CS61C – Machine Structures

Lecture 36 - Input/Output

4/21/2006

John Wawrzynek

(www.cs.berkeley.edu/~johnw)

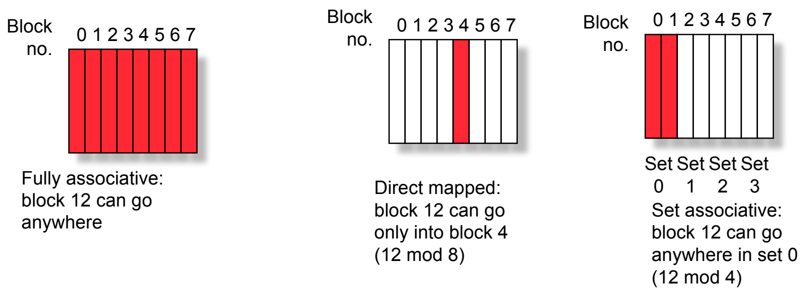
www-inst.eecs.berkeley.edu/~cs61c/

4 Qs for any Memory Hierarchy

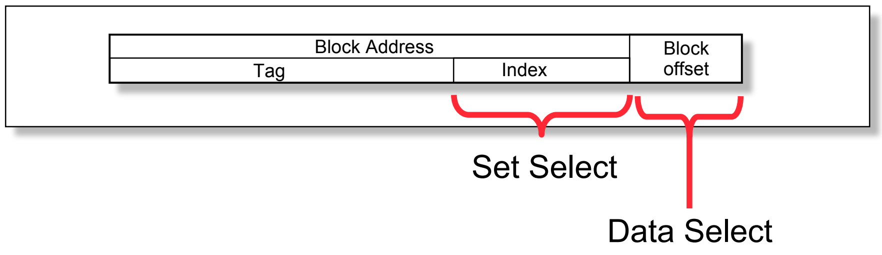
- Q1: Where can a block be placed?
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)
- Q2: How is a block found?
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)
- Q3: Which block is replaced on a miss?
  - Least recently used (LRU)
  - Random
- Q4: How are writes handled?
  - Write through (Level never inconsistent w/lower)
  - Write back (Could be "dirty", must have dirty bit)

Q1: Where block placed in upper level?

- Block #12 placed in 8 block cache:
  - Fully associative
  - Direct mapped
  - 2-way set associative
    - Set Associative Mapping = Block # Mod # of Sets


Block no.
0 1 2 3 4 5 6 7
Fully associative: block 12 can go anywhere
Block no.
0 1 2 3 4 5 6 7
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Block no.
0 1 2 3 4 5 6 7
Set Set Set Set
0 1 2 3
Set associative: block 12 can go anywhere in set 0 (12 mod 4)

Q2: How is a block found in upper level?



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

Q3: Which block replaced on a miss?

° Easy for Direct Mapped

° Set Associative or Fully Associative:

- Random
- LRU (Least Recently Used)

Miss Rates

Associativity:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What to do on a write hit?

° Write-through

- update the word in cache block and corresponding word in memory

° Write-back

- update word in cache block
- allow memory word to be "stale"

=> add 'dirty' bit to each line indicating that memory be updated when block is replaced

=> OS flushes cache before I/O !!!

° Performance trade-offs?

- WT: read misses cannot result in writes
- WB: no writes of repeated writes

Three Advantages of Virtual Memory

1) Translation:

- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program ("Working Set") must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later

Three Advantages of Virtual Memory

2) Protection:

- Different processes protected from each other
- Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs ⇒ Far more "viruses" under Microsoft Windows
- Special Mode in processor ("Kernel mode") allows processor to change page table/TLB

3) Sharing:

- Can map same physical page to multiple users ("Shared memory")

Why Translation Lookaside Buffer (TLB)?

- Paging is most popular implementation of virtual memory (vs. base/bounds)
- Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection
- Cache of Page Table Entries (TLB) makes address translation possible without memory access in common case to make fast

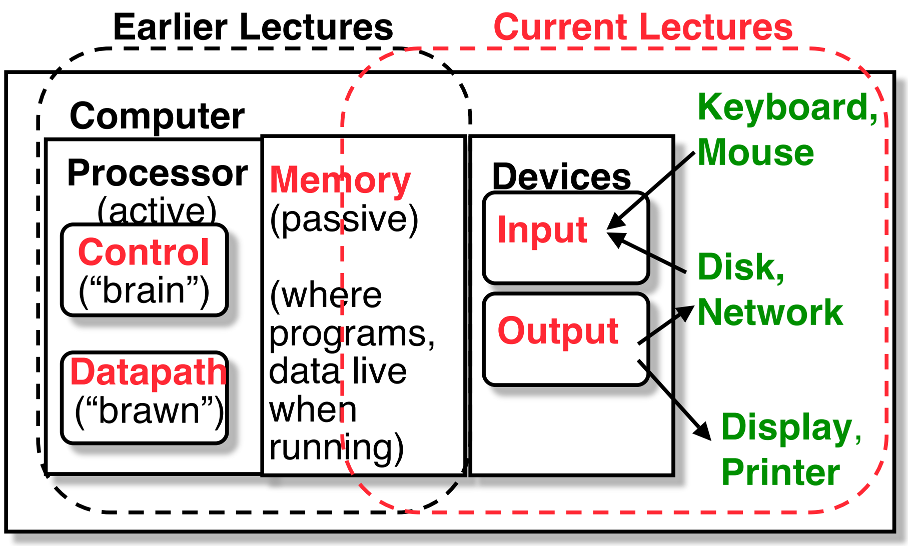
And in Conclusion…

- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
- Virtual Memory allows protected sharing of memory between processes with less swapping to disk

Administrivia

- Switch in order of lectures from original schedule
  - "Performance" will come after I/O section.
- We're late getting the homework out this week (sorry, busy with exam stuff).
  - Will be posted later today.
- New set of reading assignments posted.

Recall : 5 components of any Computer



Motivation for Input/Output

- I/O is how humans interact with computers
- I/O is how computers interconnect (Internet/www)
- I/O is how computers sense and control the environment.
- I/O gives computers long-term memory.
- Computer without I/O like a car without wheels; great technology, but won't get you anywhere

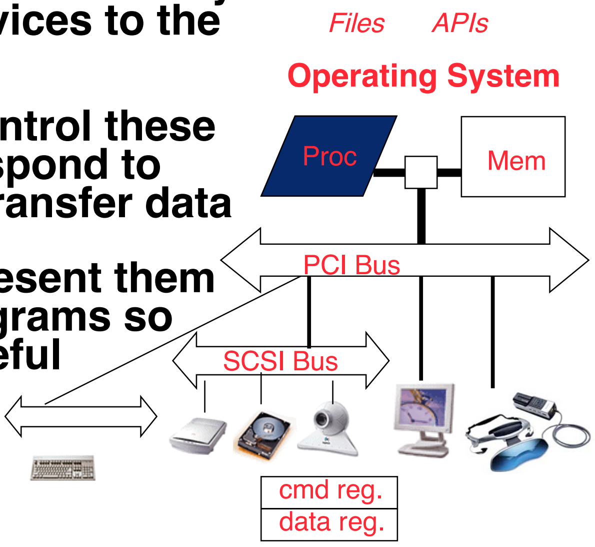
I/O Device Examples and Speeds

- I/O Speed: bytes transferred per second (from mouse to Gigabit LAN: 10-million-to-1)

Device	Behavior	Partner	Data Rate (KBytes/s)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Wireless Network	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00
Wired LAN Network	I or O	Machine	125,000.00

What do we need to make I/O work?

- A way to connect many types of devices to the Proc-Mem
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful

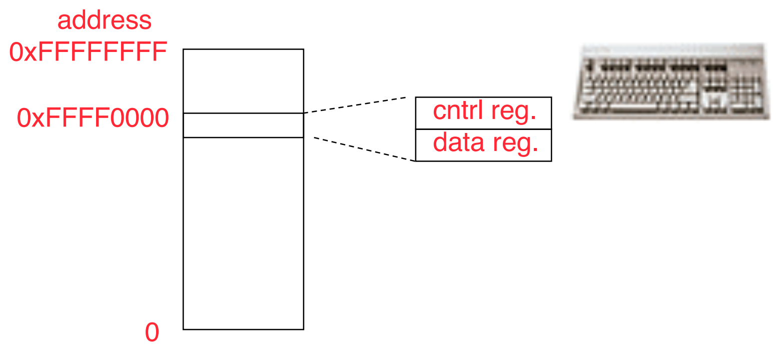


Instruction Set Architecture for I/O

- What must the processor do for I/O?
  - Input: reads a sequence of bytes
  - Output: writes a sequence of bytes
- Some processors have special input and output instructions
- Alternative model (used by MIPS):
  - Use loads for input, stores for output
  - Called "Memory Mapped Input/Output"
  - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)

Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices



Processor-I/O Speed Mismatch

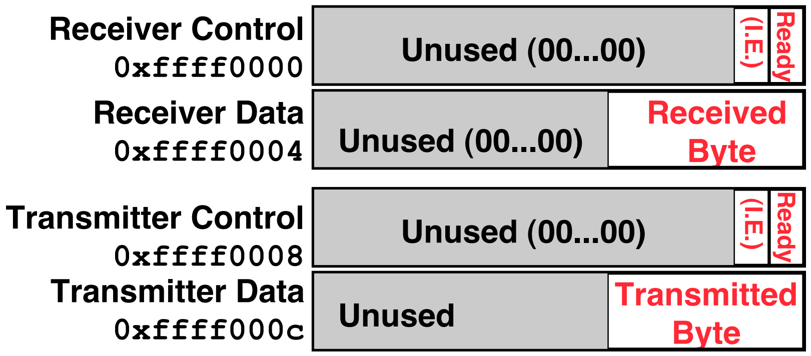
- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate
  - I/O devices data rates range from 0.01 KB/s to 125,000 KB/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device not be ready to accept data as fast as processor stores it
- What to do?

Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - Control Register, says it's OK to read/write (I/O ready) [think of a flagman on a road]
  - Data Register, contains data
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg (0 ⇒ 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit (1 ⇒ 0) of Control Register

SPIM I/O Simulation

- SPIM simulates 1 I/O device: memory-mapped terminal (keyboard + display)
  - Read from keyboard (receiver); 2 device regs
  - Writes to terminal (transmitter); 2 device regs



SPIM I/O

- Control register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register not yet been read;
    1 ⇒ 0 when data is read from Data Reg
  - Transmitter: Ready==1 means transmitter is ready to accept a new character;
    0 ⇒ Transmitter still busy writing last char
    - I.E. bit discussed later
- Data register rightmost byte has data
  - Receiver: last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display

I/O Example

- Input: Read from keyboard into `$v0`

```
              lui  $t0, 0xffff #ffff0000
Waitloop:     lw   $t1, 0($t0) #control
              andi $t1,$t1,0x1
              beq  $t1,$zero, Waitloop
              lw   $v0, 4($t0) #data
```

- Output: Write to display from `$a0`

```
              lui  $t0, 0xffff #ffff0000
Waitloop:     lw   $t1, 8($t0) #control
              andi $t1,$t1,0x1
              beq  $t1,$zero, Waitloop
              sw   $a0, 12($t0) #data
```

- Processor waiting for I/O called "Polling"
- "Ready" bit from processor's point of view!

Cost of Polling?

° Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling

- Mouse: polled 30 times/sec so as not to miss user movement
- Floppy disk: transfers data in 2-Byte units and has a data rate of 50 KB/second. No data transfer can be missed.
- Hard disk: transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.

% Processor time to poll [p. 677 in book]

Mouse Polling, Clocks/sec

= 30 [polls/s] * 400 [clocks/poll] = 12K [clocks/s]

° % Processor for polling:

$12*10^3$ [clocks/s] / $1*10^9$ [clocks/s] = 0.0012%

⇒ Polling mouse little impact on processor

Frequency of Polling Floppy

= 50 [KB/s] / 2 [B/poll] = 25K [polls/s]

° Floppy Polling, Clocks/sec

= 25K [polls/s] * 400 [clocks/poll] = 10M [clocks/s]

° % Processor for polling:

$10*10^6$ [clocks/s] / $1*10^9$ [clocks/s] = 1%

⇒ OK if not too many I/O devices

% Processor time to poll hard disk

Frequency of Polling Disk

= 16 [MB/s] / 16 [B] = 1M [polls/s]

- Disk Polling, Clocks/sec
  = 1M [polls/s] * 400 [clocks/poll]
  = 400M [clocks/s]
- % Processor for polling:

  $400*10^6$ [clocks/s] / $1*10^9$ [clocks/s] = 40%

  ⇒ Unacceptable

What is the alternative to polling?

- Wasteful to have processor spend most of its time "spin-waiting" for I/O to be ready
- Would like an unplanned procedure call that would be invoked only when I/O device is ready
- Solution: use exception mechanism to help I/O. Interrupt program when I/O ready, return when done with data transfer

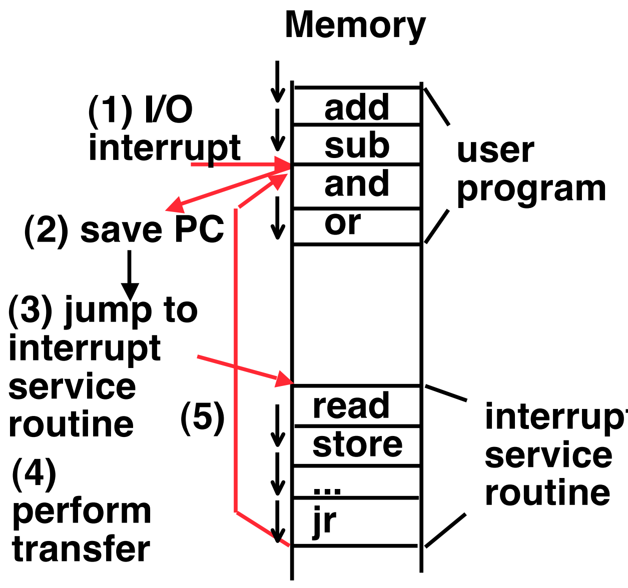
I/O Interrupt

- An I/O interrupt is like overflow exceptions except:
  - An I/O interrupt is "asynchronous"
  - More information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  - I/O interrupt does not prevent any instruction from completion

Definitions for Clarification

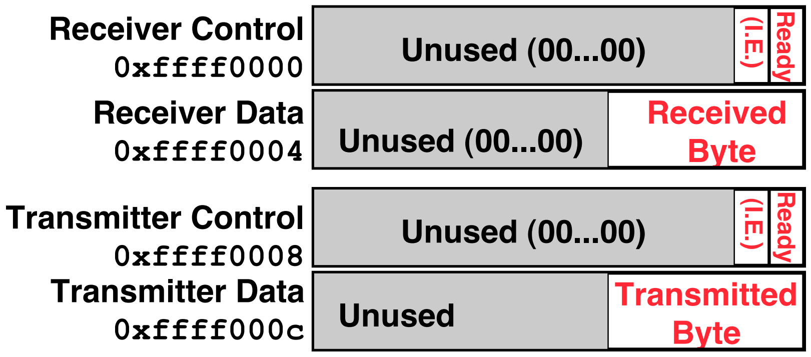
- Exception: signal marking that something "out of the ordinary" has happened and needs to be handled
- Interrupt: asynchronous exception
- Trap: synchronous exception
- Note: Many systems folks say "interrupt" to mean what we mean when we say "exception".

Interrupt Driven Data Transfer



SPIM I/O Simulation: Interrupt Driven I/O

- I.E. stands for Interrupt Enable
- Set Interrupt Enable bit to 1 have interrupt occur whenever Ready bit is set



Benefit of Interrupt-Driven I/O

- Find the % of processor consumed if the hard disk is only active 5% of the time. Assuming 500 clock cycle overhead for each transfer, including interrupt:
    - Disk Interrupts/s = 16 MB/s / 16B/interrupt
      = 1M interrupts/s
    - Disk Interrupts, clocks/s
      = 1M interrupts/s * 500 clocks/interrupt
      = 500,000,000 clocks/s
    - % Processor for during transfer:
      $500*10^6 / 1*10^9 = 50\%$
- Disk active 5% ⇒ 5% * 50% ⇒ 2.5% busy

"And in conclusion…"

- I/O gives computers their 5 senses
- I/O speed range is 100-million to one
- Processor speed means must synchronize with I/O devices before use
- Polling works, but expensive
    - processor repeatedly queries devices
- Interrupts works, more complex
    - devices causes an exception, causing OS to run and deal with the device
- I/O control leads to Operating Systems