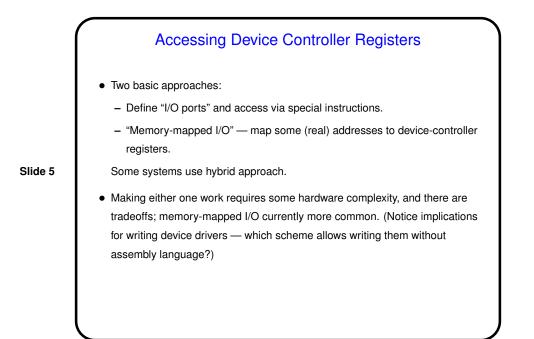


Slide 3

	I/O Hardware, Revisited
•	First, a review of I/O hardware — simplified and somewhat abstract view, mostly focusing on how low-level programs communicate with it.
•	Many, many kinds of I/O devices — disks, tapes, mice, screens, etc., etc. Car be useful to try to classify as "block devices" versus "character devices".
•	Many/most devices are connected to CPU via a "device controller" that manages low-level details — so o/s talks to controller, not directly to device.
•	Interaction between CPU and controllers is via registers in controller (write to tell controller to do something, read to inquire about status), plus (sometimes) data buffer.
	Example — parallel port (connected to printers, etc.) has control register (example bit — linefeed), status register (example bit — busy), data register (one byte of data). These map onto the wires connecting the device to the CPU.



Direct Memory Access (DMA) When reading more than one byte (e.g., from disk), device controller typically reads into internal buffer, checking for errors. How to then transfer to memory? One way — CPU makes transfer, byte by byte. Another way — DMA controller makes transfer, having been given a target memory location and a count. Which is better? consider speed of DMA versus speed of CPU, potential for overlapping data transfer and computation.

$\left(\right)$	Interrupts, Revisited
	 When I/O device finishes its work, it generates interrupt, typically actually signalling interrupt controller. Interrupt controller signals CPU, with indication of which device caused interrupt, or ignores interrupt (so device controller keeps trying) if interrupt can't be processed right now.
	 Processing is now similar to what happens on traps (interrupts generated by system calls, page faults, other errors): Hardware locates proper interrupt handler (probably using interrupt vector), saves critical info such as program counter, and transfers control (probably switching into supervisor mode).
	Interrupt handler saves other info needed to restart interrupted process, tells interrupt controller when another interrupt can be handled, and performs minimal processing of interrupt.

Slide 7

Interrupts, Revisited, A Bit More

Notice how pipelining complicates things — restarting is much easier with
precise interrupts (all instructions before interrupted one complete, none past
interrupted one complete, etc.), but these are difficult to get with pipelined
processor.

