Input/Output

Week 11 – Tutorial

Problem 5.1 (1/2)

- Advances in chip technology have made it possible to put an entire controller, including all the bus access logic, on an inexpensive chip.

- How does that affect the model of Fig. 1-6?

Problem 5.1 (2/2)

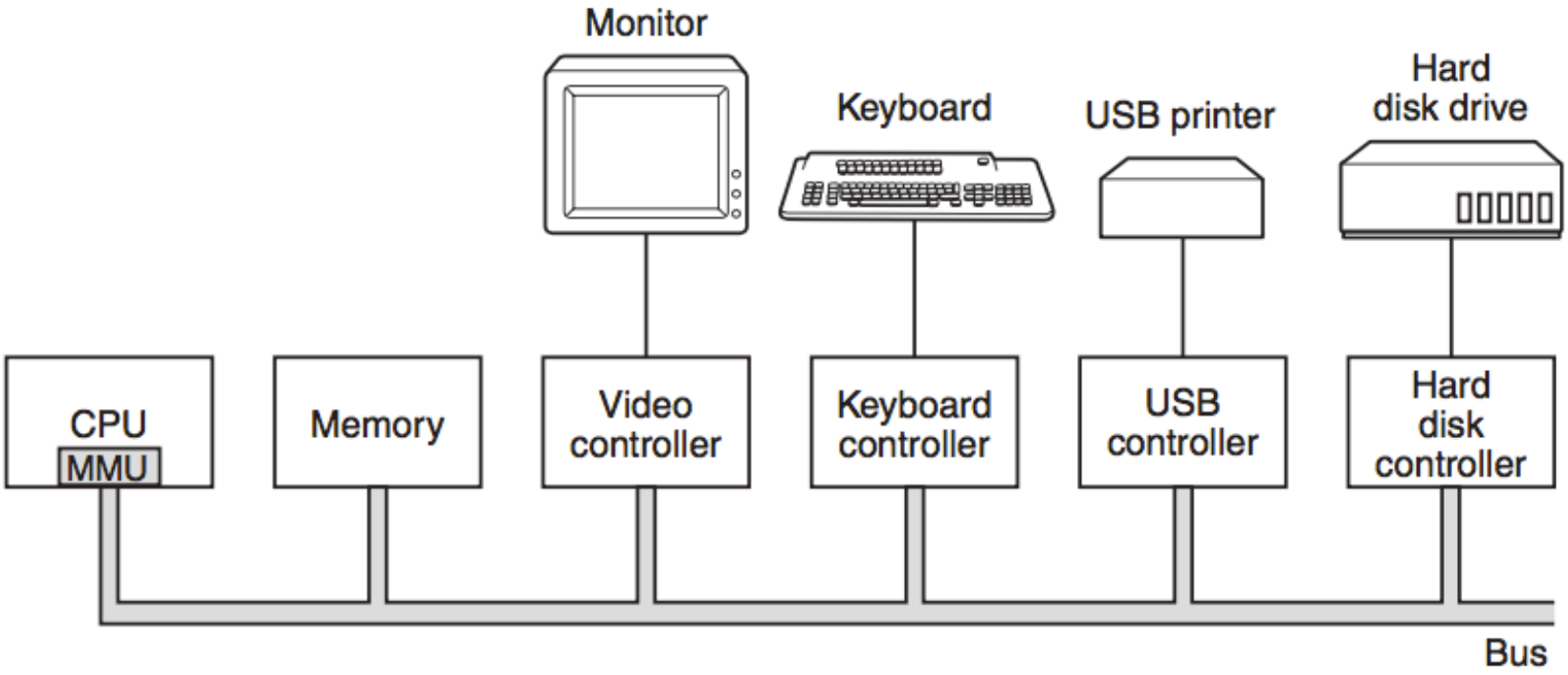


Figure 1-6. Some of the components of a simple PC

Problem 5.1 – Solution

- In the figure, we see controllers and devices as separate units. The reason is to allow a controller to handle multiple devices, and thus eliminate the need for having a controller per device
- If controllers become cheap enough, it will be simpler just to build the controller into the device
- This design will also allow multiple transfers in parallel and thus give better performance

Problem 5.2 (1/2)

- Given the speeds listed in Fig. 5-1, is it possible to scan documents from a scanner and transmit them over an 802.11g network at full speed? Explain

Problem 5.2 (2/2)

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

Figure 5-1. Some typical device, network, and bus data rates

Problem 5.2 – Solution

- The 802.11g specification is a standard for wireless local area networks (WLANs) that offers transmission over relatively short distances at up to 54 megabits per second (**Mbps**), compared with the 11 **Mbps** theoretical maximum with the earlier 802.11b standard.
  (Source: https://searchmobilecomputing.techtarget.com/definition/80211g)

- Scanner: According to the table, it's data rate is 1 MB/s

Speed discrepancy: MB/s and Mb/s difference
(Source: https://www.softperfect.com/contact/knowledgebase.php?article=10)

Problem 5.3 (1/2)

- Figure 5-3(b) shows one way of having memory-mapped I/O even in the presence of separate buses for memory and I/O devices, namely, to first try the memory bus and if that fails try the I/O bus. A clever computer science student has thought of an improvement on this idea: try both in parallel, to speed up the process of accessing I/O devices.
- What do you think of this idea?

Problem 5.3 (2/2)

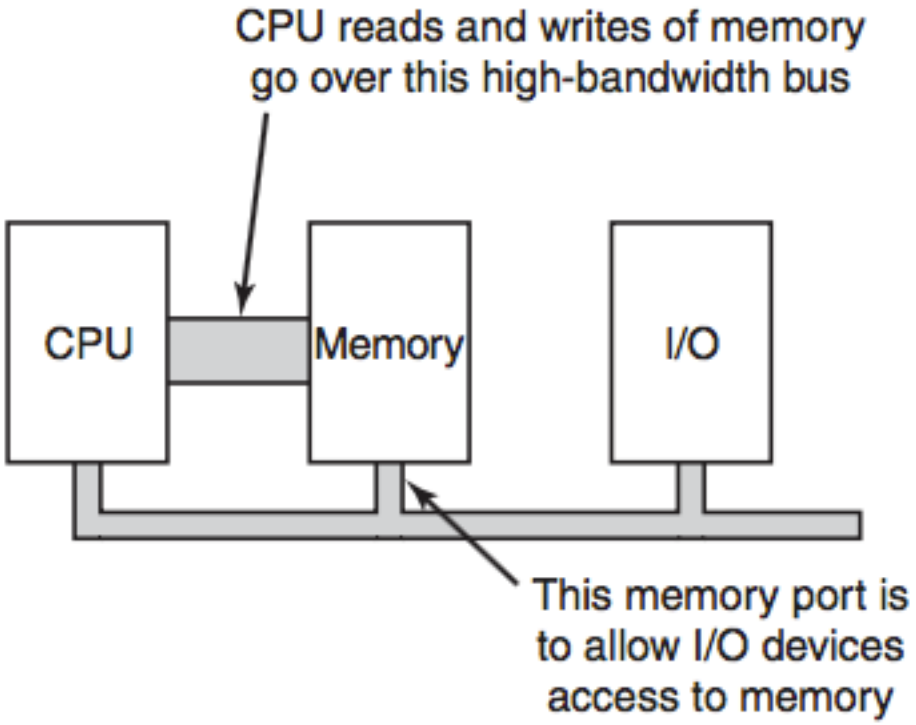


Figure 5-3(b) A dual-bus memory architecture

Problem 5.3 – Solution

- A normal memory request. The memory bus finishes first, but the I/O bus is still busy.
  - If the CPU waits until the I/O bus finishes, it has reduced memory performance
  - If it just tries the memory bus for the second reference, it will fail if this one is an I/O device reference
  - If there were some way to immediately cancel the previous I/O bus reference to try the second one, the improvement might work (there is never such an option)
- All in all, it is a bad idea.

Problem 5.5 (1/2)

- A DMA controller has five channels. The controller is capable of requesting a 32-bit word every 40 nsec. A response takes equally long.

- How fast does the bus have to be to avoid being a bottleneck?

Problem 5.5 (2/2)

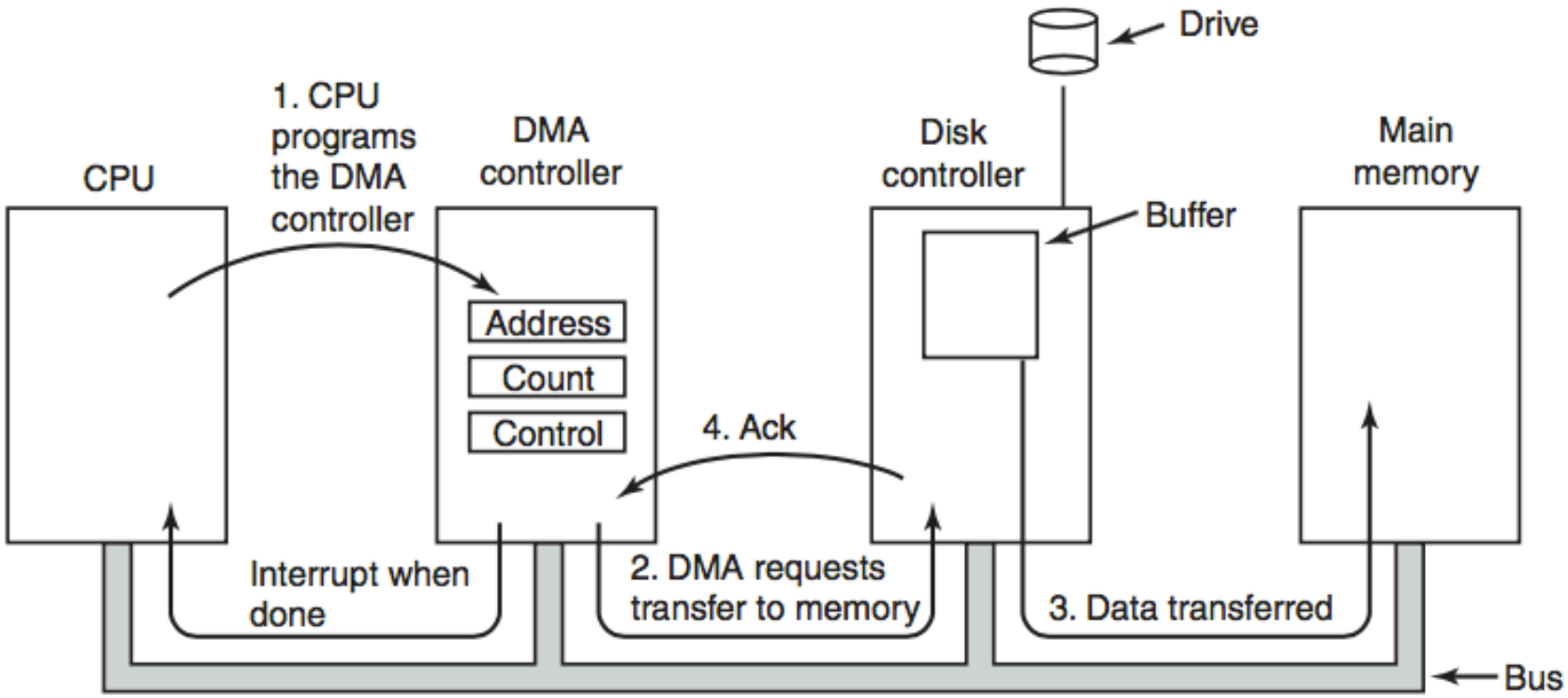


Figure 5-4. Operation of a DMA transfer

Problem 5.5 – Solution (1/2)

- Given:
  - 5 channels
  - 32-bit word is requested every 40 nsec. A response takes 40 nsec too
- The round trip time for a request is 80 nsec
- Data transfer rate is:

  32 bit/80 nsec = 4 bit/10 nsec = 4 bit * 10^{+8} sec = 400 Mbit/sec = 50 MB/sec

Problem 5.5 – Solution (2/2)

- This is the speed that the bus should have to avoid being a bottleneck
- Having five channels is irrelevant, since each request and response takes the bus

Problem 5.6

- Suppose that a system uses DMA for data transfer from disk controller to main memory. Further assume that it takes *t1* nsec on average to the DMA acquire the bus and *t2* nsec to transfer one word over the bus (*t1* >> *t2*). After the CPU has programmed the DMA controller, how long will it take to transfer 1000 words from the disk controller to main memory, if
  A. word-at-a-time mode is used?
  B. burst mode is used?
- Assume that commanding the disk controller requires acquiring the bus to send one word and acknowledging a transfer also requires acquiring the bus to send one word

Problem 5.6 – Solution (1/3)

- Word-at-a-time mode: the DMA controller requests the transfer of one word and gets it. If the CPU also wants the bus, it has to wait
- Burst mode: the DMA controller tells the device to acquire the bus, issue a series of transfers, then release the bus

Problem 5.6 – Solution (2/3)

- **Word-at-a-time mode:**
  - Acquire the bus (step 2 in Figure 5-4): *t1* + *t2*
  - Transfer the word (step 3): *t1* + *t2*
  - Acknowledge (step 4): *t1* + *t2*
  - *T = (t1 + t2 + t1 + t2 + t1 + t2) × 1000 =*

    = 3000 × (t1 + t2) nsec

Problem 5.6 – Solution (3/3)

- **Burst mode:**
  - Acquire the bus (step 2): $t_1 + t_2$
  - Transfer the word (step 3): $t_1 + 1000 \times t_2$
  - Acknowledge (step 4): $t_1 + t_2$
  - $T = (t_1 + t_2) + (t_1 + 1000 \times t_2) + (t_1 + t_2) =$

    $= 3 \times t_1 + 1000 \times t_2$ nsec

Problem 5.8

- Suppose that a computer can read or write a 32-bit memory word in 5 nsec. Also suppose that when an interrupt occurs, all 32 CPU registers, plus the program counter and the Program Status Word (PSW of 64 bits) are pushed onto the stack.
- What is the maximum number of interrupts per second this machine can process?

Problem 5.8 – Solution (1/2)

- An interrupt requires pushing CPU registers, program counter and Program Status Word onto the stack. Returning from the interrupt requires fetching 34 words from the stack.

"If no other interrupts are pending, the interrupt controller handles the interrupt immediately. However, if another interrupt is in progress, or another device has made a simultaneous request on a higher-priority interrupt request line on the bus, the device is just ignored for the moment. In this case it continues to assert an interrupt signal on the bus until it is serviced by the CPU.
(…) By having the CPU delay this acknowledgement until it is ready to handle the next interrupt, race conditions involving multiple (almost simultaneous) interrupts can be avoided."
(TB pages 347-8)

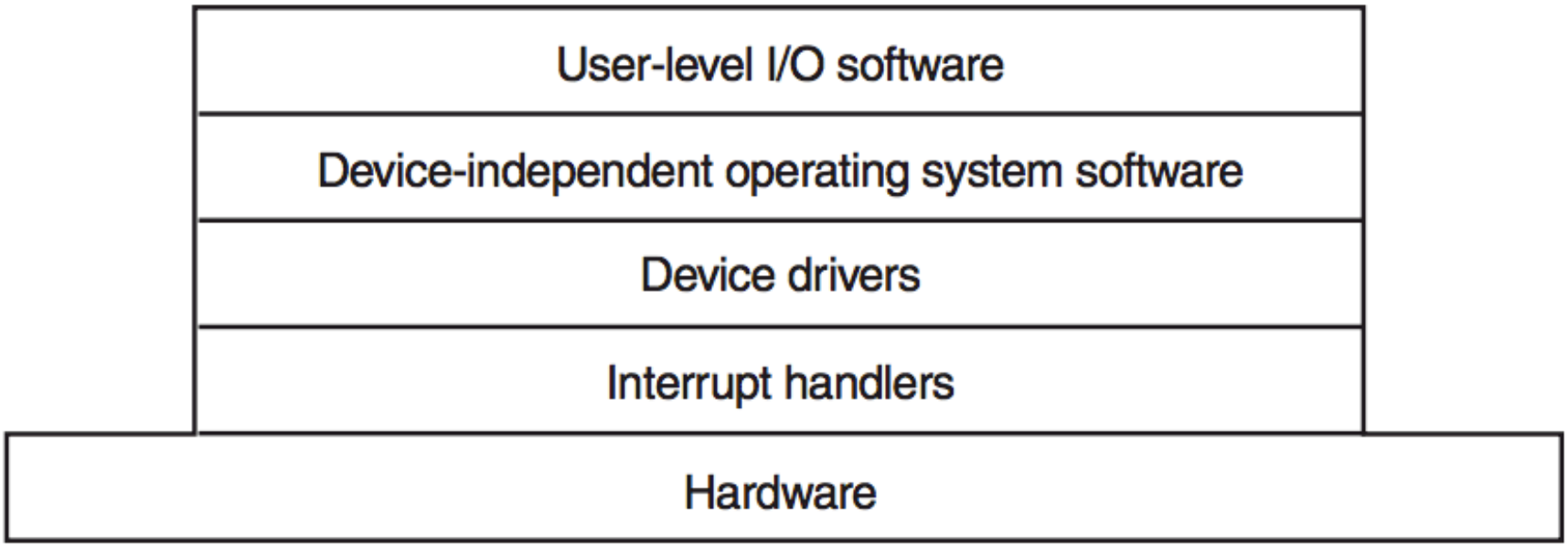
Problem 5.8 – Solution (2/2)

- The maximum number of interrupts per second this machine can process can be calculated as follows:
  - In this machine there are 34 words and 1 program counter
  - That is, total of 35 I/O operations where each takes 10 nsec take 350 nsec ($350 * 10^{-9}$ seconds) in total
  - Max number of interrupts per second: $1/(350 * 10^{-9}) = 2.85 * 10^{6}$
- Therefore, the machine can handle 2.85 million interrupts per second

Problem 5.14 (1/2)

- In which of the four I/O software layers is each of the following done?
  A. Computing the track, sector, and head for a disk read
  B. Writing commands to the device registers
  C. Checking to see if the user is permitted to use the device
  D. Converting binary integers to ASCII for printing

Problem 5.14 (2/2)



Layers of the I/O software system

Problem 5.14 – Solution (1/5)

- The User Level I/O software
  - As the name suggests, this layer works as an interface between user programs and operating system. The user makes call to library procedures. This layer converts these calls into system calls. With this, the security of system is made even if CPU or memory access is needed by the user process

Problem 5.14 – Solution (2/5)

- The Device Independent I/O software
  - The device drivers belonging to same class have some similarities among them. Such drivers are used for devices that have similar functionality and can be handled by same driver
  - In this layer, all such drivers are implemented and it acts as an interface between device driver and kernel

Problem 5.14 – Solution (3/5)

- Device Dependent I/O software or **Device Driver Layer**
  - Unlike the device independent layer, in device driver layer, specifications for driver for each device is provided
  - This layer becomes an interface between device independent layer and kernel. All calls from device independent layer are converted to device specific calls. The job of this layer is to translate request through the device-independent standard interface into appropriate sequence of commands for the particular hardware

Problem 5.14 – Solution (4/5)

- The Interrupt Handler
  - Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
  - Set up a context for the interrupt-service procedure. Doing this may involve setting up the Translation Lookaside Buffer (TLB), MMU and a page table.
  - Set up a stack for the interrupt service-procedure.
  - Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
  - Copy the registers from where they were saved (possibly some stack) to the process table.
  - Run the interrupt-service procedure. It will extract information from the interrupting device controller's registers.
  - Choose which process to run next. If the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now.
  - Set up the MMU context for the process to run next. Some TLB setup may also be needed.
  - Load the new process' registers, including its PSW.
  - Start running the new process.

Problem 5.14 – Solution (5/5)

- Computing the track, sector, and head for a disk read
  - **Device driver** layer is used
- Writing commands to the device registers
  - **Device driver** layer is used
- Checking to see if the user is permitted to use the device
  - **Device independent software**
- Converting binary integers to ASCII for printing
  - **User-level software**

End

Week 11 – Tutorial