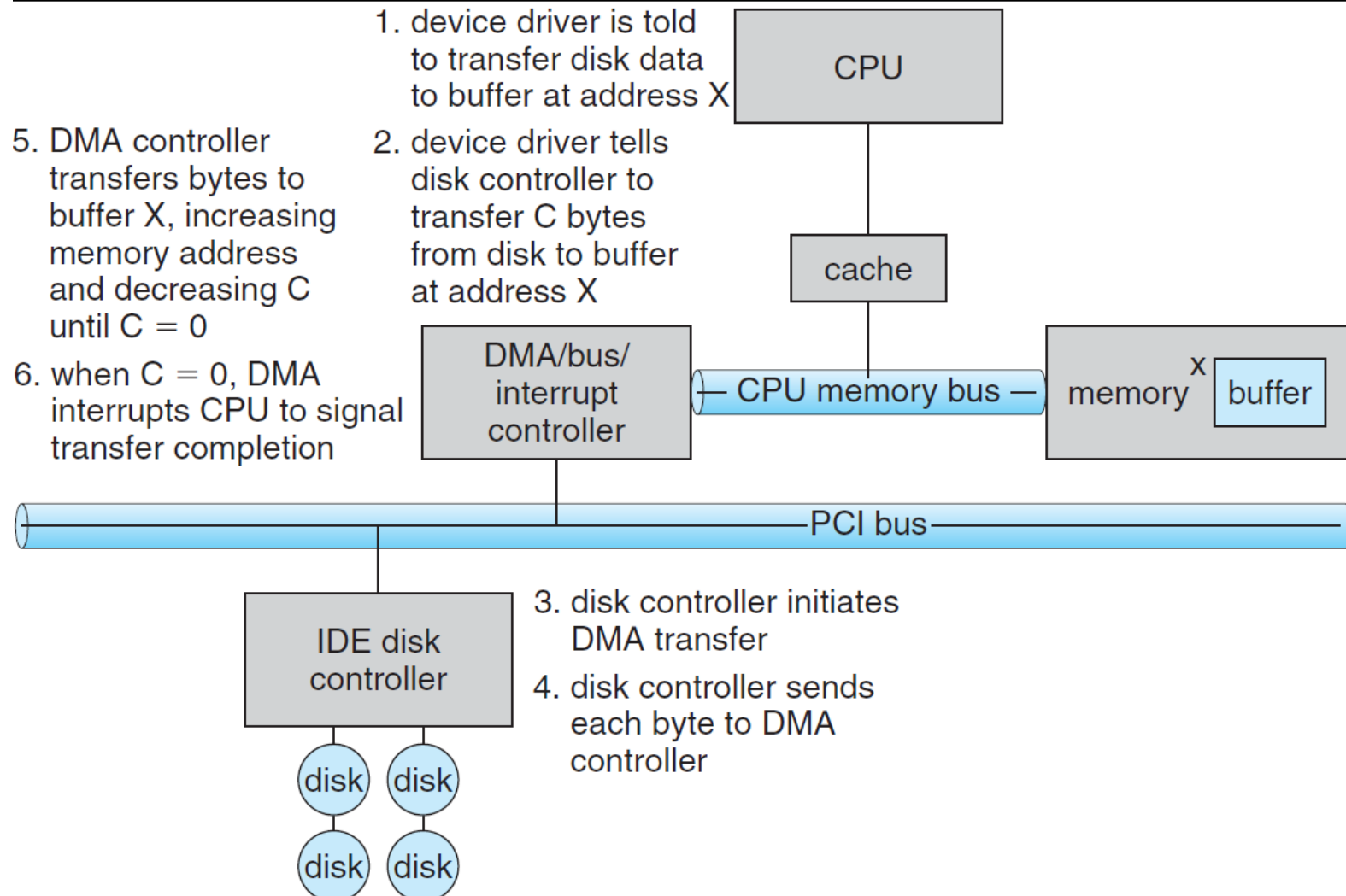
Copy 1KB from Disk 1 to
Memory (Disk Read)

A simple way:

1. CPU req Disk1 controller
for 1st byte
2. Disk1 prepares the byte.
Put it in Data-IN register.
3. Interrupt CPU
4. ISR reads the byte in Data-
IN



DMA Scenario (Fig 13.5 Textbook)



Direct Memory Access - I

- Allow devices to read/write memory directly without CPU intervention
- Does not mean accessing the memory in the same cycle
 - Whether multiple devices (e.g. CPU) can access the memory at the same time depends on the memory design
- Allows CPU processing to be in parallel with device to memory communication
 - Reduce number of interrupts

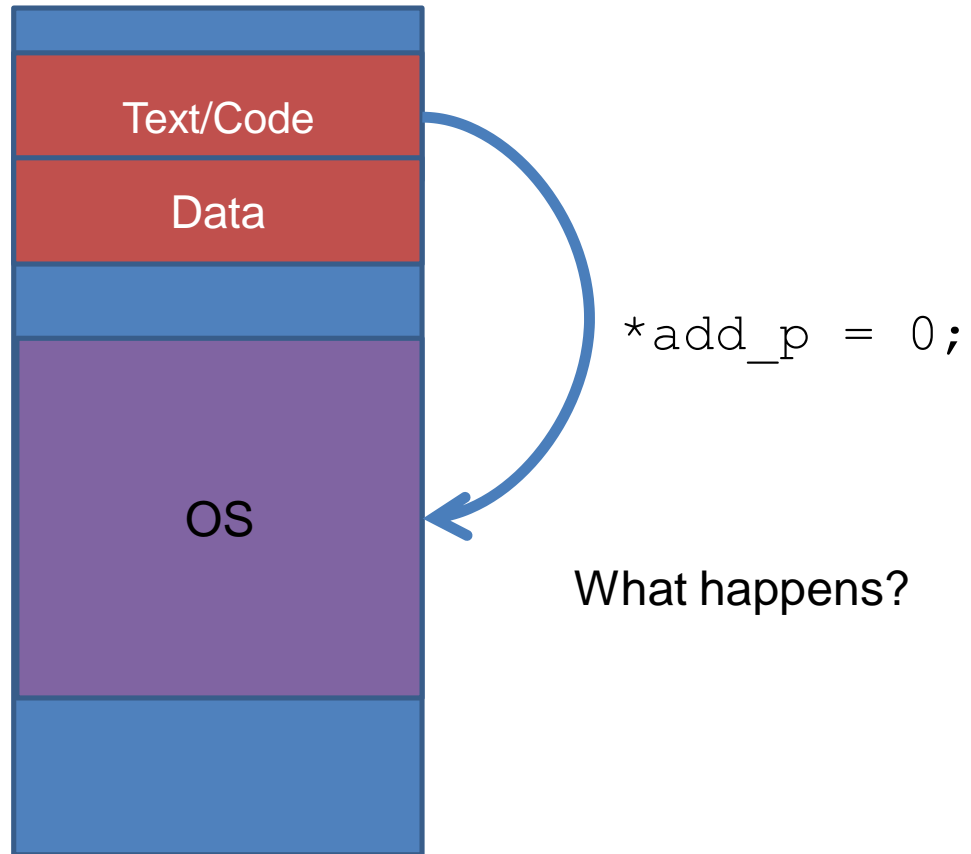
Direct Memory Access - II

- Used to avoid **programmed I/O** (one byte at a time) for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
 - Source and destination addresses
 - Read or write mode
 - Count of bytes
 - Writes location of command block to DMA controller
 - Bus mastering of DMA controller – grabs bus from CPU
 - **Cycle stealing** from CPU but still much more efficient
 - When done, interrupts to signal completion
- Version that is aware of virtual addresses can be even more efficient – **DVMA** (allows a device to perform a transfer from two **memory**-mapped devices without the intervention of the CPU or the use of main **memory** as a staging ground)

Access HW device

- User program access device directly?
 - User program talks with device driver via system calls `open()`, `read()`, `write()` etc
 - Protect data from access-control violations
 - Erroneous use leads to crash

MS-DOS and Intel 8088



CPU privilege modes

- Kernel (Supervisor mode) and user mode
- Privileged instructions
 - e.g. change the idtr value
- Other instructions
 - e.g. add, sub etc

