18-447 Lecture 26: I/O

James C. Hoe
Dept of ECE, CMU
April 29, 2009

Announcements: Complete UCA online!!
Final Thursday, May 7 5:30-8:30p.m. BH 136A

Handouts:

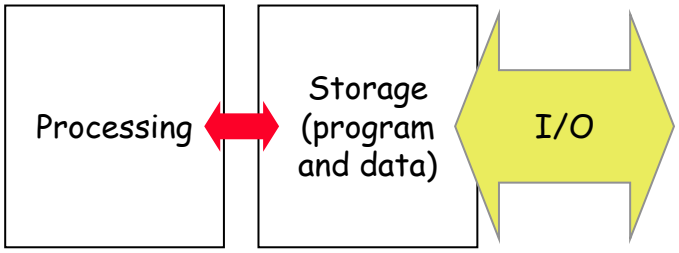
Format of the Big Quiz

- Coverage
  - lectures (L1~L26, except 1, 15, 23,24), HWs, projects, assigned readings (textbooks and papers)
- Types of questions
  - freebies: can you remember the materials
  - probing: did you understand the materials
  - applied: can you apply the materials in original thoughts
- 180 minutes, 180 points
  - if a question is worth 5 points, don't spend 20 minutes on it
  - skip questions you can't do and come back to them later
  - closed-book, **three** 2-sided 8½x11 crib sheets
  - no calculators

*** Use pencil or black/blue ink only

- Study what you missed on little Quiz 1 and Quiz 2

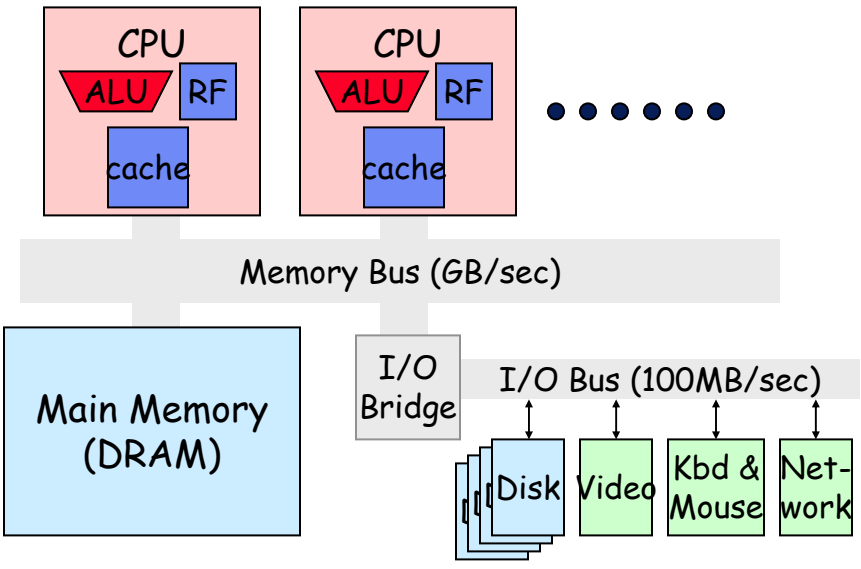
Processor I/O



How does a computer communicate with the outside world?

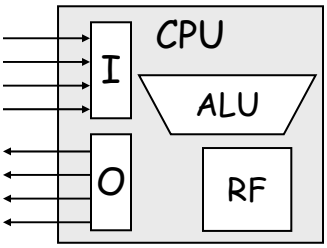
- Reasons for I/O
  - User Interface: keyboard, mouse, video display
  - Data transfer: disk, tapes, punch cards
  - Communication: network interface
  - Sensor/control

How do CPUs talk to the I/O?



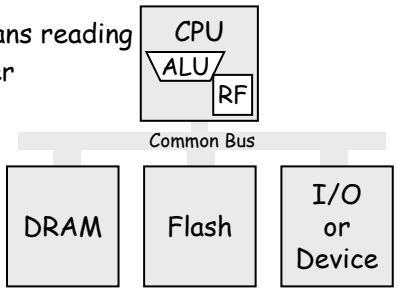
I/O Port Registers (special purpose)

- Special input and output registers as part of the ISA programmer-visible state
  - **output**: values written to an output register appear on the output pins of an external port
  - **input**: reading from an input register returns the values at the input pins of an external port
  - common scheme on microcontrollers
- Simple
  - easy to use, low latency
  - can be specialized for applications
- Not general
  - predetermined number of I/O signals
  - specialized for whose application?



Memory Mapped I/O: a general approach

- Load/store instructions perform I/O to memory from the processor's perspective
  - ld/st address identifies a specific memory location
  - ld/st data conveys information
- "Map" a subset of the "unused" memory addresses (e.g., the high ones) to registers of external devices
  - LW from a "mmap" address means reading from the corresponding register
  - similar for SW
- Memory and devices on the bus are programmed to respond only to their own address ranges

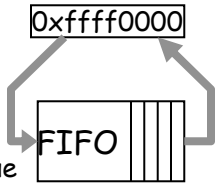


Side-effects and Idempotency

- Memory load/store semantics are idempotent
  - recall, memory semantics says when I "read" location M[A], I should get back the last value I "wrote" to location M[A]

  ⇒ writing to M[A] once has the same effect as writing to M[A] with the same value 10 times in a row

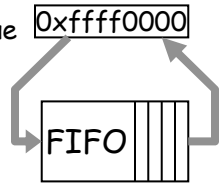
  ⇒ reading from M[A] always return the same value unless I write something else to M[A]
- mmap load/store often have side-effects
  - the receiver of mmap load/store may not be memory-like!!
  - the action of reading or writing a device register can implied other state changes
  - consider a FIFO example
    - SW to 0xffff0000 pushes the store value
    - LW from 0xffff0000 returns the popped value



Interaction with Caches

- Review: automatically managed cache hierarchy
  - keeps copies of recently used memory locations on chip
  - issuing a LW or SW to a cacheable address may not lead to an external bus transaction
  - a cacheable address may be appear in a bus transaction "spontaneously" without being directed by either a LW or SW

    No problem for memory. Very bad news for mmap I/O
- if 0xffff0000 is allowed to be cached
  - reading 0xffff0000 may return the old value copied into the cache previously
  - spontaneous read/write bus transactions changes the FIFO state unexpectedly



Solution: disallow caching on mmap addresses

Direct Memory Access

- I/O devices are not always dumb and passive
- mmap I/O is slow and consumes CPU cycles

  ⇒ Let the I/O device read/write memory directly to move large data blocks to/from memory!!
- Commands to a DMA device
  - "read (or write) 1024 KBytes starting from location 0x54100"
  - CPU issue commands using mmap writes to device registers

  How do you know when a DMA transfer is finished?
- Commands to a DMA "engine"
  - "copy a source block (i.e., base & size) to a destination block"
  - source and destination region could be memory or mmap'ed

  Allows DMA with mmap-only devices

Interaction with VM

- A contiguous block in VA
  - is not necessarily contiguous in PA
  - may not be in memory at all
- Software solutions
  - user must allocate special pages if it is intended for DMA transfer later
  - kernel copies from user buffer to pinned, contiguous buffer before DMA
- Smarter DMA engines
  - OS creates in memory a "linked list" of commands for moving non-contiguous blocks
  - DMA engine follows linked list to perform gather/scatter
- Virtually-address I/O bus
  - I/O devices refer to contiguous data blocks by VA
  - translate by I/O TLB into PA before accessing main memory
  - can TLB-miss or page-fault

Device Servicing Schemes

the mailman did it!

- You are waiting at home for mail to be delivered
- How do you find out when the mail is delivered?
  - keep going outside to check the mailbox
  - rig the mailbox door to ring your cell phone
- Which method is better?
  - easier to implement
  - less work for you
  - lets you know sooner
- What if
  - you are very busy
  - you know roughly when mail should come
  - the mail is extremely urgent
  - many deliveries each day

also consider inverse cases

Polling I/O

- Consider a keyboard device with 2 mmap registers
  - **READY**: a read returns true if a new character is available
  - **DATA**: a read returns the new character typed and
    resets **READY** if no more characters are available
- Polling-based service routine

```
_checkkbd:   LW   r16 _READY
             BEQ  r16 r0 _end
             LW   r3 _DATA
             JAL  _keystroke
_end:        JR   r31
```

  - must be called frequently and repeatedly from an outer loop
  - most of the time nothing happens
  - inefficient for infrequent, but latency-sensitive I/O events

Interrupts
(remember Lecture 14?)

- Interrupt Vector
  - a set of input pins that can be asserted by I/O devices who need "servicing"
  - In other words, let the device ring a door-bell at the CPU

    Who and when to answer the door?
- Can't ask the executing thread to keep checking if there is an interrupt (otherwise, same as polling)
- Better idea, only when interrupts are asserted
  - stop the running thread
  - switch to a special program (aka, interrupt handler) to service the interrupt
  - return to the running thread

Interrupt-Driven I/O

- Recall the keyboard device with 2 mmap registers
  - **READY**: a read returns true if a new character is available
  - **DATA**: a read returns the new character typed and resets **READY** if no more characters are available
- Now, add interrupt capability to signal readiness
- Instead of polling, `_checkkbd` is called by the interrupt handler only if the corresponding interrupt line is raised
- Interrupt-Driven I/O is suitable for
  - very infrequent events (e.g. any human input interface)
  - very long-latency operations (e.g. signaling the end of a DMA transfer)

Which I/O Mechanism to use?

- First of all, you are limited by what is available

  Hopefully who ever designed the processor or I/O device had the right application in mind
- Performance considerations
  - I/O Bandwidth = transfer size / transfer time
  - Transfer time = { overhead }+{transfer size / raw_bandwidth}
    - DMA has high raw bandwidth but large setup overhead
    - mmap I/O has low bandwidth but no overhead
- CPU considerations
  - What fraction of processing is lost to I/O operations?
  - Can the CPU be doing something else useful while I/O is happening
  - How long can afford to let I/O wait?