Input/Output (I/O)

Prof Rajeev Barua

E-A 001 -- Slide set 10

PLAKSHA UNIVERSITY

How to invoke Input/Output from the ISA

- **Special hardware instructions.**
  - In this, physical hardware registers are accessed by special instructions, say IN and OUT.
  - Hardware registers need to store status and data words per device type. These registers may store bits for:
    - Is data available bit (for input devices)
    - Is device ready for output bit (for output devices)
    - Data to be input or output
    - Are interrupts to be generated upon arrival of input (for input devices)
      - Set by software beforehand. E.g.,, programs can register exception handlers.
- **Memory mapped I/O.**
  - Here we have exact same status and data words as above, but mapped to portions of the memory address space.
  - E.g., address 0x800 may be status word for a device type, and not a real address in DRAM.
    - Read and written using regular LW and SW instructions ⇒ no special instructions needed.
    - Similarly 0x804 could be the associated data word for that device type.
  - Advantage over special hardware instructions: no special ISA instructions are needed.

Methods for I/O

There are three methods for I/O, listed below:

- Programmed I/O with busy wait.
- Interrupt-driven I/O
- Direct Memory Access (DMA)

These methods are related to the scheduling of I/O, and associated hardware needed.

Programmed I/O with busy wait

In this method, blocking instructions are available to read and write a single word of I/O.

- Blocking instructions are those that do not exit until they are complete.
- In contrast, non-blocking instructions exit before they are complete, allowing later instructions to run.

For output, blocking instructions may be tolerable, especially for shorter-running I/O instructions.

But for input, we may wait for a long time with blocking instructions, since input is unpredictable.

- Disadvantage: Cannot do anything in the meantime!
    - Especially wasteful since most inputs arrive at unpredictable times.

Consequently, not used for desktops, laptops, servers; and not even in mobile phones.

But it **<u>is</u>** used in low-end embedded processors that simply respond to inputs. (E.g., a digital home thermostat).

Interrupt-driven I/O

This method aims to avoid busy waiting.

What is an interrupt?

- An interrupt is a mechanism by which the Operating System (OS) software can handle exceptional conditions during run-time such as errors and I/O events.
- To implement interrupts, the program switches to a special procedure called an interrupt handler during execution, BUT WHICH IS TRANSPARENT TO THE RUNNING PROGRAM. (Like a hardware-initiated procedure call).
- The memory address of each interrupt handler is maintained by the OS.
- Each type of interrupt has a different address for it's handler.

We will describe interrupts in more detail in a later lecture.

Can avoid busy waiting as follows

For input devices:

- CPU is interrupted by device when input data arrives, so it does not need to busy wait.

For output devices:

- CPU writes data to output device, then signals done, and then switches to other tasks, without busy waiting for the output to complete.
- When the output device is done, if the output requires an acknowledgment (ack) when done, then it generates an interrupt to let the CPU know.

Does the CPU need to request I/O?

For input, two kinds of devices:

- Input provided when requested. (Eg: disk, floppy, tape, CD-ROM).
  - Here the CPU requests the input.
- Input provided by outside world (Eg: keyboard, mouse, external sensor).
  - Here the CPU does not request the input.

For output, CPU always requests output.

Cost of interrupt-driven I/O

- Huge improvement over busy waiting
  - No need to wait for input to arrive or output to complete.
- However, interrupt handlers can be slow (often hundreds of cycles in run-time) because of tasks such as clearing the processor pipeline (especially superscalar fetched instruction windows), and saving and restoring registers.
- Really slow if interrupts needed for each word.

Direct memory access (DMA)

Programmed I/O, but not done by CPU one word at a time, instead done by DMA controller (engine) for blocks of words at once.

- The DMA controller interrupts the CPU when it’s done with the entire block.
- Advantages of a DMA controller handling I/O :
  - Enables combining multiple interrupts (for each word) into one (for a block of words).
  - DMA hardware engine eliminates software loop-control instruction overhead needed for I/O block transfers when done in CPU.
  - Leaves the CPU free to do other tasks in the meantime.

The CPU starts off each DMA transfer by specifying the following:

- <u>Start address</u> of memory block that is source or destination of transfer.
- <u>Number of bytes</u> to be transferred.
- <u>The device number</u> to communicate with.
- <u>Direction</u> of the transfer.

Direct memory access (DMA)

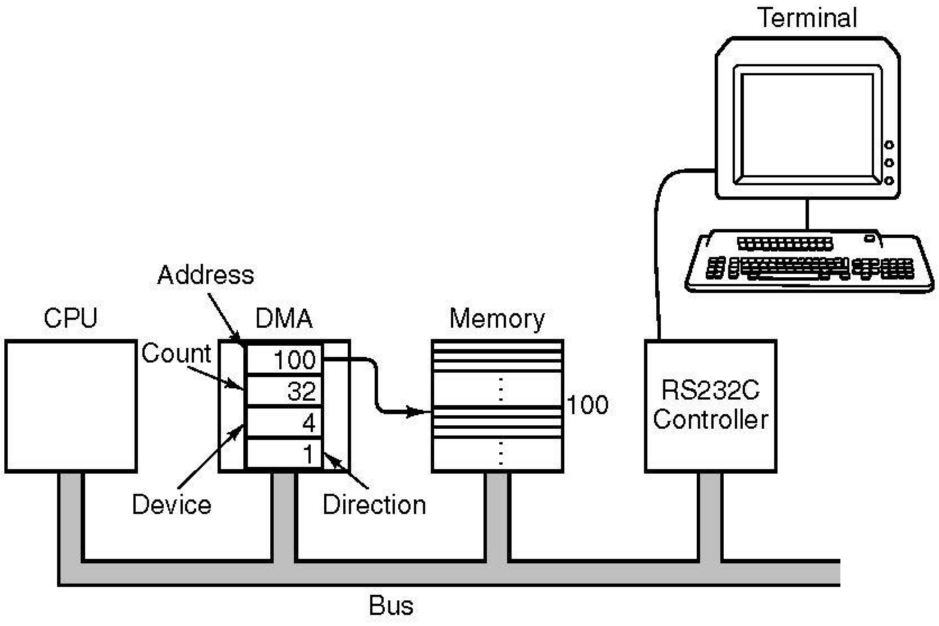

Figure 5-32. A system with a DMA controller.

Picture showing computer organization with DMA controller.

- The DMA controller is a type of small CPU for conducting I/O.
- It transfers a block of data between main memory and any I/O device.
- Here, for example, it may transfer a block of data from memory to a terminal (monitor).
- The DMA engine does the transfer word-by-word (or groups of words as wide as the system bus)
- The parameters of the DMA transfer are shown inside controller.
- RS232C is a communication standard.

Comparing DMA and interrupt-driven I/O

Advantage of DMA over interrupt IO:

- Many interrupts combined into one. (this is a big win in run-time)

Disadvantages:

- Additional DMA hardware needed (interrupts needed anyway, so that is not extra over interrupt-driven I/O).
- Bus usage by DMA controller ⇒ the bus is not available to the CPU to read memory during DMA (this is called cycle stealing, and is small loss in run-time)
  - This is not a loss over interrupts as bus would be used anyway.
  - Instead, it is just the cost of doing I/O over the shared system bus.