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

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

<table>
<thead>
<tr>
<th>Size</th>
<th>LRU 16 KB</th>
<th>LRU 64 KB</th>
<th>LRU 256 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ran</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
</tr>
<tr>
<td>Ran</td>
<td>4.7%</td>
<td>5.3%</td>
<td>4.7%</td>
</tr>
<tr>
<td>Ran</td>
<td>4.4%</td>
<td>5.0%</td>
<td>4.4%</td>
</tr>
<tr>
<td>Ran</td>
<td>5.3%</td>
<td>4.4%</td>
<td>5.0%</td>
</tr>
</tbody>
</table>

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

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

Computer

Processor (active)
  Control ("brain")
  Datapath ("brawn")

Memory (passive)
  (where programs, data live when running)

Devices
  Input
  Output

Keyboard, Mouse
Disk, Network
Display, Printer
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)

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

<table>
<thead>
<tr>
<th>Receiver Control</th>
<th>Unused (00...00)</th>
<th>Ready (I.E.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffffff0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver Data</td>
<td>Unused (00...00)</td>
<td>Received Byte</td>
</tr>
<tr>
<td>0xffffffff0004</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmitter Control</th>
<th>Unused (00...00)</th>
<th>Ready (I.E.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffffff0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmitter Data</td>
<td>Unused</td>
<td>Transmitted Byte</td>
</tr>
<tr>
<td>0xffffffff000c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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

```assembly
lui $t0, 0xffff  #ffffff0000
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

```assembly
lui $t0, 0xffff  #ffffff0000
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

\[
\text{Frequency of Polling Disk} = \frac{16 \text{ [MB/s]}}{16 \text{ [B]}} = 1 \text{M [polls/s]}
\]

° Disk Polling, Clocks/sec

\[
\text{Disk Polling, Clocks/sec} = 1 \text{M [polls/s]} \times 400 \text{ [clocks/poll]}
\]

\[
\text{Disk Polling, Clocks/sec} = 400 \text{M [clocks/s]}
\]

° % Processor for polling:

\[
\frac{400 \times 10^6 \text{ [clocks/s]}}{1 \times 10^9 \text{ [clocks/s]}} = 40\%
\]

⇒ 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

(1) I/O interrupt
(2) save PC
(3) jump to interrupt service routine
(4) perform transfer
(5) jump to interrupt service routine

Memory

add
sub
and
or

user program

read
store
jr

interrupt service routine

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

<table>
<thead>
<tr>
<th>Receiver Control</th>
<th>0xffff0000</th>
<th>Unused (00...00)</th>
<th>Unused (00...00)</th>
<th>Receiver Data</th>
<th>0xffff0004</th>
<th>Unused (00...00)</th>
<th>Received Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitter Control</td>
<td>0xffff0008</td>
<td>Unused (00...00)</td>
<td>Unused (00...00)</td>
<td>Transmitter Data</td>
<td>0xffff000c</td>
<td>Unused</td>
<td>Transmitted Byte</td>
</tr>
</tbody>
</table>
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